A Multi-Objective Genetic Programming Hyper-Heuristic Approach to Uncertain Capacitated Arc Routing Problems

Shaolin Wang, Yi Mei, Mengjie Zhang
Victoria University of Wellington,
School of Engineering and Computer Science,
PO Box 600, Wellington, New Zealand
Email: {shaolin.wang, yi.mei, mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—The Uncertain Capacitated Arc Routing Problem (UCARP) is a very important problem which has many real world applications. Genetic Programming Hyper-heuristic (GPHH), which can automatically evolve effective routing policies, is considered as a promising technique that can handle UCARP effectively. However, GP-evolved routing policies are often very complex and hard to be understood and trusted by human users. In this paper, we aim to improve the interpretability of the GP-evolved routing policies by reducing the size of the GP-evolved routing policies since smaller routing policies tend to be easier to understand. We propose a new Multi-Objective GP (MOGP) to optimise the performance (total cost) and size simultaneously. One main challenge is that the size is much easier to be optimised than the performance. Thus, the population tends to be biased to the small but poor routing policies and quickly lose the ability of exploration. To address this issue, we propose a MOGP approach with α dominance strategy (α-MOGP) which can balance the tradeoff between performance and individual size. The experimental results showed that α-MOGP could obtain much smaller routing policies than the state-of-the-art single-objective GPHH, without deteriorating the performance. Compared with traditional MOGP, α-MOGP can obtain a much better and more widespread Pareto front.**

I. Introduction

The Capacitated Arc Routing Problem (CARP) [1] is a traditional combinatorial optimisation problem, which has been proved to be NP-hard [2]. To better reflect the reality, Uncertain Capacitated Arc Routing Problem (UCARP) [3] was proposed. One of the main challenges in UCARP is *Route failures*. A *Route failure* occurs when the actual demand of an edge that needs to be served exceeds the expected demand. The remaining capacity of the vehicle becomes insufficient to serve the edge. The vehicle has to go back to the depot to refill and come back again. This leads to a considerable recourse cost.

Traditional optimisation approaches, such as mathematical programming [1] and evolutionary algorithms [4], are not directly applicable to UCARP since they cannot deal with route failures effectively. Typically, they try to optimise a robust solution beforehand. However, it is hard to optimise a single robust solution that is suitable for all the environments. The pre-planned solution must be re-optimised when route failures occur. Traditional optimisation approaches can hardly adjust the solutions effectively on-the-fly due to their high computational cost.

Routing policy [5] is considered as a promising technique that can handle the uncertain environment in UCARP effectively. Comparing with other solutions optimisation approaches, using a routing policy does not need to optimise a solution beforehand. It can respond to the uncertain environment immediately. A routing policy works as a heuristic that helps a vehicle to determine which task to serve once it becomes idle [5]. It can construct different solutions based on different environments. There are some manually designed routing policies (Path Scanning [4]) based on some existing constructive heuristics.

Many factors can affect the effectiveness of the manually designed routing policy, such as the scenario, the objective(s), and the graph topology [6]. In response to this issue, Genetic Programming Hyper-heuristic (GPHH) approaches have been applied to UCARP to automatically evolve routing policies [7], [8], [9], [10]. However, the routing policies evolved by GPHH are usually too complex to interpret.

The interpretability of the GP-evolved routing policies can be improved by reducing the complexity of the individuals. An intuitive measure for complexity is the size, i.e. the number of nodes in the GP tree since smaller routing policies tend to be easier to interpret. In this paper, we propose a novel multi-objective GPHH, which optimises the performance (total cost) and the individual size of the routing policy simultaneously.

The main challenge for optimising both the performance and the individual size using multi-objective GP is that the size is much easier to be optimised than the performance. It is much easier to generate routing policies with small size than with good performance. As the evolutionary process continues, the population will be filled up with small (but poor) routing policies gradually. However, the evolutionary process requires large routing policies to maintain the ability of exploration. To handle the above issue, Bleuler [11] summarises two basic strategies, using a bias against small individuals and increasing the diversity in the population. Besides that, Bleuler [11] propose a Strength Pareto Evolutionary Algorithm 2 (SPEA2) algorithm to deal with the issue further. However, to our

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

preliminary work, SPEA2 does not work well in UCARP since it is hard to maintain the diversity of the population in a dynamic and stochastic problem. The population will still be taken over by small individuals. α dominance strategy [12] has the potential to handle the above issue of one objective is much easier to optimise than the other since it sets tradeoff rates between objectives in the dominance area.

In this paper, we aim to evolve more compact and interpretable routing policies by using the MOGP approach with α dominance strategy (α-MOGP). We expect that the α-MOGP approach can deal with the issue of losing the ability of exploration while the size is much easier to be optimised than the performance. On the other hand, we expect that it can evolve more compact and interpretable routing policies and achieve comparable performance as the routing policies evolved by the current state-of-the-art GPHH. Specifically, this paper has the following research objectives:

1) Develop a novel α-MOGP to evolve interpretable and effective routing policies for UCARP;
2) Investigate the effects of different adaption schemes for α parameter;
3) Verify the efficacy of α-MOGP by comparing with SimpleGP and other MOGP algorithms.

