EURO Meets NeurIPS 2022 Vehicle Routing Competition

Wouter Kool* (wouter.kool@ortec.com) Laurens Bliek (l.bliek@tue.nl)
Danilo Numeroso Robbert Reijnen Reza Refaei Afshar Yingqian Zhang
Tom Catshoek Kevin Tierney Eduardo Uchoa Thibaut Vidal
Joaquim Gromicho

Abstract

The EURO Meets NeurIPS 2022 Vehicle Routing Competition seeks to unite the OR and ML communities to propose new, innovative techniques to solve routing problems. This document describes the problem setting and rules for the competition. We look forward to your participation.

1 Introduction

The goal of this competition is to solve the vehicle routing problem with time windows (VRPTW) in two different settings, static and dynamic, using both the CPU and, if desired, the GPU. The setting is structured to encourage the use of machine learning (ML) techniques, but we note that there is no requirement to use ML in your solution method. We encourage participants to try out new ideas, using combinations of OR and ML techniques. Top participants are invited to present their ideas at the virtual workshop of NeurIPS 2022 following the competition (a NeurIPS 2022 registration required and more information will follow).

The competition consists of two phases: the qualification phase and the final phase. During the qualification phase, teams must submit their solvers, and these will be evaluated automatically on a small set of instances (50% static and 50% dynamic problems). The results of the evaluation will be listed on a public leaderboard, which will be available starting from August 1st. The top 10 teams will automatically qualify for the finals, starting November 1st, and their submitted solvers will be evaluated on a larger set of final test instances with longer time limits. Note that the solver cannot be changed for the finals. The final results will be announced during the workshop at NeurIPS 2022.

1.1 Background

Every day, millions of trucks and other vehicles perform deliveries or services around the globe. To reduce costs and environmental impact, the problem of optimally routing vehicles has become increasingly relevant. The vehicle routing problem (VRP), a generalization of the traveling salesperson problem (TSP), has been studied for many years. The capacitated vehicle routing problem (CVRP) considers the problem of finding of a set of routes to serve a set of customers with minimum total distance, while respecting a limited capacity per vehicle. Many variants of routing problems exist [1], for example including time windows (known as the VRPTW), which require deliveries at customers to happen during certain periods of time, as well as stochastic variants (e.g., travel durations that are uncertain) or dynamic variants that reveal requests over time [2].

Traditionally, vehicle routing problems have been studied in the field of operations research (OR), and solution methods primarily rely on searching the space of feasible solutions, either using exact methods or heuristics [3, 4, 5]). In recent years, the problem has also attracted the attention of

*Corresponding author.

August 2, 2022. This is not an official NeurIPS publication.

machine learning (ML) researchers [6, 7, 8, 9, 10, 11, 12, 13, 14], who aim to train models that can exploit patterns when repeatedly solving problem instances from a specific distribution to solve new problem instances faster or with better results.

While ML-based methods have made advances at solving routing problems, the best known results for static problems (see, e.g., http://vrp.galgos.inf.puc-rio.br/) are generally achieved by more traditional OR algorithms. This competition aims to bridge the gap between the OR community and ML by bringing a relevant vehicle routing problem to NeurIPS, and inviting researchers from both communities to work on the same problem. In addition, to create more opportunities of exploiting learning and advanced ML algorithms, this competition also considers a dynamic variant in which information is revealed over time.

1.2 Static problem setting

The VRPTW is a more constrained variant of the capacitated vehicle routing problem (CVRP), in which each customer requires their delivery within a specified interval of time called time window. In the static setting, the problem is defined as follows.

A VRPTW problem instance considered in this competition consists of n locations (a depot and a set of $n-1$ customers), an $n \times n$ matrix D specifying the time (in seconds) to travel between each pair of locations (it represents actual road driving times in seconds and is NOT euclidean nor symmetric), a quantity q_i that specifies the demand for some resource by each customer i, and the maximum quantity, Q, of the resource that a vehicle can carry, also known as the vehicle's capacity. In addition, each node i is associated with a duration of time s_i, denoting the time it takes to serve customer i, and a time window $[e_i, l_i]$, where $e_i < l_i$, within which delivery must begin. A vehicle is allowed to arrive at a customer location before the beginning of the time window, but it must wait for the window to "open" to make the delivery. A delivery cannot be started after the time window closes. In this way, the competition considers the time-window constraints to be hard constraints. There is no limit on the number of available vehicles. A feasible solution to the VRPTW consists of a set of routes that begin and end at the depot, such that each customer is visited on exactly one route within its specified time window and the total demand assigned to a route does not exceed the vehicle capacity Q. The objective is finding a feasible solution that minimizes the total combined driving duration of all routes. Note that the number of vehicles, the service times and waiting times are NOT part of the objective.

