Genetic Programming Hyper-Heuristic for the Dynamic Electric Dial-a-Ride Problem

William Huang*, Yi Mei*, Günther Raidl†, Fangfang Zhang*, Laurenz Tomandl†,
Steffen Limmer‡, Mengjie Zhang* and Tobias Rodemann‡
*Centre for Data Science and Artificial Intelligence, Victoria University of Wellington, Wellington, New Zealand
†Algorithms and Complexity Group, TU Wien, Vienna, Austria
‡HRI-EU – Honda Research Institute Europe, Offenbach/Main, Germany

2025 IEEE Congress on Evolutionary Computation (CEC) | 979-8-3315-3431-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/CEC65147.2025.11042942

***Abstract*—This paper studies the Dynamic Electric Dial-A-Ride Problem (DEDARP), which is a combinatorial optimisation problem that has applications in real-world ridesharing services with electric vehicles. In addition to the challenges from classical scheduling and route planning, we consider here the extra challenge of making real-time dispatching decisions in dynamic environments with new requests arriving over time and selecting proper times for the vehicles to recharge. To solve DEDARP effectively, we propose a Genetic Programming Hyper-Heuristic (GPHH) that evolves heuristics/policies to dispatch vehicles in real time. We have developed a simulation process that generates a solution for any given instance by two policies, one for vehicle allocation and the other for request allocation, and design fitness evaluations based on the simulation. Moreover, we propose a multi-tree GP to evolve these two policies simultaneously, which makes use of advanced terminals to comprehensively represent the state. Experimental results on a wide range of instances show that GPHH can evolve effective policies that make significantly better real-time dispatching decisions than human-designed policies based on prior knowledge.**

I. Introduction

The Dial-a-Ride Problem (DARP) is a well known challenging combinatorial optimisation problem with applications in modern on-demand public transport services [1]. It aims to dispatch vehicles to serve a set of customer requests with pick-up and drop-off locations within given time windows. Unlike traditional taxi/Uber services, passengers may share the rides in DARP; thus, limited detours from the most direct routes between the customers' pick-up and drop-off locations may occur. This way, the overall efficiency may be greatly improved and costs reduced. This paper investigates more specifically the Dynamic Electric Dial-a-Ride Problem (DEDARP), which is a special DARP variant in which electric vehicles are considered with their necessity of charging. Moreover, online decision-making is necessary as knowledge of requests to serve only becomes available over time.

For dynamic optimisation problems, existing methods can be mainly divided into three categories: (1) robust proactive approaches which aim to find robust solutions that can perform reasonably well (possibly with naive recourse operators) in all possible environments; (2) completely reactive approaches that make real-time decisions without needing a preplanned solution; and (3) proactive-reactive approaches that consist of an offline-optimised solution and a (fast) re-optimisation process applied when an environment change is detected.

We consider solving the DEDARP by a reactive approach, i.e., learning policies that can make effective real-time decisions. This approach has shown success in many dynamic optimisation and decision-making problems [2]–[4] due to its ability to make effective and truly real-time decisions after an off-line training procedure. In contrast, the robust proactive approach is often highly inflexible, and the robustness of the obtained single solution is usually limited. The proactive-reactive approach is more widely used in a rolling horizon manner, but it is non-trivial to predefine the trade-off between the effectiveness and efficiency of re-optimisation, since the time allowed for re-optimisation is often unknown in advance.

For learning real-time decision-making policies, existing studies mainly consider reinforcement learning and Genetic Programming (GP). Reinforcement learning methods formulate the problem as a Markov Decision Process (MDP) by defining the states, possible actions, a policy, and a reward function. However, the performance of reinforcement learning highly depends on the problem-specific design, especially the reward function, and is also sensitive to parameter settings such as the learning rate. Moreover, with the commonly adopted neural network representation of the policy, it is difficult for users to understand the learned policy. A GP approach learns the policies as a hyper-heuristic. Each possible policy is an individual in GP, and its fitness is evaluated by applying the policy to a set of real-time decision-making simulations. Compared to reinforcement learning, GP requires no specific reward function, the learning process often is more stable, and it has higher potential to obtain more interpretable policies due to its symbolic representation.

In this paper, we aim to solve DEDARP by a GP Hyper-Heuristic (GPHH), with the following research objectives:

1) We develop a DEDARP simulation to reflect real-world scenarios. Given a DEDARP instance and a policy for the real-time decisions, the simulation always generates a feasible solution in terms of all the constraints.
2) We design a set of potentially useful terminals/features to form a rich search space of policies.
3) We develop a GPHH method with a simulation-based fitness evaluation based on the designed terminal set.

4) We verify the effectiveness of the GPHH method by comparing with manually designed rules on a wide range of test instances.

The rest of the paper is organised as follows. Section II gives background information, including the problem description and related work. Section III describes the proposed GPHH method. The experimental studies are conducted in Section IV. Finally, the paper is concluded in Section V.

II. Background

A. Problem Description

The Dynamic Electric Dial-a-Ride Problem (DEDARP) we consider here is defined as follows. We are given a transportation network $G(V, A)$. Each node $v \in V$ is a relevant location in the considered geographical area, and each directed arc $(u, v) \in A$ represents the fastest route from node u to node v. For each arc (u, v), we know the travel time $\Delta t(u, v)$ and the battery charge consumption $\Delta b(u, v)$. There are in total N requests arriving in real time. Each request $i \in \{1, ..., N\}$ is characterised by an arrival time $t_i^{\text{arr}} \in [0, T]$ (T is the considered time horizon), pick-up and drop-off locations $\langle v_i^{\downarrow}, v_i^{\uparrow} \rangle \in V \times V$, a demand (number of customers) d_i, a time window for the pick-up $[t_i^{\text{start}}, t_i^{\text{end}}] \subset [0, T]$, and a maximal ride time $\Delta t_i^{\max}$. These requests are to be served by a fleet of K Electric Autonomous Vehicles (EAVs). Each EAV $k \in \{1, ..., K\}$ has a customer capacity Q, a battery capacity B and an initial battery charge level $b_k(0) \in [0, B]$. There are C charging stations in the transportation network, where the EAVs can recharge. Each charging station $c \in \{1, ..., C\}$ has a location $\ell_c \in V$ and a charging rate α.

