Evolving Ensembles of Routing Policies using Genetic Programming for Uncertain Capacitated Arc Routing Problem

Shaolin Wang, Yi Mei, John Park and Mengjie Zhang
Victoria University of Wellington
Wellington, New Zealand
Email: {wangshao3, yi.mei, john.park, mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—The Uncertain Capacitated Arc Routing Problem (UCARP) has a wide range of real-world applications. Genetic Programming Hyper-heuristic (GPHH) approaches have shown success in solving UCARP to evolve routing policies that generate routes in real time. However, existing GPHH approaches still have a drawback. Despite the effectiveness in many benchmarks, the single routing policy evolved by GPHH is too complex to interpret. On the other hand, the users need to be able to understand the evolved routing policies to feel confident to use them. In this paper, we aim to employ three ensemble methods, BaggingGP, BoostingGP and Cooperative Co-evolution GP (CCGP) to evolve a group of interpretable routing policies. The ensemble can be used to compare with single complex routing policy from GPHH. Experiment studies show that CCGP significantly outperformed BaggingGP and BoostingGP, and can generate much smaller and simpler routing policies to form ensembles with comparable test performance as the routing policy evolved by SimpleGP. This demonstrates the potential of improving the interpretability issue of GPHH using ensemble methods.**

***Keywords*—UCARP; GPHH; ensemble learning; routing policy.**

I. Introduction

The Capacitated Arc Routing Problem (CARP) [1] is an important optimisation problem of serving a set of edges in a graph using a fleet of vehicles with minimum cost. CARP has a wide range of real-world applications such as waste collection [2] and winter gritting [3], [4].

As a NP-hard problem, CARP has received much research interest, and there have been extensive studies for solving it [1], [5]. However, so far most studies have been focused on static environment, where all the problem parameters (e.g. task demand, travel cost) are fixed and known in advance. This assumption is usually not true in reality, where the problem parameters are often stochastic. For example, the amount of waste to be collected on a street varies from one day to another, and can be quite different from expected. To better reflect the reality, the Uncertain Capacitated Arc Routing Problem (UCARP) was proposed [6].

Traditional solution optimisation approaches such as genetic algorithms and ant colony optimisation generally show poor performance in UCARP. The stochastic nature of UCARP can result in situations where the actual demand of the tasks can be greater than expected, leading to infeasible routes which violate the capacity constraint. Traditional optimisation generally requires a long computational time to adjust the preplanned solution to account for the failure. Instead, heuristic approaches such as *routing policies* [7] are better able to handle UCARP than traditional optimisation techniques. A routing policy [7] helps the vehicle determine the next edge to serve in real time once it becomes idle. The effectiveness of a routing policy largely depends on the scenario, objective(s), and graph topology [8]. It is very time-consuming to design effective routing policy for a given problem scenario manually. To overcome this drawback, Genetic Programming Hyper-heuristic (GPHH) approaches have been applied to UCARP to evolve routing policies automatically [9], [10], [11].

However, a major limitation of existing GPHH approaches is that they use large tree depth (e.g. depth of 8 [11]), which generally results in too complex routing policies. These large and complicated rules are often too difficult for a human to interpret effectively. In a real-world scenario, the users need to be able to understand the evolved routing policies to feel confident to use them. In addition, if we can interpret the evolved routing policies, we can determine what contribution each feature makes in the decision-making process, and understand the inner mechanism of each routing policy so that we can reuse the knowledge to other cases. Evolving effective, small and interpretable heuristics has not yet been investigated for UCARP.

The overall research goal of this paper is to evolve effective and interpretable ensemble of routing policies for UCARP. For this purpose, we first adapt three existing ensemble GPHH approaches that have been applied to other combinatorial optimisation problems [12], [13] to handle the UCARP. The ensemble GPHH approaches have shown effectiveness over GPHH approaches that evolve single rules, but they have not been used to evolve small rules. Therefore, our aim is to evolve ensembles with small and interpretable rules that are competitive with large and complex rules, and to identify the behaviours of the rules from the ensembles. The research goal is broken down into the following objectives:

1) Investigate the effectiveness of three ensemble GPHHs for UCARP: BaggingGP [12], BoostingGP [12], and Cooperative Co-evolution GP (CCGP) [13].

2) Evaluate the evolved ensembles against a standard GPHH that evolves single rules, testing multiple levels of depth (and complexity) of evolved rules.
3) Analyse the structures and the behaviors of the evolved ensemble members.

The rest of this paper is organised as follows. Section II introduces the background. Section III describes the ensemble approaches extended for UCARP. Section IV gives the experimental studies. Finally, Section V gives the conclusions and possible future directions.

II. Background

This section briefly describes some background of the UCARP, previous approaches and the hyper-heuristic approaches that have been applied to UCARP.

A. Uncertain Capacitated Arc Routing Problem

A UCARP instance can be described as follows: consider a graph $G(V, E)$, where V is the set of vertices, E is the set of edges. Each edge $e \in E$ has a positive random deadheading cost $dc(e)$, indicating the cost of traversing the edge. A set of edge *tasks* $E_R \subseteq E$ are required to be served by the vehicles. Each task $e_R \in E_R$ has positive random demand $d(e_R)$ which represents the demand to serve, and a positive serving cost $sc(e_R)$. A set of vehicles with capacity Q are located at the depot $v_0 \in V$. The goal is to serve all the tasks with the least total cost. An edge can be traversed multiple times to meet the demands of the tasks. A vehicle must start at v_0 and finish at v_0, and the total served demand for each route cannot exceed the capacity of the vehicle.