This problem definition follows the convention used in the recent DIMACS VRPTW challenge, except that the matrix D is not euclidean, and the number of vehicles is unlimited (in the instances considered in the DIMACS challenge the number of vehicles was large enough to form no practical restriction). Using an unlimited number of vehicles, feasibility of the problem is guaranteed, even in the dynamic setting (see next section).

1.3 Dynamic problem setting

For the dynamic variant of the problem, requests arrive at different 1-hour epochs during the day. The solver can choose which requests to dispatch in the current epoch, for which it must create feasible routes, and which ones to postpone to consolidate with new requests that may arrive later. Some requests are must-go requests that must be dispatched during the current epoch, as delaying them until next epoch will render their time window infeasible. In the last epoch, all request are must-go. Locations, demands, service times and time windows of requests are sampled for each epoch uniformly from the data of a static VRPTW instance, after which requests with infeasible time windows are filtered. To facilitate the evaluation, an environment is provided, to which a solution for the current epoch can be submitted. This is then validated by the environment, after which requests for the next epoch are sampled by the environment.

For the dynamic variant, the static instance from which requests are sampled can thus be interpreted as a 'customer base' from which requests may arrive. The coordinates, time windows, service times and demands define the distribution for the requests, and this information can be used by the solver in its dispatching strategy. The solver should be able to interact with the environment through a controller script, which will be used for final evaluation of the solver.

2 Participation

Participation is open to anyone, either alone or as a team. Teams (or individuals) should register through the form on https://euro-neurips-vrp-2022.challenges.ortec.com/ before September 30th, and include all relevant information (name and affiliation of team leader). Teams are free to choose a name for their team. A quickstart repository[2] is provided to facilitate participation.

3 Datasets & evaluation protocol

In the quickstart repository, a public dataset with 250 static VRPTW instances is provided. These are real instances provided by ORTEC, using data from a US-based grocery delivery service that have been anonymized. The instances contain between 200 and 900 customers. From these static instances, dynamic instances are created by the environment by sampling 100 requests (after which requests with infeasible time windows are filtered) from the static set of customers during a number of epochs that is between 5 and 9, depending on the instance. The public instances and the provided environment can be used to develop a solver or train a machine learning model.

Qualification phase During the qualification phase, teams should submit their solver which will be evaluated automatically on a small hidden set of instances, with statistics similar to the 250 public instances. Each instance will be solved by the solver twice: once as static problem and once as dynamic, hence the distribution of tested problems is 50% static and 50% dynamic. During the qualification phase, time limits are set as follows: for the static variant, the time limits are 3/5/8 minutes for instances with <300/300-500/500+ customers, respectively, whereas for the dynamic variant, it is 1 minute for each epoch. Note that the time limit is enforced per epoch, so unused time can't be 'transferred' to the next epoch. The results during the qualification phase will be shown on a public leaderboard, available as of August 1st. Participants are encouraged to submit their solvers early and a maximum of 1 submission per day is allowed during the competition.

Final phase Final testing will be performed using 100 hidden test instances with similar statistics. Teams should be qualified for final testing by submitting their solver during the qualification phase and scoring among the top 10 on the public leaderboard. Similar to the qualification phase, each instance is solved twice, as a static variant and as a dynamic variant. For final testing, the static problem time limits are 5/10/15 minutes for <300/300-500/500+ customers and for the dynamic problem 2 minutes per epoch.

