A local-search based algorithm for the ICAPS 2021 Dynamic Pickup and Delivery Problem

Markó Horváth[a,*], Tamás Kis[a], Péter Györgyi[a]

[a]*Institute for Computer Science and Control, H-1111 Budapest, Kende u. 13-17.*

Abstract

In this paper we consider a dynamic vehicle routing problem with pickups and deliveries which was introduced in a competition hosted by the International Conference on Automated Planning and Scheduling in 2021. Given a homogeneous fleet of vehicles to serve pickup-and-delivery requests arriving online. Loading items has to respect the vehicle's capacity limit, while unloading items has to follow a last-in-first-out order, and the factories have a limited number of docking ports for loading and unloading, which may force the vehicles to wait. The goal is to satisfy all the requests such that a combination of tardiness penalties and traveling costs is minimized. We present a local-search based algorithm to solve this problem, and compare it to the best known methods.

Keywords: Dynamic Vehicle Routing Problem, Dynamic Pickup and Delivery Problem, Variable Neighborhood Search

1. Introduction

Dynamic vehicle routing problems constitute a rapidly developing field of transportation research, which is certified by a series of recent review papers, see e.g., Berbeglia et al. (2010); Pillac et al. (2013); Bektaş et al. (2014); Psaraftis et al. (2016); Rios et al. (2021). The growing interest is due to the rapidly developing real-world application areas, and to fact that real-time communication technologies and solutions make it possible to implement efficient algorithms in practice.

Briefly stated, the familiar *vehicle routing problem* aims to determine an optimal set of routes to be performed by a fleet of vehicles to fulfill customer demands at different locations. The problem was introduced more than 60 years ago by Dantzig and Ramser (1959), then generalized by Clarke and Wright (1964), and many variations have appeared since then, see e.g., Toth and Vigo (2002); Eksioglu et al. (2009); Braekers et al. (2016); Zhang et al. (2022). The

*Corresponding author

Email addresses: marko.horvath@sztaki.hu (Markó Horváth), tamas.kis@sztaki.hu (Tamás Kis), peter.gyorgyi@sztaki.hu (Péter Györgyi)

variant where each customer request has a given origin and a given destination is called the *one-to-one pickup and delivery problem*, using the terminology of Berbeglia et al. (2010). In addition, if the vehicles can serve more than one request at a time, the problem can be categorized as the *vehicle routing problem with pickups and deliveries*.

A vehicle routing problem is *dynamic*, if the requests are not known in advance, instead, they arrive in an online fashion over the planning horizon, e.g., over one day. The dynamic nature brings new challenges, and makes the problem more complex. Most importantly it requires to make decisions also in an online fashion, namely, upon the arrival of new requests, either the vehicle routes have to be changed, or new routes have to be planned on top of the already planned ones. However, these decisions have to be made quickly, since the more time is spent on searching for good solutions, the less time remains for reacting to the changes taken place in the meantime. Thus only a limited computation time (a few minutes) is available for updating the routes of the vehicles, and thus the exact methods devised for solving the static problems typically cannot be applied in the dynamic variants. Instead, heuristic methods are applied, such as ant colony optimization (Montemanni et al., 2005), local search and variable neighborhood search based methods (Branchini et al., 2009; Xu et al., 2013), genetic algorithms (Elhassania et al., 2014), particle swarm optimization (Demirtaş et al., 2015), etc.

Assessing the performance of an online algorithm is not straightforward. An option is to apply competitive analysis, which aims to determine the maximum ratio between the value of the online solution and the offline optimum in the worst case. However, this type of analysis focuses on the worst case scenario, which is typically far from the real-life cases, moreover, it is hard or even impossible to apply for complex problems and algorithms. Thus, in most of the cases a simulator is also developed for assessing the merits of different algorithms for the dynamic problem.

Dynamic vehicle routing problems with pickups and deliveries often arise in practice, for example, in courier services such as meal delivery services (Reyes et al., 2018; Ulmer et al., 2021) or other same-day delivery services (Attanasio et al., 2007), in internal logistics especially when automated guided vehicles are used (Györgyi and Kis, 2019), and in external logistics (Yang et al., 2004). When the travel times for serving the requests are bounded from above, the problem is called the *dial-a-ride problem* which also has a considerable literature (Cordeau and Laporte, 2003; Ho et al., 2018; Nasri et al., 2021).

In the course of the International Conference on Automated Planning and Scheduling in 2021 (ICAPS 2021), the Dynamic Pickup and Delivery Problem (DPDP) challenge[1] was organized by Huawei Technologies Co. Ltd., where the proposed dynamic vehicle routing problem with pickups and deliveries is based on their real-life scenario. A large amount of cargoes (including raw materials, products and semi-finished goods) need to be delivered between the factories

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

every day. Due to the uncertainties of customers' requirements and production processes, most delivery requirements cannot be fully decided beforehand. The delivery orders, with the information including the pickup factories, delivery factories, the amount of cargoes and the time requirements, arrive over the day and a fleet of homogeneous vehicles are periodically scheduled to serve these orders. Due to the large amount of delivery requests, even a small improvement of the logistics efficiency can bring significant benefits. Therefore, it is of great significance to develop an efficient optimization algorithm to dispatch orders and plan the route of the vehicles.

A slight variant of the above problem was investigated in (Ma et al., 2021; Li et al., 2021). The participants of the DPDP challenge were provided with a public benchmark dataset and a simulator to dynamically evaluate their algorithms, see (Hao et al., 2022). The submitted algorithms were evaluated on a hidden dataset, and teams were ranked based on their resulted scores. The top three placed teams were invited to ICAPS 2021 to shortly present their solution approaches. These presentations along with the source codes are available on the website of the competition[2].

As it turned out, all of the top three teams of the competition applied classical techniques of Operations Research, such as constructive heuristics and local search. The winning team, Zhu et al. (2021) proposed a variable neighborhood search method for the problem. The team finished on the second place, Ye and Liang (2021) developed a rule based algorithm. Our team took third place using a local-search based method (Horváth et al., 2021). In this paper we revise and improve our solution approach, and perform computation experiments to compare it to the aforementioned ones.