The goal of DEDARP is to dispatch the EAVs to serve the requests in real time. In the dynamic environment, the information of a request i is unknown until its arrival time t_i^{arr}. A DEDARP solution $X = \{X_1, ..., X_K\}$ consists of a route for each of the K vehicles. Each vehicle route X_k is a sequence of visits of the pick-up, drop-off, and charging station locations. The objective is to minimise the total travel time of the vehicles in serving the requests, plus a penalty for requests served after t_i^{end}. Specifically,

$$\min \sum_{k=1}^{K} \Delta t(X_k) + \rho \cdot \sum_{i=1}^{N} \max\{t_i^{\downarrow} - t_i^{\text{end}}, 0\}, \qquad (1)$$

where $\Delta t(X_k)$ is the total travel time of the vehicle route X_k, $t_i^{\downarrow}$ is the pick-up time of request i, and ρ is a weight to balance between the total travel time and the penalty.

The following constraints are to be met in DEDARP.

- Each request i is served by exactly one vehicle, and $v_i^{\downarrow}$ must be visited before $v_i^{\uparrow}$ in the route.
- If a vehicle arrives at a pick-up location $v_i^{\downarrow}$ earlier than t_i^{start}, it has to wait until t_i^{start} to start the service.
- For each request, the service duration is zero (it is assumed to be integrated in the travel time).
- For each request i, the maximal ride time must not be exceeded, i.e., $t_i^{\uparrow} - t_i^{\downarrow} \leq \Delta t_i^{\max}$.
- Each vehicle must always retain a positive charge level.
- Only an empty vehicle that currently fulfilling no requests can go to recharge. Once it starts recharging, it cannot leave until fully charged. The charging time is ${}^{(B-b_k)}\!/\!_{\alpha}$, where b_k is the battery level of the vehicle when it arrives at the charging station.
- Once a vehicle becomes empty, it can accept requests from the waiting pool (see below) or go to recharge.
- Once a vehicle finishes charging, it can accept requests from the waiting pool, but cannot go to recharge again.
- If an empty vehicle does not accept any requests and does not go to recharge, it goes into waiting, where it may only accept newly arrived requests.

B. Related Work

1) Existing Related Methods: DEDARP inherits the technical challenges of the complex characteristics of DARP, and introduces the challenges of battery charging decisions and real-time dynamic environment decision-making. So far, there are only a few existing studies that consider all these challenges. Bongiovanni et al. [5] considered a DEDARP with dynamic request arrivals and proposed a two-phase heuristic approach to deal with each dynamic request arrival. Specifically, once a new request arrives in real time, it is first attempted to be inserted into the current remaining vehicle routes. Then, local search is used to improve the routes until the next request arrives. In [6], a new learning large neighborhood search was developed to improve performance during the second phase of the aforementioned method. Kullman et al. [7] formulated the problem as a Markov Decision Process (MDP) and used deep reinforcement learning to learn a policy to make real-time decisions for electric vehicles.

Other variants of the problem that consider only subsets of the above challenges are more widely studied. The basic DARP model has been extensively investigated in the past decades, and there have been many exact methods (e.g., [8], [9]) and meta-heuristic approaches (e.g., [10]–[12]) proposed. Comprehensive surveys can be found in [1], [13]. The static electric DARP has been less investigated than the basic DARP variant, but still some exact methods (e.g., [14]) and meta-heuristics ([15]–[19]) have been proposed for it.

The dynamic DARP without the battery/recharging constraints is considered in some existing studies. For example, Coslovich et al. [20] proposed a two-phase heuristic approach for it. In the first phase, the method searches a neighbourhood of the current route until the vehicle arrives at the next stop. In the second phase, if a new request arrives, then it is inserted into the route based on the updated neighbourhood in real time. Tafreshian et al. [21] developed a similar two-phase method, where a pool of candidate solutions are found offline, and the best solutions are selected on-the-fly. Sayarshad et al. [22], [23] formulated the dynamic DARP as a MDP, and proposed non-myopic pricing policies to make effective and efficient decisions during the process. Heitmann et al. [24] proposed a method that accelerates value function approximations under the MDP formulation. Liu et al. [25] developed a guided insertion method for solving dynamic large-scale DARP instances with up to 300,000 requests with real-time request arrivals. It uses machine learning techniques to learn simplified routes, called vehicle travel patterns, from historical data of the same/similar distributions. The learned travel patterns can perform effective real-time insertion of new requests. Gaul et al. [26] proposed a rolling horizon approach that divides the decision process into sliding windows, each of which is a static subproblem to be re-optimised by mathematical programming in real time. Amiri et al. [27] developed an accelerated column generation method to carry out real-time re-optimisation whenever a request arrives. In addition to dynamic DARP, there are existing studies for related dynamic routing problems, such as [28], [29].

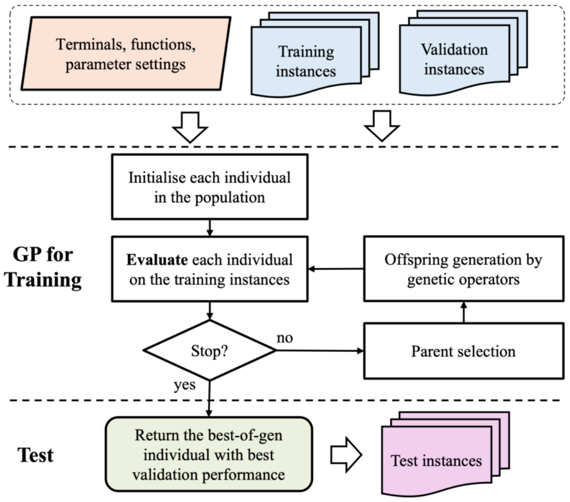
2) Genetic Programming for Policy Learning: GP has been successfully employed in hyper-heuristic optimisation for learning policies that can make effective and efficient real-time decisions in complex optimisation problems involving dynamic and/or stochastic environments, such as scheduling heuristics for dynamic job shop scheduling [3], [30], [31], routing policies for dynamic routing problems [32]–[35], and real-time resource allocation in cloud computing [36], [37]. However, GP has not yet been considered for DEDARP.

