Optimize the Future of Delivery
-
DH's VRPPD Challenge

CO@Work

September 2024

1 Motivation

Are you passionate about solving complex logistic problems in actual operations? Delivery Hero (DH) invites you to participate in its 'Optimize the Future of Delivery - DH's VRPPD Challenge' - A unique opportunity to tackle a Vehicle Routing Problem with Pickup and Delivery (VRPPD) arising in the real world.

As one of the world's leading online local delivery platforms, Delivery Hero (DH) relies on cutting-edge optimization techniques to ensure that millions of orders are delivered efficiently every day. Besides applying a manifold of heuristic methods, we at DH are currently broadening our portfolio of solution approaches by incorporating exact algorithms. And this is where you can actually help us and make a difference: In this challenge, you'll be tasked with implementing and solving the VRPPD using a Mixed-Integer Linear Programming (MIP) based approach.

What makes this challenge truly exciting? Not only will you work on one of the most crucial logistical problems in our industry and solve problem instances from the real-world, you will additionally have the freedom to combine MIP with the heuristic methods of your choice. The created hybrid approach shall allow you to explore creative and efficient ways to achieve high-quality solutions within short computational times.

Whether you're interested in pure optimization or eager to experiment with a blend of exact and heuristic techniques, this challenge not only offers you the platform to demonstrate your skills, innovate, and make a real impact, but you can learn a lot on the way and have a lot of fun. Participants will develop a working prototype capable of handling real-world complexities, with the potential to influence how a global leader like Delivery Hero operates.

Highlights

- Model and solve the VRPPD - Implement a MIP model for the VRPPD, incorporate constraints such as vehicle capacity, time windows, and pickup-delivery precedence.
- Explore hybrid approaches - While the primary focus is on MIP, you're encouraged to enhance your solution by integrating heuristic methods, e.g. local search operators, to improve solution quality and computational efficiency.
- Create scalable solutions - Your prototype should handle large-scale problems representing the most crucial real-world scenarios at Delivery Hero, striking a balance between optimality and tractability.
- Showcase your innovation - Last but not least, present your solution to industry experts, and gain recognition for your creativity, technical skills, and problem-solving abilities.

This challenge is not about finding the best theoretical solution – it's about delivering practical, innovative solutions that can operate effectively in the dynamic world of food delivery logistics. Are you ready to optimize the future of delivery?

2 Problem Description

As discussed above, in this challenges, we will be looking at the Vehicle Routing Problem with Pickup and Delivery (VRPPD) and Time Windows from the perspective of Delivery Hero. In particular, the goal is to get a feeling for how to handle the planning complexities of routing riders, picking up meals from restaurants, and delivering them to customers. All of this while adhering to time and capacity constraints to ensure food quality and customer satisfaction, and to guarantee a great rider experience.

Now, what are the crucial components of the VRPPD to consider?

- Vehicles (Riders):
  - We consider a fleet consisting of delivery drivers, so-called riders, on bikes, motorcycles, or cars. For the sake of simplicity, in our challenge we assume that they all travel at the same speed.
  - Each rider starts from a designated location and is responsible for delivering food from multiple restaurants to multiple customers as instructed by us.
  - The vehicle's capacity is limited. In this challenge, we consider a simplified capacity that incorporates the weight as well as the size of the ordered items.

- Pickup and Delivery:
  - Pickup Locations: Each delivery is associated with a restaurant, which serves as the pickup location. A rider must first go to the restaurant to pick up the food. Moreover, every delivery is also assigned a load value that corresponds to the capacity that needs to be available in the vehicle.
  - Delivery Locations: After picking up the food, the rider must deliver it to the customer's location. However, note that a rider can also pick up other deliveries or go to drop off orders that they already picked up before. Having multiple deliveries in the bag at the same time we call stacking.
  - Pairing Constraint: Each delivery has a clear sequence – the rider must first pick up the food before delivering it to the customer, and the pickup and delivery are linked as a pair.
- Time Windows:
  - Restaurant Time Windows: Restaurants have specific time windows during which they prepare the food and expect it to be picked up. In particular, if a rider arrives too early, the food may not be ready and they have to wait.
- Capacity Constraints:
  - A rider can only carry a certain maximum load at any time. In particular, the sum of the load of the deliveries that a rider has picked up but not delivered is not allowed to exceed the vehicle's capacity.
- Objective:
  - Minimizing Delivery Time: The company aims to deliver food as quickly as possible to ensure freshness. In particular, we aim at minimizing the sum of all delivery times.