The paper is organized as follows. In Section 2 we describe the problem in detail, starting with the basic static version (Section 2.1) and then the dynamic one (Section 2.2). In Section 3 we present our modeling approach, while the new solution approach is summarized in Section 4, we also discussed the similarities and differences to the methods of Zhu et al. (2021) and Horváth et al. (2021) in Section 4.3. In Section 5 we present the results of our computational experiments. Finally, we conclude the paper in Section 6.

2. Problem definition

In the next two sections we define the problem in detail. In the rest of the paper, we simply refer to this problem as the *Dynamic Pickup and Delivery Problem (DPDP)*.

2.1. The static problem

Given a set of factories, such that the distance and the traveling time between factories f_i and f_j are denoted with $\text{dist}(f_i, f_j)$ and $\text{travel}(f_i, f_j)$, respectively.

[2] `https://competition.huaweicloud.com/information/1000041411/Winning`

Given a set $\mathcal{O}$ of orders, where each order $o_i \in \mathcal{O}$ is described by a tuple $(f_i^p, f_i^d, \mathcal{I}_i, t_i^p, t_i^d)$, where f_i^p and f_i^d represent the *pickup factory* and the *delivery factory*, respectively, $\mathcal{I}_i$ is the set of *order items* (i.e., cargoes) to be delivered, t_i^p refers to the *release time* and t_i^d is the *committed completion time*. Order o_i can only be served after its release time t_i^p.

Given a fleet $\mathcal{V}$ of homogeneous vehicles to serve orders. Vehicles have a uniform loading capacity. Initially, each vehicle is empty and parks at a given factory.

Each order item has a *size*, which represents the amount of space the item occupies in a vehicle. The size of an order is the total size of its order items. An order is *splittable* if its size exceeds the uniform capacity limit of vehicles, otherwise the order is *unsplittable*. The order items of an unsplittable order must be loaded at once to a single vehicle, and must be unloaded from that vehicle at once. For each order item, the amount of time required to load and unload is also given. The loading/unloading time of an order is the total loading/unloading time of its order items. Unloading order items has to follow a last-in-first-out (LIFO) order.

Each factory has a given number of docking ports for loading and unloading. When a vehicle arrives at a factory, it approaches a free port (may have to wait until one of the ports becomes free), then some order items may be unloaded from the vehicle, and some order items may be loaded to the vehicle. Then, the vehicle can start to its next destination, if any. The dock approaching time is denoted by t^{docking}. A port is in use if a vehicle is currently approaching it, or is performing loading or unloading operations at the port, otherwise it is free. Recall that if all of the ports are in use when a vehicle arrives at a factory, it has to wait until one of them becomes free. The allocation of vehicles to ports is done on a first-come first served basis. If multiple vehicles arrive at the same factory at the same time, then their service order is determined randomly. However, in the static problem, we can use any tie-breaking rule, such as always serve the vehicle of smallest index.

In Figure 1 we depict a situation where four vehicles arrive at a factory with two docking ports. Vehicle 1 arrives at the factory at time t_1 and occupies a free docking port. Vehicle 2 arrives at time t_2 and occupies the other free port. Vehicle 3 arrives at time t_3, however, since both of the ports are in use (that is, currently vehicle 1 is under unloading at the first port, and vehicle 2 approaches the other one), it has to wait until a port is freed at time t_5. Vehicle 4 arrives at time t_4, however, both of the ports are in use, moreover, vehicle 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.

The output, i.e., the solution for the problem is the route plans of the vehicles, which are the sequences of visited factories along with the list of unloaded and loaded order items. That is, the route plan of vehicle v_k is $\Pi_k = ((f_1^k, D_1^k, P_1^k), (f_2^k, D_2^k, P_2^k), \ldots, (f_{\ell_k}^k, D_{\ell_k}^k, P_{\ell_k}^k))$, where f_i^k refers to the ith visited factory by the vehicle, D_i^k and P_i^k are the lists of order items delivered and picked up at this visit, if any, respectively. The total distance traveled by

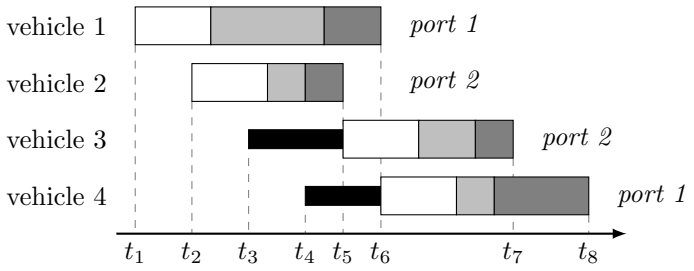


Figure 1: Example for vehicles arriving at a factory with two docking ports. White rectangles represent dock approaching, gray rectangles represent unloading and loading order items, black rectangles represent waiting for ports to become free.

vehicle v_k is $\text{dist}(\Pi_k) = \sum_{i=1}^{\ell_k - 1} \text{dist}(f_i^k, f_{i+1}^k)$.

The objective of the problem is to minimize the weighted sum of the total tardiness of serving the orders and the average traveling distance of the vehicles. For an order $o_i \in \mathcal{O}$ let C_i denote its completion time (that is, the arrival time of the order to its destination) in a solution, and $T_i = \max(0, C_i - t_i^d)$ its tardiness. Then, the objective is

$$\text{minimize } \lambda \times \sum_{o_i \in \mathcal{O}} T_i + \frac{1}{|\mathcal{V}|} \times \sum_{v_k \in \mathcal{V}} \text{dist}(\Pi_k), \tag{1}$$

where λ is a large positive constant. The value of the objective function (1) for a given schedule is called *score*. Note that the completion time of a splitted order is the arrival time of its latest items. For example, if in the situation depicted in Figure 1 vehicles 1 and 3 deliver the items of an order split in two, the completion time of that order is the arrival time of the latest vehicle carrying some items of the order, that is, t_3.

