Evolving Heuristics for Dynamic Vehicle Routing with Time Windows Using Genetic Programming

Josiah Jacobsen-Grocott, Yi Mei, Gang Chen, Mengjie Zhang
[*] School of Engineering and Computer Science, Victoria University of Wellington
PO Box 600, Wellington 6140, New Zealand
{yi.mei, aaron.chen, mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—Dynamic vehicle routing problem with time windows is an important combinatorial optimisation problem in many real-world applications. The most challenging part of the problem is to make real-time decisions (i.e. whether to accept the newly arrived service requests or not) during the execution of the routes. It is hardly applicable to use the optimisation methods such as mathematical programming and evolutionary algorithms that are competitive for static problems, since they are usually time-consuming, and cannot give real-time responses. In this paper, we consider solving this problem using heuristics. A heuristic gradually builds a solution by adding the requests to the end of the route one by one. This way, it can take advantage of the latest information when making the next decision, and give immediate response. In this paper, we propose a *meta-algorithm* to generate a solution given any heuristic. The meta-algorithm maintains a set of routes throughout the scheduling horizon. Whenever a new request arrives, it tries to re-generate new routes to include the new request by the heuristic. It accepts the new request if successful, and reject otherwise. Then we manually designed several heuristics, and proposed a genetic programming-based hyper-heuristic to automatically evolve heuristics. The results showed that the heuristics evolved by genetic programming significantly outperformed the manually designed heuristics.**

I. Introduction

The Vehicle Routing Problem (VRP) [1] is an important combinatorial optimisation problem that has a wide range of real-world applications in supply chain and logistics-related areas. It is to design a set of routes, each for a vehicle, to serve the given requests from different locations subject to some constraints such as the capacity constraint. VRP has a number of variations, among which the VRP with Time Windows (VRPTW) [2], [3] is one of the most commonly investigated one. In VRPTW, each request has a time window that specifies the desired period that the customer likes the service to occur. Then, VRPTW aims to design routes that minimises the violation of the time windows of the requests. There have been extensive studies for solving VRPTW, and many effective algorithms (e.g. [4]–[7]) have been proposed to solve it.

In reality, the environment is usually dynamic and new requests may arrive in real time during the execution of the routes. A typical example is the same-day pick-up and delivery service provided by the delivery companies. In this case, the vehicles are sent out to serve the existing requests, and new requests may arrive while the vehicles are still on their ways. The company aims to accept as many of the new requests as possible subject to the capacity of the vehicles and the fulfillment of the time windows. Whenever a new request arrives, it is necessary to decide whether to accept or reject the new request immediately so that the customer will be notified in time. This raises the Dynamic VRPTW (DVRPTW) [8].

In the dynamic environment, the commonly used solution-optimisation algorithms such as evolutionary algorithms [9], variable neighborhood search [10] and ant colony optimisation [11] are hardly applicable due to their high computational complexity. They are based on iterative search framework, which normally takes time to reach high-quality solutions. On the contrary, heuristics (e.g. the savings heuristic [12]) can give immediate and reasonably good responses, and thus are well suited for the dynamic environment. The recent advance of telecommunication technologies also makes it much easier to communicate between the management centre and the vehicles. Therefore, more and more studies consider solving dynamic VRPs using heuristics (or so-called *routing policies*) [13].

There have been a number of works for designing different types of heuristics for constructing routes or modifying the predetermined routes in real time [13]. However, the effectiveness of a heuristic largely depends on the scenario, objective(s), and even the graph topology. Therefore, it is hard to manually design an effective heuristic for a particular problem scenario of interest.

Recently, automated design of heuristics using hyper-heuristics [14] has attracted more and more research interests. Genetic Programming (GP) [15], [16] is one of the most commonly used hyper-heuristic method due to its flexible representation. It has been used for automatically evolving heuristics for a wide range of combinatorial optimisation problems such as production scheduling [17], knapsack problem [18], timetabling problem [19] and arc routing problem [20]. However, no effort has been made for DVRPTW so far.

In this paper, we aim to develop a GP-based hyper-heuristic approach for automatically evolving heuristics for DVRPTW. To be specific, the paper consists of the following goals:

- Develop a meta-algorithm that can generate a solution given any heuristic and any problem instance;
- Manually design heuristics for DVRPTW based on domain knowledge. These manually designed heuristics will be used as the baseline benchmark heuristics;
- Develop a GP-based Hyper-Heuristic (GPHH) for automatically evolving heuristics;

- Investigate the effectiveness of the GP-evolved heuristics in comparison with the manually designed heuristics;
- Investigate the effectiveness of different routing strategies in the meta-algorithm including the *driving-first* and *waiting-first* strategies [21].

The rest of the paper is organised as follows: Section II gives the background introduction of the problem and related work. Section III describes the proposed GPHH method for evolving heuristics for DVRPTW. Then, Section IV shows the experimental studies and discussions. Finally, Section V concludes the paper and discusses about future directions.

II. Background

A. Dynamic Vehicle Routing Problem with Time Windows

DVRPTW was formally defined in [22]. In DVRPTW, suppose we have a connected graph $G = (V, E)$, where V is the vertex set and E is the edge set. There is a special vertex $v_0 \in V$ which is called the depot vertex. There are k vehicles, each with capacity Q, located at the depot vertex for serving the requests. Each pair of vertex $(v_i, v_j) \in V \times V$ is associated with a travel time $c(v_i, v_j)$. Without loss of generality, we assume that the graph G is fully connected, and $c(v_i, v_j)$ indicates the travel time of the shortest path from v_i to v_j, $\forall\ v_i, v_j \in V$.