Obviously, the problem stated above is only a simplified version of what a company such as Delivery Hero has to deal with in the real world. For example, one of the most important aspects that we completely exclude here is that deliveries dynamically come in over time and that some tasks may take less or more time to be fulfilled than expected. This makes it necessary to continuously revisit the currently planned routes.

3 A Graph Model of the VRPPD

Next, we are defining a directed graph model for the VRPPD. An example for two riders an two orders is shown in Figure 1.

3.1 Vertices

First of all, we are given a set of n vehicles $K := \{1, \dots, n\}$. Each vehicle $k \in K$ is associated with a nonnegative capacity value $Q_k \in \mathbb{R}_{\geq 0}$. Let us denote the maximum capacity of all vehicles by $Q_{max} := \max\{Q_1, \dots, Q_n\}$. Further, each vehicle $k \in K$ is associated with a depot vertex s_k representing its starting point. In the following, we denote the set of all depots by $S := \{s_1, \dots, s_n\}$.
Next, we consider a set of m orders $O := \{1, \dots, m\}$. Each order $o \in O$ is represented by a pickup vertex p_o and the corresponding delivery vertex d_o. We denote the set of all pickup vertices by $\mathcal{P} := \{p_1, \dots, p_m\}$ and the set of delivery vertices by $\mathcal{D} := \{d_1, \dots, d_m\}$.
This concludes the construction of the vertex set $\mathcal{V}$, which is the union of the depot vertices and the pickup as well as the delivery vertices, i.e., $\mathcal{V} := S \cup \mathcal{P} \cup \mathcal{D}$.
Next, we introduce some more parameters associated with the vertices.
Every delivery (synonymously order) $o \in O$ is associated with a load Q_o. In this context, we define the load of the corresponding pickup vertex as $Q_{p_o} := Q_o$ and for the delivery vertex as $Q_{d_o} := -Q_o$. Additionally, we set $Q_{s_k} := Q_{max} - Q_k$ for all $s_k \in S$.
Moreover, each pickup $p_i \in \mathcal{P}$ is associated with a time window $[\ell_{p_i}, u_{p_i}]$, where $\ell_{p_i}, u_{p_i} \in \mathbb{R}_{\geq 0}$ and $\ell_{p_i} \leq u_{p_i}$, representing the time within which the vertex can be visited by a vehicle. For all $v \in S \cup \mathcal{D}$ we consider $[\ell_v, u_v] := [0, \infty]$, i.e, there are no time-related restrictions.

3.2 Arcs

The arc set consists of four different subsets: $\mathcal{A}_1 := \{(s_k, p_i) \,|\, s_k \in S, p_i \in \mathcal{P}\}$, $\mathcal{A}_2 := \{(u, v) \,|\, u, v \in \mathcal{P} \cup \mathcal{D} \text{ with } u \neq v\}$, $\mathcal{A}_3 := \{(d_i, s_k) \,|\, d_i \in \mathcal{D}, s_k \in S\}$, and $\mathcal{A}_4 := \{(s_k, s_k) \,|\, s_k \in S\}$,. While set $\mathcal{A}_1$ represents arcs from the depots towards the pickup vertices, $\mathcal{A}_2$ describes the arcs between pickup and dropoff vertices. Additionally, $\mathcal{A}_3$ is modelling arcs between the dropoff vertices and the depots. Finally, in $\mathcal{A}_4$ we have a loop at each depot vertex. Each arc $a = (v, w) \in \mathcal{A}$ is associated with a travel time $T_a = T_{vw} \in \mathbb{R}_{\geq 0}$ from vertex v to vertex w. Additionally, let $T := \sum_{a \in \mathcal{A}} T_a$ represent the sum of all travel times.