2.2. The dynamic problem

In the dynamic problem, the orders are not known a priori, but arrive online. Order o_i arrives at the release time t_i^p. Consequently, the route plans of the vehicles are built dynamically, that is, route plans are always determined for the currently known undelivered order items, and once a new subset of orders becomes known, replanning is needed.

When solving the dynamic problem, replannings occur at *update time points*, e.g., at every 10 minutes. Let t^{update} be such a time point. Although initially all vehicles are empty and parked at a factory, this may not be the case in an intermediate state of the dynamic problem, that is, at the update time vehicles may be on their routs to some factories, and may have order items already loaded on them. Thus, a status regarding to the update time is given for each vehicle as follows. Each vehicle has a *current factory* associated with a *leave time* or a *destination factory* associated with an *arrival time*, or both. Moreover, the *carrying order items*, i.e., the list of items currently carried by the vehicle in the order of loading is also given, if any.

The destination factory is the next factory which will be visited by the vehicle, if any, and the associated arrival time is the calculated time when the vehicle will arrive at that factory. Note that a vehicle is either at a factory or on the way to a factory at time t^{update}. If a vehicle is not at a factory, then it has no current factory, but it has a destination (the factory to which the vehicle is on its way). If a vehicle is parking at a factory at time t^{update}, then this factory is the *current factory* of the vehicle with leave time t^{update}. If a vehicle is at a factory at time t^{update} such that loading/unloading is not finished yet (that is, the vehicle has arrived at the factory, but waits for a docking port to become free, approaches a port, or uses a port), then this factory is the vehicle's current factory, and the leave time corresponds to the calculated time when loading/unloading is finished. If a vehicle has both of a current factory and a destination, then the arrival time associated with the destination equals to the leave time associated with the current factory plus the travel time between them, that is, the vehicle will start to the destination without any delays. If a vehicle has a destination at time t^{update}, then this cannot be changed in the course of replanning, and it must remain the fist factory visited in the updated route of the vehicle.

For an example, return to the situation depicted in Figure 1, consider vehicle 3, and assume that this vehicle visits factory f_j after it leaves the depicted factory f_i. If $t_3 \leq t^{\text{update}} \leq t_7$, then f_i is the current factory of vehicle 3 with leave time t_7, and f_j is the destination with arrival time $t_7 + \text{travel}(f_i, f_j)$. If $t_7 < t^{\text{update}} < t_7 + \text{travel}(f_i, f_j)$, then vehicle 3 has no current factory, but has destination f_j with arrival time $t_7 + \text{travel}(f_i, f_j)$.

The order items, and thus the orders also have statuses regarding the update time, namely, each item is either *finished*, *ongoing*, or *unallocated*. An order item is finished if it is already delivered to its destination, and an order is finished if all of its order items are finished. An order item is ongoing, if it is already loaded on a vehicle, but not yet delivered to its delivery factory, in other words, the ongoing order items are the carrying order items of the vehicles. Finally, an order item is unallocated if it is not yet loaded on any vehicle.

To sum up, the constraints of the problem are the followings:

- Destination constraint: the destination factory of a vehicle, if any, cannot be changed.
- Capacity constraint: the total quantity loaded on a vehicle cannot exceed its capacity.
- LIFO constraint: unloading items has to follow a last-in-first-out order.
- Splitting constraint: unsplittable orders cannot be split.

2.3. Dynamic evaluation

The organizer of the competition provided the participants a benchmark dataset from real business scenarios along with a simulator to support the dynamic evaluation of the solution approaches. Now, we briefly describe the operation of this simulator, and for details we refer to (Hao et al., 2022).

The simulator is initialized with the complete data of the problem, that is, all the orders are known a priori. In the simulator, the planning horizon is split into several time intervals with the same duration, e.g., $\Delta T = 10$ minutes. At the beginning of the current time interval, say $[t, t + \Delta T]$, the simulator passes the environment information (these are, the status of the vehicles and the order items regarding to update time t) to the developed algorithm, and invokes it. Once the algorithm outputs the planned routes, the simulator simulates the events (these are, vehicle movements, loading, unloading, etc.) in time interval $[t, t + \Delta T]$ based on the resulted routes, updates the environment information, and steps to the next time interval $[t + \Delta T, t + 2\Delta T]$, or terminates with the score of the complete solution if all the orders are finished.

Note that the simulator terminates with error, if the invoked algorithm does not output the planned routes until the end of the time interval (i.e., within ΔT minutes), or the routes violate any of the constraints.

3. Modeling approach

In this section we describe our modeling approach. The main idea is to maintain a *schedule* for the vehicles which is evaluated with an event-based simulation procedure.

3.1. Definitions and notation

Consider the current update time t^{update}, and suppose $\mathcal{O}'$ is the subset of orders known at this time point, that is, $\mathcal{O}' = \{o_i \in \mathcal{O} : t_i^p \leq t^{\text{update}}\}$. First, we group order items into *packages* subject to the following. The order items of a package belong to the same order, have the same status (ongoing or unallocated), if the items are ongoing then carried by the same vehicle, the total size of the items does not exceed the uniform capacity limit of the vehicles, and the order items of an unsplittable order are in the same package. We just split a splittable order into multiple packages and put an unsplittable order in one package. For this, consider the order items $\mathcal{I}_i$ of an order $o_i \in \mathcal{O}'$, and let $\mathcal{I}_i^f$, $\mathcal{I}_i^o$, and $\mathcal{I}_i^u$ be its partition into finished, ongoing, and unallocated items, respectively. Finished order items $\mathcal{I}_i^f$ are not relevant anymore. Recall, that ongoing items are currently carried by vehicles, thus let $\mathcal{I}_{i,k}^o$ be the set of ongoing items of $\mathcal{I}_i^o$ carried by the vehicle v_k. If such a set $\mathcal{I}_{i,k}^o$ is non-empty, then we associate a package with these items. Finally, consider the unallocated items. If $\mathcal{I}_i^u$ is non-empty, we first sort these items in non-increasing order of their size, then we put them into packages in this order respecting the capacity limit. That is, if an item does not fit into the currently used package, then we close this package, open a new one, put the item into the freshly opened package, and continue the process with the remaining items, if any. After the packages are determined, we do not modify them, that is, we do not move items between packages. Moreover, in our solution approach we require that packages not be split, that is, the order items of a package are loaded at once on a single vehicle, and are unloaded at once. Note that by this, we never violate the splitting constraint. Also note that

