A Two-Stage Genetic Programming Hyper-Heuristic 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*—Genetic Programming Hyper-heuristic (GPHH) has been successfully applied to automatically evolve effective routing policies to solve the complex Uncertain Capacitated Arc Routing Problem (UCARP). However, GPHH typically ignores the interpretability of the evolved routing policies. As a result, 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. To this end, we propose a new Multi-Objective GP (MOGP) to optimise the performance and size simultaneously. A major issue here is that the size is much easier to be optimised than the performance, and the search tends to be biased to the small but poor routing policies. To address this issue, we propose a simple yet effective Two-Stage GPHH (TS-GPHH). In the first stage, only the performance is to be optimised. Then, in the second stage, both objectives are considered (using our new MOGP). The experimental results showed that TS-GPHH could obtain much smaller and more interpretable routing policies than the state-of-the-art single-objective GPHH, without deteriorating the performance. Compared with traditional MOGP, TS-GPHH can obtain a much better and more widespread Pareto front.**

***Keywords*—UCARP; GPHH; routing policy; multi-objective.**

I. Introduction

The Capacitated Arc Routing Problem (CARP) [1] has broad real-world applications such as waste collection [2]. Theoretically, CARP has been proved to be NP-hard [3]. The problem aims to serve a set of *required edges* in a graph using a fleet of vehicles with minimum cost.

CARP has received much research interest, and there have been extensive studies for solving it [1], [3]. However, most of the previous studies consider that all the problem parameters (e.g., travel time) are fixed and known in advance. This assumption is usually not true in reality, where there are a lot of uncertainties. For example, in snow removal, the amount of snow to be removed on a street varies from one day to another, and cannot be exactly known in advance.

The Uncertain Capacitated Arc Routing Problem (UCARP) [4] was proposed to reflect the reality better, where the demands and travel costs are uncertain. *Route failures*, which is caused by the stochastic demand, is one of the main main challenges in UCARP. When *Route failures* occur, the vehicle has to go back to the depot to replenish in the middle of the service, because the actual demand of a required edge is greater than expected, the remaining capacity of the vehicle is insufficient to complete the service. This will lead to huge recourse cost.

Traditional solution optimisation approaches, such as mathematical programming [5] and evolutionary algorithms [6], are less effective when applied to UCARP, due to the challenge of dealing with route failures. In general, they try to optimise a robust solution beforehand. When route failures occur, a recourse operator is applied to modify it. However, it is hardly possible to have a single robust solution that can fit all the environments. On the other hand, it is time-consuming to re-optimise the entire remaining solution from scratch on-the-fly.

Routing policy [7] is considered as a promising technique to deal with the uncertain environment in UCARP. Unlike other solutions optimisation approaches, a routing policy creates the solution based on the latest information on-the-fly. Specifically, whenever a vehicle becomes idle, it uses the routing policy to decide the next service. Routing policy has the capability to efficiently give responses to the environment change, and is much more flexible than traditional solution optimisation approaches, as it can provide different solutions based on different environments. Some existing constructive heuristics, such as Path Scanning [6], designed some effective routing policies manually.

The effectiveness of a routing policy largely depends on the scenario, the objective(s), and even the graph topology [8]. It is very time-consuming to manually design effective routing policy for a given problem scenario. To overcome this drawback, Genetic Programming Hyper-heuristic (GPHH) approaches have been applied to UCARP to automatically evolve routing policies [9]–[12].

In the real world, it is important for human users to understand and trust a routing policy. However, the existing studies on GPHH only focused on the effectiveness but ignored the interpretability. On the other hand, GP tends to continuously increase the size of its individuals, known as *bloat* [13]–[15]. As a result, the GP-evolved routing policies are typically very large and complex and hard to interpret.

A commonly used approach, which can improve the interpretability of the GP-evolved routing policies, is to minimise the complexity of the individuals along with the original objective(s). A simple measure of complexity is the size (number of nodes in the GP tree), as a smaller routing policy tends to be easier to interpret. Therefore, in this paper, we propose to solve the multi-objective UCARP, which is to optimise the performance and the size of the routing policy simultaneously.