III. The Proposed Method

Our GPHH for DEDARP is illustrated in Fig. 1. It consists of a *training* phase and a *test* phase. During the training phase, given a set of training DEDARP instances, along with the defined terminals, functions, and other parameter settings, GP is used to evolve a population of individuals. Each individual is a pair of a vehicle allocation policy (a policy used to select vehicles) and a request allocation policy (a policy used to select requests) $(\pi^{\text{veh}}, \pi^{\text{req}})$, where each policy is represented as a tree.

The evolutionary process follows a standard GP process. At first, we initialise each policy in each individual by the ramp-half-and-half method and evaluate each initial individual (details shown in Section III-C). In each generation, we create an empty new population, and copy the top individuals from the current population into it (i.e., elitism). Then, we keep generating offspring to fill the new population until it is full. For each offspring generation, we select the parent(s) by tournament selection and randomly select a genetic operator (crossover, mutation, or reproduction) to generate offspring from the parent(s). For crossover, we apply the diversity-preserving crossover operator [38], which has shown to be effective in multi-tree GP. Given two pairs of policies as parents, this operator randomly selects one type of policy from both parents and conducts standard tree-based crossover on them. Then, it swaps the other type of policy between the two parents. For mutation, given a pair of policies as the parent, we randomly select a policy and conduct standard tree-based mutation on it. More details about these crossover and mutation operators can be found in [38]. For reproduction, the select parent is directly copied as the offspring. Finally, we evaluate the fitness of each offspring individual.

The evolutionary process yields at its end the overall best individual found to be later deployed to the unseen test instances during the test phase. The best individual is the one across all generations that generalises the best to an unseen validation instance set.



Terminals, functions, parameter settings
Training instances
Validation instances
Initialise each individual in the population
GP for Training
Evaluate each individual on the training instances
Offspring generation by genetic operators
Stop?
no
yes
Parent selection
Test
Return the best-of-gen individual with best validation performance
Test instances

Fig. 1. Flowchart for the proposed GPHH method for DEDARP.

A. Representation of Individuals

The proposed GPHH evolves the vehicle allocation policy and request allocation policy simultaneously, and thus adopts a multi-tree representation. Each individual contains two trees, one for the vehicle allocation policy and the other for the request allocation policy. An example of the multi-tree representation is shown in Fig. 2. In the example, the vehicle allocation policy $\pi^{\text{veh}} = \text{TVPU} * \text{CRD}$, where TVPU indicates the travel time from the vehicle to the pick-up location of the request, and CRD represents the crowdedness of the region of the request. The request allocation policy $\pi^{\text{req}} = \text{TVPU} + \text{SLACK}$, where SLACK is the slack of the request[1]. Depending on the defined terminal, function sets, and maximal tree depth, we are able to obtain much more complex policies than the two shown in Fig. 2.

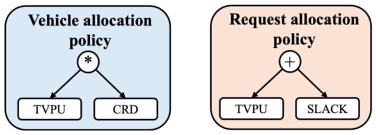


Fig. 2. An example of the multi-tree representation in the proposed GPHH.

B. Terminals and Functions

Suitable problem-specific terminals and functions are essential for determining the search space of GP. For DEDARP, we aim to design a minimal terminal set that enables a most effective decision-making for the vehicles and requests. To this end, we design the terminals based on our prior knowledge

[1]Feature details will be given in Table I.

about the most relevant/useful features for dispatching vehicles to requests, which are given in Table I. They are categorised into three groups: vehicle features V, request features R, and features of a (vehicle, request) pair (V,R).

TABLE I
TERMINALS USED IN THE PROPOSED GPHH FOR DEDARP.

Symbol	**Group**	**Description**
TVPU	(V,R)	Travel time from the vehicle to the pick-up location of the request.
COST	(V,R)	Expected cost (additional objective value caused) for the vehicle to serve the request.
OBV	(V,R)	The best (smallest) waiting time out of all vehicles other than the specified vehicle.
DEM	R	Demand of the request.
DUR	R	Request duration: travel time from the pick-up to the drop-off location of the request.
SLACK	R	Slack of the request: t_i^{end} minus current time minus earliest arrival time of all the vehicles.
CRD	R	Crowdedness of the request, estimated by the average travel time from its pick-up location to all the pick-up and drop-off locations of the existing requests.
CHRQ	R	Whether the request is a recharging decision or an actual request (the two are considered in the same capacity by the problem simulation).
RQ	V	Remaining capacity of the vehicle.
RT	V	Remaining travel time of the vehicle due to its remaining battery.
FRT	V	Remaining travel time of the vehicle due to its remaining battery at its latest known point.
VSLACK	V	Vehicle slack: the minimal slack of the requests in the vehicle's planned route.
TVC	V	Travel time from the vehicle to the closest charging station.

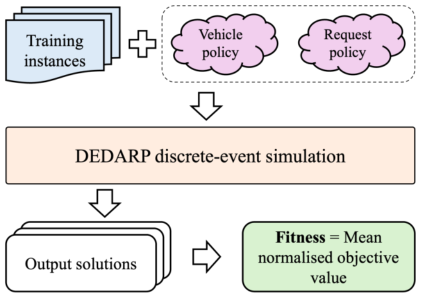
As function set we use $\{+,-,*,/,\min,\max\}$, which are the commonly used functions in GPHH for dynamic combinatorial optimisation problems. The "/" operator is the protected division, which returns one in case of a division by zero.