The request arrival process is defined as a discrete event simulation within a time horizon $[0, T]$. Each request τ is characterised as $\tau = (v(\tau), t(\tau), s(\tau), d(\tau), l(\tau), u(\tau))$, where $v(\tau) \in V \setminus v_0$ is the vertex (location) where the request is invoked, $t(\tau)$ means the time when the request is invoked, $s(\tau)$ is the service time of the request, $d(\tau)$ indicates the demand of the request, and $l(\tau)$ and $u(\tau)$ stand for the lower and upper bounds of the time window of the request. Obviously, $0 \le t(\tau) \le l(\tau) \le u(\tau) \le T$. The request set Υ contains a subset Υ_1 of requests that already exist at the beginning of the period and a subset Υ_2 of requests that arrive during the execution of the routes. That is, $t(\tau) = 0, \forall \tau \in \Upsilon_1$ and $t(\tau) > 0, \forall \tau \in \Upsilon_2$. The information of a request is not known until it arrives.

The solution of DVRPTW can be seen as a decision making process, in which a decision is made whenever a new request arrives or a vehicle becomes available to serve the next request. When a new request arrives, a decision needs to be made on whether to accept or reject the new request while not changing the decisions that have already made for the previous requests (including the requests in Υ_1). When a vehicle becomes available and there are requests waiting in the queue, a decision needs to be made on which is its next request to serve. The solution has to satisfy the following constraints:

- Each vehicle starts and ends its route at the depot;
- Each request is served exactly once by one vehicle (no interruption);
- The total demand of the requests served by each vehicle does not exceed its capacity (*capacity constraint*);
- The starting time of each service cannot be outside (earlier than the lower bound or later than the upper bound of) the time window of the request (*time window constraint*);

If a request cannot be served feasibly (i.e. in terms of the capacity and time window constraints), then it has to be rejected. The objective of the problem is to find a feasible solution that maximises the number of accepted requests.

A solution S to DVRPTW can be represented as a set of routes $S = \{R_1, R_2, \ldots, R_k\}$. Each route $R_i = (\tau_0, \tau_{i1}, \ldots, \tau_{i,l_i}, \tau_0)$ is a sequence of requests, where $\tau_0 = (v_0, 0, 0, 0, 0, T)$ represents a special request of visiting the depot node. Then, the problem can be formulated as follows:

$$\max \sum_{i=1}^{k} l_k, \tag{1}$$

$$\text{s.t.: } \tau_{ij} \neq \tau_{uv}, \forall i \neq u \text{ or } j \neq v, i \in \{1, \ldots, k\}, j \in \{1, \ldots, l_i\}, \tag{2}$$

$$\sum_{j=1}^{l_i} d(\tau_{ij}) \le Q, \forall i \in \{1, \ldots, k\}, \tag{3}$$

$$l(\tau_{ij}) \le t_s(\tau_{ij}) \le u(\tau_{ij}), i \in \{1, \ldots, k\}, j \in \{1, \ldots, l_i\}, \tag{4}$$

$$t_s(\tau_{ij}) \ge arr(\tau_{ij}), i \in \{1, \ldots, k\}, j \in \{1, \ldots, l_i\}, \tag{5}$$

$$\tau_{ij} \in \Upsilon, i \in \{1, \ldots, k\}, j \in \{1, \ldots, l_i\}. \tag{6}$$

where Eq. 1 is the objective function, which is maximising the number of accepted requests. Note that in [22], the objective was defined as minimising the rejected requests. Given the same number of requests in total, these two objectives are equivalent. Eq. (2 means that each request is served exactly once by one vehicle. Eq. (3 indicates the capacity constraint, and Eq. (4) refers to the time window constraint, where $t_s(\tau)$ stands for the time to start serving τ. Eq. (5) specifies that the service of each request cannot be started until the vehicle arrives its location, where $arr(\tau)$ indicates the time that the vehicle arrives $v(\tau)$. It is decided as follows:

$$arr(\tau_{i1}) \ge \max\{0, t(\tau_{i1})\} + c(v_0, v(\tau_{i1})), \quad \forall i \in \{1, \ldots, k\}, \tag{7}$$

$$arr(\tau_{ij}) = dep(\tau_{i,j-1}) + c(v(\tau_{i,j-1}), v(\tau_{ij})), \quad \forall i \in \{1, \ldots, k\}, j \in \{2, \ldots, l_i\}, \tag{8}$$

$$dep(\tau_{ij}) \ge t_s(\tau_{ij}) + s(\tau_{ij}), \quad \forall i \in \{1, \ldots, k\}, j \in \{1, \ldots, l_i - 1\}. \tag{9}$$

$$dep(\tau_{i,j-1}) \ge t(\tau_{ij}), \quad \forall i \in \{1, \ldots, k\}, j \in \{2, \ldots, l_i\}. \tag{10}$$

where $dep(\tau)$ represents the departure time of the vehicle after finishing the service of τ. Eq. (7) means that for each route, the arrival time of the first request is no earlier than the travel cost from v_0 to its location since its arrival (0 if it already exists initially). Eq. (8) indicates that for each subsequent request, the arrival time equals the departure time of its predecessor plus the travel time in between. Eq. (9) means that the vehicle cannot depart until it finishes serving the request. Eq. (10) means that the vehicle cannot go to the next request before its arrival.

B. Related Work on Dynamic Vehicle Routing

Pillac et al. [8] and Ritzinger et al. [13] gave two comprehensive surveys for DVRP. They introduced different problem variations such as deterministic and stochastic DVRPs, as well as commonly used approaches for solving DVRP. For example, one can divide the whole time horizon into short time slices, and periodically solve the corresponding optimisation problem at the beginning of each time slice (e.g. [23]). Another alternative is to keep a pool of good and diversely distributed solutions during the search process, so that when the environment changes, at least one of the solutions in the pool still performs well. An example is the adaptive memory proposed by Taillard et al. [24].

During the execution process, a number of heuristics have been proposed to decide the next customer to visit, such as the consensus, expectation and regret methods. The consensus method [25] selects the customer that appears the most frequently in the past history or sampled scenarios. The expectation method [25] evaluates the cost of visiting each customer by forcing its visit and then optimise for the remaining customers. The regret method [26] is an approximation of the expectation method.

Note that after deciding the next customer to go, there is still another issue to determine the departure time. For example, the vehicle can depart immediately to arrive the next customer as soon as possible (i.e. *drive-first*) or wait at the current location for a while (i.e. *wait-first*), as long as all the remaining customers can still be served in time. The waiting strategy has been demonstrated to be effective in many scenarios (e.g. [27]).