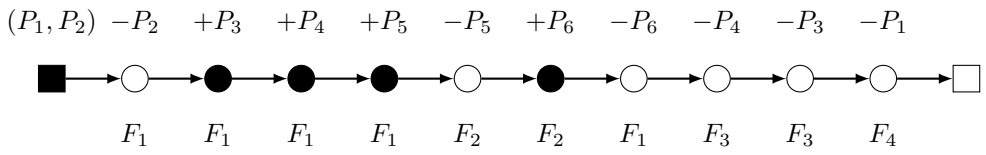


Figure 2: Example for a route. Pickup/delivery nodes are depicted with black/white circles. Head/tail is depicted with black/white rectangle.

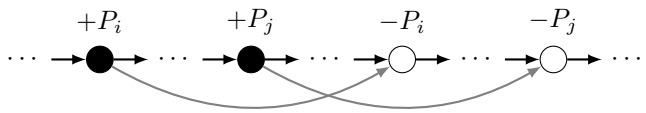


Figure 3: Example for a route violating LIFO constraint.

this packaging procedure may be too naive in general, however, it is appropriate for the provided instances (see Section 5.1).

3.2. Schedule

The schedule consists of the routes of the vehicles, where a *route* is a sequence of nodes such that each *node* refers to a pickup or a delivery of a certain package. Technically, a route is a doubly linked list with head and tail. A *pickup node* refers to the pickup of a certain package, and associated with that package and with the pickup factory of the package. A *delivery node* refers to the delivery of a certain package, and associated with that package and with the delivery factory of the package.

In Figure 2 we depict an example route, where black circles are pickup nodes, white circles are delivery nodes, and the black and white rectangles refer to the head and tail, respectively. Corresponding packages are indicated above the nodes (carrying packages are indicated above the head), and corresponding factories are indicated below the nodes. Note that carrying packages could be also depicted at the start of the route as a sequence of pickup nodes of the corresponding packages in the order of loading.

A route is *feasible* if the pickup node (which is the head in case of carrying packages) precedes the corresponding delivery node for each package, and also fulfills the LIFO, the capacity, and the destination constraints. A route satisfies the LIFO constraint if there are no two packages such that the order of pickup nodes and the order of delivery nodes are the same (see Figure 3 for an example). A route fulfills the capacity constraint if for each starting slice of the route, the total size of carrying packages, plus the total size of picked up packages, minus the total size of delivered packages does not exceed the capacity limit. A route satisfies the destination constraint if the first (not head) node belongs to the corresponding vehicle's destination factory, if any. A schedule is *feasible* if all of its routes are feasible.

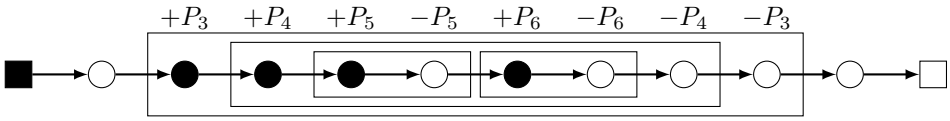


Figure 4: Example for blocks. Blocks are framed by rectangles.

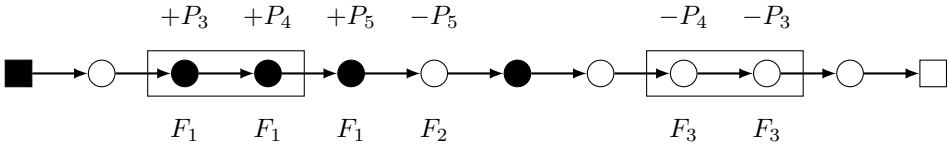


Figure 5: Example for a (maximal) bridge. Left and right sequences of the bridge are framed by rectangles.

A *couple* is an ordered pair of a pickup node and a delivery node corresponding to the same package.

A *block* is a sequence of consecutive nodes such that the first and the last nodes of the block refer to the pickup and the delivery of the same package, respectively, see Figure 4. Note that a block is a slice of vehicle activity where exactly those packages are delivered which are picked up in this block, that is, the loaded packages are the same before and after the block. Thus if a block is removed from a feasible route, the route still meets the capacity and the LIFO constraints.

A sequence $(\ell_1, \ldots, \ell_k)$ of consecutive pickup nodes and a sequence $(r_k, \ldots, r_1)$ of consecutive delivery nodes constitute a *bridge*, if ℓ_j and r_j are the pickup and the delivery node of the same package, respectively, for $j = 1, \ldots, k$, and moreover, the pickup nodes belongs to the same factory, and the delivery nodes belongs to the same factory, see Figure 5. Note that a bridge refers to a set of packages that are picked up at the same time and delivered at the same time. Thus, if a bridge is removed from a feasible route, the route still meets the capacity and the LIFO constraints. A bridge $((\ell_1, \ldots, \ell_k), (r_k, \ldots, r_1))$ is *maximal*, if neither $((\mathrm{pred}(\ell_1), \ell_1, \ldots, \ell_k), (r_k, \ldots, r_1, \mathrm{succ}(r_1)))$ nor $((\ell_1, \ldots, \ell_k, \mathrm{succ}(\ell_k)), (\mathrm{pred}(r_k), r_k, \ldots, r_1))$ constitutes a bridge, where $\mathrm{pred}(n)$ and $\mathrm{succ}(n)$ denote the immediate predecessor and the immediate successor of node n in the route, respectively. Remark that a couple can be considered as a minimal bridge.

3.2.1. Evaluating schedule

In order to apply a local-based procedure we need to assign a value to each schedule, which is the corresponding objective value (see (1)), called *evaluated value*. The evaluated value of a feasible schedule S is denoted by $\mathrm{eval}(S)$.