C. Fitness Evaluation

The fitness evaluation is the key component that distinguishes the proposed GPHH for DEDARP from existing GPHH methods for other problems. The evaluation scheme we use is defined as follows (also shown in Fig. 3). Given an individual $(\pi^{\text{veh}},\pi^{\text{req}})$ to be evaluated and a set of training instances, our scheme runs a *DEDARP discrete-event simulation* on each training instance by using the individual's policies as the decision-making policies for the simulation, generating a corresponding solution. Each generated solution is then evaluated in terms of the objective function. Finally, the fitness is calculated as the mean normalised objective value among all the generated solutions. Specifically,

$$\mathrm{fit}(\pi^{\text{veh}},\pi^{\text{req}}) = \frac{1}{|\mathcal{P}^{\text{train}}|}\sum_{P\in\mathcal{P}^{\text{train}}} \widehat{\mathrm{obj}}(X(\pi^{\text{veh}},\pi^{\text{req}};P)), \quad (2)$$

where $\widehat{\mathrm{obj}}(\cdot)$ is the normalised objective value, and $X(\pi^{\text{veh}},\pi^{\text{req}};P)$ is the solution obtained by the DEDARP disrete-event simulation with $(\pi^{\text{veh}},\pi^{\text{req}})$ on instance P.



Training instances
Vehicle policy
Request policy
DEDARP discrete-event simulation
Output solutions
Fitness = Mean normalised objective value

Fig. 3. The fitness evaluation process.

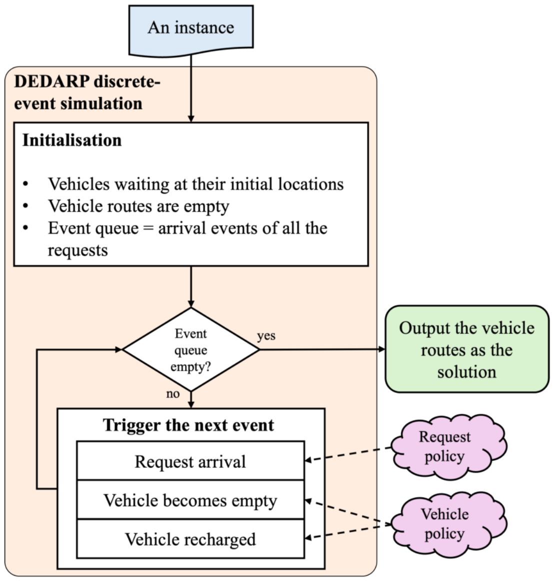


Fig. 4. The DEDARP discrete-event simulation.

D. Simulation Description

Details of the DEDARP discrete-event simulation are shown in Fig. 4. Given an instance, the state and event queue are first initialised: All vehicles are waiting at their initial locations with empty routes, and the event queue is set to the arrival events of all the requests. Then, the events in the queue are triggered one by one until the event queue becomes empty. Finally, the vehicle routes along with the departure and arrival times are returned as solution.

There are three types of events: (1) when a new request arrives, (2) when a vehicle completes a *sub-route*, i.e., a sequence of connected requests has been fulfilled and the EAV becomes empty, and (3) when a vehicle has been recharged. These events are triggered in increasing order of their time (e.g., arrival time of the request, time when charging finished). The processes triggered by the three types of events are shown in Figs. 5–8. Note that when triggering each event, a new event might be created and inserted into the event queue.

When a new request i arrives (Fig. 5), the vehicle allocation policy is called to calculate the priority value for each feasible waiting vehicle to the new request. Feasibility is determined by whether a vehicle can serve a request subject to battery,

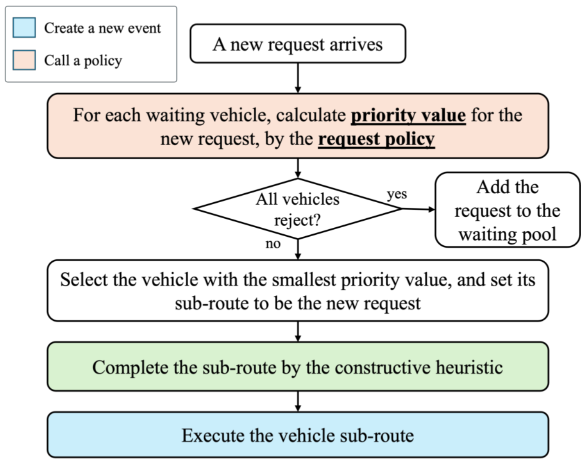

Create a new event
Call a policy
A new request arrives
For each waiting vehicle, calculate priority value for the new request, by the request policy
All vehicles reject?
yes
no
Add the request to the waiting pool
Select the vehicle with the smallest priority value, and set its sub-route to be the new request
Complete the sub-route by the constructive heuristic
Execute the vehicle sub-route

Fig. 5. Handling a request arrival event.

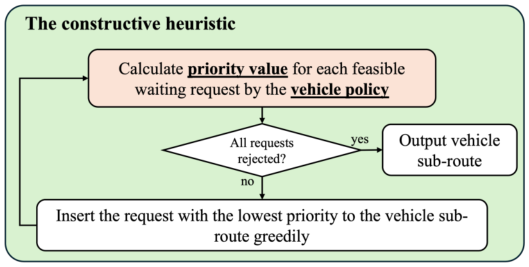

The constructive heuristic
Calculate priority value for each feasible waiting request by the vehicle policy
All requests rejected?
yes
no
Output vehicle sub-route
Insert the request with the lowest priority to the vehicle sub-route greedily

Fig. 6. The constructive heuristic to complete a vehicle sub-route.

demand, and maximal ride time constraints. If all the vehicles reject the new request (e.g., their priority values for the request exceed the threshold of zero), then we add the request to the waiting pool. Otherwise, we select the vehicle with the smallest priority value among those accepting the request. Then, we use a constructive heuristic to complete that vehicle's sub-route, and lastly, we execute the sub-route.