II. Background

A. Uncertain Capacitated Arc Routing Problem

A UCARP instance can be described as a connected graph $G(V, E)$. There are a set of vertices V and edges E in the graph. A subset of edges $E_R \subseteq E$, denoted as *tasks*, are required to be served by the vehicles. Deadheading cost $dc(e)$, which is a positive random value, indicates the cost of traversing an edge $e \in E$. Each *task*, an edge that needs to be served, has a positive random demand $d(e_R)$ which represents the demand to serve and a positive serving cost $sc(e_R)$ which represents the cost to serve the task. Each vehicle has a capacity Q. The goal is to minimise the total cost of serving all the *tasks*. There is a depot $v_0 \in V$. Each vehicle must start at v_0 and finish at v_0, the edges that the vehicle goes along are denoted as a *route*, and the total demand of a *route* cannot exceed the capacity Q of the vehicle.

There are many random variables in a UCARP instance. Each of the random variables can have different samples. Accordingly, a UCARP instance may contain different samples. In a UCARP instance, each random variable (i.e. task demand and deadheading cost) has a realised value. For example, the actual demand of a task is unknown until the vehicle finishes serving it, and the actual deadheading cost of an edge is unknown until the vehicle finishes traversing over the edge.

Due to the uncertain environment, there are two kinds of failures, *Route failure* and *Edge failure*, that may occur during the serving process. *Route failure* occurs when the remaining capacity of the vehicle is insufficient for the actual demand of a task. A recourse operator can repair it. A typical recourse operator is that the vehicle goes back to the depot to refill and come back to the failed task to complete the remaining service. *Edge failure* occurs when the edge in the route becomes infeasible suddenly. *Edge failure* can be repaired by finding the shortest path (e.g., Dijkstra's algorithm) under the current situation.

$S = (\{S.R^{(1)}, \ldots, S.R^{(m)}\}, \{S.D^{(1)}, \ldots, S.D^{(m)}\})$ can be a representation of a UCARP instance sample solution. Each $S.R$, such as $S.X^{(1)}$, is a route represented as a set of vertex sequences. For example, $S.R^{(k)} = (S.r_1^{(k)}, \ldots, S.r_{L_k}^{(k)})$ represents the k^{th} route. Each $S.D$ is a continuous vector, each number in the vector refers to the workload has been served for this edge. Specifically, $S.D^{(k)} = (S.d_1^{(k)}, \ldots, S.d_{L_k-1}^{(k)})$ ($S.d_i^{(k)} \in [0, 1]$) refers to the fraction of demand that has been served along the route $S.R^{(k)}$. For example, if $S.d_5^{(3)} = 0.5$, then $(S.r_5^{(3)}, S.r_6^{(3)})$ is a task and 50% of its demand is served at position 5 of the route $S.R^{(3)}$.

B. Related Work

Many studies have been made on dealing with CARP, including Integer linear programming model [1], Tabu search [13], [14], [15] approach CARP, Genetic Algorithm [16], Memetic Algorithms [4], [17], [18], ant colony schema [19], ant colony optimization [20]. All these effort make great contributions to CARP. However, they cannot apply to UCARP directly since they are not able to deal with the route failure effectively.

There are two main types of approaches for solving UCARP, the proactive approaches [21], [22], [23] and the reactive approaches [7], [8], [9]. The proactive approaches 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 which construct the solution gradually in real-time.

GPHH approaches have been applied to solve UCARP effectively. Weise et al. [24] first proposed a GPHH for UCARP with a single vehicle and examined its performance. The result showed that routing policies evolved by GPHH could outperform manually designed routing policies. Liu et al. [7] proposed a novel and effective meta-algorithm to filter irrelevant candidate tasks during the decision-making process. To better reflect the reality, the model was extended from single-vehicle to multiple-vehicle version by Mei et al. [9]. Then the solution can be generated with multiple vehicles on the road simultaneously. To further improve the GPHH, MacLachlan et al. [8] proposed a novel task filtering method and an effective look-ahead terminal.

The interpretability of the evolved routing policies is also an important aspect for the routing policies evolved by GPHH. It can guarantee the confidence of the users while using them. However, to our best knowledge, most of the current GPHH approaches cannot achieve satisfactory interpretability. The main reason of why routing policies evolved by GPHH is hard to interpret is that GP tends to generate large trees. Thus, tree size control can be a good approach to reduce the structural complexity of GP so that we can improve the interpretability of routing policies evolved by GPHH. Therefore, it is important to find a good way to control tree size.

There are two commonly used approaches to control tree size. The first one is Tree size or depth limitation [25], [26]. The second one is using a penalty term [27], which penalize those trees with a larger size, in the fitness function. However, both of them cannot handle it perfectly. For the tree size limitation approach, it is not very easy to predefine a reasonable limit for size or depth and maintain the diversity near the root node [28]. For the penalty term approach, it is also challenging to predefine the weights of the penalty term, and a linear weighted sum of two objectives prefers individuals that perform very well in one of the objectives rather than individuals that perform fair well in both objectives when the trade-off surface is concave [29].

Multi-objective optimisation (MO) approach is considered as a more effective approach to control the tree size [28]. The multi-objective optimisation approach not only does not need to predefine any aggregation functions to combine those objective values but also obtain a set of solutions which can satisfy different objectives. Multiple multi-objective optimization approaches [28], [30], [31] can be applied to control tree size. However, there is one issue that the size is much easier to optimise than the performance. This leads to premature convergence to small individuals [29], [32]. It is difficult to deal with the above issue using traditional EMO algorithms. Bleuler [11] summaries two basic strategies, using a bias against small individuals [33], [34] and increase the diversity in the population [35], [29], [36], to deal with the above issue. Besides that, a Pareto-based strategies [28] was applied to GP for a even-parity problem. However, SPEA2 [11] does not work well in UCARP since it is difficult to maintain the diversity of the population in a dynamic and stochastic problem. It cannot prevent small individuals from taking over the entire population. The problem can also be partially addressed by Preference-Based EMO approach [37], [38]. To our knowledge, the basic idea of Preference-Based EMO approach is to find solutions in some specific regions. Thus, we can only get a small portion of the Pareto front in the"preferred region". However, we aim to cover the entire Pareto front as much as possible in this paper.