Saint-Guillain et al. [22] proposed the DVRPTW, and proposed to solve the problem using a Multi Scenario Approach (MSA). The main idea is to randomly sample a number of scenarios based on certain distribution, and conduct robust optimisation so that the obtained solution performs well on all the sampled scenarios. Then, whenever a decision needs to be made (i.e. a new request arrives or a vehicle becomes available), a short re-optimisation is carried out by means of an online decision rule called the Global Stochastic Assessment (GSA) rule.

C. Genetic Programming for Evolving Heuristics

GP has been used for evolving heuristics for many challenging combinatorial optimisation problems. For example, Branke et al. gave a review for evolving production scheduling dispatching rules with GP [17]. Burke et al. [16] explored the potential of GP as a hyper-heuristic for evolving heuristics, and showed the applications to the SAT problem and online bin packing. This paper pointed out the key steps for designing a hyper-heuristic: (1) designing a framework (*meta-algorithm*) for heuristics to operate in; (2) decide on the terminal and function sets and (3) identify the fitness function.

Weise et al. [20] proposed a GPHH approach for the Arc Routing Problem (ARP), which is the counterpart of VRP that serves edges/arcs instead of nodes. They proposed a meta-algorithm that can generate a feasible solution given any instance, and defined a heuristic as a priority function of each unserved arc. The proposed GPHH showed promising results on both static benchmark instances and stochastic instances in comparison with manually designed heuristics.

Sim and Hart [28] proposed a hyper-heuristic approach for VRP, which is a combination of a GP-evolved generative heuristic and a perturbative heuristic to further improve the solution generated by the generative heuristic. The results showed that the proposed hyper-heuristic performed competitively when applied to solve a wide range of different types of VRP instances. However, there is no GPHH proposed for DVRPTW so far. In this paper, we fill this gap by propose a new GPHH that considers both dynamic request arrivals and time window constraint.

III. Genetic Programming Hyper-Heuristic for DVRPTW

There are two major differences between DVRPTW and other VRP variants. First, in VRP, while adding the new customers, it is possible to remove some less profitable customers from the remaining routes. However, in DVRPTW, when deciding whether to accept each new request or not, all the requests that have already been accepted cannot be rejected again. That is, rejecting a request that is previously accepted will cause an infinite penalty. Second, in many VRP variants, it may not be necessary to make acceptance/rejection decisions for all the existing customers. However, in DVRPTW, such decision has to be made for each request as soon as it arrives. Due to these two differences, the existing meta-algorithms (i.e. frameworks that heuristics operate on) cannot be applied directly to DVRPTW. Therefore, we developed a new meta-algorithm for DVRPTW.

A. Meta-algorithm

The proposed meta-algorithm can be seen as a decision process. Whenever a new request arrives, a decision needs to be made on whether to accept or reject the new request without rejecting any requests that have already been accepted before. Note that at the beginning of the scheduling horizon, there may already exists a set of initial requests (i.e. $\Upsilon_1 \neq \emptyset$). In this case, the acceptance/rejection decisions are made for all the initial requests at the beginning.

Note that in DVRPTW, the requests that have been accepted before cannot be rejected again. In other words, when accepting a new request, one has to make sure all the previously accepted requests can still be served. For this purpose, we maintain a complete feasible solution for serving all the accepted requests at all times. At the beginning, we generate a feasible solution (by heuristic or optimisation method) that serves as many initial requests as possible, and reject the initial requests that cannot be served. Then, whenever a new request arrives, we generate a new solution based on the current vehicle states and request set plus the new request. If the new solution successfully serves all the requests, then we accept the new request, otherwise we reject it.

In this algorithm, a solution S is represented as a sequence of actions, i.e. $S = (\pi_1, \pi_2, \dots)$, where each action $\pi = \langle ty_\pi, st_\pi, et_\pi, ve_\pi, \tau_\pi \rangle$ is characterised by its type ty_π, start time st_π, end time et_π, vehicle ve_π and request τ_π. Here we define two types of actions: (1) *travel* from one node to another and (2) *serve* a request.

Given the above representations, the proposed meta-algorithm is described in Algorithm 1.

Algorithm 1: The meta-algorithm of the GPHH for DVRPTW.

```
   Input: A problem instance $I$, a heuristic $h(\cdot)$.
   Output: A solution $S$.
   // Initialise the state
1  Set $t \leftarrow 0$, $\Upsilon_t \leftarrow \Upsilon_1$;
2  for $i = 1 \rightarrow k$ do
3      Set vehicle location at $loc_i \leftarrow v_0$, remaining capacity
         $\bar{Q}_i \leftarrow Q$, available time $a_i \leftarrow 0$;
4  end
5  Set the state $\Omega \leftarrow \prod_{i=1}^{k} \langle loc_i, \bar{Q}_i, a_i \rangle$;
   // Generate a solution for initial requests
6  if $\Upsilon_t \neq \emptyset$ then $S \leftarrow$ InitialSolution($\Omega, \Upsilon_t$) ;
   // Online decision making
7  while $t \leq T$ do
8      Extract the next arrived request $\tau'$ from the random arrival
         process;
9      for unfinished action $\pi \in S$ do
10         if $st(\pi) < t(\tau')$ then Execute $\pi$ and update $\Omega$ ;
11     end
12     $\Upsilon_t \leftarrow \Upsilon_t \cup \tau'$, $S' \leftarrow S$;
13     $S \leftarrow$ GenerateSolution($t, \Omega, \Upsilon_t$);
14     if $S$ serves all the requests in $\Upsilon_t$ then
15         Accept $\tau'$;
16     else
17         Reject $\tau'$, $\Upsilon_t \leftarrow \Upsilon_t \setminus \tau'$, $S \leftarrow S'$;
18     end
19     $t \leftarrow t(\tau')$;
20 end
21 return $S$;
```

In line 10, an action updates the state Ω as follows:

- When a vehicle travels from one node to another, its location is changed to the new node and its available time becomes the end time of the action;
- When a vehicle serves a request, its remaining capacity is deducted by the demand of the request and its available time becomes the end time of the action. The request is removed from the request set.