The constructive heuristic is shown in Fig. 6. At each step, the request allocation policy is called to calculate the priority value for each feasible waiting request to the vehicle whose sub-route is being constructed. If all the requests are rejected (e.g., their priority values for the vehicle exceed the threshold of zero), then it yields the vehicle sub-route. Otherwise, it inserts the accepted request with the lowest priority value to the sub-route greedily.

When a vehicle becomes empty again (Fig. 7), its sub-route is set to empty, and the request allocation policy is called to calculate the priority value for each feasible waiting request, as well as a recharging decision (a proxy request that involves recharging the vehicle at the nearest charging station). If all the candidates are rejected (e.g., their priority values for the vehicle exceed the threshold of zero), then let the vehicle wait. Otherwise, we select the candidate with the lowest priority value. If the recharging decision is selected, then the vehicle goes to recharge. Otherwise, we set the vehicle sub-route to be the selected request, complete the sub-route by the constructive heuristic, and then execute the sub-route.

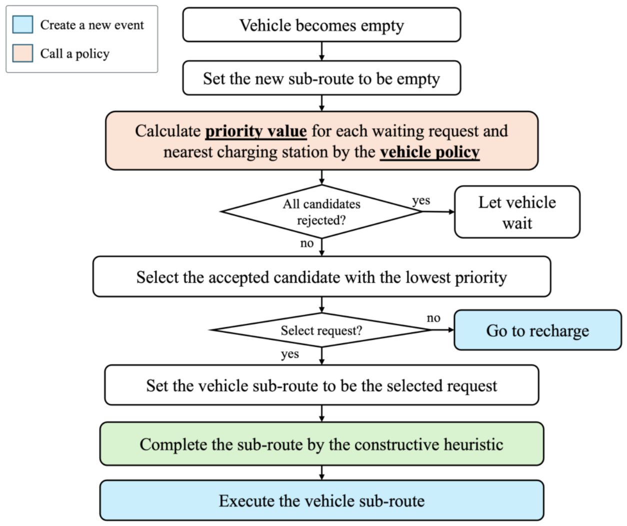

Create a new event
Call a policy
Vehicle becomes empty
Set the new sub-route to be empty
Calculate priority value for each waiting request and nearest charging station by the vehicle policy
All candidates rejected?
yes
no
Let vehicle wait
Select the accepted candidate with the lowest priority
Select request?
no
yes
Go to recharge
Set the vehicle sub-route to be the selected request
Complete the sub-route by the constructive heuristic
Execute the vehicle sub-route

Fig. 7. Handling a vehicle empty event.

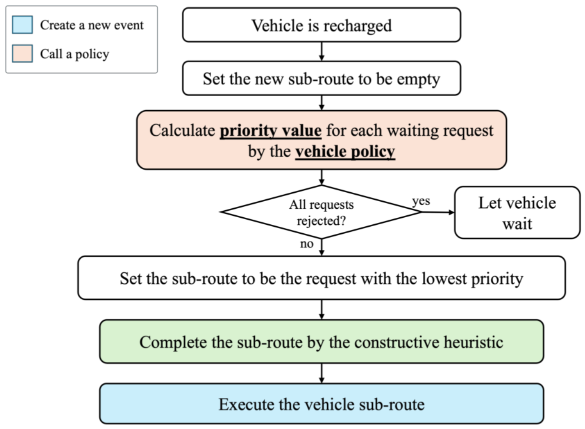

Create a new event
Call a policy
Vehicle is recharged
Set the new sub-route to be empty
Calculate priority value for each waiting request by the vehicle policy
All requests rejected?
yes
no
Let vehicle wait
Set the sub-route to be the request with the lowest priority
Complete the sub-route by the constructive heuristic
Execute the vehicle sub-route

Fig. 8. Handling the vehicle recharged event.

When a vehicle finishes recharging (Fig. 8), follow the same process as when a vehicle becomes empty, except that the charging decision is excluded from the candidate pool.

IV. Experimental Studies

A. Experiment Settings

In the experiments, we generated a set of synthetic DEDARP instances as follows. We randomly sampled N requests over a 24-hr time horizon. The locations of the requests were sampled uniformly at random from the $[-1000, 1000] \times [-1000, 1000]$ euclidean square. Travel times between each pair of locations were set to the Euclidean distance multiplied by a factor ϕ plus a constant service duration t^{serv}, i.e., $\Delta t(u, v) = \phi \cdot ||u - v||_2 + t^{\text{serv}}$. The energy consumption was set to the Euclidean distance multiplied by a factor β, i.e., $\Delta b(u, v) = \beta \cdot ||u - v||_2$. The arrival time of the requests follows a homogeneous Poisson process. The start time for each pickup is sampled uniformly after its arrival time $t_i^{\text{start}} \in [t_{\text{arr}}, t_{\text{arr}} + \Delta T]$ with the end of the time window being 15 minutes after its arrival time $t_i^{\text{end}} = t_{arr} + 15$. For

TABLE II
THE PARAMETERS FOR GENERATING THE INSTANCES.

Parameter	Meaning	Value(s)
N	Number of requests	10, 20, 50
K	Number of EAVs	2
C	Number of charging stations	3
T	Time horizon (mins)	1440
Q	Request capacity	3
B	Battery capacity	15.0
β	Energy consumption rate	0.0005
α	Charging rate	0.05
t^{serv}	Service duration	0
ϕ	Travel time rate	0.01
ρ	Weight of penalty in the objective	2.0
ΔT	Arrival time constant	180.0

the sake of simplicity, the demand of all the requests were set to one. Table II shows the parameters for instance generation.

Based on the number of requests, there are three DEDARP scenarios, i.e., 10, 20, and 50 requests. For each scenario, we randomly generated 250 training instances, 500 validation instances, and 500 test instances.