In a *sample* of a UCARP instance, the demand and deadheading costs are defined in advance, i.e., have a realised value. However, although the *expected* demand $E[d(e_R)]$ of a task e_R is known to the routing policy in advance, the *actual* demand $d(e_R)$ of the task is unknown to the routing policy until the vehicle finishes serving the task at edge e. Likewise, the value of deadheading cost $dc(e)$ is unknown to the routing policy until the vehicle finishes traversing over the edge e.

Due to the uncertain environment, routes can have failures during execution. Specifically, the vehicle may be able to meet the actual demand of a task while it traverses the edge, where the demand of the tasks from the edges traversed by the vehicle is greater than the capacity of the vehicle. In this case, a *route failure* occurs, and the route has to be repaired. A typical approach to handling a route failure is to return the vehicle back to the depot, and re-traverse the edge to meet the remaining demand required by the task. Fig. 1 shows an example of a *route failure*. Red lines refer to edges that required to be served. Dotted lines refer to shortest path between two vertices. In this case, the actual demand of A exceed the expected demand, and a *route failure* occurs. The vehicle returns the depot through the shortest path (through Y in this case) and refill. Then the vehicle goes back and finish serving A.

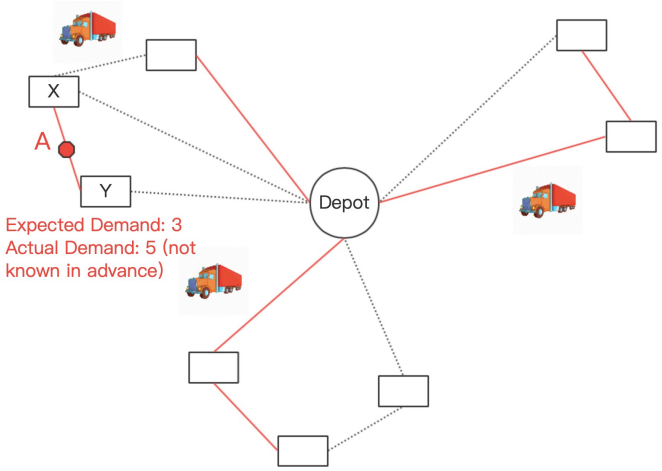


Fig. 1: An example of a *route failure*.

B. Related Work

The CARP has received extensive research interest. Golden and Wong [14] developed an integer linear programming model and solve it using branch-and-cut. However, this approach can only solve small instances. Tabu search [15], [16], [17] approach was proposed for static CARP for improvement. After that, Genetic Algorithm [18] and Memetic Algorithms [19], [20], [21] were proposed to solve the problem better. Lacomme et al. [22] proposed an ant colony schema, and Doerner et al. [23] developed an ant colony optimization. However, these studies are not directly applicable to UCARP, since they are not able to deal with the route failure effectively.

To solve UCARP, both proactive and reactive approaches have been proposed. Proactive approaches [24], [25] aim to find robust solutions that are expected to handle all the possible realisations of the random variables. On the other hand, reactive approaches mainly use GPHH to evolve routing policies [9], [10], [11] to generate solution on-the-fly. In this paper, we focus on the GPHH approaches.

Effective GPHH approaches have been applied to UCARPs in the literature. Liu et al. [9] proposed a GPHH approach to UCARP by designing a novel and effective meta-algorithm that incorporate stochastic information from the problem. The proposed meta-algorithm improves the learning efficiency of GPHH by employing domain knowledge and filtering redundant candidate tasks, and lead to improved performance than an existing GPHH [26]. Mei et al. [11] extended the decision making process from single-vehicle to multiple-vehicle version, so that the solution can be generated with multiple vehicles on the road simultaneously. MacLachlan et al. [10] further improved the GPHH by proposing a novel task filtering method and an effective look-ahead terminal. Liu et al. [27] proposed a proactive-reactive GPHH to co-evolve a robust solution and a route recourse strategy simultaneously.

Ensemble learning is widely used in classification problems [28], [29], [30] to improve performance. Ensemble learning has been combined with various algorithms, such as decision

tree [31], active example selection algorithm [28], neural network [29], SVM and Naive Bayes algorithm [30], and GPHH [13], [12]. The nature of Ensemble approach, using a group of weak learners to form up a stronger learner [32], has the potential to overcome the drawback of GPHH.

III. Ensemble GP Hyper-Heuristics for UCARP

In this section, we describe the three Ensemble GP (EGP) methods for UCARP, namely BaggingGP, BoostingGP and Cooperative Co-evolution GP (CCGP). BaggingGP and BoostingGP employ the idea of bagging and boosting in traditional ensemble learning [33], and evolves the routing policies in the ensemble in a sequential manner. CCGP, on the other hand, co-evolves the routing policies in the ensemble simultaneously. These ensemble approaches were proposed to create ensembles for job shop scheduling problems [13], [12], and showed good performance. In this paper, the idea of BaggingGP and BoostingGP is similar with Durasevic and Jakobovic's [12] idea, which were applied to a problem outside of CARP. The idea of CCGP is similar with the EGP-JSS [13].

A. GP Representation and Evaluation

During the GP process, a GP tree represents a routing policy, which is essentially an arithmetic priority function. For example, the routing policy, "CFH−CTD", can be represented as a GP tree that the root node is the function "-", the left child node is the terminal "CFH" and the right child node is the terminal "CTD". The routing policy is applied in a solution generation process, which is modeled as a decision making process. During the process, the vehicles serve a task at a time. Once a vehicle completes serving the current task and becomes idle, the routing policy is called to calculate the priority of the unserved candidate tasks. Then, the candidate task with the best priority is selected to be served next. The process completes when all the tasks have been served, and the routes of the vehicles are returned as the solution generated by the routing policy. Fig. 2 shows the simulation process for generating solutions. Given an instance sample and a routing policy, the simulation process generates a corresponding solution.