To address the above issues, we aim to propose a novel Multi-Objective Genetic Programming approach using α-dominance strategy (α-MOGP) which uses performance and size as two objectives. Besides that, α-MOGP is expected to avoid premature convergence to small individuals so that we can have compact and effective routing policies.

III. Multi-Objective Genetic Programming using α-dominance strategy

A. α-dominance strategy

Comparing with traditional single objective GPHH approach, MOGP uses Pareto dominance criteria to evaluate the fitness of individuals. α-dominance strategy [12] is a more general form of traditional dominance. The basic idea of α-dominance is to set tradeoff rates between objectives. For example, we have two solutions, A and B, with two objectives, obj_1 and obj_2. $\mathbf{f}(A) = (a_1, a_2)$ and $\mathbf{f}(B) = (b_1, b_2)$. When using traditional Pareto-dominance, if $a_1 < b_1$ and $a_2 > b_2$, A and B are nondominated. However, A might dominate B when using α-dominance strategy. For a problem to minimize all objectives. If a feasible solution x dominates another solution x' using α-dominance strategy. The problem can be formulated as follows.

$$\min(f_1(o),\ f_2(o),\ f_3(o), ...,\ f_m(o)), \tag{1}$$

$$s_i(x, x') \leq 0,\ \forall i \in \{1, 2, 3, ..., m\},\ \wedge \tag{2}$$

$$s_i(x, x') < 0,\ \exists i \in \{1, 2, 3, ..., m\} \tag{3}$$

where

$$s_i(x, x') = f_i(x) - f_i(x') + \sum_{j \neq i}^{1...m} \alpha_{ij}(f_j(x) - f_j(x')) \tag{4}$$

There is one thing that needs to be noticed that traditional dominance is a special form of α dominance while all α_{ij} equal to 0.

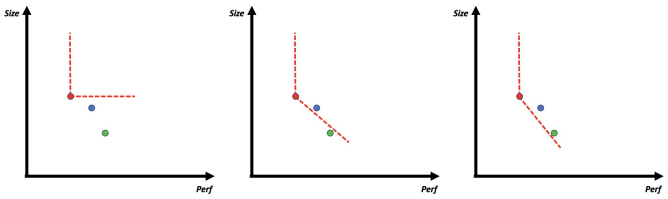
In our problem, we expect that a solution with medium size and good performance will not be dominated by a solution with extremely small size but poor performance. Thus, we modified Eq. 3 as follows.

$$s_{size}(x, x') = f_s(x) - f_s(x') + \alpha_s(f_p(x) - f_p(x')) \tag{5}$$

$$s_{perf}(x, x') = f_p(x) - f_p(x') + \alpha_p(f_s(x) - f_s(x')) \tag{6}$$

where $f_s(x)$ refers to the size of the individual x. $f_p(x)$ refers to the performance of the individual x. α_s equals to α (all the α value mentioned later in the paper) and α_p equals to 0.

Figure 1 shows the change of the dominance area while α value increasing in our problem. We can see that three solutions (red dot, blue dot and green dot) are nondominated while α equals to 0. But, the red dot dominates the blue dot when α equals to 10. While α equals to 100, the green dot is also dominated by the red dot.



(a) α = 0 (b) α = 10 (c) α = 100

Fig. 1: The dominance area with different α value

One of the main problems of the α-dominance strategy is that it is quite challenging to identify a proper α value that can perfectly balance the tradeoff between different objectives during the evolution process. For our problem, while the α value increase, the population will be biased to performance, and the population will be biased to size while the α value decrease. Since the bias to different objectives can change during the evolutionary process, we examine the behaviour of

three different adaptation schemes, linear, sigmoid and cosine, for the α value. Linear scheme will decrease at a constant rate. Sigmoid scheme will decrease slowly in the early stage and decrease rapidly in the middle stage and then back to slowly at the final stage. Cosine scheme will repeat the process of increasing and decreasing. The three adaption scheme curves are shown in Fig. 2. The formulas for all three schemes are

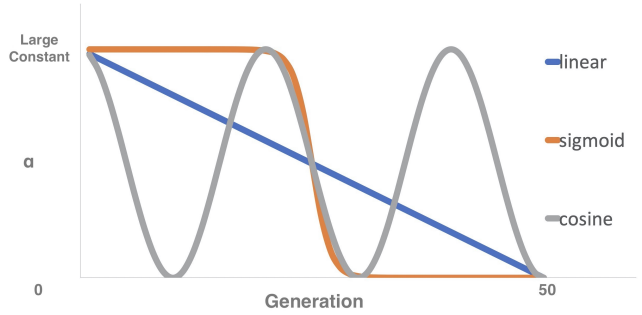


Fig. 2: Three adaption scheme curves.

shown as follow.

$$f_{linear}(x) = \mathrm{C} + \frac{-\mathrm{C} * x}{50} \tag{7}$$

$$f_{sigmoid}(x) = \mathrm{C} * \frac{1}{1 + e^{x-25}} \tag{8}$$

$$f_{cosine}(x) = \frac{\mathrm{C}}{2} * (\cos(\frac{\pi * x}{10}) + 1) \tag{9}$$

where C is a sufficiently large constant which is 99999999 in this paper.