3.3 Solution

A solution for the VRRPD with respect to this graph model is a vertex-disjoint cycle cover of the graph where each cycle contains exactly one depot vertex. A possible solution for the example from Figure 1 can be found in Figure 2. A solution is feasible, if all the criteria explained above are met. In particular, consider the depot vertex as starting point of the corresponding tour, both the pickup and the dropoff vertices of a delivery have to be contained in the cycle. Additionally, the pickup vertex has to be visited before the dropoff vertex. Finally, the capacity condition has to be met, i.e., the load at any point in time of the cycle cannot exceed the capacity of the vehicle.

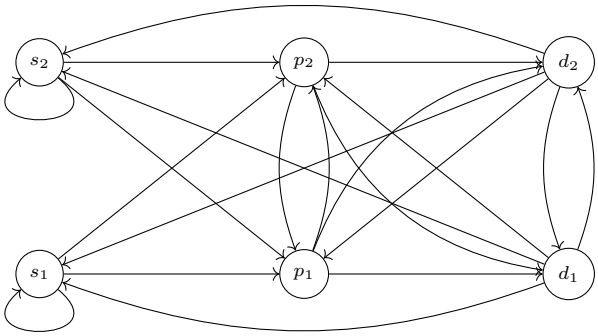


Figure 1: Example graph for two riders and two deliveries.

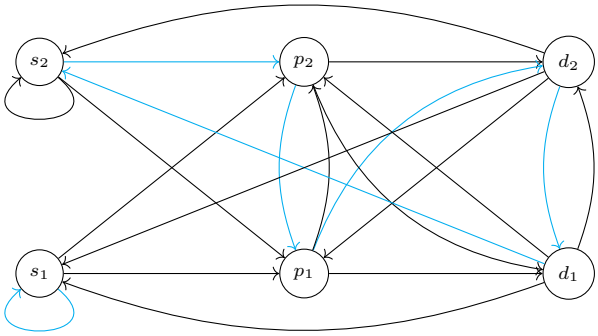


Figure 2: A possible solution to the VRPPD. Here, rider 1 is idle and not assigned any delivery, indicated by the blue loop above its depot. Rider 2 on the other hand picks up delivery 2 first and then delivery 1, thereby creating a so-called stack. Afterwards, they drop off delivery 2 and then delivery 1 before "returning" to its depot.

4 Input Data

The training dataset for the challenge is provided as a zip file containing a structured directory with 150 instances. Each instance is contained in a separate folder, and in every instance folder, there are three CSV files: *couriers.csv*, *deliveries.csv*, and *traveltimes.csv*. Below is a detailed description of each file

- **couriers.csv**
  This file contains the list of available couriers. It has three columns:
  - **ID:** A unique identifier for each courier.
  - **Location:** A location ID representing the courier's starting point. Multiple couriers can share the same starting location, which may differ from their courier ID.

- **Capacity:** This field describes the total capacity each courier can handle in terms of delivery volume.

Figure 3 shows an example *couriers.csv* file. Here, the rider with ID 1 is currently located at the location with ID 1. Additionally, the rider can carry a maximum load of 100 as this is its capacity.