As there is no existing rule or learning algorithm for DEDARP, we manually designed intuitive policies based on two heuristics: nearest neighbour and lowest cost. For the nearest neighbour heuristic, at each decision point, the priority value is defined as the distance between the considered vehicle and request. For the lowest cost heuristic, at each decision point, the priority value is defined as the increase in objective value caused by the resulting sub-route. During training, validation, and test, the nearest neighbour policy was used to normalise the objective values, i.e.,

$$\widehat{\text{obj}}(X(\pi^{\text{veh}}, \pi^{\text{req}}; P)) = \frac{\text{obj}(X(\pi^{\text{veh}}, \pi^{\text{req}}; P))}{\text{obj}(X(\pi^{\text{NN}}, \pi^{\text{NN}}; P))}. \quad (3)$$

In training, the GP parameters were set in a standard manner, in which the population size is 1000, the number of generations is 50, the tournament size for parent selection is 7, and the crossover, mutation, and reproduction rates are 0.8, 0.15, and 0.05, respectively. The 250 training instances were divided into 50 subsets, each with 5 instances to be used in the fitness evaluation for a particular generation. After the training, the best individual of each generation was evaluated using the 500 validation instances, and the one with the best validation performance was selected. We conducted 30 independent GP runs, and compared the 30 results with the nearest neighbour policy by normalisation. All the experiments were run on Intel Core 4.8 GHz and 16G memory.

B. Results and Discussions

1) Test Performance: The first graph in Fig. 9 shows the test performance of the GP-evolved rules in the three scenarios. It is clear that in all the scenarios, the GP-evolved rules achieved much better performance than the nearest neighbour policy (i.e., the normalised values are significantly smaller than one). Furthermore, the advantage of GP greatly increases with the

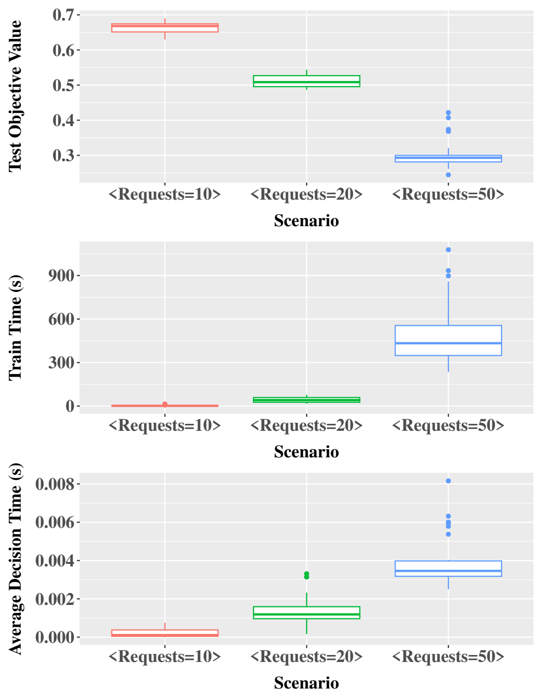


Fig. 9. Normalised test objective values, GPHH training time, and average decision time of the simulation during test.

TABLE III
RAW COMPARISON OF TEST OBJECTIVE VALUES (MEAN ± STANDARD DEVIATION).

Scenario	Nearest Neighbour	Lowest Cost	GP Policies
10-Request	371.47	309.39	246.46 ± 5.92
20-Request	1088.43	791.73	557.78 ± 20.15
50-Request	8216.66	9308.85	2478.54 ± 336.21

problem size. For the small scenario (10 requests), the objective values were about 65% of those obtained by the nearest neighbour policy, and for the large scenarios (50 requests), the objective values were around 30% of those obtained by the nearest neighbour policy. This trend is particularly noticeable given the additional context provided by the raw objective values as shown in Table III, where GP policies significantly outperformed the two manual policies on all the scenarios.

2) Computational Time: The second graph in Fig. 9 shows the training time of the GP runs. For the scenario with 10 requests, the training time was very short (less than 10 seconds). For the scenario with 20 requests, the training time increased to about 50 seconds. For the scenario with 50 requests, the training time increased to 500 seconds on average. In summary, while GP training time increases drastically with problem size, it ultimately remains affordable—though it is yet to be seen if this holds true for even larger problem instances with hundreds or thousands of requests.

The third graph in Fig. 9 shows the distribution of time taken by the GP-evolved rules at each decision point in the simulation for the three scenarios. It can be seen that the

TABLE IV
AVERAGE POLICY SIZE OF THE BEST INDIVIDUALS (MEAN ± STANDARD DEVIATION).

Scenario	VP Size	RP Size	Combined Size
10 Requests	31.67 ± 20.17	41.33 ± 17.59	73.00 ± 27.78
20 Requests	39.80 ± 17.42	40.87 ± 21.67	80.67 ± 31.62
50 Requests	21.87 ± 16.99	46.80 ± 21.97	68.67 ± 27.85

decision time always is negligible (less than 1 millisecond) in all scenarios. Of course, the decision time increases with the problem size, but it remains within limits to clearly meet practical real-time decision-making requirements.

C. Further Analysis

Table IV shows the size (number of nodes) of the vehicle allocation policy (VP) and request allocation policy (RP) obtained by GP in the three scenarios. The combined size is the sum of the two policy sizes. From the table, we can see that the policy size is relatively independent of the instance size. In other words, the policy does not become more complex for larger instances. In addition, we also observe that the vehicle allocation policy tends to have a smaller size than the request allocation policy. This is possibly due to the fact that the request allocation policy is used in a greater variety of decision states and thus, requires higher complexity.

1) Feature Importance: Fig. 10 shows the frequency of terminals in the best GP individuals obtained in the three scenarios. From the figure, we see that the terminal frequencies are relatively consistent across all scenarios. The COST, SLACK, and OBV were among the most often-used terminals in the best policies. This is consistent with our prior knowledge that (1) COST is directly related to the objective value; (2) SLACK is directly related to the likelihood of delaying the serving of a request, and (3) OBV reflects the best alternative option. Besides, the other terminals were also used reasonably often as illustrated in the figure.