To evolve the routing policies, terminal set and function set commonly used by GPHH approaches to UCARP [11] is shown in Table I. The function set consists of the operators $+, -, \times$, protected division $/$ which returns 1 if divided by 0, $\max$, and $\min$. The $\max$ and $\min$ operators take two child nodes, and return the maximum and minimum value between them, respectively. As an example, a routing policy "CFH − CTD" calculates the priorities of unserved candidate task by taking the difference of the deadheading cost of the shortest path between the current location and the candidate task, and the deadheading cost of the shortest path between the candidate task and the depot. Therefore, the rule prioritises candidate tasks that are closest to the current location and farthest to the depot. To evaluate the fitness of a routing policy or an ensemble x, a set of training UCARP instance samples T_{train} is used. Specifically, x is applied to each of the training instance samples and generate a corresponding solution. For each training instance sample $Sample_t \in T_{train}$, the total cost of the solution obtained by x is $tc(x, Sample_t)$. Then, the

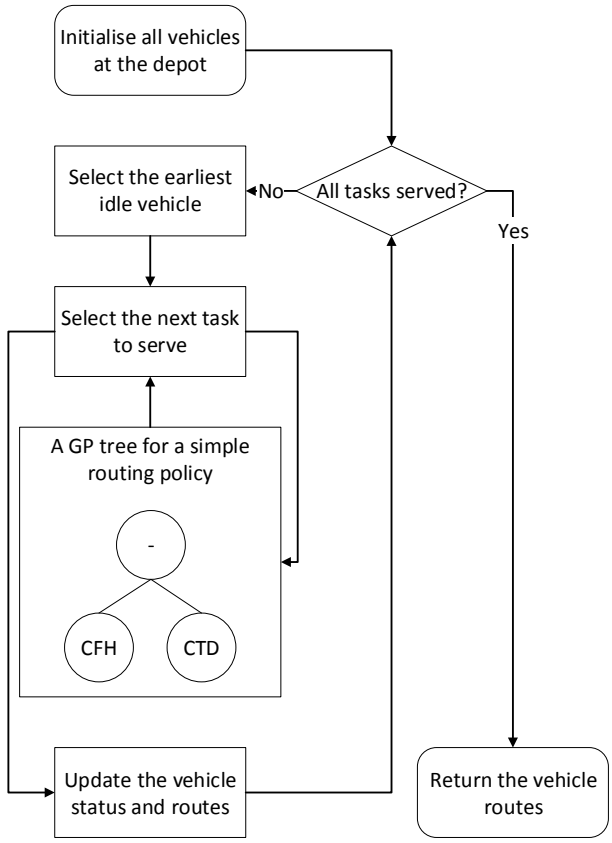


Fig. 2: The simulation for generating a solution by a routing policy.

TABLE I: The terminal set used to evolve the routing policies.

Terminal	Description
SC	the serving cost of the candidate task e_R
CFH	the deadheading cost of the shortest path from the current location v to the candidate task e_R
CTD	the deadheading cost of the shortest path from the candidate task e_R to the depot v_0
CR	the deadheading cost of the shortest path from the current location v to the depot v_0
DEM	the expected demand of the candidate task e_R
DEM1	the demand of closest unserved task e'_R to the candidate task e_R
RQ	the remaining capacity of the vehicle
FULL	the fullness (served demand over capacity) of the vehicle
FRT	the fraction of unserved tasks
FUT	the fraction of unassigned tasks
CFR1	the deadheading cost of the closest feasible alternative route (i.e. the second shortest path) to the candidate task e_R
RQ1	the remaining capacity of the closest other route to the candidate task e_R
CTT1	the deadheading cost from the candidate task e_R to its closest remaining task e'_R

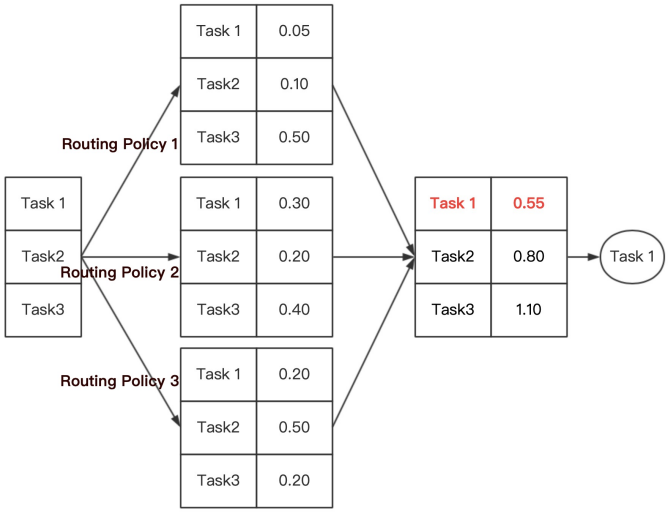


Fig. 3: A simple task selection process.

fitness of x is defined as the average total cost of the solution obtained by x over the set of training instance samples, i.e.

$$fit(x) = \frac{1}{|T_{train}|} \sum_{Sample_t \in T_{train}} tc(x, Sample_t), \quad (1)$$

where $|T_{train}|$ is the number of training instance samples.