- **deliveries.csv**
  This file contains the information about all deliveries. Each delivery is associated with a pickup and dropoff and contains six columns:
  - **ID:** The unique identifier for each delivery. These IDs are sequentially numbered, continuing from the last courier ID. For example, if there are 485 couriers, the first delivery ID would be 486 (as shown in the dataset image Figure 4).
  - **Capacity:** The necessary capacity required to transport this delivery. When a courier picks up a delivery, the available capacity of the courier decreases by this value, and when the delivery is dropped off, the capacity is restored.
  - **Pickup Loc:** The ID of the location where the delivery is picked up.
  - **Time Window Start:** The earliest time (in minutes) the pickup can occur.
  - **Pickup Stacking_Id:** If some deliveries share the same stacking ID, it indicates that they can be picked up at the same vendor.
  - **Dropoff Loc:** The ID of the location where the delivery will be dropped off.

  Figure 4 shows an example deliveries.csv file. Here, the order with ID 5 has a load/capacity of 17. Moreover, it has to be picked up at the location with ID 5 and has to be delivered to the location with ID 6. The earliest point in time when the delivery can be picked up is in 10 minutes. Its stacking ID is 108056.

- **traveltimes.csv**
  This file consists of a matrix representing the travel times (in minutes) between different locations. Each value corresponds to the time it takes to travel from one location to another. The matrix is symmetrical, and diagonal values are zero, representing that there is no travel time between a location and itself.

  The matrix layout can be seen in the example image Figure 5.

Note: A script in Python is provided to facilitate the reading of these input data files directly from the directory structure. However, using the script is not mandatory.

ID	Location	Capacity
1	1	100
2	2	100
3	3	100
4	4	100

Figure 3: A *couriers.csv* example file.

ID	Capacity	Pickup Loc	Time Window St	Pickup Stacking	Dropoff Loc
5	17	5	10	108056	6
6	24	5	11	108056	7
7	3	8	10	151391	9
8	7	10	13	147289	11
9	4	12	23	132201	13
10	15	10	16	147289	14
11	14	15	8	98018	16
12	12	15	7	98018	17
13	15	12	28	132201	18

Figure 4: A *deliveries.csv* example file.

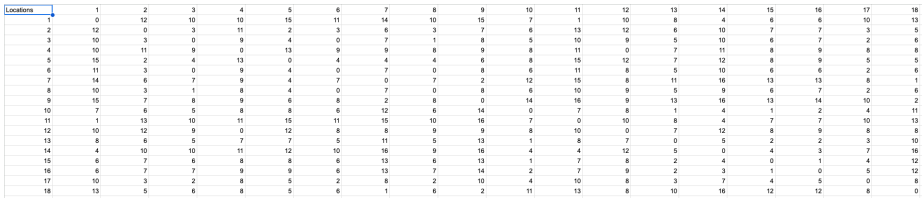
5 Output Data Format

Result files are structured as follows, see also the example stated in Figure 6. In the first column, the IDs of the couriers are stated. In particular, a result file must contain the ID and therefore a row corresponding to every courier listed in the corresponding courier.csv file.
Next, following the courier's ID, in every row the sequence of deliveries visited by the corresponding courier is stated. For a route to be feasible, every delivery has to appear twice. The first appearance marks the pickup and the second appearance the dropoff. Moreover, every delivery is contained in exactly one row, i.e., the delivery is part of the route of exactly one courier. Additionally, all other constraints, e.g. the capacity condition, needs to be met.
Consider the example solution presented in Figure 6. Here, the courier with ID 1 first of all picks up the delivery with ID 13. Next it picks up the delivery with ID 7. Afterwards, the courier drops off delivery 13 and subsequently delivery 7. On the other hand, the route of the courier with ID 3 is empty, i.e., no delivery is assigned to this courier.

6 Rules

The following rules have to be respected during the challange.