2) Structural Analysis: Fig. 11 and Fig. 12 show an example vehicle allocation policy and request allocation policy obtained for the scenario with 50 requests. It can be seen that the request allocation policy tends to be more complex than the vehicle allocation policy. This is likely due to its usage in a wider variety of decisions, i.e., the vehicle allocation policy is only used for incoming requests, whereas the request allocation policy is used for the constructive heuristic, allocation upon a vehicle becoming empty, and allocation upon a vehicle becoming fully recharged. Furthermore, the request allocation policy is in charge of recharging and waiting decisions, which are some of the most complex decisions in the entire problem simulation. Thus, the request allocation policy requires a higher degree of sophistication than the vehicle allocation policy.

V. CONCLUSIONS

This work aimed to automatically design effective policies to make real-time decisions in the DEDARP for dispatching

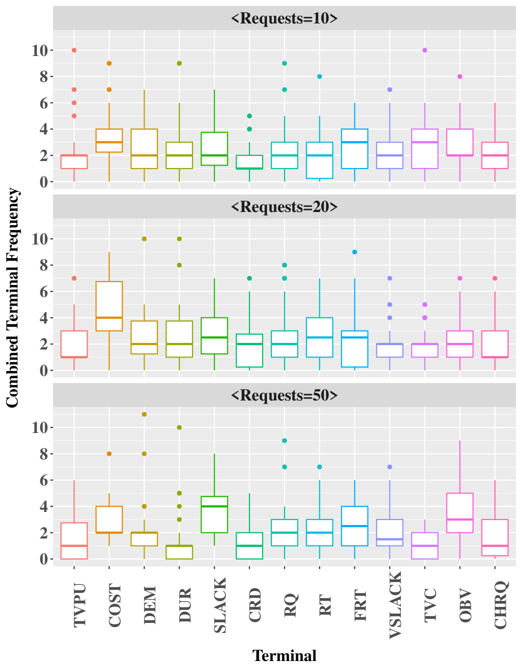


Fig. 10. Frequency of terminals in the best policies.

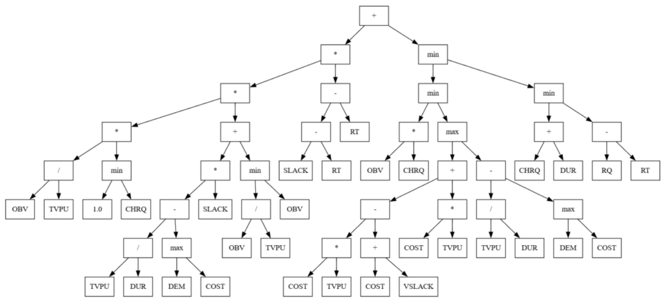


Fig. 11. An example vehicle allocation policy.

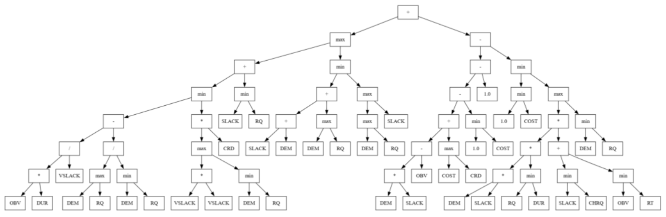


Fig. 12. An example request allocation policy.

EAVs to fulfill dynamically arriving request. This goal has been successfully achieved by the developed GPHH method with the sophisticated event-driven simulation adopting the vehicle and request allocation policies. The experimental results show that the GP-evolved policies can greatly outperform manual policies designed based on intuition. The learned policies can make effective decisions in milliseconds, meeting real-time decision requirements. This demonstrates the high potential of GPHH in addressing this problem.

Future research directions include further improving the simulation to capture more nuanced behaviour, designing more potentially informative terminals to enrich the GP search space, testing on larger DEDARP instances and instances with inhomogeneous arrival rates of requests, and considering advanced technologies for improving training efficiency and generalisation. We will also consider more realistic scenarios on real-world data.

References

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

[2] D. Vengerov, "A reinforcement learning approach to dynamic resource allocation," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 3, pp. 383–390, 2007.

[3] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated design of production scheduling heuristics: A review," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, 2015.

[4] F. Zou, G. G. Yen, L. Tang, and C. Wang, "A reinforcement learning approach for dynamic multi-objective optimization," *Information Sciences*, vol. 546, pp. 815–834, 2021.

[5] C. Bongiovanni, M. Kaspi, and N. Geroliminis, "A two phase heuristic approach for the dynamic electric autonomous dial-a-ride problem," in *7th Symposium of the European Association for Research in Transportation*, 2018.

[6] C. Bongiovanni, M. Kaspi, J.-F. Cordeau, and N. Geroliminis, "A learning large neighborhood search for the dynamic electric autonomous dial-a-ride problem," in *8th Symposium of the European Association for Research in Transportation*, 2019.

[7] N. D. Kullman, M. Cousineau, J. C. Goodson, and J. E. Mendoza, "Dynamic ride-hailing with electric vehicles," *Transportation Science*, vol. 56, no. 3, pp. 775–794, 2022.

[8] J.-F. Cordeau, "A branch-and-cut algorithm for the dial-a-ride problem," *Operations research*, vol. 54, no. 3, pp. 573–586, 2006.

[9] M. Liu, Z. Luo, and A. Lim, "A branch-and-cut algorithm for a realistic dial-a-ride problem," *Transportation Research Part B: Methodological*, vol. 81, pp. 267–288, 2015.

[10] J.-F. Cordeau and G. Laporte, "A tabu search heuristic for the static multi-vehicle dial-a-ride problem," *Transportation Research Part B: Methodological*, vol. 37, no. 6, pp. 579–594, 2003.