3.1 Input format

All instances are provided in the VRPLib format, following the convention used by LKH3[3]. Following the convention in this format, the depot is referred to as node 1, and the remaining locations are 2 to n, but we generally refer to the depot as node 0 and locations as 1 to $n-1$ (e.g. in how the solution is represented). Instances files contain an 'EDGE_WEIGHT_SECTION' which contains the full duration matrix. Examples of the instance files and how to read them using Python can be found in the quickstart repository.

3.2 Environment

As the static problem is a special case of the dynamic problem, solvers will be evaluated on static and dynamic problem variants using the same protocol, using the environment that is implemented in Python and follows the OpenAI gym interface. It implements two methods: env.reset() and env.step(). The env.reset() function initializes the environment and returns an observation, as well as extra information that may be used by the solver. The env.step() function accepts an action and returns an observation, a reward, a flag indicating whether the environment is done and extra information. This terminology comes from reinforcement learning, where the solver can be seen as an agent interacting with an environment. In this terminology, the observation defines the VRPTW/request selection problem during each epoch, an action is a solution defining a set of routes to dispatch, and the reward

[2] https://github.com/ortec/euro-neurips-vrp-2022-quickstart
[3] http://webhotel4.ruc.dk/~keld/research/LKH-3/

is the negative of the cost (driving duration) of the solution (set of dispatched routes) for that epoch. The env.step() function moves the problem to the next epoch. Solving a single problem to completion (with multiple epochs) can be referred to as an episode of the environment.

Observation The observation contains the following information: the current epoch number, the current time, the start time of the planning for the current epoch (which equals the current time plus one hour margin to dispatch the vehicles) and the so called epoch instance, specifying the VRPTW/request selection problem to be solved for that epoch. The epoch instance contains the requests that are available and have not yet been dispatched in previous epochs. The epoch instance contains all information corresponding to a normal static VRPTW instance (coordinates, demands, vehicle capacity, time windows, service times, duration matrix), but additionally has a vector to indicate which requests must be dispatched in the current epoch, as their time windows prevent them from being dispatched in the next epoch. It also contains a vector with request ids (which can be used to identify requests across multiple epochs while dispatched requests are removed; request ids are thus not consecutive) and a vector with customer ids, indicating that the coordinates for that request correspond to a specific customer in the static instance (note that the demand, time window and service time do not correspond to the customer id but are sampled independently!). The time windows in the epoch instance are shifted to start at 0, so they can be solved by a static VRPTW solver that assumes the schedule always starts at time 0. For example, if the current time is 36000 (10:00), the planning start time is 39600 (11:00) (assuming one hour for dispatch), which means that a time window starting at 50400 (14:00) will start (after shifting) at 50400 - 39600 = 10800 (3:00) and the shifted depot start time will be 0 (0:00). Time windows starting before 0 (after shifting) are truncated to start at 0.

Action The agent/solver is free to select a subset of the requests to dispatch during the current epoch, as long as the subset contains all the orders with 'must-go' status. For the subset of requests selected, it must create a feasible set of routes according to the VRPTW problem definition. In the final epoch, all requests are 'must-go' and the agent/solver must dispatch all remaining orders. The solution/action should be a set of routes, where each route is a list of request ids in the order that they should be visited, represented as a list of list of integers. The solution must be submitted to the environment through the env.step() function, after which the environment will check the validity of the solution and move to the next epoch, and return the next observation, reward and an indicator if the environment is done.

Reward The reward returned by the environment is the negative of the VRPTW objective corresponding to the routes created (pure driving duration). The objective for the solver/agent solving the dynamic problem is to maximize the total sum of the rewards (also called the return), which amounts to minimizing the total driving duration of all routes created, i.e. the cost of the overall solution.

Termination The environment will return done as true if either the final epoch has been reached, or the agent/solver has submitted an invalid solution, in which case a large penalty is received as reward (i.e., solvers should ALWAYS ensure feasibility before submitting a solution during an epoch). If the solution is infeasible, the info object will have a non-empty property 'error' indicating the infeasibility. Also, if a submission is submitted using env.step() while the time limit per epoch (measured in wall clock time seconds) is exceeded, an error is returned and the environment terminates with a large penalty (there is a two second grace period to account for overhead and timing inaccuracies).