1. If the algorithm is based on the MIP implementation FICO Xpress or SCIP have to be used. Other than that, you are free to implement any



Locations	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	12	10	10	15	11	14	10	15	7	1	10	8	4	6	6	10	13
2	12	0	3	11	2	3	6	3	7	6	13	12	6	10	7	7	3	5
3	10	3	0	9	4	0	7	1	8	5	10	9	5	10	6	7	2	6
4	10	11	9	0	13	9	9	8	9	8	11	0	7	11	8	9	8	8
5	15	2	4	13	0	4	4	4	6	8	15	12	7	12	8	9	5	5
6	11	3	0	9	4	0	7	0	8	6	11	8	5	10	6	6	2	6
7	14	6	7	9	4	7	0	7	2	12	15	8	11	16	13	13	8	1
8	10	3	1	8	4	0	7	0	8	6	10	9	5	9	6	7	2	6
9	15	7	8	9	6	8	2	8	0	14	16	9	13	16	13	14	10	2
10	7	6	5	8	8	6	12	6	14	0	7	8	1	4	1	2	4	11
11	1	13	10	11	15	11	15	10	16	7	0	10	8	4	7	7	10	13
12	10	12	9	0	12	8	8	9	9	8	10	0	7	12	8	9	8	8
13	8	6	5	7	7	5	11	5	13	1	8	7	0	5	2	2	3	10
14	4	10	10	11	12	10	16	9	16	4	4	12	5	0	4	3	7	16
15	6	7	6	8	8	6	13	6	13	1	7	8	2	4	0	1	4	12
16	6	7	7	9	9	6	13	7	14	2	7	9	2	3	1	0	5	12
17	10	3	2	8	5	2	8	2	10	4	10	8	3	7	4	5	0	8
18	13	5	6	8	5	6	1	6	2	11	13	8	10	16	12	12	8	0

Figure 5: A *traveltimes.csv* example file.

ID								
1	13	7	13	7				
2	5	6	5	9	9	6		
3								
4	10	8	10	11	12	11	12	8

Figure 6: Example for structure in *result.csv*.

preprocessing or heuristic algorithm to find feasible or even optimal solutions.

2. Do not use any specialized VRP algorithm library to obtain solutions.
3. The program should run in the provided online environment.

7 Submission

On the last day of the challenge (Thursday, 26.09.2024), we will provide an additional test set with 50 unknown instances at the beginning of the challenge session via the common git repository. Each team is required to prepare and submit the following:

1. Solution Description Document (PDF)

- **Filename**: The document should be named following this format: `teamnumber_teamname_solution_description.pdf`.
- **Length**: 1-2 pages.
- **Content**: The document should include:
  - **Final Approach**: A detailed description of the final solution the team used for the challenge. Include brief explanations of the model, methodology, and any preprocessing steps.
  - **Alternative Attempts**: A summary of other approaches the team tried throughout the competition, regardless of their success. Briefly describe why those approaches were less effective.
  - **Challenges and Solutions**: Discuss any difficulties the team encountered and how they overcame them.

2. Solution Files (ZIP)

- **Filename**: `teamnumber_teamname_solutions.zip`.
- **Contents**: The zip file should include one csv file per test instance in the format described in Section5 and with the same name as instance name (folder name).

  Additionaly:

  - Code files (scripts, notebooks, etc.)
  - Configuration files (if applicable)
  - Any other resources required for evaluation of the solution.
- Make sure the solution is well-documented and easy to run.

Both the PDF document and the ZIP file should be submitted by 18:00 26.09.2024.

8 Scoring

The submissions will be evaluated according to the following criteria:

- **Feasibility**: For each instance with a feasible solution, the team will be awarded one point.
- **Ranking**: For each instance, the top 5 objective function values will receive points as follows: 10, 8, 6, 4, and 2 points. In case of ties, the corresponding points will be awarded to every team.