While generating solutions, the routing policy or ensemble is applied to select the next task for vehicles to serve next. A single policy simply selects the one with the best priority. An ensemble, on the other hand, needs a combination scheme. Here, we use the simple aggregation combination. That is, the final priority of a task is calculated as the sum of the priority given by the routing policies in the ensemble. A simple ensemble task selection process example can be found in Fig. 3.

B. BaggingGP

The basic idea of BaggingGP is to evolve a set of routing policies with different biases using different training subsets [12]. Then, the evolved routing policies are combined to form an ensemble. Durasevic and Jakobovic's [12] bagging approach to job shop scheduling evolves the ensemble from multiple separate GP runs, where each run outputs a single GP rule. Each ensemble element is evolved using a different training subset randomly sampled from the training set. The individual with the best fitness in the population will be added to the final ensemble after each run. When applying the trained ensemble $\mathcal{RP}$ to an unseen test instance sample, the ensemble makes decisions by the aggregating priorities from each routing policies and select the task with the best total priority.

To extend Durasevic and Jakobovic's [12] bagging approach to UCARP and make the fair comparison with SimpleGP which evolve a single routing policy in one single GP run, we make some modifications. The ensemble is formed in a single GP run, which consists of a number of *cycles*. Each routing policy is evolved in a cycle. Given the number of generations G which is same as the number of generations of SimpleGP and the cycle length $cycleLength$, the algorithm will evolve $G/cycleLength$ routing policies to be included in the ensemble.

C. BoostingGP

The basic idea of BoostingGP [12] is that GP evolved rules focus on those instances that were solved poorly in the previous GP runs so that those instances get higher importance in the following GP runs. In this way, each rule in the ensemble can solve different problematic instances. Durasevic and Jakobovic's [12] boosting approach for job shop scheduling evolves the ensemble from multiple separate GP runs, where each run outputs a single GP rule. Each training instance is given a weight, which is initially set equally to one divided by the number of training instances. After each run, the weights of the instances that are poorly solved are increased, and the weights of instances that the evolved policies show good performance are decreased based on Adaboost algorithm [12]. This means that poorly solved instances are more important in the subsequent cycle.

To extend Durasevic and Jakobovic's [12] boosting approach to UCARP and make the fair comparison with SimpleGP which evolve a single routing policy in one single GP run, we make some modifications. The GP run with G generations is split into cycles of length $cycleLength$, resulting in an ensemble that consists of $G/cycleLength$ routing policies. Each cycle represents the independent GP runs in Durasevic and Jakobovic's original BoostingGP, and the weight updates are carried out after each cycle. In addition, we modified the fitness function of a routing policy or ensemble x as the weighted sum of the total cost of the solutions obtained by it over the training set, since fitness function shown in Eq. (1) is not satisfied any more for BootingGP. The new fitness function is shown below as

$$fit(x) = \sum_{t=1}^{N_{train}} w(Sample_t) \cdot tc(x, Sample_t). \quad (2)$$

where $w(Sample_t)$ is the weight of the training instance $Sample_t$, and N_{train} is the number of training samples.

D. Cooperative Co-evolution GP

The basic idea of the Cooperative Coevolution (CC) approach is to divide the original problem into several subproblems. Each sub-problem is solved by a sub-population of the GP population. Finally, the best individuals from each subpopulation are combined to form a complete solution to solve the original problem. The CCGP has been successfully used for evolving an ensemble of dispatching rules for dynamic job shop scheduling [13]. In this paper, we adapt Park's [13] approach to UCARP, which is described in Algorithm 1. The CCGP approach partitions the population into several smaller sub-populations. Each sub-population has the same number of individuals. An ensemble is composed of representatives from the sub-populations. To evaluate the fitness of an individual

Algorithm 1: The CCGP approach.

Input: number of generation G, evolutionary method $evolve$, ensemble size n
Output: An ensemble of routing polices $\mathcal{RP}$

initialize n sub-populations $pop_1, \ldots, pop_n$;
for $i = 1 \rightarrow n$ **do**
  randomly select a representative $rp_i^* \in pop_i$;
end
$\mathcal{RP} = \{rp_1^*, rp_2^*, \ldots, rp_n^*\}$;
$g = 0$;
while $g < G$ **do**
  for $i = 1 \rightarrow n$ **do**
    for *each routing policy* $rp_i \in pop_i$ **do**
      Form an ensemble $\mathcal{RP}' = \{rp_1^*, \ldots, rp_{i-1}^*, rp_i, rp_{i+1}^*, \ldots, rp_n^*\}$;
      for *each training sample* $Sample_t \in T_{train}$ **do**
        while *tasks in queue is not empty* **do**
          if *vehicle is ready* **then**
            select which task to serve using $\mathcal{RP}'$;
          end
        end
        get the total cost of S using $\mathcal{RP}'$;
      end
      calculate $fit(rp_i) = fit(\mathcal{RP}')$ using Eq. (2);
    end
    update rp_i^* in $\mathcal{RP}$ if $fit(rp_i)$ is better than $fit(rp_i^*)$;
  end
  $evolve(pop)$;
  $g = g + 1$;
end
return the ensemble $\mathcal{RP}$;

in a sub-population, CCGP first replaces the corresponding representatives in the ensemble with the individual being evaluated. The fitness of the individual is the fitness of the resultant ensemble.

Initially, the representative of each sub-population is randomly selected. Then, in each generation, each representative is replaced by the best individual in the corresponding sub-population if the fitness of the ensemble is improved. Finally, the ensemble with the best fitness will be returned.

IV. Experimental Studies

To evaluate the performance of the proposed BaggingGP, BoostingGP and CCGP, we conduct experiments on a number of UCARP instances and compare with the SimpleGP [11], which evolves a single complex routing policy.