[11] R. M. Jorgensen, J. Larsen, and K. B. Bergvinsdottir, "Solving the dial-a-ride problem using genetic algorithms," *Journal of the operational research society*, vol. 58, no. 10, pp. 1321–1331, 2007.

[12] S. N. Parragh, K. F. Doerner, and R. F. Hartl, "Variable neighborhood search for the dial-a-ride problem," *Computers & Operations Research*, vol. 37, no. 6, pp. 1129–1138, 2010.

[13] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem: models and algorithms," *Annals of operations research*, vol. 153, pp. 29–46, 2007.

[14] Y. Su, N. Dupin, S. N. Parragh, and J. Puchinger, "A Branch-and-Price algorithm for the electric autonomous Dial-A-Ride Problem," *Transportation Research Part B: Methodological*, vol. 186, p. 103011, Aug. 2024.

[15] M. A. Masmoudi, M. Hosny, E. Demir, K. N. Genikomsakis, and N. Cheikhrouhou, "The dial-a-ride problem with electric vehicles and battery swapping stations," *Transportation Research Part E: Logistics and Transportation Review*, vol. 118, pp. 392–420, Oct. 2018.

[16] Y. Molenbruch, K. Braekers, O. Eisenhandler, and M. Kaspi, "The electric dial-a-ride problem on a fixed circuit," *Transportation Science*, vol. 57, no. 3, pp. 594–612, 2023.

[17] Y. Su, N. Dupin, and J. Puchinger, "A deterministic annealing local search for the electric autonomous dial-a-ride problem," *European Journal of Operational Research*, vol. 309, no. 3, pp. 1091–1111, Sep. 2023.

[18] S. Limmer, "Bilevel large neighborhood search for the electric autonomous dial-a-ride problem," *Transportation Research Interdisciplinary Perspectives*, vol. 21, p. 100876, Sep. 2023.

[19] M. Bresich, G. R. Raidl, and S. Limmer, "Letting a large neighborhood search for an electric dial-a-ride problem fly: On-the-fly charging station insertion," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2024, pp. 142–150.

[20] L. Coslovich, R. Pesenti, and W. Ukovich, "A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem," *European Journal of Operational Research*, vol. 175, no. 3, pp. 1605–1615, Dec. 2006.

[21] A. Tafreshian, M. Abdolmaleki, N. Masoud, and H. Wang, "Proactive shuttle dispatching in large-scale dynamic dial-a-ride systems," *Transportation Research Part B: Methodological*, vol. 150, pp. 227–259, Aug. 2021.

[22] H. R. Sayarshad and J. Y. J. Chow, "A scalable non-myopic dynamic dial-a-ride and pricing problem," *Transportation Research Part B: Methodological*, vol. 81, pp. 539–554, Nov. 2015.

[23] H. R. Sayarshad and H. Oliver Gao, "A scalable non-myopic dynamic dial-a-ride and pricing problem for competitive on-demand mobility systems," *Transportation Research Part C: Emerging Technologies*, vol. 91, pp. 192–208, Jun. 2018.

[24] R. J. O. Heitmann, N. Soeffker, F. Klawonn, M. W. Ulmer, and D. C. Mattfeld, "Accelerating value function approximations for dynamic dial-a-ride problems via dimensionality reductions," *Computers & Operations Research*, vol. 167, p. 106639, Jul. 2024.

[25] C. Liu, A. Quilliot, H. Toussaint, and D. Feillet, "A Guided Insertion Mechanism for Solving the Dynamic Large-Scale Dial-a-Ride Problem," 2024.

[26] D. Gaul, K. Klamroth, and M. Stiglmayr, "Solving the Dynamic Dial-a-Ride Problem Using a Rolling-Horizon Event-Based Graph," in *21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021)*, M. Müller-Hannemann and F. Perea, Eds., vol. 96. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 8:1–8:16.

[27] E. Amiri, A. Legrain, and I. El Hallaoui, "Accelerated column generation: Application in real-time dial-a-ride problem," *Les Cahiers du GERAD ISSN*, vol. 711, p. 2440, 2024.

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

[29] H. Tong, L. L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "A novel generalized metaheuristic framework for dynamic capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 6, pp. 1486–1500, 2022.

[30] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: A survey with a unified framework," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 41–66, Mar. 2017.

[31] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 28, no. 1, pp. 147–167, 2023.

[32] Y. Mei and M. Zhang, "Genetic Programming Hyper-Heuristic for Stochastic Team Orienteering Problem with Time Windows," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. IEEE, Jun. 2018, pp. 1–8.

[33] ——, "Genetic Programming Hyper-Heuristic for Multi-Vehicle Uncertain Capacitated Arc Routing Problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO)*. ACM, Jul. 2018, pp. 141–142.

[34] D. Jakobović, M. Durasević, K. Brkić, J. Fosin, T. Carić, and D. Davidović, "Evolving dispatching rules for dynamic vehicle routing with genetic programming," *Algorithms*, vol. 16, no. 6, p. 285, 2023.

[35] F. J. Gil-Gala, M. Durasević, and D. Jakobović, "Evolving routing policies for electric vehicles by means of genetic programming," *Applied Intelligence*, vol. 54, no. 23, pp. 12 391–12 419, 2024.

[36] M. Xu, Y. Mei, S. Zhu, B. Zhang, T. Xiang, F. Zhang, and M. Zhang, "Genetic programming for dynamic workflow scheduling in fog computing," *IEEE Transactions on Services Computing*, vol. 16, no. 4, pp. 2657–2671, 2023.

[37] Z. Sun, Y. Mei, F. Zhang, C. Gu, H. Huang, and M. Zhang, "Multi-tree genetic programming hyper-heuristic for dynamic flexible workflow scheduling in multi-clouds," *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 2687–2703, 2024.

[38] F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with multi-tree representation for dynamic flexible job shop scheduling," in *Proceedings of the Australasian Joint Conference on Artificial Intelligence (AI)*. Springer, 2018, pp. 472–484.