Because of the possible queuing at factories, the routes can have a big impact on each other, so they cannot be independently evaluated. Thus to evaluate a schedule we apply a simple discrete-event simulation procedure, similar to the

evaluation procedure of the provided simulator.

3.2.2. Basic operations on a schedule

In our solution approach we apply several operations to modify a schedule. The basic operations are to optimally insert a couple, a bridge, or a block into a schedule. The optimal insertion of such a structure means that we consider those feasible schedules that can be resulted from the current one by the insertion of the given structure, and we choose the one which yields the smallest evaluated value.

The simplest operation is the *optimal insertion of a block*. Briefly stated, for each route and for each position we try to insert the block, check whether the resulting schedule is feasible, and if so, we evaluate it, and finally, we choose the one with the smallest evaluated value.

In case of the *optimal insertion of a couple*, for each route and for each position we try to insert the pickup node, then for each subsequent position the delivery node, then check whether the resulting schedule is feasible, and if so, we evaluate it, and finally, we choose the one with the smallest evaluated value. The *optimal insertion of a bridge* into a schedule is analogous to the previous procedure (recall, that a couple can be considered as a minimal bridge).

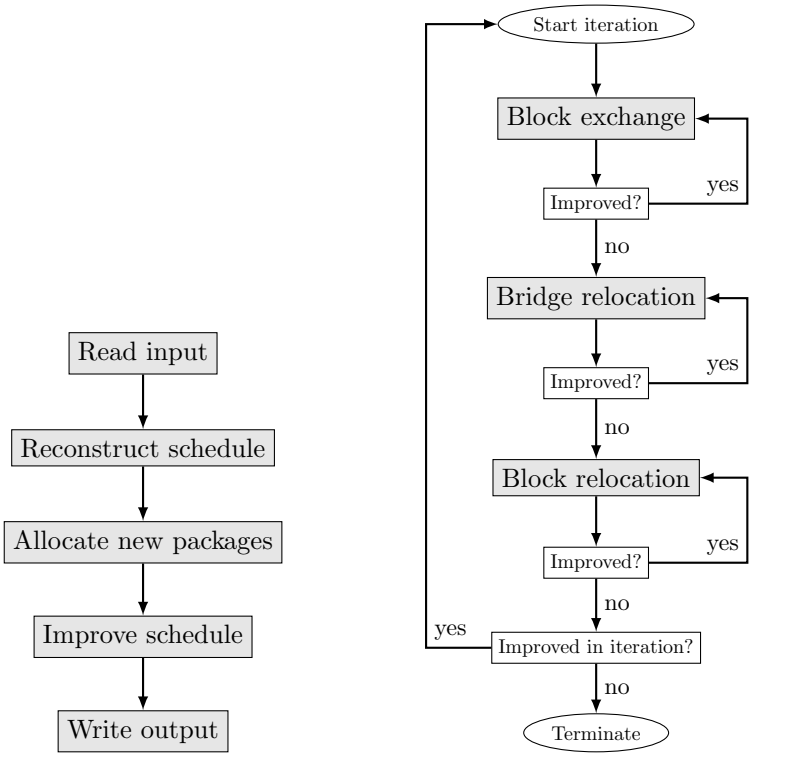
4. Solution approach

In this section we present our solution approach to solve an iteration of the dynamic pickup and delivery problem described in Section 2.2. This approach consists of two main phases. First, we reconstruct the schedule obtained in the previous iteration, if any, and insert newly arrived packages into this schedule (see Section 4.1). Second, we improve this initial schedule by a local-search procedure (see Section 4.2). The sketch of the whole approach is depicted in Figure 6.

4.1. Initial schedule

Note that improving a schedule may consume a lot of time, and this final schedule could be a good starting point for the next iteration. Thus, in the end of each iteration we save final schedule into a file, and in the beginning of each iteration we try to build an initial schedule based on the one obtained in the previous iteration. This is nothing, but we reconstruct the previous schedule from the file and remove expired nodes from the beginning of the routes, i.e., pickup nodes corresponding to packages already loaded on vehicles, and delivery nodes corresponding to packages already unloaded from the vehicles.

New packages (that is, packages corresponding to orders that have become known since the last iteration, and thus not in this reconstructed schedule) are inserted into the schedule as follows. The main idea is to insert packages (more precisely, the corresponding couples) into the schedule one by one, starting with



(a) Main steps of the approach.

(b) Main steps of improving a schedule.

Figure 6: Sketch of the local-search based solution approach.

the most urgent in some sense. That is, for a package P that belongs to order o_i we determine an *estimated delay value* calculated as

$$\Delta_P := t_i^d - t^{\text{update}} - \left(t^{\text{docking}} + t_P^{\text{loading}} + \text{travel}(f_i^p, f_i^d)\right)$$

where t_P^{loading} is the loading time of package P. There is no chance to deliver the package on time, if the assigned vehicle does not arrive at the pickup factory Δ_P time units later. Clearly, if Δ_P is negative, the package will surely be late. We call a package *urgent* if its estimated delay value is less then 3600 (seconds), otherwise it is *easy*. We start insertion with urgent packages, then continue with the remaining ones. When we insert such a couple into the schedule, we apply the optimal insertion procedure described in Section 3.2.2.

4.2. Improving a schedule

In order to improve a schedule we apply a local-search procedure (see Section 4.2.2), where we use three different types of operations to modify the schedule (see Section 4.2.1).

4.2.1. Operations on schedules

A *block relocation* is an operation when a block is removed from its current position and inserted back to an other place. After a block is removed from its current place, it can be inserted into either the same route or into an other one. In Figure 7a we depict an example for a block relocation between routes.

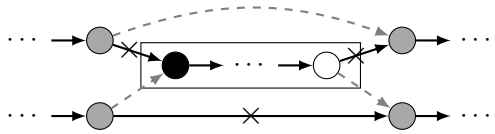
Similarly, a *bridge relocation* is an operation when a maximal bridge is removed from its current position and inserted back to an other place. Again, after a bridge is removed from its current place, it can be inserted into either the same route or into an other one. Note, however, that for inserting a bridge into a route there are many more options, since in this case two node sequences have to be inserted. In Figure 7b we depict an example for a bridge relocation between routes where there are nodes between the bridge's two parts in the initial schedule, but there are no nodes between these parts after relocation.