Resetting/initializing the environment The environment can be initialized with different seeds to generate different instances/scenarios for the dynamic problem. Also, an option can be provided to use the environment to evaluate the static problem, in which case there will be only a single epoch, and requests are not sampled but correspond directly to the customers in the static instance. In this case, all requests are must-go requests.

When solving the dynamic variant, the info object returned when calling env.reset() will provide the dynamic context information, which corresponds to the static instance coordinates, demands, time windows, etc. This information can be seen as defining the empirical distributions for requests (as the requests are sampled from these values), which can be taken into account by the solver/agent. We emphasize that coordinates, time windows and service times for requests are sampled independently, thus for the dynamic variant there is **no meaningful link** between coordinates and demands, time

windows or service times at the same position (e.g., for a machine learning model it has no meaning to combine the i-th coordinate with the i-th demand value as features). In addition to this context, the environment will indicate the start epoch (which is typically not 0, as 0 corresponds to 0:00), the end epoch and the wall-clock time limit per epoch that is allowed for the solver/agent.

3.3 IMPORTANT: the controller script

During development, solvers can directly use the Python environment, but at test time, **solvers should interact with the environment through the controller.py script**, by interfacing json messages! More specifically, at test time, the controller.py script (provided in the quickstart) will run the solver as a subprocess, while the solver can interact with the environment managed by the controller by writing and reading json from stdin/stdout. This way, internal info of the environment (specifically future requests) are hidden from the participant solver. Therefore **participants can only use the information returned by environment.reset() and environment.step()** and not rely on other methods or internal properties of the environment. Note that there is no requirement to implement the solver using Python, as long as the solver implements the same protocol of communicating through json messages. However, we recommend wrapping a non-Python solver in Python so the example implementation can be used for communication with the controller (the quickstart provides an example of this).

Note: it is possible to make the solver strategy depend on whether the problem variant is static or dynamic, as well as on the epoch time limit, number of epochs, etc. as this information is returned by env.reset().

To ensure that local testing is representative of the code submission process, the controller.py script as well as the implementation of the environment **should not be modified**.

4 Baselines

Static baseline. As a baseline, we provide an implementation of the Hybrid Genetic Search (HGS) algorithm [15, 16], which has been adapted to handle time-window constraints and dynamic request information. HGS iteratively creates new solutions by recombination and local search. This algorithm has produced the current state-of-the-art results for dozens of vehicle routing problem variants. [16] provide a simple C++ implementation of the HGS for the capacitated VRP (HGS-CVRP – https://github.com/vidalt/HGS-CVRP) that includes an additional local-search operator called SWAP*. The HGS algorithm has also been extended to vehicle routing problems with delivery time-windows [17] as well as many other variants (e.g., in [18]).

An extension of the HGS-CVRP has recently won the VRPTW track of the DIMACS implementation challenge [19]. This method uses the strategy of [17] to efficiently evaluate time-window constraints through the search. It includes an additional crossover from [20], dynamic parameters for population- and neighborhood-size management, along with some code optimizations. This implementation is used as the static baseline for this competition, and included in the "baselines/hgs_vrptw" folder.

Dynamic policies. All the vehicle routing variants solved with HGS until now were static (i.e., all problem information is known at the start of the solution process). The EURO Meets NeurIPS 2022 Vehicle Routing Competition goes beyond this scope, as the delivery requests are revealed dynamically in one of the two problem variants. Extending a static solver to deal with dynamic requests requires, in our situation, to choose which requests should be visited on any given epoch. Such an extension is non-trivial, and requires additional policies for customer selection.

To select the customers visited in each epoch, we provide three initial simple policies in the quickstart code base:

- **Greedy**: every available request is going to be dispatched within the current epoch. This means that there is no request who is not included in HGS route optimization.
- **Lazy**: only must-go requests will be dispatched. Must-go requests are requests that cannot be delayed to further epochs because of time windows violation (indicated by the environment).
- **Random**: requests to be dispatched in the current epoch are selected randomly: each request is dispatched with 50% probability, independently. Note that must-go customers will always be included in order to have feasible solutions.

These policies provide initial results for comparison, but other strategies and improvements are very likely possible.