In Algorithm 1, a key component is the function GenerateSolution$(\cdot)$, which generates a complete solution based on the current state Ω and request set Υ_t. Since the acceptance/rejection decision of the new request needs to be made immediately, we choose an efficient heuristic which is similar to those used for VRP [28] and ARP [20]. The algorithm is described in 2. At each step, the simulation time is incremented to the time when the next vehicle becomes available. Then, the servable requests are identified (line 4). A request is servable to a vehicle if (1) the vehicle has sufficient remaining capacity to serve it, and (2) the vehicle can arrive the location no later than the upper bound of the time window. If there is no servable request for the vehicle, it goes back to the depot, refill its capacity and wait for potential future requests. Otherwise, we select the request with the minimal value of the heuristic value $h(\cdot)$.

After deciding the next request, the departure time of the vehicle is decided (line 10). The decision on the departure time is not a trivial task, since there may be a wide feasible range to choose from. It is known that $t \leq dep \leq u(\tau') - c(loc_{i^*}, v(\tau'))$. In this case, we consider two different strategies for deciding the departure time: (1) driving-first and (2) wait-first. The two strategies are described as follows:

- *Driving-first* strategy: the vehicle departs as soon as possible, and waits at the request location if necessary. That is, $dep \leftarrow t$;
- *Wait-first* strategy: the vehicle waits at the current location if it is expected to arrive earlier than the time window. That is, $dep \leftarrow \max\{l(\tau') - c(loc_{i^*}, v(\tau')), t\}$.

Algorithm 2: $S \leftarrow$ GenerateSolution(t, Ω, Υ_t)

```
   Input: Current time $t$, state $\Omega$ and request set $\Upsilon_t$.
   Output: A solution $S$.
1  while $t \leq T$ do
2      Find the next available vehicle and time
         $(i^*, a^*) = \min_{i=1}^{k}\{a_i\}$;
3      $t \leftarrow a^*$;
4      $\bar{\Upsilon}_t \leftarrow$ ServableRequests($\Upsilon_t, i^*$);
5      if $\bar{\Upsilon}_t = \emptyset$ then
           // Go back to depot
6          $S \leftarrow (S, \langle \text{travel}, t, t + c(loc_{i^*}, v_0), i^*, \emptyset \rangle)$;
7          $\bar{Q}_{i^*} \leftarrow Q$, $a_{i^*} \leftarrow t + c(loc_{i^*}, v_0)$, $loc_{i^*} \leftarrow v_0$;
8      else
           // Select the request with the minimal
              heuristic value
9          $\tau' \leftarrow \arg\min_{\tau \in \bar{\Upsilon}}\{h(\tau)\}$;
10         Decide the departure time $dep$;
11         $arr \leftarrow dep + c(loc_{i^*}, v(\tau'))$;
12         $t_s \leftarrow \max\{arr, l(\tau')\}$;
13         $S \leftarrow (S, \langle \text{travel}, dep, dep + c(v_{\text{curr}}, v(\tau')), i^*, \emptyset \rangle)$;
14         $S \leftarrow (S, \langle \text{serve}, t_s, ts + s(\tau'), i^*, \tau' \rangle)$;
15         $loc_{i^*} \leftarrow v(\tau')$, $\bar{Q}_{i^*} \leftarrow \bar{Q}_{i^*} - d(\tau')$, $a_{i^*} \leftarrow t_s + s(\tau')$;
16     end
17 end
18 return $S$;
```

The initial solution S is generated by the function InitialSolution$(\cdot)$. The function can be defined as a constructive heuristic or a search-based optimisation method. For the sake of simplicity, in our experiment we generate the initial solution using Algorithm 2. That is, InitialSolution(Ω, Υ_t) = GenerateSolution$(0, \Omega, \Upsilon_t)$.

B. Manually Designed Heuristics

From Algorithm 2, it can be seen that the heuristic function $h(\cdot)$ plays an important role in generating a high-quality solution. Here, we design three simple heuristic functions manually based on our domain knowledge and intuition. They are the *earliest-first* (EF), *urgent-first* (UF) and *linear-combination* (LC) heuristic functions. They are defined as follows:

- EF heuristic: first serve the request that can be started the earliest, i.e. $h(\tau) = \max\{t + c(loc, v(\tau)), l(\tau)\}$, where loc is the current location of the vehicle;
- UF heuristic: first serve the request that is the most urgent, i.e. whose time window will close earliest. $h(\tau) = u(\tau)$;
- LC heuristic is a more sophisticated heuristic that considers a number of attributes including the following attributes: $A_1 = \max\{t + c(loc, v(\tau)), l(\tau)\}$ (EF heuristic), $A_2 = u(\tau)$ (UF heuristic), $A_3 = c(loc, v(\tau))$ (travel time from the current location), $A_4 = (1 - \bar{Q}/Q)c(v(\tau), v_0)$ (travel time to the depot times the current normalised load), $A_5 = d(\tau)$ (demand), and $A_6 = \min_{i=1}^{k}\{c(loc_i, v(\tau))\}$ (travel time to the nearest vehicle). The heuristic function is defined as a weighted linear combination as follows: $h(\tau) = A_1 + A_2 + A_3 + 0.1A_4 + 0.1A_5 - A_6$.

In the experimental studies, these three heuristic will be used as the baseline heuristics to compare with the GP-evolved heuristics.

C. Genetic Programming-based Hyper-Heuristic

In the GPHH, a heuristic function is represented as a syntax tree. The terminal set is given in Table I. The *vertex density* reflects the density of vertices around the given vertex, where the scaling parameter s was set to $s = \max\{d(v_i, v_j)\}/100$ after some parameter tuning. The *vehicle density* indicates the density of vehicles around the given vertex. It was defined in the same way as the vertex density, except that the scaling parameter was set to $s' = \max\{d(v_i, v_j)\}/8$.