A. Experiment Setup

We selected 8 representative UCARP instances from the widely used Ugdb and Uval datasets [11], [9], where the problem size ranges from small (22 tasks and 5 vehicles) to large (97 tasks and 10 vehicles). This way, we can examine the effectiveness of the proposed methods in different problem scenarios.

In the experiments, for each UCARP instance and each algorithm, a routing policy or an ensemble is first trained in the training phase. Then, the trained policy or ensemble is tested on a test set, which contains 500 unseen test instance samples.

Due to the different nature of the algorithms, we use different settings for the training set, so that all the algorithms have the same number of sample evaluations during the training phase. The setting of the training set is given in Table II.

TABLE II: The setting of the training set of the compared algorithms.

Algorithm	Training set setting
SimpleGP	5 training samples, re-sampled at each generation
BaggingGP	5 training samples, re-sampled at each cycle
BoostingGP	5 training samples, reweight at each cycle
CCGP	5 training samples, re-sampled at each generation

In the experiments, the terminal set of all the compared algorithms is shown in Table I, and the population size of SimpleGP, BaggingGP and BoostingGP are set to 1000. For CCGP, there are 5 sub-populations, each with a size of 200. The number of generations is set to 100. For BaggingGP and BoostingGP, the cycle length is set to 20, so that all the EGP methods have an ensemble size of 5. All the methods use ramp-half-and-half initialisation and size-7 tournament selection. The crossover, mutation and reproduction rates are 80%, 15% and 5%, respectively. To evolve simple and interpretable routing policies in the ensemble, we use the most straightforward way that restricts the maximal depth of the EGP methods. We intentionally set the maximal depth for the EGP methods to 4. In addition, we investigate two maximal depth settings for SimpleGP: 4 to match the maximal depth used by the EGPs investigated, and 8 to match the maximal depth setting used by existing GPHH for UCARP [11].

B. Results and Discussions

Table III shows the mean and standard error of the test performance of the compared algorithms. We also conduct Wilcoxon rank sum test with significance level of 0.05 to compare each EGP method with SimpleGP-4 (SimpleGP with max depth 4) and SimpleGP-8 (SimpleGP with max depth 8).

From Table III, one can see that BaggingGP and BoostingGP performed significantly worse than SimpleGP-4 and SimpleGP-8 for all the test instances. This is likely due to the low number of generations to evolve the rules in each cycle for BaggingGP and BoostingGP. CCGP, on the other hand, performed much better than BaggingGP and BoostingGP. It performed significantly better than SimpleGP-8 on Ugdb8, and statistically comparable with SimpleGP-8 on Ugdb2 and Uval9A. For the other instances, although CCGP performed statistically significantly worse, the results of CCGP were much closer to that of SimpleGP than BaggingGP and

TABLE III: The mean and standard error of the test performance of the compared algorithms. For each EGP method, (+), (-) and (=) indicates it is significantly lower (better), higher (worse) and comparable with SimpleGP.

Instance	SimpleGP-4	SimpleGP-8	BaggingGP	BoostingGP	CCGP
Ugdb1	367.8(17.6)	354.8(15.0)	397.1(62.9)(-)(-)	399.4(56.5)(-)(-)	364.9(13.9)(=)(-)
Ugdb2	386.1(10.4)	377.1(26.6)	424.9(34.8)(-)(-)	433.0(32.0)(-)(-)	372.3(9.7) (+)(=)
Ugdb8	485.0(38.7)	499.8(35.5)	548.7(26.0)(-)(-)	568.9(39.4)(-)(-)	467.7(33.7)(+)(+)
Ugdb23	255.5(3.1)	252.1(3.3)	266.4(5.1) (-)(-)	268.8(7.5) (-)(-)	256.0(4.2) (=)(-)
Uval9A	348.0(8.6)	340.3(13.3)	374.0(18.3)(-)(-)	375.9(16.5)(-)(-)	341.3(13.6)(+)(=)
Uval9D	501.1(27.3)	478.3(18.7)	561.4(33.1)(-)(-)	542.2(22.7)(-)(-)	490.4(30.5)(+)(-)
Uval10A	444.4(6.3)	440.6(5.6)	475.2(24.8)(-)(-)	471.9(13.2)(-)(-)	445.2(9.7) (=)(-)
Uval10D	641.7(24.6)	630.2(62.7)	693.9(29.6)(-)(-)	685.2(26.0)(-)(-)	649.1(16.3)(=)(-)

TABLE IV: The mean and standard deviation of the number of nodes in the routing policies or routing polices in the ensembles evolved by the compared algorithms.

Instance	SimpleGP-4	SimpleGP-8	BaggingGP	BoostingGP	CCGP
Ugdb1	11.9(3.1)	75.7(19.3)	12.2(2.3)	12.9(2.0)	5.2(3.6)
Ugdb2	11.9(2.6)	68.3(16.3)	12.8(2.1)	12.2(2.9)	6.7(3.7)
Ugdb8	12.8(2.4)	75.7(19.3)	12.2(2.3)	13.1(1.9)	6.9(3.7)
Ugdb23	12.4(2.3)	68.3(16.9)	12.7(2.0)	12.7(2.1)	5.6(3.5)
Uval9A	13.1(2.0)	65.3(21.6)	12.2(2.3)	12.9(2.0)	7.6(3.6)
Uval9D	13.5(1.8)	58.1(17.7)	11.8(2.4)	12.1(2.3)	7.4(4.0)
Uval10A	11.8(2.4)	58.1(17.7)	11.8(2.4)	12.1(2.3)	6.9(3.7)
Uval10D	13.1(2.0)	65.7(13.3)	12.9(1.9)	12.6(2.1)	6.7(3.2)

BoostingGP. In addition, it can be seen that CCGP clearly outperformed the SimpleGP-4. Overall, there is still a tradeoff in the depth and the complexity of evolving routing policies, but the impact on the performance is minimised by using CCGP to combine the weak routing policies to an ensemble. This demonstrates that EGP is more effective than SimpleGP when both of them produce interpretable routing policies.