Finally, a *block exchange* is an operation when two blocks are exchanged with each other. In Figure 7c we depict an example for a block exchange between routes.

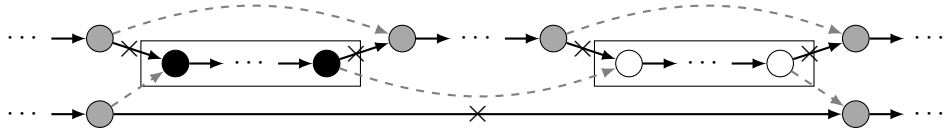
4.2.2. Local-search approach

For a given operation type (i.e., a block relocation, a bridge relocation, or a block exchange) we go through all the feasible schedules that can be resulted from the current incumbent schedule S by such an operation, and choose from these a one, say S', with minimal evaluated value. If $\text{eval}(S') < \text{eval}(S)$ then we continue with improved schedule S', otherwise, we say that schedule S cannot be improved by such an operation.

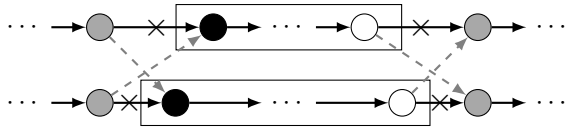
The local-search procedure consists of iterations, where in each iteration we apply the block exchange operation until it improves the current incumbent schedule, then we do the same with bridge relocation and then with block relocation operations. If the schedule is improved at least once during an iteration,



(a) Block relocation between routes.



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



(c) Block exchange between routes.

Figure 7: Example for schedule operations. Crossed/dashed arrows refer to old/new links.

we start a new iteration, otherwise, we terminate the local-search procedure. The sketch of the procedure is depicted in Figure 6b. We also apply a 540 seconds time limit for the local-search procedure in order to meet the requirements of the simulator (see Section 2.3), that is, before each improvement attempt we check whether the time limit is reached, and if yes, we terminate the procedure.

Note that the presented order of the operations is not determined a priori, but based on our computational experiments (see Section 5.2).

4.3. Final remarks

As we mentioned in Section 1, Zhu et al. (2021) and Horváth et al. (2021) also propose a local-search type solution approach for the DPDP. Since the current approach is a revised and improved version that of Horváth et al. (2021), the two algorithms show many similarities. First, to split orders and to create initial schedule, Horváth et al. (2021) apply the same procedure as described in this paper, however, the authors use a different, compact – and in a sense inconvenient – schedule representation where each node is associated with a factory and two lists of packages to be delivered and picked up at this factory, respectively. To improve schedules, the authors apply a local-search procedure with a time limit of one minute, using an operation equivalent to *bridge relocation* (see Section 4.2.1). Since this time limit is rarely reached in practice (i.e., on the benchmark instances), the authors felt no need to apply a schedule reconstruction procedure, but build initial schedules always from scratch.

To split orders, Zhu et al. (2021) use a similar approach described in Section 3.1, apply a similar schedule structure presented in Section 3.2, and create

initial schedules similarly as in Section 4.1, and the authors also apply a schedule reconstruction step. The authors apply a variable neighborhood search with a time limit of 9 minutes to improve their schedule. They used four types of operations to explore the ever-growing neighborhood of the current incumbent solution. These are, *couple exchange*, *couple relocation*, *block exchange*, *block relocation*. As their names suggest, in case of couple relocation, a couple is removed from its current position and inserted somewhere else, while in case of couple exchange, two couples are exchanged between two different routes. Block exchange and block relocation are similar as described in Section 4.2.1. If a schedule cannot be improved by these operations, the authors apply a 2-opt like operator to disturb the schedule in order to avoid falling into a local optimum.

5. Computational experiments

In this section we present our computational experiments. First, in Section 5.1 we briefly describe the public instances used in our computational experiments, for the the detailed description we refer the reader to (Hao et al., 2022). In Section 5.2 we evaluate the performance of the operations introduced in Section 4.2.1 in order to find their best combination. Finally, in Section 5.3 we compare our best method to the solution approaches submitted to the competition. The public dataset and the simulator can be download from the website `https://competition.huaweicloud.com/information/1000041601/Download`.

All experiments were performed on a workstation with an i9-7960X 2.80 GHz CPU with 16 cores, under Debian 9 operating system using a single thread.

5.1. Instances

Participants were provided with a public dataset for testing purpose, while submitted algorithms were evaluated on a hidden dataset. The public dataset consists of 64 instances based on 30 days of historical data. These instances contain 50-4000 orders of a single day to be satisfied with 5-100 vehicles, see Table 1. As it turned out afterwards, hidden dataset consists of 3 instances from the biggest groups of public dataset, these are, instances 50, 54, and 61.

In case of all instances, there are 4 hours to complete an order on time, that is, for each order $o_i \in \mathcal{O}$ we have $t_i^d - t_i^p = 14\,400$ seconds. Each order item is either a *box*, a *small pallet*, or a *standard pallet*, and has a size 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 seconds, respectively. The underlying network is the same for all instances and consists of 153 factories, such that each factory has 6 docking ports. Docking to a port takes half an hour (i.e., $t^{\text{docking}} = 1800$ seconds). Finally, the constant, λ, in the objective function (1) equals to $10\,000/3600$, that is, a cost of ≈ 2.78 units must be paid for every second of delay.