TABLE I
The terminals used in the GPHH for DVRPTW

Notation	Description
TD	Normalised travel time to the depot $c(v(\tau), v_0)/T$
T	Normalised travel time from current location $c(loc, v(\tau))/T$
OT	Normalised relative open time $\max\{l(\tau) - t, 0\}/T$
CT	Normalised relative close time $(u(\tau) - t)/T$
ST	Normalised service time $s(\tau)/T$
Q	Normalised remaining capacity $\bar{Q}/Q$
DEM	Normalised demand $d(\tau)/Q$
VTD	Vertex density $\sum_{v' \in V, v' \neq v(\tau)} e^{-\frac{1}{2}\left(\frac{c(v', v(\tau))}{s}\right)^2}$
VHD	Vehicle density $\sum_{i=1, i \neq i^*}^{k} e^{-\frac{1}{2}\left(\frac{c(loc_i, v(\tau))}{s'}\right)^2}$
TOV	Travel time from nearest other vehicle $\min_{\substack{i=1\\ i \neq i^*}}^{k} \{c(loc_i, v(\tau))\}$
1	The constant 1

In addition to the above terminals, we designed two more terminals about the probabilistic information of the requests as follows:

- *Expected number of future requests at* $v(\tau)$ (NFR)
- *Probability that a new request arrives at* $v(\tau)$ *within the next* $T/6$ *time* (PNR)

These two terminals are calculated based on the stochastic information (temporal distributions of requests during the horizon) given by the dataset. Based on the terminals, we propose the following two GP versions:

- *GP1* with the terminals shown in Table I;
- *GP2* with the terminals shown in Table I as well as NFR and PNR.

For the function set, we used the basic arithmetic operators including addition, subtraction, multiplication and protected division (returns 1 if denominator is zero), along with the non-linear operators $\max(\cdot, \cdot)$ and $exp(\cdot)$.

The fitness of a GP individual is simply defined as the average objective value over a set of training instances. Given a set of training instances $\mathcal{I}_{tr}$, the fitness function of a heuristic function h is defined as follows:

$$fit(h) = \frac{1}{|\mathcal{I}_{tr}|} \sum_{I \in \mathcal{I}_{tr}} f(S(I, h)), \tag{11}$$

where $S(I, h)$ is the solution obtained by applying the meta-algorithm (Algorithm 1) to the instance I and heuristic h, and $f(S)$ indicates the number of rejected requests in the solution S.

IV. Experimental Studies

We use the benchmark instances designed by Saint-Guillain et al. [22]. The benchmark consists of 6 classes, with the degree of dynamism from ranging low to high. Our experiments focus on classes 4, 5 and 6, which are highly dynamic instances (with average degree of dynamism of 57% for class 4, 81% for class 5 and 100% for class 6). Each class contains 3 scenarios (namely rc101, rc102 and rc104), which are based on the same graph with 100 nodes. Each scenario has the same distribution of request arrivals. In the datasets, for each scenario, 5 instances were randomly sampled from the distribution. More details of the benchmark instances can be found in [22].

In our experiments, we train a heuristic function using GP for each of the 9 scenarios (3 classes, each with 3 scenarios). For each scenario, the 5 instances given in the dataset were used as the test instances. At each generation of the GP process, we randomly sample 20 training instances from the distribution of the scenario. We change the random seed for sampling the training set at each generation to prevent overfitting.

The parameter setting of GP is as follows: the population size is set to 1024. The maximal depth is 8. The crossover, mutation and reproduction rates are 0.8, 0.15 and 0.05 respectively. The best 10 individuals are considered as elitists. The parents are selected by tournament selection of size 7. The number of generations is set to 25. The algorithms were run for 30 times independently on desktops with Intel(R) Core(TM) i7 CPU @3.60GHz.

A. Results and Discussions

Tables II–IV show the test performance (with the format of "mean(std)") of the proposed GP1 and GP2 methods as well as the three manually designed heuristics on the Class 4–6 instances. As mentioned in Algorithm 2, we consider either drive-first or wait-first strategies for deciding the departure time. For each test instance, we conducted Wilcoxon's rank sum test with significance level of 0.05, and highlight the tables as follows:

- For drive-first/wait-first strategy, if the results of GP1/GP2 is significantly better than all the manually designed heuristics, then the results are highlighted in bold;
- For drive-first/wait-first strategy, if the results of one GP approach is significantly better than the other, then the better results are marked with *;
- For each heuristic, if drive-first (wait-first) strategy is better than the other, then the better results are marked with underline.

From the tables, we have the following observations:

- GP1 and GP2 can evolve significantly better heuristics than the manually designed heuristics in most cases. As the degree dynamism increases, such advantage becomes more obvious. For example, when using the drive-first strategy, GP1 obtained significantly better results for 5 Class-4 instances, 6 Class-5 instances and 14 Class-6 instances.
- GP1 and GP2 obtained similar test performance in most cases. The only exceptions are Class-5 rc101-4 with drive-first strategy and rc101-2 with wait-first strategy, Class-6 rc101-3 and rc102-2 with wait-first strategy. GP1 outperformed GP2 in three out of these four exceptions. This implies that simply including the probability information as terminals failed to help the discovery of better heuristics.
- The wait-first strategy is generally significantly better than the drive-first strategy. There are much more underlines in the wait-first side than the drive-first side. This observation is consistent with intuition and those found in other literatures such as [21], [22].
- The UF heuristic is usually worse than the EF and LC heuristics. When the degree of dynamism is not high (Class-4), the EF heuristic performs better than LC more often. However, as the degree of dynamism increases (Classes 5 and 6), the LC heuristic performs better.

B. Further Analysis

Fig. 1 shows the convergence curves of GP1 on Class-6 rc101 using the wait-first strategy. The other instances and other algorithms show similar patterns. From the figure, we have two observations. First, the search almost converged at around generation 15. Therefore, 25 generations is enough to guarantee convergence. Second, the test curve is consistent with the training curve. This implies that the rotation of training instances successfully prevented overfitting. The test performance also converged after generation 15.