B. Overall Framework

The basic idea of α-MOGP is using α-dominance strategy to replace the traditional Pareto-dominance. The α-dominance strategy is combined with tournament selection as α tournament selection. After evaluating each routing policy, α tournament selection will be applied to select parent individuals from the old population. The parent individuals will be used to breed new population. The, $crossover$, $mutation$ and $reproduction$, operators will be applied in the breeding process. The pseudocode of the α-MOGP approach is shown in Algorithm 1. Given the number of generations G. Each routing policy is evolved using a different training subset randomly sampled from the training set T_{train} (line 4).

The α tournament selection uses the α dominance to select the parents. Specifically, it first randomly selects T individuals, and then select the best one among them in terms of α dominance. If there are multiple non-dominated individuals, the first one identified during the comparison is selected. The pseudocode of the α Tournament Selection method is shown in Algorithm 2.

Algorithm 1: The overall framework of α-MOGP.

Input: Training set $\mathcal{S}_{train}$, number of generations G
Output: A set of non-dominated routing policies $\mathcal{RP}$

```
1  initialise the population pop;
2  g = 0;
3  while g < G do
4      randomly sample a training subset S' ⊆ S_train;
5      evaluate pop using S';
6      while |pop'| < popsize do
7          Breed pop' using parent selection (Algorithm
           2) and genetic operators;
8          g = g + 1;
9      end
10     pop = pop';
11     update the α value;
12 end
13 return the non-dominated routing policies in pop;
```

Algorithm 2: The α tournament selection approach

Input: size of Tournament T, random select method $random\_rp$ which random select a routing policy from population, α-dominance strategy method $\alpha - dominates$ using Eq. (2,3)
Output: best routing policy

```
1  t= 1;
2  best = random_rp;
3  for t < T do
4      rp = random_rp;
5      if rp α − dominates best then
6          best = rp;
7      end
8      t = t + 1;
9  end
10 return best;
```

TABLE I: The terminal set in the experiments.

Terminal	Description
CR	cost from the depot to the current location
CFH	cost from the candidate task to the current location
CFD	cost from the head node of the task to the depot
CFR1	cost from the closest other route to the candidate task
CTT1	the cost from the candidate to its closest remaining task
CTD	cost from the depot to the candidate task
DC	the deadheading cost of the candidate task
DEM	expected demand of the candidate task
DEM1	demand of unserved task that closest to the candidate task
FULL	fullness (served demand over capacity) of the vehicle
FRT	fraction of unserved tasks
FUT	fraction of unassigned tasks
RQ	remaining capacity of the vehicle
RQ1	remaining capacity of the closest other route to the candidate task
SC	cost of serving the candidate task

C. Individual Representation

In α-MOGP, each routing policy is essentially represented as an arithmetic priority function. The priority function is

TABLE II: The mean and standard deviation for HV of the compared algorithms in test process. For each method, (+), (-) and (=) indicates it is significantly higher (better) than, lower (worse) than, and comparable with NSGA-II (the first parentheses) and SPEA2 (the second parentheses).

Instance	NSGA-II [30]	SPEA2 [11]	TS-GPHH[39]	α-MOGP-l	α-MOGP-s	α-MOGP-c
Ugdb1	0.9071(0.0252)	0.8645(0.0408)	0.9378(0.0258)(+)(+)	0.9389(0.0356)(+)(+)	0.9427(0.0263)(+)(+)	0.9423(0.0292)(+)(+)
Ugdb2	0.9153(0.0154)	0.8894(0.0355)	0.9399(0.0199)(+)(+)	0.9395(0.0281)(+)(+)	0.9572(0.0121)(+)(+)	0.9423(0.0175)(+)(+)
Ugdb8	0.9142(0.0228)	0.8625(0.0466)	0.9395(0.0333)(+)(+)	0.9404(0.0302)(+)(+)	0.9505(0.0195)(+)(+)	0.9427(0.0251)(+)(+)
Ugdb23	0.8889(0.0206)	0.8738(0.0259)	0.9303(0.0198)(+)(+)	0.9295(0.0376)(+)(+)	0.9416(0.0186)(+)(+)	0.9341(0.0247)(+)(+)
Uval9A	0.9756(0.0052)	0.9577(0.0178)	0.9838(0.0037)(+)(+)	0.9781(0.0159)(+)(+)	0.9853(0.0115)(+)(+)	0.9811(0.0084)(+)(+)
Uval9D	0.919(0.0174)	0.8528(0.053)	0.9508(0.012)(+)(+)	0.9393(0.0324)(+)(+)	0.9581(0.0121)(+)(+)	0.948(0.0215)(+)(+)
Uval10A	0.9736(0.0067)	0.9534(0.0161)	0.9861(0.0097)(+)(+)	0.9832(0.0112)(+)(+)	0.9905(0.0047)(+)(+)	0.9859(0.0117)(+)(+)
Uval10D	0.9302(0.0238)	0.8986(0.0325)	0.9643(0.0123)(+)(+)	0.9518(0.0411)(+)(+)	0.9724(0.0106)(+)(+)	0.963(0.0138)(+)(+)

TABLE III: The WDL table for the pairwise comparisons between the algorithms in terms of HV.