When optimising the performance and size of the routing policies together, a major issue is that the size is much easier to be optimised than the performance. In other words, it is much easier to generate routing policies with small size than with good performance. Therefore, the population tends to contain a large number of small (but poor) routing policies. On the other hand, GP requires large individuals to explore the search space effectively. In this case, a traditional Multi-Objective GP (MOGP) tends to have many small routing policies in the population, and gradually loses its exploration ability. Bleuler [16] summarises two basic strategies, using a bias against small individuals and increasing the diversity in the population, which can deal with this problem. Strength Pareto Evolutionary Algorithm 2 (SPEA2) [16], which outperforms two basic strategies, is also applied to deal with this issue.

In this paper, we aim to address the above issue by proposing a simple yet effective Two-Stage GPHH (TS-GPHH). In TS-GPHH, the whole search process is divided into two stages. In the first stage, only the performance is considered, and the problem is solved as a single-objective optimisation. In the second stage, both the performance and the size are to be optimised based on the final population of the first stage. We expect the final population of the first stage can provide individuals with good performance, which can be used in the second stage as *anchors* to reduce the search bias towards the routing policy size. Specifically, we have the following research objectives:

1) Develop a novel TS-GPHH to evolve interpretable and effective routing policies for UCARP;
2) Investigate different elitism mechanism in the second multi-objective stage of TS-GPHH;
3) Verify the efficacy of TS-GPHH and analyse the evolved routing policies.

II. Background

A. Uncertain Capacitated Arc Routing Problem

UCARP is based on a connected graph $G(V, E)$, where V and E are the sets of vertices and edges. Each edge $e \in E$ has a positive random deadheading cost $\tilde{dc}(e)$, indicating the cost of traversing the edge. A set of edges $E_R \subseteq E$ are required to be served. Each required edge (also called *task*) $e_R \in E_R$ has a positive random demand $\tilde{d}(e_R)$ 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 minimise the total cost of serving all the tasks in E_R. A vehicle must start and finish at v_0, and the total demand served between two subsequent depot visits cannot exceed the capacity of the vehicle.

A UCARP instance contains many random variables, each of which can have different samples. Thus, one UCARP instance contains different samples. In an instance sample, each random variable such as a task demand has a realised value that is not known in advance. For example, the actual task demand 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.

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

A solution to a UCARP instance sample is represented as $S = (S.X, S.Y)$. $S.X = \{S.X^{(1)}, \dots, S.X^{(m)}\}$ is a set of vertex sequences, where $S.X^{(k)} = (S.x_1^{(k)}, \dots, S.x_{L_k}^{(k)})$ stands for the k^{th} route. $S.Y = \{S.Y^{(1)}, \dots, S.Y^{(m)}\}$ is a set of continuous vectors, where $S.Y^{(k)} = (S.y_1^{(k)}, \dots, S.y_{L_k-1}^{(k)})$ ($S.y_i^{(k)} \in [0, 1]$) is the fraction of demand served along the route $S.X^{(k)}$. For example, if $S.y_3^{(1)} = 0.7$, then $(S.x_3^{(1)}, S.x_4^{(1)})$ is a task and 70% of its demand is served at position 3 of the route $S.X^{(1)}$.

Under the above representation, the problem can be formulated as follows.