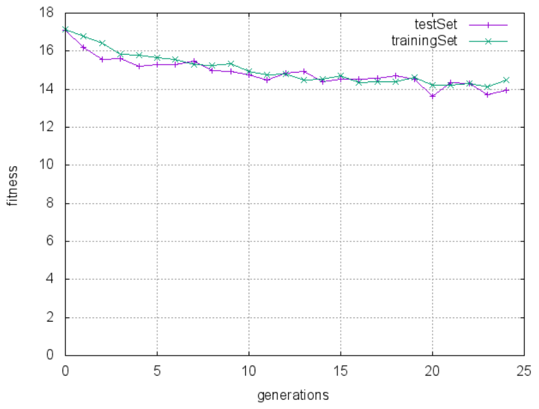

Fig. 1. Convergence curves of GP2 on **Class-6** rc101 using wait-first strategy.

Fig. 2 shows the average frequency of terminals used in the GP2-evolved heuristics over the 30 runs on Class-6 rc104 using wait-first strategy. All the other cases show similar patterns. It can be seen that CT and T are the most important terminals in the heuristic, followed by OT. This makes sense, since CT is equivalent to the UF heuristic, and $\max\{\mathrm{T}, \mathrm{OT}\}$ is essentially equivalent to the EF heuristic. On the contrary, NFR and PNR were rarely used in the heuristics, which means GP failed to effectively use these terminals. A wiser way of incorporating the probability information is needed. Finally, the constant terminal was not often used, indicating that it can be removed without much affecting the performance of GP.

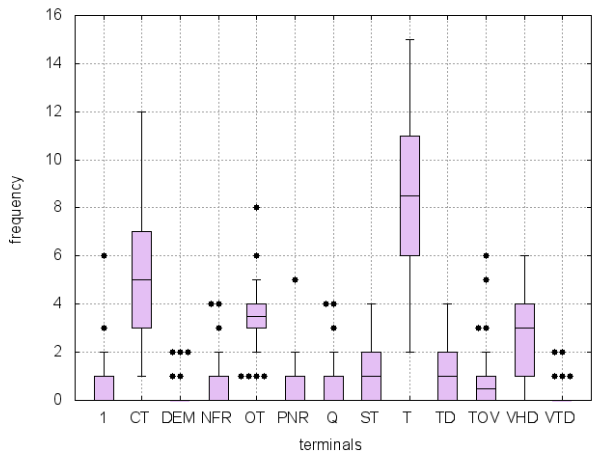

Fig. 2. The average frequency of terminals used in the GP2-evolved heuristics on Class-6 rc104 using wait-first strategy.

V. Conclusions and Future Work

This paper solves the Dynamic Vehicle Routing Problem with Time Windows (DVRPTW), which requires an immediate decision on accepting/rejecting the newly arrived request in real time. We consider solving DVRPTW using a GP-based

TABLE II
THE TEST PERFORMANCE OF THE COMPARED ALGORITHMS ON THE CLASS-4 INSTANCES.

	Drive-first					Wait-first				
Instance	EF	UF	LC	GP1	GP2	EF	UF	LC	GP1	GP2
rc101-1	11.0	13.0	20.0	12.47(4.03)	10.80(4.28)	12.0	8.0	20.0	**5.23(1.45)**	**4.80(1.79)**
rc101-2	14.0	15.0	13.0	12.87(3.77)	12.90(4.25)	10.0	12.0	21.0	**7.70(2.91)**	**7.20(2.35)**
rc101-3	7.0	12.0	17.0	8.80(3.67)	8.53(3.66)	5.0	12.0	11.0	**4.23(1.74)**	**3.70(1.86)**
rc101-4	6.0	15.0	17.0	12.07(4.70)	12.53(3.95)	7.0	8.0	5.0	5.97(1.94)	6.20(1.56)
rc101-5	24.0	22.0	23.0	**14.67(5.01)**	**14.53(4.40)**	14.0	12.0	16.0	**7.47(1.36)**	**7.77(2.05)**
avg	12.4	15.4	18.0	12.17(2.20)	11.86(1.97)	9.6	10.4	14.6	**6.12(1.04)**	**5.93(0.93)**
rc102-1	18.0	19.0	15.0	**8.83(6.34)**	**7.40(4.57)**	7.0	8.0	8.0	**2.57(2.03)**	**1.97(1.38)**
rc102-2	19.0	17.0	7.0	11.33(5.78)	9.77(5.30)	8.0	13.0	11.0	10.50(3.65)	9.90(4.00)
rc102-3	14.0	20.0	11.0	11.87(6.63)	12.13(4.92)	21.0	14.0	16.0	**11.43(4.75)**	**11.23(3.04)**
rc102-4	16.0	19.0	6.0	8.77(2.91)	8.90(2.58)	7.0	12.0	2.0	7.87(3.55)	7.50(3.71)
rc102-5	9.0	14.0	17.0	**5.60(4.38)**	**7.27(4.76)**	19.0	15.0	9.0	**4.57(3.32)**	**4.23(3.49)**
avg	15.2	17.8	11.2	**9.28(2.65)**	**9.09(2.06)**	12.4	12.4	9.2	**7.39(1.36)**	**6.97(1.39)**
rc104-1	14.0	46.0	16.0	**8.60(5.79)**	**8.70(6.28)**	6.0	46.0	8.0	6.80(3.93)	7.90(7.69)
rc104-2	20.0	38.0	7.0	6.67(5.84)	**5.37(3.85)**	17.0	33.0	7.0	9.80(5.73)	8.93(8.12)
rc104-3	33.0	36.0	19.0	**8.00(5.02)**	**6.87(5.28)**	19.0	39.0	34.0	**8.30(5.34)**	**7.40(4.14)**
rc104-4	24.0	45.0	2.0	10.67(6.26)	11.43(6.03)	24.0	44.0	12.0	**8.17(4.23)**	10.13(6.56)
rc104-5	11.0	38.0	23.0	10.20(5.10)	10.73(4.72)	8.0	41.0	2.0	8.70(4.19)	9.10(4.78)
avg	20.4	40.6	13.4	**8.83(2.50)**	**8.62(2.34)**	14.8	40.6	12.6	**8.35(2.14)**	**8.69(2.94)**

TABLE III
THE TEST PERFORMANCE OF THE COMPARED ALGORITHMS ON THE CLASS-5 INSTANCES.