Approach	NSGA-II [30]	SPEA2 [11]	TS-GPHH[39]	α-MOGP-l	α-MOGP-s	α-MOGP-c
NSGA-II	0-8-0	0-0-8	8-0-0	8-0-0	8-0-0	8-0-0
SPEA2	8-0-0	0-8-0	8-0-0	8-0-0	8-0-0	8-0-0
TS-GPHH	0-0-8	0-0-8	0-8-0	0-8-0	6-2-0	0-8-0
α-MOGP-l	0-0-8	0-0-8	0-8-0	0-8-0	5-3-0	0-8-0
α-MOGP-s	0-0-8	0-0-8	0-2-6	0-3-5	0-8-0	0-5-3
α-MOGP-c	0-0-8	0-0-8	0-8-0	0-8-0	3-5-0	0-8-0

a combination of the state features, such as the cost from the current location to the candidate task, and the current remaining capacity of the vehicle. For example, if the priority function is "CFH + DEM", as shown in Fig. 3, then the priority function tends to select the tasks that are closer to the current location and have smaller demand. The routing policy works as a decision maker during the solution construction process. Normally, each vehicle can only serve one task at one time. The routing policy is applied to each unserved task to determine the priority of each unserved task once one vehicle becomes idle. The vehicle will go to serve the task with best priority value. When all the tasks have been served, the routes that the vehicle went through will be returned as the solution constructed by the routing policy.

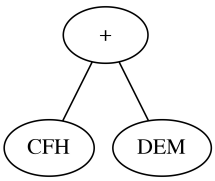


Fig. 3: A representation of GP tree.

D. Fitness Evaluation

To evaluate the fitness of a routing policy, we need know the size and performance of a routing policy. Given a routing policy rp, we can easily calculate the $size(rp)$ by counting the number of nodes in rp. The performance need to be calculate based on the average quality (i.e. total cost) of the routes that it returns. Specifically,

$$perf(rp) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} tc(rp, s), \tag{10}$$

where $\mathcal{S}$ is a set of instance samples, $|\mathcal{S}|$ is the number of instance samples. $tc(rp, S)$ stands for the total cost of the solution obtained by rp on sample s. The solution is constructed based on a simulation process which is commonly used in UCARP literature [10], [26].

In Algorithm 1, we randomly re-sample a subset of training instances for the fitness evaluation (lines 4). Such instance rotation has been commonly used in other studies [7], [40], [41] and has been demonstrated to be able to improve the generalisation of the evolved solutions.

IV. Experimental Studies

To evaluate the proposed approach, we test them on a number of UCARP instances which are commonly used in UCARP literature [7], [8], [9], [10]. For the sake of convenience, we denote the α-MOGP with different adaption schemes as follows: α-MOGP-linear (α-MOGP-l), α-MOGP-sigmoid (α-MOGP-s) and α-MOGP-cosine (α-MOGP-c) . α-MOGP-l adjust the α value based on a linear adaption scheme. α-MOGP-s adjusts the α value based on a sigmoid adaption scheme. α-MOGP-c adjusts the α value based on a cosine adaption scheme. We compare these three algorithms with the SimpleGP [9], which evolves a single routing policy using GPHH, Two-stage GPHH [39], which apply single-objective GPHH in the first stage and multi-objective GPHH in the second stage, and some other traditional MOGP algorithms, such as NSGA-II [30] and SPEA2 [11].

A. Experiment Setup

We select 8 commonly used UCARP instances to evaluate the performance of the proposed approach. The problem size varies from 22 tasks and 5 vehicles (small) to 97 tasks and 10 vehicles (large). Thus, we can examine the effectiveness of the proposed methods in different problem scenarios. In the training phase, routing policies are trained based on the training set $\mathcal{S}_{train}$. There are 5 training samples during the evaluation and they are re-sampled each generation (Algorithm 1 line: 4). In the test phase, the routing policy will be tested on an unseen test set $\mathcal{S}_{test}$, which contains 500 test samples that can avoid testing bias.

Commonly used function set $\{+, -, \times, /, \min, \max\}$. The "/" operator is protected divide which returns 1 if divided by 0, are applied in the experiments. The terminal set is shown in Table I.

TABLE IV: The mean and standard deviation for IGD of the compared algorithms in test process. For each method, (+), (-) and (=) indicates it is significantly lower (better) than, higher (worse) than, and comparable with NSGA-II (the first parentheses) and SPEA2 (the second parentheses).