Appendix

A Mixed-Integer Programming Model

Based on the graph model $G = (\mathcal{V}, \mathcal{A})$ from the previous section, we introduce the following MIP-model.

$$\min \sum_{v \in \mathcal{D}} t_v \tag{1}$$

$$\text{s.t.} \sum_{k \in K} \sum_{(u,v) \in \delta^-(v)} x_{uvk} = 1 \qquad \forall v \in \mathcal{V} \tag{2}$$

$$\sum_{(v,u) \in \delta^+(v)} x_{vuk} = \sum_{(u,v) \in \delta^-(v)} x_{uvk} \qquad \forall v \in \mathcal{V}, \forall k \in K \tag{3}$$

$$\sum_{(u,p_o) \in \delta^-(p_o)} x_{up_ok} = \sum_{(u,d_o) \in \delta^-(d_o)} x_{ud_ok} \; \forall o \in O, \forall k \in K \tag{4}$$

$$\sum_{(s_k,v) \in \delta^+(s_k)} x_{s_kvk} = 1 \qquad \forall k \in K \tag{5}$$

$$t_u + (T_{uv} + T)\, x_{uvk} \leq t_v + T \qquad \forall (u,v) \in \mathcal{A}_1 \cup \mathcal{A}_2, \forall k \in K \tag{6}$$

$$t_{d_o} - t_{p_o} \geq T_{p_o d_o} \qquad \forall o \in O \tag{7}$$

$$q_u + (Q_v + Q_{max})\, x_{uvk} \leq q_v + Q_{max} \qquad \forall (u,v) \in \mathcal{A}_1 \cup \mathcal{A}_2, \forall k \in K \tag{8}$$

$$q_{s_k} = Q_{s_k} \qquad \forall k \in K \tag{9}$$

$$\ell_v \leq t_v \leq u_v \qquad \forall v \in \mathcal{V} \tag{10}$$

$$0 \leq q_v \leq Q_{max} \qquad \forall v \in \mathcal{V} \tag{11}$$

$$x_{uvk} \in \{0,1\} \qquad \forall (u,v) \in \mathcal{A}, \forall k \in K \tag{12}$$

Variables

The MIP model features three types of variables.
First, for every vertex $v \in \mathcal{V}$ we have a continuous variable t_v, see (10), representing the time when a vertex is visited by a vehicle. As we can see, this variable has to respect the bounds of the respective time window.
Moreover, for every vertex $v \in \mathcal{V}$ we have a continuous variable q_v, see (11), which represents the load of the vehicle after visiting a vertex. The load cannot be negative and is not allowed to exceed Q_{max}.
Finally, for every arc $a = (u,v) \in \mathcal{A}$ and every vehicle $k \in K$ we have a binary decision variable x_{uvk} indicating whether this vehicle is travelling via this arc or not, see (12).

Constraints

Constraints (2) ensure that every vertex is visited exactly once. In addition, constraints (3) ensure that the same vehicle that visits a vertex does also leave it. Moreover, constraints (4) guarantee that the same vehicle that visits a pickup vertex does also visit the corresponding delivery vertex. Finally, due

to constraint (5), only the vehicle corresponding to its depot can leave it. Note that if the loop at the depot is used, this means that the vehicle is not assigned any deliveries and it stays idle.
Next, constraints (6) ensure that the time variables for the vertices are correctly updated. In particular, if a vehicle travels an arc $(u, v) \in \mathcal{A}$, the constraint ensures that the time at v has to be greater than or equal to the time at u plus the travel time T_{uv}. Furthermore, constraints (7) ensure that the pickup is visited before the corresponding dropoff.
Similar to the time-related constraints above, constraints (8) ensure that the load variables for the vertices are correctly updated. In particular, if a vehicle travels an arc $(u, v) \in \mathcal{A}$, the constraint ensures that the load at v has to be greater than or equal to the load at u plus the load at v which is Q_v. Note that Q_v is negative for dropoff vertices, which corresponds to unloading the order. Additionally, due to constraint (9) the load at the depots is already fixed. In particular, for depot $s_k \in S$ we fix the starting load to $Q_{s_k} = Q_{max} - Q_k$. By putting this artificial load into the vehicles, which won't be unloaded at any point in time, we can w.l.o.g. assume that all vehicles have Q_{max} as their maximum capacity.

Objective Function

The objective function (1) aims at minimizing the sum of the visiting times of the delivery vertices, i.e., to minimize the sum of the delivery times.