	Drive-first					Wait-first				
Instance	EF	UF	LC	GP1	GP2	EF	UF	LC	GP1	GP2
rc101-1	15.0	15.0	22.0	**13.53(4.67)**	**13.23(5.48)**	13.0	14.0	19.0	**10.67(4.69)**	**10.13(3.59)**
rc101-2	14.0	13.0	16.0	**10.97(5.36)**	**8.90(3.32)**	6.0	8.0	4.0	8.43(3.49)	6.23(2.66)*
rc101-3	9.0	21.0	10.0	10.90(7.48)	12.27(5.34)	8.0	8.0	19.0	**5.60(3.51)**	**6.23(4.17)**
rc101-4	8.0	17.0	19.0	7.80(4.32)*	9.10(3.69)	6.0	10.0	19.0	7.47(3.06)	7.37(3.27)
rc101-5	20.0	19.0	16.0	15.27(5.75)	**13.47(4.61)**	17.0	15.0	20.0	**10.97(3.52)**	**9.83(3.68)**
avg	13.2	17.0	16.6	**11.69(3.23)**	**11.39(2.34)**	10.0	11.0	16.2	**8.63(1.73)**	**7.96(1.50)**
rc102-1	14.0	27.0	6.0	11.50(6.21)	10.83(4.93)	10.0	19.0	2.0	5.97(4.68)	6.00(3.49)
rc102-2	19.0	33.0	30.0	**5.93(3.78)**	**5.13(3.14)**	18.0	15.0	8.0	**3.17(2.96)**	**3.43(2.46)**
rc102-3	8.0	20.0	5.0	6.60(4.85)	7.00(5.14)	5.0	3.0	3.0	3.00(2.12)	2.97(2.28)
rc102-4	7.0	17.0	7.0	**3.90(2.67)**	**3.87(2.29)**	2.0	10.0	7.0	3.30(3.06)	3.27(2.16)
rc102-5	22.0	16.0	14.0	**1.77(2.08)**	**2.10(2.09)**	10.0	14.0	6.0	**4.57(3.40)**	**4.27(2.92)**
avg	14.0	22.6	12.4	**5.94(1.86)**	**5.79(1.95)**	9.0	12.2	5.2	**4.00(1.48)**	**3.99(1.25)**
rc104-1	11.0	36.0	5.0	10.20(4.16)	9.87(4.25)	22.0	32.0	13.0	14.77(3.74)	13.57(4.22)
rc104-2	29.0	42.0	22.0	25.90(7.01)	23.30(6.90)	32.0	39.0	27.0	**19.50(5.53)**	**21.30(8.17)**
rc104-3	11.0	39.0	4.0	7.50(5.73)	8.63(7.82)	19.0	33.0	6.0	**5.07(2.98)**	**5.07(3.42)**
rc104-4	27.0	33.0	12.0	11.10(5.12)	11.80(5.44)	22.0	37.0	13.0	16.57(4.78)	16.90(5.62)
rc104-5	15.0	46.0	16.0	**6.70(6.50)**	**7.03(6.03)**	28.0	32.0	8.0	9.07(7.10)	12.13(8.81)
avg	18.6	39.2	11.8	12.28(2.54)	12.13(2.75)	24.6	34.6	13.4	12.99(1.60)	13.79(3.60)

Hyper-Heuristic (GPHH). To this end, we proposed a meta-algorithm that maintains a set of routes throughout the scheduling horizon, and updates it by heuristic in an attempt to accept new requests. Then we designed three heuristics manually, and developed a GPHH for automatically evolving heuristics. Experimental results show that the GP-evolved heuristics significantly outperformed the manually designed heuristics, and such advantage becomes more obvious as the degree of dynamism increases. This demonstrates the efficacy of GPHH in designing heuristics for DVRPTW. On the other hand, it is shown that simply including the probability information as terminals is not effective for finding good heuristics. In the future, we will investigate wiser ways of using such information, e.g. manually designing new composite terminals based on the raw information.

REFERENCES

[1] B. L. Golden, S. Raghavan, and E. A. Wasil, *The vehicle routing problem: latest advances and new challenges*. Springer Science & Business Media, 2008, vol. 43.

[2] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, part i: Route construction and local search algorithms," *Transportation science*, vol. 39, no. 1, pp. 104–118, 2005.

[3] ——, "Vehicle routing problem with time windows, part ii: Metaheuristics," *Transportation science*, vol. 39, no. 1, pp. 119–139, 2005.

TABLE IV
THE TEST PERFORMANCE OF THE COMPARED ALGORITHMS ON THE CLASS-6 INSTANCES.