Instance	NSGA-II [30]	SPEA2 [11]	TS-GPHH[39]	α-MOGP-l	α-MOGP-s	α-MOGP-c
Ugdb1	0.0972(0.0205)	0.1316(0.0373)	0.0835(0.028)(+)(+)	0.0786(0.0375)(+)(+)	0.0733(0.021)(+)(+)	0.0729(0.0284)(+)(+)
Ugdb2	0.1619(0.0078)	0.1806(0.0277)	0.112(0.0252)(+)(+)	0.1197(0.023)(+)(+)	0.105(0.0218)(+)(+)	0.1212(0.0214)(+)(+)
Ugdb8	0.0748(0.0198)	0.1205(0.0406)	0.0672(0.0306)(+)(+)	0.0778(0.0448)(=)(+)	0.0466(0.0169)(+)(+)	0.0676(0.041)(+)(+)
Ugdb23	0.1376(0.0171)	0.1487(0.0225)	0.0909(0.0203)(+)(+)	0.083(0.0421)(+)(+)	0.0717(0.0177)(+)(+)	0.0791(0.0366)(+)(+)
Uval9A	0.1018(0.0215)	0.1194(0.0237)	0.0548(0.0154)(+)(+)	0.0608(0.0195)(+)(+)	0.0568(0.025)(+)(+)	0.0499(0.0194)(+)(+)
Uval9D	0.1403(0.02)	0.195(0.0507)	0.0767(0.0194)(+)(+)	0.0742(0.0387)(+)(+)	0.0713(0.0191)(+)(+)	0.0696(0.0319)(+)(+)
Uval10A	0.1029(0.0196)	0.1332(0.0282)	0.0635(0.016)(+)(+)	0.0619(0.0167)(+)(+)	0.0647(0.0189)(+)(+)	0.0593(0.0232)(+)(+)
Uval10D	0.0751(0.0198)	0.098(0.0275)	0.0507(0.0226)(+)(+)	0.0645(0.0509)(+)(+)	0.0331(0.0131)(+)(+)	0.0543(0.0251)(+)(+)

TABLE V: The WDL table for IGD on all 8 instances in W-D-L format for each compared approach.

Approach	NSGA-II [30]	SPEA2 [11]	TS-GPHH[39]	α-MOGP-l	α-MOGP-s	α-MOGP-c
NSGA-II	0-8-0	0-0-8	8-0-0	7-1-0	8-0-0	8-0-0
SPEA2	8-0-0	0-8-0	8-0-0	8-0-0	8-0-0	8-0-0
TS-GPHH	0-0-8	0-0-8	0-8-0	1-7-0	3-5-0	2-6-0
α-MOGP-l	0-1-7	0-0-8	0-7-1	0-8-0	3-5-0	1-7-0
α-MOGP-s	0-0-8	0-0-8	0-5-3	0-5-3	0-8-0	0-6-2
α-MOGP-c	0-0-8	0-0-8	0-6-2	0-7-1	2-6-0	0-8-0

The population size for all the compared algorithms is 1000. Besides that, the total number of generations for all the compared algorithms set to 50. For the initialisation, ramp-half-and-half initialisation is used. The ratio of crossover to mutation to reproduction is $0.8 : 0.15 : 0.05$. For all α-MOGP algorithms, the α-tournament selection size is 7. For NSGA-II, the tournament selection size is 2 which is a common setting for NSGA-II. For SPEA2, the tournament selection size is 7 which is a common setting for SPEA2. For SimpleGP, the tournament selection size is 7 which is commonly used in UCARP experiments. The maximal depth is 8, which has been commonly used in previous studies (e.g., [7]). Elitism size for all compared approaches is 10 except NSGA-II and SPEA2. There is no elitism parameter for NSGA-II and SPEA2 approaches.

We use Evolutionary Computation Java (ECJ) package [42] to implement all the algorithms. The result is collected based on 30 independent runs for each algorithm on each UCARP instance.

B. Results and Discussions

We use two commonly adopted measures for multi-objective optimisation, i.e. hyper-volume (HV, the larger the better) and Inverted Generational Distance (IGD, the smaller the better). The Wilcoxon rank sum test with a significance level of 0.05 is used to verify the performance of the proposed approaches. Each $+$, $-$ and $=$ in parentheses refer the Wilcoxon rank sum test significance. The first parentheses refer to the Wilcoxon rank sum test significance between compared algorithms and NSGA-II. The second parentheses refer to the Wilcoxon rank sum test significance between compared algorithms and SPEA2.

Table II shows the mean and standard deviation of HV of the compared algorithms. It can be seen that all α-MOGP algorithms significantly outperforms NSGA-II and SPEA2 on all instances. This indicates the effectiveness of the proposed α-MOGP. To make further comparison, we also make a pairwise comparison between the algorithms. Table III shows the pairwise comparison results between the algorithms. In the table, each entry represents the comparison result between the column algorithm and the row algorithm. The entry is formatted in W-D-L format. W (L) indicates the number of instances where the column approach performs significantly better (worse) than the row approach. D indicates the number of instances where the two approaches showed no significant difference.

From the table, we can see that all α-MOGP algorithms can generate better Pareto fronts than NSGA-II and SPEA2. Also, α-MOGP-s performs best among all α-MOGP algorithms and TS-GPHH. It is significantly better than α-MOGP-l on 5 instances and significantly better than α-MOGP-c on 3 instances. It can also generate better Pareto fronts than TS-GPHH. We can see that it perform significantly better than TS-GPHH on 6 out of 8 instances. Besides that, it does not perform significantly worse on any instance than other algorithms.

Table IV shows mean and standard deviation of IGD of compared algorithms. It can be seen that α-MOGP-l outperforms NSGA-II on 7 out of total 8 instances. Besides that, all other α-MOGP algorithms and TS-GPHH outperform NSGA-II on all instances. It is clear to see that all compared algorithms can outperform SPEA2 on all instances. This is consistent with HV. We also make a pairwise comparison on each algorithm on IGD, and the results are shown in Table V. In the table, each entry represents the comparison result between the column algorithm and the row algorithm. The entry is formatted in W-D-L format. W (L) indicates the number of instances where the column approach performs significantly better (worse) than the row approach. D indicates the number of instances where the two approaches showed no significant difference. The result is consistent with HV, all compared algorithms can obtain better front than NSGA-II and SPEA2, and α-MOGP-s performs best among all α-MOGP algorithms.