Table IV shows the mean and standard deviation of the number of nodes in the routing policies or routing policies in the ensembles evolved by the compared algorithms. From the table, it is obvious that the EGP approaches obtained much smaller (shorter) routing policies than SimpleGP-8. This is mainly because the maximal tree depth of the EGP approaches was set to 4, while that of SimpleGP-8 was set to 8. Therefore, the EGP approaches generally evolve shorter and simpler routing policies than SimpleGP. Although they may be weaker, we aim to form a much stronger ensemble by combining these weak routing policies. Furthermore, even with the same maximal depth, the sizes of the routing policies evolved by CCGP are significantly smaller than those evolved by BaggingGP and BoostingGP. This shows that evolving the ensemble as a whole tends to lead to shorter and simpler routing policies in the ensemble than bagging and boosting.

Fig. 4 shows the **test** performance convergence curves of the routing policies evolved by SimpleGP and the ensembles evolved by CCGP on three representative instances. Since BaggingGP and BoostingGP evolve the routing policies sequentially, they obtain the final ensemble only at the end of the training process and do not have a curve for the test performance of the ensemble. Therefore, BaggingGP and BoostingGP are not included in the figure.

From Fig. 4, one can see that for Ugdb8, the ensemble evolved by CCGP always performed better than the routing policy evolved by SimpleGP. For val9D, on the other hand, the curve of CCGP is always above that of SimpleGP. For val9A, the two curves are almost the same, especially in the final stage. This shows the relative performance of the algorithms is consistent throughout the GP process.

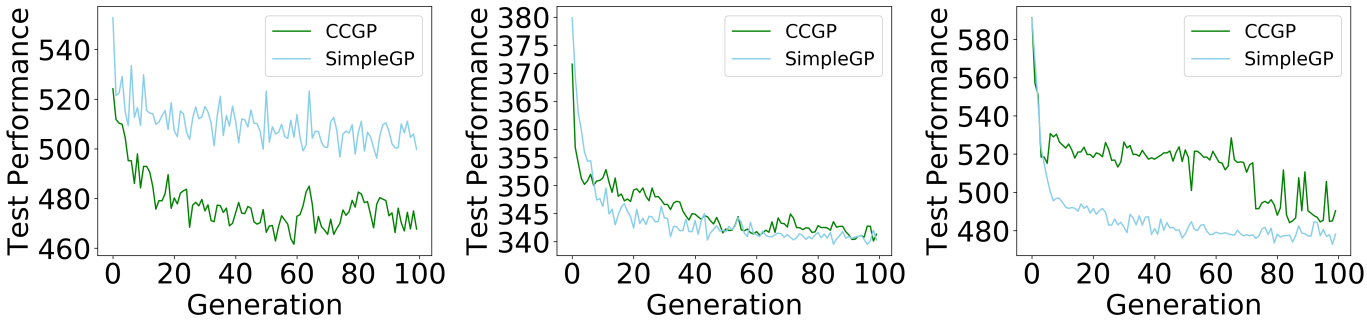
Fig. 5 shows the **training** performance convergence curve of SimpleGP and CCGP on the same instances. Again, BaggingGP and BoostingGP were excluded since they do not store intermediate ensembles. From the figure, it is obvious that the training performance of CCGP is worse than that of SimpleGP on all the instances. This shows that the learning ability of CCGP is still weaker than SimpleGP. Note that the routing policies in CCGP are naturally weaker than those in SimpleGP due to the smaller tree depth. To form a much stronger ensemble from the weak routing policies, an important requirement is the diversity of the routing policies, i.e. they compliment each other. The training performance of CCGP implies that the routing policies in the ensemble may not be complimentary to each other sufficiently, either due to the small ensemble size, or the way to co-evolve the routing policies in the ensemble. Nevertheless, Figs. 4a and 5a shows that CCGP leads to a better generalisation than than SimpleGP on the Ugdb8 instance.

C. Further Analysis

To gain further understanding of the behaviours of the routing policies in the ensemble, we selected an effective ensemble for Ugdb8, and show its routing policies as follows:

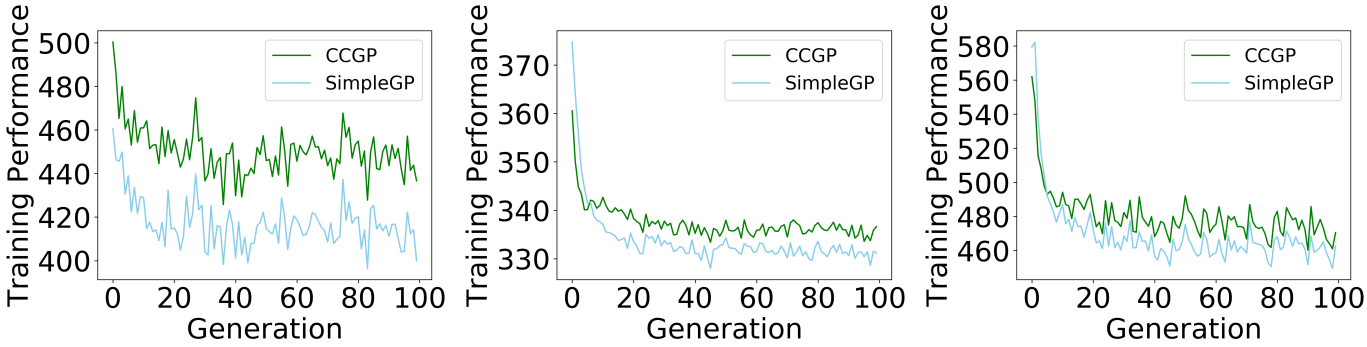
$$
\begin{aligned}
rp_1 &= (\text{DEM1} - \text{FULL}) \times \text{CTT1} \times \text{CTT1},\\
rp_2 &= \text{SC} \times \text{CR} \times \text{CFH} + \text{SC} \times \text{RQ} \times \text{FRT},\\
rp_3 &= \min\{\text{CFR1}, \text{DEM}\},\\
rp_4 &= \min\{\text{CFR1}, \text{DEM1}\},\\
rp_5 &= \text{CTD} + \text{CFH} + \text{DEM} + \text{DEM}.
\end{aligned}
$$

The above ensemble is applied to a Ugdb8 sample, and Table V shows the behavior of the routing policies in some decision situations, which start from 92 candidate tasks to 56 candidate tasks. In the table, each column indicates a decision situation. The first row is the number of candidate tasks ("pool size") of the decision situation, and each row below is the rank of the task selected by the ensemble calculated by each routing



(a) Ugdb8 (b) Uval9A (c) Uval9D

Fig. 4: The **test** performance curves of SimpleGP and CCGP on the representative instances.



(a) Ugdb8 (b) Uval9A (c) Uval9D

Fig. 5: The **training** performance curves of SimpleGP and CCGP on the representative instances.

policy (the lower the better). For example, in the first decision situation, there are 92 candidate tasks, and the ranks of the selected task is ranked 0th by both rp_1 and rp_2, 7th by rp_3 and rp_4, and 9th by rp_5. From the table, rp_1 always assigns the highest priorities to the same edges selected by the ensemble, i.e., rp_1's decision making process is most similar to the ensemble's decisions than the other ensemble members. This is then followed by rp_2.

Further analysis into the behaviours of the rules show interesting properties. For example, many of the tasks in the problem instances are directly adjacent to an unserved tasks, meaning that the deadheading cost from the candidate task to the nearest task is generally 0. This results in rp_1 assigning zero priorities to the candidate tasks, since CTT1 $= 0$. This indicates that rp_1 favours tasks that have at least one edge in between the candidate task and other unserved tasks, as this results in the task being assigned a non-zero priority value. rp_2 has a very important feature CFH with a large coefficient SC $\times$ CR. In other words, rp_2 tends to select the tasks with a short distance from the current location (CFH) and small serving cost (SC). rp_3 and rp_4 disagree with the ensemble decisions most of the time, which is consistent with their uninterpretable structures. rp_5 also prefers nearest neighbours (CFH) with less demand (DEM), as well as the ones close to the depot (CTD). This is consistent with Table V, which shows that the task selected by the ensemble has a better rank in rp_5 in the later decisions situations (pool size ≤ 70) where the routes are more full and tend to return to the depot. The above ensemble shows that each routing policy in the ensemble is short, and each of them is more interpretable than those long and complex routing policies.

Overall, we can see that the CCGP can evolve ensembles that are competitive single rules with significantly more depth while being presented in piecewise format that makes interpretation of the rules easier for the policy maker. Although understanding the interactions between the ensemble members can result in added complexity, the individual rules that contribute to the ensemble are much easier to interpret than standard rules evolved by GP. Besides that, these routing policies play different roles at different stages of the decision process. However, there are some redundant policies (e.g. rp_3 and rp_4 in the example) which limits the effectiveness of the ensemble. There is potential to further improve the performance by further enhancing the collaborations between the routing policies.

V. Conclusions

This paper aims to improve the interpretability of the routing policies evolved by GPHH for UCARP. To this end, we em-

TABLE V: Behaviour of the routing policies in a decision making process.

Pool size	92	90	88	86	84	82	80	78	76	74	72	70	68	66	64	62	60	58	56
Rank in rp1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rank in rp2	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Rank in rp3	7	24	30	61	0	77	41	31	0	31	30	0	0	41	39	52	0	13	14
Rank in rp4	7	16	34	22	0	56	39	27	0	17	24	0	0	40	53	53	0	18	14
Rank in rp5	9	17	42	42	1	77	53	54	66	67	66	1	2	5	7	6	7	0	9

ployed ensemble methods with a smaller tree depth to evolve a group of shorter and more interpretable routing policies which is the first attempt of applying ensemble learning to UCARP to increase the interpretability of routing policies evolved by GPHH. We examined three widely used ensemble methods, namely BaggingGP, BoostingGP and Cooperative Co-evolution GP (CCGP). The experimental results show that although there is still a significant impact on the performance due to the small rule depths, CCGP can perform well against more complex single rules. The routing policies in the ensembles evolved by CCGP also show interpretable structures that are consistent with their behaviors. For future work, we will identify the reason why Bagging GP and Boosting GP achieve bad performance. Apart from that, we will try to develop more effective ensemble approaches for UCARP.

REFERENCES

[1] M. Dror, *Arc routing: theory, solutions and applications*. Springer Science & Business Media, 2012.