Table 1: Basic properties of public instances

Group	Instances	Orders	Vehicles
1	1 – 8	50	5
2	9 – 16	100	10
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

Table 2: Scenarios for the evaluation of the local-search based algorithm

Scenario	Block relocation	Bridge relocation	Block exchange
Scenario 1 (○ ○ ○)	no	no	no
Scenario 2 (● ○ ○)	yes	no	no
Scenario 3 (○ ● ○)	no	yes	no
Scenario 4 (○ ○ ●)	no	no	yes
Scenario 5 (● ● ○)	yes	yes	no
Scenario 6 (● ○ ●)	yes	no	yes
Scenario 7 (○ ● ●)	no	yes	yes
Scenario 8 (● ● ●)	yes	yes	yes

5.2. Evaluation of the local-search based approach

In Section 4.2.1 three operations are introduced to improve schedules. We performed computational experiments to investigate the effect of these operations and to find their best combination. That is, we examined the eight scenarios summarized in Table 2, where for each scenario we marked whether block relocation, bridge relocation, or block exchange was applied. Note that symbols are only indicated to provide visual assistance in identifying the corresponding scenario (that is, '●' refers to the presence of the corresponding operation, and '○' indicates if it is not applied).

In Tables 3 and 4 we summarize the results, that is, the average scores for each group and for the complete dataset, while the scores per instance can be found in the Supplementary Material associated with this paper. For each group and for the complete dataset the best results are highlighted.

The average score for the complete dataset is 3 070 172.1 when no local-search is applied (Scenario 1). As expected, using any of the three operations (Scenarios 2-4) gives a significantly better result, while the best improvement (40.9%) is resulted when block exchange operation is applied (Scenario 4). Based on these observations, we applied these operations in the order described in Section 4.2.2, that is, in each iteration we started to improve the schedule by block exchanges, then by bridge relocations, and finally by block relocations. These results depicted in Table 4. We get the best average score for the complete

Table 3: Average scores of the local-search approach (Scenarios 1-4)

Instances	Scenario 1 ○ ○ ○	Scenario 2 ● ○ ○	Scenario 3 ○ ● ○	Scenario 4 ○ ○ ●
Group 1	1 231.3	1 226.9	**1 225.5**	1 227.4
Group 2	51 591.1	39 576.2	**34 038.5**	34 681.3
Group 3	1 205.0	725.0	**691.8**	725.5
Group 4	14 735.4	9 542.2	10 766.9	**6 847.5**
Group 5	12 544.7	12 086.8	11 984.1	**11 600.9**
Group 6	1 028 109.1	207 503.5	**118 418.4**	232 882.5
Group 7	3 394 542.1	1 650 329.9	**1 456 570.2**	1 831 027.1
Group 8	20 057 417.7	14 715 948.6	13 517 726.3	**12 398 856.7**
All	3 070 172.1	2 079 617.4	1 893 927.7	**1 814 731.1**

Table 4: Average scores of the local-search approach (Scenarios 5-8)

Instances	Scenario 5 ● ● ○	Scenario 6 ● ○ ●	Scenario 7 ○ ● ●	Scenario 8 ● ● ●
Group 1	**1 226.6**	1 306.6	1 226.8	1 306.7
Group 2	**30 829.7**	31 191.4	38 344.1	34 046.3
Group 3	809.8	690.4	927.5	**687.8**
Group 4	8 733.5	**4 820.8**	7 303.1	6 831.4
Group 5	11 343.9	10 645.7	11 049.2	**10 344.8**
Group 6	110 274.7	70 594.5	47 553.4	**42 249.8**
Group 7	1 610 791.6	1 374 656.6	**1 025 329.8**	1 103 798.1
Group 8	13 472 511.8	11 120 565.8	9 314 094.5	**8 913 545.2**
All	1 905 815.2	1 576 809.0	1 305 728.6	**1 264 101.3**

Table 5: Average scores of the solution approaches

Instances	LSBA*	1st Team[1]	2nd Team[2]	3rd Team[3]
Group 1	**1 306.7**	2 896.4	13 676.2	1 763.8
Group 2	**34 046.3**	41 535.3	-	62 180.2
Group 3	**687.8**	5 860.4	2 310.7	8 969.7
Group 4	6 831.4	**6 544.5**	105 049.4	26 938.3
Group 5	**10 344.8**	10 459.1	17 284.3	94 794.9
Group 6	42 249.8	**41 494.3**	153 419.1	651 944.9
Group 7	1 103 798.1	**798 240.7**	904 586.3	1 941 385.3
Group 8	**8 913 545.2**	11 359 466.4	18 678 529.1	15 122 816.2
All	**1 264 101.3**	1 533 312.1	-	2 238 849.2

* Local-search based approach proposed in this paper
[1] Zhu et al. (2021)
[2] Ye and Liang (2021)
[3] Horváth et al. (2021)

dataset, namely 1 264 101.3, if all of the operations are in use (Scenario 8).

5.3. Comparison of the solution approaches

Finally, we compare our best solution approach determined in the previous experiments (that is, Scenario 8) to the three approaches submitted to the competition, namely, the variable neighborhood search method of Zhu et al. (2021), the rule-based algorithm of Ye and Liang (2021), and the local-search based approach of Horváth et al. (2021). In Table 5 we indicate the average scores for each group and for the complete dataset, while the scores per instance can be found in the Supplementary Material associated with this paper. For each group and for the complete dataset the best results are highlighted.

Note that the algorithms of Ye and Liang (2021) and Horváth et al. (2021) never reach the set time limit, thus give the same scores for each execution. This is why we got the same average scores on the hidden dataset (i.e., on instances 50, 54, and 61) as resulted by the evaluation on Huawei's servers (namely, 3 765 994.1 and 3 905 011.1). In contrast, the algorithm of Zhu et al. (2021) often reaches the 9 minutes time limit set by the authors, thus in a faster computer their variable neighborhood search method may perform more iterations, thus outputs a better schedule, and thus finally gives a better score. That is why we cannot reproduce the hidden score of this algorithm on our server (namely, we get 2 541 925.6 instead of 2 291 704). However, we emphasize that since all of our experiments were performed on the same workstation, our comparisons are fair.

Note that the submitted algorithm of Ye and Liang (2021) fails on seven instances of Group 2, thus in this case no average score is depicted for the group and for the complete dataset, however, even in the best case scenario (that is, assuming a zero score for all of these instance) the latter would be 2 487 183.2. The solution approach of the first place team – apart from the

smallest groups — significantly outperforms the approaches of the other two awarded teams. That is, this approach gives the lowest score on 49 out of 64 instances, and the average score for the five biggest group is much better than the other two cases.