Performance is still the primary objective of this study. Thus, we take the routing policies with the best performance from every front to compare the mean performance and size for all compared algorithms. The result is shown in Table VI and

TABLE VI: The mean and standard deviation for test performance (total cost) of the compared algorithms. For each method, (+), (-) and (=) indicates it is significantly lower (better) than, higher (worse) than, and comparable with SimpleGP.

Instance	SimpleGP	NSGA-II [30]	SPEA2 [11]	TS-GPHH[39]	α-MOGP-l	α-MOGP-s	α-MOGP-c
Ugdb1	355.47(14.83)	373.2(9.8)(-)	389.15(15.56)(-)	356.4(10.7)(=)	354.8(12.0)(=)	358.4(10.6)(=)	354.6(12.5)(=)
Ugdb2	371.72(7.67)	392.8(6.5)(-)	404.08(15.62)(-)	372.0(9.1)(=)	370.2(7.9)(=)	370.8(6.4)(=)	370.5(7.0)(=)
Ugdb8	463.34(54.3)	476.3(15.1)(-)	509.82(29.91)(-)	452.0(23.0)(=)	441.1(10.8)(+)	448.9(12.7)(=)	443.7(11.2)(+)
Ugdb23	252.47(3.11)	260.2(2.9)(-)	262.35(3.66)(-)	253.0(3.3)(=)	250.5(2.2)(+)	252.0(3.1)(=)	251.1(2.7)(=)
Uval9A	335.13(3.8)	351.3(6.0)(-)	371.24(19.77)(-)	336.7(4.6)(=)	336.0(3.7)(=)	336.4(3.4)(=)	336.3(3.8)(=)
Uval9D	478.14(16.68)	522.7(17.4)(-)	586.58(51.35)(-)	480.7(10.1)(=)	474.2(11.9)(=)	479.3(13.4)(=)	474.8(12.2)(=)
Uval10A	439.41(5.97)	460.0(7.0)(-)	481.59(16.89)(-)	442.0(10.9)(=)	440.5(3.9)(=)	440.9(4.4)(-)	439.7(4.0)(=)
Uval10D	620.91(7.97)	668.9(24.0)(-)	699.8(32.83)(-)	624.2(8.6)(=)	619.6(8.4)(=)	622.2(10.0)(=)	617.5(10.4)(+)

TABLE VII: The mean and standard deviation for size of routing policies of the compared algorithms.

Instance	SimpleGP	NSGA-II [30]	SPEA2 [11]	TS-GPHH[39]	α-MOGP-l	α-MOGP-s	α-MOGP-c
Ugdb1	74.6(23.84)	10.0(3.99)	8.4(5.18)	30.53(18.02)	27.67(15.97)	17.33(11.98)	27.87(15.27)
Ugdb2	71.93(23.79)	6.93(3.13)	5.73(3.22)	38.33(19.8)	28.07(14.12)	26.53(13.27)	29.0(14.08)
Ugdb8	65.47(24.33)	7.07(3.08)	5.6(4.49)	42.27(36.62)	51.67(19.03)	30.87(13.97)	41.6(22.49)
Ugdb23	71.8(25.22)	8.27(3.66)	8.6(4.53)	31.13(19.5)	47.53(26.57)	33.07(24.67)	43.27(24.64)
Uval9A	56.93(18.27)	9.73(4.65)	8.53(4.83)	30.6(13.72)	28.4(14.79)	26.07(14.04)	30.6(12.68)
Uval9D	69.27(29.46)	10.33(4.85)	7.53(6.15)	42.67(19.31)	58.0(29.24)	37.0(22.18)	49.87(20.02)
Uval10A	60.47(18.59)	8.07(3.47)	4.53(4.54)	24.07(10.86)	19.27(9.74)	15.73(6.0)	23.6(12.99)
Uval10D	65.33(14.35)	8.93(3.46)	9.2(5.18)	35.0(21.22)	47.67(23.7)	34.13(17.92)	45.73(21.52)

Table VII, respectively. Table VI shows the mean and standard deviation of test performance of the compared algorithms. Wilcoxon rank sum test with a significance level of 0.05 is used to compare each approach with SimpleGP. From Table VI, one can see that both NSGA-II and SPEA2 perform worse than SimpleGP since they cannot handle the challenge of premature convergence on smaller individuals. All α-MOGP algorithms can obtain a comparable result with SimpleGP. Especially, both α-MOGP-l and α-MOGP-c perform significantly better than SimpleGP on 2 out of 8 instances.

Table VII shows the mean and standard deviation of size of the compared algorithms. All the compared approaches can evolve much smaller routing policies than SimpleGP. This is as expected since we use size as an objective. The result indicates that using size as an objective can reduce routing policy size effectively. There is one thing that needs to be noticed that both NSGA-II and SPEA2 achieves much smaller size on all instances than other algorithms. This is because either NSGA-II or SPEA2 has the problem of premature convergence to small individuals which will lead to poor performance (shown in Table VI).

Overall, the proposed α-MOGP can generate better Pareto fronts than NSGA-II, SPEA2 and TS-GPHH. Besides that, it can achieve much smaller rule size than SimpleGP with comparable performance. Primarily, with proper adaption scheme, it can obtain better performance.