Given the way the baseline code is structured, we note that there are two main directions for improvement: improvement of the request selection strategy and improvement of the underlying HGS algorithm itself. The request selection strategy is only relevant for the dynamic problem, whereas the static variant of the problem accounts for 50% of the final evaluation. However, as HGS has been battle-tested on static vehicle routing problems for over a decade, we can expect that the magnitude of the improvements achieved by better solving the dynamic problem with enhanced selection policies is larger than that of purely improving the solution process on the static case. This does not affect the importance of the two variants of the problem, given the way the final ranking of the solvers is computed (see the next session).

5 Scoring and submission system

Teams are expected to submit code that can generate a solution given an instance: for example, a hand-designed heuristic, or a policy trained with reinforcement learning. This allows us to test the submitted code on hidden test instances, which will determine the final winners based on the solution quality on those instances. The score for each instance is determined by the total time that all vehicles spent driving, but does not include the number of vehicles, service times, or waiting times of the vehicles.

These scores per instance are averaged for the hidden instances of the static problem for all teams, and also for the hidden instances of the dynamic problem for all teams, leading to a separate ranking of the teams on both problem variants. Then, the two rankings of each team are averaged, leading to a final ranking for each team for the whole competition. This ensure both problems are weighted equally regardless of the actual scores. In case of a tie, the scores themselves are averaged instead of the rankings.

For enabling code submission, we use the Codalab Competition[4] platform, which is available on August 1st. Instructions on code submission are available in the quickstart repository. The code will be executed inside an Apptainer/Singularity container, which is based on a Dockerfile[5]. See the Dockerfile for pre-installed dependencies. The environment has a single CPU core (Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz), a single GPU (Nvidia TitanRTX 24GB) and 32 GB of RAM.

6 Prizes

We provide the following awards, in total over 5000€.

1. **Best performance – First place:** 2022 €
2. **Best performance – Second place:** 750 €
3. **Best performance – Third place:** 500 €
4. **Jury prize:** 250 €
5. **Young talent prize:** 250 €
6. **Leaderboard prizes:** 1300 € total

The top three prizes are scored according to the leaderboard criteria described above on the hidden dataset in the finals. To further encourage innovative ideas, even if they do not lead to top performance, we include a 'jury prize' that will be awarded at the organizers' discretion. For this award, we seek an approach that (1) offers a novel solution approach, (2) uses ML, (3) achieves "good" performance[6] within the time limit, and (4) does not win first place.

The 'young talent prize' is awarded to the best performing team consisting only of students and/or PhD candidates that does not win any prize for best performance. While we acknowledge that student teams may be advised by a non-student, this person is not allowed to be part of the team, should contribute

[4] https://codalab.lisn.upsaclay.fr/

[5] https://github.com/TCatshoek/codalab-dockers/blob/master/legacy-gpu/Dockerfile

[6] We leave "good" performance vague on purpose.

no code/experimentation to the submission, and thus receives no credit for the accomplishments of the team. Note that the organizers reserve the right to request proof of student/PhD candidate status, which must have been valid at the time of the team's registration. Finally, the leaderboard prizes are 50/30/20€ awarded to the number 1/2/3 on the leaderboard every Monday (end of day UTC), from August 8th until October 31st (13 weeks). The leaderboard prizes will be paid at the end of the competition, only to teams with a minimum of 100€ total prize.

7 Schedule

1. **July 1** Quickstart guide and problem description made available
2. **August 1 (tentative)** Code submission opens & public leaderboard available
3. **September 31** Registration deadline
4. **October 31** Qualification phase: code submission deadline
5. **November** Finals: Top 10 public participants will be tested on hidden instances
6. **28 Nov – 9 Dec** Virtual Event at NeurIPS 2022 (NeurIPS registration is required!)

8 The fine print

Competition participants are expected to act ethically and fairly throughout the entire competition (and, hopefully, beyond). The organizing team reserves the right to disqualify any team that is found to not be acting in the spirit of the competition or upholding the standards expected of the research community. Furthermore, we will do our best to ensure a fair comparison between methods, but mistakes can happen. The decision of the organizing team on the winners of each prize is, thus, final.

9 Additional resources

Our github repository is located at: https://github.com/ortec/euro-neurips-vrp-2022-quickstart.