On the complete dataset the local-search based approach presented in this paper (indicated as *LSBA*) gives the best result, i.e., the lowest average score, namely, 1 264 101.3, which is 17.6% better than the score of the first place team. Also this approach gives the lowest average score for 5 out of 8 groups including the biggest one, where it gives the best result on 6 out of 8 instances.

For the sake of completeness, we note that on hidden dataset (i.e., on instances 50, 54, and 61) the local-search based approach gives average score equals to 3 564 390.4 and in the best case (that is, Scenario 7) it equals to 3 124 566.8 which is also higher then that of (Zhu et al., 2021).

6. Conclusions

In this paper we proposed a local-search based solution approach for the Dynamic Pickup and Delivery Problem introduced by the Huawei Technologies Co. Ltd. The problem has some unique real-world characteristics, such as limited number of ports at the facilities for serving the vehicles. Our computational experiments proved that the proposed algorithm outperforms the other known approaches.

In the future we plan to further develop our method to better exploit some hidden characteristics of the problem, in which machine learning techniques may have a great potential.

7. Acknowledgments

This work was supported by the National Research, Development and Innovation Office, grant numbers SNN 129178 and TKP2021-NKTA-01. The authors are thankful to Xiang Xiang for all the technical help he assisted us during the competition.

References

Attanasio, A., Bregman, J., Ghiani, G., Manni, E., 2007. Real-time fleet management at Ecourier Ltd, in: Dynamic fleet management. Springer, pp. 219–238.

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

Berbeglia, G., Cordeau, J.F., Laporte, G., 2010. Dynamic pickup and delivery problems. European Journal of Operational Research 202, 8–15.

Braekers, K., Ramaekers, K., Van Nieuwenhuyse, I., 2016. The vehicle routing problem: State of the art classification and review. Computers & Industrial Engineering 99, 300–313.

Branchini, R.M., Armentano, V.A., Løkketangen, A., 2009. Adaptive granular local search heuristic for a dynamic vehicle routing problem. Computers & Operations Research 36, 2955–2968.

Clarke, G., Wright, J.W., 1964. Scheduling of vehicles from a central depot to a number of delivery points. Operations research 12, 568–581.

Cordeau, J.F., Laporte, G., 2003. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. Quarterly Journal of the Belgian, French and Italian Operations Research Societies 1, 89–101.

Dantzig, G.B., Ramser, J.H., 1959. The truck dispatching problem. Management science 6, 80–91.

Demirtaş, Y.E., Özdemir, E., Demirtaş, U., 2015. A particle swarm optimization for the dynamic vehicle routing problem, in: 2015 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO), IEEE. pp. 1–5.

Eksioglu, B., Vural, A.V., Reisman, A., 2009. The vehicle routing problem: A taxonomic review. Computers & Industrial Engineering 57, 1472–1483.

Elhassania, M., Jaouad, B., Ahmed, E.A., 2014. Solving the dynamic vehicle routing problem using genetic algorithms, in: 2014 International Conference on Logistics Operations Management, IEEE. pp. 62–69.

Györgyi, P., Kis, T., 2019. A probabilistic approach to pickup and delivery problems with time window uncertainty. European Journal of Operational Research 274, 909–923.

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. arXiv:2202.01256.

Ho, S.C., Szeto, W.Y., Kuo, Y.H., Leung, J.M., Petering, M., Tou, T.W., 2018. A survey of dial-a-ride problems: Literature review and recent developments. Transportation Research Part B: Methodological 111, 395–421.

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.

Li, X., Luo, W., Yuan, M., Wang, J., Lu, J., Wang, J., Lü, J., Zeng, J., 2021. Learning to optimize industry-scale dynamic pickup and delivery problems, in: 2021 IEEE 37th International Conference on Data Engineering (ICDE), IEEE. pp. 2511–2522.

Ma, Y., Hao, X., Hao, J., Lu, J., Liu, X., Xialiang, T., Yuan, M., Li, Z., Tang, J., Meng, Z., 2021. A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. Advances in Neural Information Processing Systems 34.

Montemanni, R., Gambardella, L.M., Rizzoli, A.E., Donati, A.V., 2005. Ant colony system for a dynamic vehicle routing problem. Journal of combinatorial optimization 10, 327–343.

Nasri, S., Bouziri, H., Aggoune-Mtalaa, W., 2021. Customer-oriented dial-a-ride problems: A survey on relevant variants, solution approaches and applications, in: Emerging Trends in ICT for Sustainable Development. Springer, pp. 111–119.

Pillac, V., Gendreau, M., Guéret, C., Medaglia, A.L., 2013. A review of dynamic vehicle routing problems. European Journal of Operational Research 225, 1–11.

Psaraftis, H.N., Wen, M., Kontovas, C.A., 2016. Dynamic vehicle routing problems: Three decades and counting. Networks 67, 3–31.

Reyes, D., Erera, A., Savelsbergh, M., Sahasrabudhe, S., O'Neil, R., 2018. The meal delivery routing problem. Optimization Online 6571.

Rios, B.H.O., Xavier, E.C., Miyazawa, F.K., Amorim, P., Curcio, E., Santos, M.J., 2021. Recent dynamic vehicle routing problems: A survey. Computers & Industrial Engineering 160, 107604.

Toth, P., Vigo, D., 2002. The vehicle routing problem. SIAM.

Ulmer, M.W., Thomas, B.W., Campbell, A.M., Woyak, N., 2021. The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. Transportation Science 55, 75–100.

Xu, Y., Wang, L., Yang, Y., 2013. Dynamic vehicle routing using an improved variable neighborhood search algorithm. Journal of Applied Mathematics 2013.

Yang, J., Jaillet, P., Mahmassani, H., 2004. Real-time multivehicle truckload pickup and delivery problems. Transportation Science 38, 135–148.

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. Archives of Computational Methods in Engineering 29, 195–221.

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`.