[2] S. Amponsah and S. Salhi, "The investigation of a class of capacitated arc routing problems: The collection of garbage in developing countries," *Waste Management*, vol. 24, no. 7, pp. 711–721, 2004.

[3] H. Handa, L. Chapman, and X. Yao, "Robust route optimization for gritting/salting trucks: a cercia experience," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 6–9, 2006.

[4] ——, "Dynamic salting route optimisation using evolutionary computation," in *IEEE Congress on Evolutionary Computation*, 2005, pp. 158–165.

[5] S. Wøhlk, "A decade of capacitated arc routing," in *The vehicle routing problem: latest advances and new challenges*. Springer, 2008, pp. 29–48.

[6] Y. Mei, K. Tang, and X. Yao, "Capacitated arc routing problem in uncertain environments," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[7] 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.

[8] J. Jacobsen-Grocott, Y. Mei, G. Chen, and M. Zhang, "Evolving heuristics for dynamic vehicle routing with time windows using genetic programming," in *IEEE Congress on Evolutionary Computation*. IEEE, 2017, pp. 1948–1955.

[9] Y. Liu, Y. Mei, M. Zhang, and Z. Zhang, "Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 290–297.

[10] J. MacLachlan, Y. Mei, J. Branke, and M. Zhang, "An improved genetic programming hyper-heuristic for the uncertain capacitated arc routing problem," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 432–444.

[11] Y. Mei and M. Zhang, "Genetic programming hyper-heuristic for multi-vehicle uncertain capacitated arc routing problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '18. New York, NY, USA: ACM, 2018, pp. 141–142. [Online]. Available: http://doi.acm.org/10.1145/3205651.3205661

[12] M. Durasević and D. Jakobović, "Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment," *Genetic Programming and Evolvable Machines*, vol. 19, no. 1-2, pp. 53–92, 2018.

[13] J. Park, S. Nguyen, M. Zhang, and M. Johnston, "Evolving ensembles of dispatching rules using genetic programming for job shop scheduling," in *European Conference on Genetic Programming*. Springer, 2015, pp. 92–104.

[14] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," *Networks*, vol. 11, no. 3, pp. 305–315, 1981.

[15] R. W. Eglese and L. Y. Li, "A tabu search based heuristic for arc routing with a capacity constraint and time deadline," in *Meta-Heuristics*. Springer, 1996, pp. 633–649.

[16] A. Hertz, G. Laporte, and M. Mittaz, "A tabu search heuristic for the capacitated arc routing problem," *Operations research*, vol. 48, no. 1, pp. 129–135, 2000.

[17] J. Brandão and R. Eglese, "A deterministic tabu search algorithm for the capacitated arc routing problem," *Computers & Operations Research*, vol. 35, no. 4, pp. 1112–1126, 2008.

[18] P. Lacomme, C. Prins, and W. Ramdane-Chérif, "A genetic algorithm for the capacitated arc routing problem and its extensions," in *Workshops on Applications of Evolutionary Computation*. Springer, 2001, pp. 473–483.

[19] P. Lacomme, C. Prins, and W. Ramdane-Cherif, "Competitive memetic algorithms for arc routing problems," *Annals of Operations Research*, vol. 131, no. 1-4, pp. 159–185, 2004.

[20] K. Tang, Y. Mei, and X. Yao, "Memetic algorithm with extended neighborhood search for capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1151–1166, 2009.

[21] Y. Mei, K. Tang, and X. Yao, "Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 2, pp. 151–165, 2011.

[22] P. Lacomme, C. Prins, and A. Tanguy, "First competitive ant colony scheme for the carp," in *International Workshop on Ant Colony Optimization and Swarm Intelligence*. Springer, 2004, pp. 426–427.

[23] K. F. Doerner, R. F. Hartl, V. Maniezzo, and M. Reimann, "Applying ant colony optimization to the capacitated arc routing problem," in *International Workshop on Ant Colony Optimization and Swarm Intelligence*. Springer, 2004, pp. 420–421.

[24] J. Wang, K. Tang, and X. Yao, "A memetic algorithm for uncertain capacitated arc routing problems," in *Memetic Computing (MC), 2013 IEEE Workshop on*. IEEE, 2013, pp. 72–79.

[25] J. Wang, K. Tang, J. A. Lozano, and X. Yao, "Estimation of the distribution algorithm with a stochastic local search for uncertain capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 96–109, 2016.

[26] 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.

[27] Y. Liu, Y. Mei, M. Zhang, and Z. Zhang, "A predictive-reactive approach with genetic programming and cooperative co-evolution for uncertain capacitated arc routing problem," *Evolutionary Computation*, 2019.

[28] S. Oh, M. S. Lee, and B.-T. Zhang, "Ensemble learning with active example selection for imbalanced biomedical data classification," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 8, no. 2, pp. 316–325, 2011.

[29] L. Yu, S. Wang, and K. K. Lai, "Credit risk assessment with a multistage neural network ensemble learning approach," *Expert systems with applications*, vol. 34, no. 2, pp. 1434–1444, 2008.

[30] L. Shi, X. Ma, L. Xi, Q. Duan, and J. Zhao, "Rough set and ensemble learning based semi-supervised algorithm for text classification," *Expert Systems with Applications*, vol. 38, no. 5, pp. 6300–6306, 2011.

[31] M. Pal, "Random forest classifier for remote sensing classification," *International Journal of Remote Sensing*, vol. 26, no. 1, pp. 217–222, 2005.

[32] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.

[33] R. Polikar, "Ensemble based systems in decision making," *IEEE Circuits and systems magazine*, vol. 6, no. 3, pp. 21–45, 2006.