Still have a question? Feel free to join us on Slack: https://join.slack.com/t/euro-neurips-vrp-2022/shared_invite/zt-1dldr119v-lV7FMmuxhRdkfXr07Mzgfw.

To stay up-to-date, follow us on Twitter: https://twitter.com/EuroNeuripsVRP.

References

[1] T. Vidal, G. Laporte, and P. Matl, "A concise guide to existing and emerging vehicle routing problem variants," European Journal of Operational Research, vol. 286, pp. 401–416, 2020.

[2] V. Pillac, M. Gendreau, C. Guéret, and A. Medaglia, "A review of dynamic vehicle routing problems," European Journal of Operational Research, vol. 225, no. 1, pp. 1–11, 2013.

[3] G. Laporte, S. Ropke, and T. Vidal, "Heuristics for the vehicle routing problem," in Vehicle Routing: Problems, Methods, and Applications (P. Toth and D. Vigo, eds.), ch. 4, pp. 87–116, Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014.

[4] M. Poggi and E. Uchoa, "New Exact Algorithms for the Capacitated Vehicle Routing Problem," in Vehicle Routing: Problems, Methods, and Applications (P. Toth and D. Vigo, eds.), ch. 3, pp. 59–86, Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014.

[5] L. Costa, C. Contardo, and G. Desaulniers, "Exact branch-price-and-cut algorithms for vehicle routing," Transportation Science, vol. 53, no. 4, pp. 946–985, 2019.

[6] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," Advances in neural information processing systems, vol. 31, 2018.

[7] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!," in International Conference on Learning Representations, 2018.

[8] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "Pomo: Policy optimization with multiple optima for reinforcement learning," Advances in Neural Information Processing Systems, vol. 33, pp. 21188–21198, 2020.

[9] A. Hottung, B. Bhandari, and K. Tierney, "Learning a latent search space for routing problems using variational autoencoders," in International Conference on Learning Representations, 2020.

[10] L. Xin, W. Song, Z. Cao, and J. Zhang, "NeuroLKH: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem," Advances in Neural Information Processing Systems, vol. 34, 2021.

[11] P. da Costa, J. Rhuggenaath, Y. Zhang, A. Akcay, and U. Kaymak, "Learning 2-opt heuristics for routing problems via deep reinforcement learning," SN Computer Science, vol. 2, no. 5, pp. 1–16, 2021.

[12] L. Xin, W. Song, Z. Cao, and J. Zhang, "Multi-decoder attention model with embedding glimpse for solving vehicle routing problems," in Proceedings of 35th AAAI Conference on Artificial Intelligence, pp. 12042–12049, 2021.

[13] W. Kool, H. van Hoof, J. Gromicho, and M. Welling, "Deep policy dynamic programming for vehicle routing problems," in International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR), 2022.

[14] R. Bai, X. Chen, Z.-L. Chen, T. Cui, S. Gong, W. He, X. Jiang, H. Jin, J. Jin, G. Kendall, et al., "Analytics and machine learning in vehicle routing research," arXiv preprint arXiv:2102.10012, 2021.

[15] T. Vidal, T. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, "A hybrid genetic algorithm for multidepot and periodic vehicle routing problems," Operations Research, vol. 60, no. 3, pp. 611–624, 2012.

[16] T. Vidal, "Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood," Computers and Operations Research, vol. 140, p. 105643, 2022.

[17] T. Vidal, T. Crainic, M. Gendreau, and C. Prins, "A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows," Computers & Operations Research, vol. 40, no. 1, pp. 475–489, 2013.

[18] T. Vidal, T. Crainic, M. Gendreau, and C. Prins, "A unified solution framework for multi-attribute vehicle routing problems," European Journal of Operational Research, vol. 234, no. 3, pp. 658–673, 2014.

[19] W. Kool, J. O. Juninck, E. Roos, K. Cornelissen, P. Agterberg, J. van Hoorn, and T. Visser, "Hybrid genetic search for the vehicle routing problem with time windows: a high-performance implementation," 12th DIMACS Implementation Challenge Workshop, 2022.

[20] Y. Nagata, O. Bräysy, and W. Dullaert, "A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows," Computers & Operations Research, vol. 37, no. 4, pp. 724–737, 2010.