	Drive-first					Wait-first				
Instance	EF	UF	LC	GP1	GP2	EF	UF	LC	GP1	GP2
rc101-1	35.0	41.0	26.0	**16.17(6.08)**	**18.53(7.42)**	20.0	15.0	7.0	12.97(3.44)	11.93(2.85)
rc101-2	32.0	35.0	34.0	**12.93(4.35)**	**14.37(5.48)**	20.0	19.0	12.0	**11.63(2.76)**	12.13(3.17)
rc101-3	38.0	39.0	34.0	**18.17(5.50)**	**19.00(7.60)**	19.0	21.0	22.0	**12.27(4.09)***	**14.67(4.68)**
rc101-4	23.0	28.0	28.0	**18.07(4.95)**	**19.70(6.75)**	17.0	17.0	18.0	18.50(2.86)	17.73(2.80)
rc101-5	37.0	39.0	35.0	**15.53(5.18)**	**17.10(4.04)**	26.0	21.0	13.0	14.33(3.39)	14.13(3.37)
avg	33.0	36.4	31.4	**16.17(2.80)**	**17.74(3.74)**	20.4	18.6	14.4	13.94(1.76)	14.12(2.04)
rc102-1	32.0	42.0	32.0	**18.17(6.16)**	**16.00(6.32)**	26.0	28.0	26.0	**16.73(6.07)**	**14.03(6.00)**
rc102-2	41.0	37.0	37.0	**15.70(5.15)**	**15.63(5.81)**	25.0	23.0	10.0	12.87(3.39)*	14.57(3.78)
rc102-3	39.0	38.0	32.0	**14.87(3.65)**	**13.17(3.22)**	12.0	16.0	16.0	**8.00(3.10)**	**7.30(3.63)**
rc102-4	29.0	29.0	23.0	**17.57(6.16)**	**17.67(6.05)**	24.0	17.0	16.0	**14.80(2.92)**	**14.07(3.31)**
rc102-5	45.0	46.0	32.0	**19.07(5.10)**	**17.77(4.52)**	28.0	23.0	20.0	**12.47(4.38)**	**13.13(5.14)**
avg	37.2	38.4	31.2	**17.07(3.26)**	**16.05(3.65)**	23.0	21.4	17.6	**12.97(1.71)**	**12.62(2.09)**
rc104-1	46.0	56.0	31.0	**13.27(8.20)**	**13.33(7.62)**	20.0	43.0	24.0	**18.13(4.02)**	**18.10(4.63)**
rc104-2	33.0	56.0	21.0	**18.13(8.62)**	**16.33(6.70)**	18.0	44.0	25.0	**14.67(2.54)**	**15.83(4.71)**
rc104-3	40.0	55.0	43.0	**23.77(7.11)**	**22.93(5.75)**	25.0	50.0	25.0	**18.00(6.15)**	**16.97(3.91)**
rc104-4	29.0	52.0	28.0	**21.13(6.94)**	**20.83(6.20)**	33.0	44.0	28.0	**24.30(3.99)**	**22.87(4.01)**
rc104-5	22.0	51.0	16.0	21.07(5.94)	21.30(5.05)	22.0	38.0	16.0	22.20(5.63)	22.80(4.97)
avg	34.0	54.0	27.8	**19.47(3.93)**	**18.95(2.87)**	23.6	43.8	23.6	**19.46(2.25)**	**19.31(1.99)**

[4] L. M. Gambardella, É. Taillard, and G. Agazzi, "Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows," 1999.

[5] P. K. Nguyen, T. G. Crainic, and M. Toulouse, "A tabu search for time-dependent multi-zone multi-trip vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 231, no. 1, pp. 43–56, 2013.

[6] T. Vidal, T. G. 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.

[7] S. Belhaiza, P. Hansen, and G. Laporte, "A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows," *Computers & Operations Research*, vol. 52, pp. 269–281, 2014.

[8] 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.

[9] C. Prins, "A simple and effective evolutionary algorithm for the vehicle routing problem," *Computers & Operations Research*, vol. 31, no. 12, pp. 1985–2002, 2004.

[10] J. Kytöjoki, T. Nuortio, O. Bräysy, and M. Gendreau, "An efficient variable neighborhood search heuristic for very large scale vehicle routing problems," *Computers & operations research*, vol. 34, no. 9, pp. 2743–2757, 2007.

[11] B. Yu, Z.-Z. Yang, and B. Yao, "An improved ant colony optimization for vehicle routing problem," *European journal of operational research*, vol. 196, no. 1, pp. 171–176, 2009.

[12] G. Laporte, M. Gendreau, J.-Y. Potvin, and F. Semet, "Classical and modern heuristics for the vehicle routing problem," *International transactions in operational research*, vol. 7, no. 4-5, pp. 285–300, 2000.

[13] U. Ritzinger, J. Puchinger, and R. F. Hartl, "A survey on dynamic and stochastic vehicle routing problems," *International Journal of Production Research*, vol. 54, no. 1, pp. 215–231, 2016.

[14] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[15] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[16] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, "Exploring hyper-heuristic methodologies with genetic programming," in *Computational intelligence*. Springer, 2009, pp. 177–201.

[17] 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, 2016.

[18] R. Glanville, D. Griffiths, P. Baron, J. H. Drake, M. Hyde, K. Ibrahim, and E. Ozcan, "A genetic programming hyper-heuristic for the multidimensional knapsack problem," *Kybernetes*, vol. 43, no. 9/10, pp. 1500–1511, 2014.

[19] M. Bader-El-Den, R. Poli, and S. Fatima, "Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework," *Memetic Computing*, vol. 1, no. 3, pp. 205–219, 2009.

[20] T. Weise, A. Devert, and K. Tang, "A developmental solution to (dynamic) capacitated arc routing problems using genetic programming," in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM, 2012, pp. 831–838.

[21] R. Bent and P. Van Hentenryck, "Waiting and relocation strategies in online stochastic vehicle routing." in *IJCAI*, 2007, pp. 1816–1821.

[22] M. Saint-Guillain, Y. Deville, and C. Solnon, "A multistage stochastic programming approach to the dynamic and stochastic vrptw," in *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer, 2015, pp. 357–374.

[23] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati, "Ant colony system for a dynamic vehicle routing problem," *Journal of Combinatorial Optimization*, vol. 10, no. 4, pp. 327–343, 2005.

[24] É. D. Taillard, L. M. Gambardella, M. Gendreau, and J.-Y. Potvin, "Adaptive memory programming: A unified view of metaheuristics," *European Journal of Operational Research*, vol. 135, no. 1, pp. 1–16, 2001.

[25] R. Bent and P. Van Hentenryck, "The value of consensus in online stochastic scheduling." in *ICAPS*, vol. 4, 2004, pp. 219–226.

[26] ——, "Regrets only! online stochastic optimization under time constraints," in *AAAI*, vol. 4, 2004, pp. 501–506.

[27] G. Ghiani, E. Manni, A. Quaranta, and C. Triki, "Anticipatory algorithms for same-day courier dispatching," *Transportation Research Part E: Logistics and Transportation Review*, vol. 45, no. 1, pp. 96–106, 2009.

[28] K. Sim and E. Hart, "A combined generative and selective hyper-heuristic for the vehicle routing problem," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 2016, pp. 1093–1100.