C. Further Analysis

To analyse the routing policies evolved by α-MOGP, we picked one of the best performing routing policies from α-MOGP-s for Ugdb1 as an example. The routing policy evolved by α-MOGP-s, denoted as RP1, is shown in Eq. 11.

$$RP1 = \max(S_1, S_2) \tag{11}$$

where

$$S_1 = DC * CFH + CTT1 \tag{12}$$

$$S_2 = \frac{DEM1}{DC - CFR1} \tag{13}$$

From S_1, we can observe that the policy tends to select the tasks with smaller CFH (close to the current place) and smaller CTT1. From S_2, one can see that the policy tends to select the tasks with smaller DEM1. Thus RP1 tends to select the tasks close to current place or the tasks that their closest task has a small demand.

We have also analysed the final Pareto front. One interesting observation is that a large amount of small and good routing policies contains the same building block "DC * CFH". The final Pareto front contains a large number of individuals with one single terminal. Another interesting observation is that most of the single-terminal individuals are "CFH". This indicates that "CFH" is critical for build small and good individuals.

V. CONCLUSIONS

The primary purpose of this paper is to improve the interpretability of the routing policies evolved by GPHH for UCARP by optimising the performance and size simultaneously. To this end, this paper proposes a simple yet effective α-MOGP approach, and apply it with different adaption schemes that are used to adaptively adjust α value. The experimental results showed that with a proper adaption scheme, the proposed α-MOGP approach could evolve much smaller routing policies without losing the test performance. This also indicates that our newly proposed approach can deal with the issues in the multi-objective optimisation while one objective is much easier to optimise than other objectives. Especially, size is much easier to optimise than performance in UCARP. In the future,

we will consider self-adaptive α value adjustment to improve the ability to balance the tradeoff among each objective.

REFERENCES

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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*. New York, NY, USA: ACM, 2018, pp. 141–142.

[10] 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.

[11] S. Bleuler, J. Bader, and E. Zitzler, "Reducing bloat in gp with multiple objectives," in *Multiobjective Problem Solving from Nature*. Springer, 2008, pp. 177–200.

[12] K. Ikeda, H. Kita, and S. Kobayashi, "Failure of pareto-based moeas: Does non-dominated really mean near to optimal?" in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, vol. 2. IEEE, 2001, pp. 957–962.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] G. Fleury, P. Lacomme, and C. Prins, "Evolutionary algorithms for stochastic arc routing problems," in *Workshops on Applications of Evolutionary Computation*. Springer, 2004, pp. 501–512.

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

[23] 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.

[24] 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.

[25] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.

[26] S. Wang, Y. Mei, and M. Zhang, "Novel ensemble genetic programming hyper-heuristics for uncertain capacitated arc routing problem," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2019, pp. 1093–1101.

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

[28] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler, "Multiobjective genetic programming: Reducing bloat using spea2," in *Proceedings of the Congress on Evolutionary Computation*. IEEE, 2001, pp. 536–543.

[29] E. D. De Jong, R. A. Watson, and J. B. Pollack, "Reducing bloat and promoting diversity using multi-objective methods," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2001, pp. 11–18.

[30] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *International conference on parallel problem solving from nature*. Springer, 2000, pp. 849–858.

[31] F. Zhang, Y. Mei, and M. Zhang, "Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics," in *Proceedings of IEEE Congress on Evolutionary Computation*. IEEE, 2019, pp. 1366–1373.

[32] W. B. Langdon and J. Nordin, "Seeding genetic programming populations," in *European Conference on Genetic Programming*. Springer, 2000, pp. 304–315.

[33] A. Ekárt and S. Z. Nemeth, "Selection based on the pareto nondomination criterion for controlling code growth in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 2, no. 1, pp. 61–73, 2001.

[34] L. Panait and S. Luke, "Alternative bloat control methods," in *Genetic and Evolutionary Computation Conference*. Springer, 2004, pp. 630–641.

[35] Y. Bernstein, X. Li, V. Ciesielski, and A. Song, "Multiobjective parsimony enforcement for superior generalisation performance," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, vol. 1. IEEE, 2004, pp. 83–89.

[36] E. D. De Jong and J. B. Pollack, "Multi-objective methods for tree size control," *Genetic Programming and Evolvable Machines*, vol. 4, no. 3, pp. 211–233, 2003.

[37] L. Thiele, K. Miettinen, P. J. Korhonen, and J. Molina, "A preference-based evolutionary algorithm for multi-objective optimization," *Evolutionary computation*, vol. 17, no. 3, pp. 411–436, 2009.

[38] K. Deb and J. Sundar, "Reference point based multi-objective optimization using evolutionary algorithms," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, 2006, pp. 635–642.

[39] S. Wang, Y. Mei, J. Park, and M. Zhang, "A two-stage genetic programming hyper-heuristic for uncertain capacitated arc routing problem," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2019, pp. 1606–1613.

[40] F. Zhang, Y. Mei, and M. Zhang, "A two-stage genetic programming hyper-heuristic approach with feature selection for dynamic flexible job shop scheduling," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2019, pp. 347–355.

[41] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach," in *Proceedings of Genetic and Evolutionary Computation Conference*. ACM, 2010, pp. 257–264.

[42] S. Luke, L. Panait, G. Balan, S. Paus, Z. Skolicki, J. Bassett, R. Hubley, and A. Chircop, "Ecj: A java-based evolutionary computation research system," *Downloadable versions and documentation can be found at the following url: http://cs. gmu. edu/eclab/projects/ecj*, 2006.