$$\min \ \mathrm{E}_{\xi \in \Xi}[C(S_\xi)], \tag{1}$$

$$s.t. \ \ S_\xi.x_1^{(k)} = S_\xi.x_{L_k}^{(k)} = v_0, \quad \forall k = 1, 2, ..., m \tag{2}$$

$$\sum_{k=1}^{m} \sum_{i=1}^{L_k-1} S_\xi.y_i^{(k)} \times S_\xi.z_i^{(k)}(e) = 1, \forall e : d_\xi(e) > 0, \tag{3}$$

$$\sum_{k=1}^{m} \sum_{i=1}^{L_k-1} S_\xi.y_i^{(k)} \times S_\xi.z_i^{(k)}(e) = 0, \forall e : d_\xi(e) = 0, \tag{4}$$

$$\sum_{i=1}^{L_k-1} d_\xi \left(S_\xi.x_i^{(k)}, S_\xi.x_{i+1}^{(k)}\right) \times S_\xi.y_i^{(k)} \le Q, \forall k = 1, \dots, m, \tag{5}$$

$$\left(S_\xi.x_i^{(k)}, S_\xi.x_{i+1}^{(k)}\right) \in E, \tag{6}$$

$$S_\xi.y_i^{(k)} \in [0, 1], \tag{7}$$

where in Eqs. (3) and (4), $S_\xi.z_i^{(k)}(e)$ equals 1 if $\left(S_\xi.x_i^{(k)}, S_\xi.x_{i+1}^{(k)}\right) = e$, and 0 otherwise. Eq. (1) is the objective function, which is to minimise the expected total cost $C(S_\xi)$ of the solution S_ξ in all possible environments $\xi \in \Xi$. Eq. (2) indicates that in all S_ξ, the routes start and end at the depot. Eqs. (3) and (4) mean that each task is served exactly once (the total demand fraction served by all vehicles is 1), while each non-required edge is not served. Eq. (5) is the capacity constraint, and Eqs. (6) and (7) are the domain constraints of $S_\xi.X$ and $S_\xi.Y$.

B. Related Work

The CARP is a classical combinatorial optimisation problem. Extensive researches have been made on it. Golden and Wong [5] developed an integer linear programming model and solved it using branch-and-cut. However, this approach can only solve small instances. Tabu search [17], [18] approach was proposed for static CARP for improvement. After that, Genetic Algorithm [19] and Memetic Algorithms [6], [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. All these studies make great contributions to CARP. However, they are not applicable to UCARP directly. The reason is that they are not able to deal with the route failure effectively.

The proactive approaches [24]–[26] and the reactive approaches [9]–[11] are two main types of approaches for solving UCARP. The main idea of proactive approaches is to find robust solutions that can handle all the possible realisations of the random variables. In contrast, reactive approaches mainly focus on using GPHH to evolve routing policies which construct the solution gradually in real time.

Weise et al. [27] first proposed a GPHH for UCARP with a single vehicle and examined its performance. Liu et al. [9] designed a novel and effective meta-algorithm which filters irrelevant candidate tasks during the decision-making process and achieved better performance than the GPHH in [27]. After that, Mei et al. [11] extended the model from single-vehicle to multiple-vehicle version, so that the solution can be generated with multiple vehicles on the road simultaneously. A novel task filtering method and an effective look-ahead terminal were proposed by MacLachlan et al. [10] to improve the GPHH further.

When evolving routing policies by GPHH, the interpretability of the evolved routing policies is an important aspect to guarantee the confidence of the users. However, the current GPHH approaches cannot achieve satisfactory interpretability. As discussed in Section I, the reason why routing policies evolved by GPHH is hard to interpret is because of GP tends to generate larger trees. Tree size control is regarded as a good approach to reduce the structural complexity of GP so that we can have more interpretable routing policies. So it is wise to find some methods that can control the tree size.

The most common and straightforward way to control tree size is to limit the tree size or depth directly [28]. However, there are two limitations to this method. First, the diversity near the root node would be decreased when used with crossover [29]. Second, it is difficult to set a reasonable limit for size or depth [13]. To this end, a penalty term is added to fitness function. This penalty term is used to reward those trees with smaller size and penalize those trees with a larger size. However, this approach also has its own limitations. First, the weights of the penalty term and the fitness have to be predefined. Second, 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. [14]. The reason is that it is difficult to find a weight value that can perfectly balance the original fitness and size of individuals for the whole objective space [30]. Multi-objective optimisation approach has been considered as an effective approach to control the tree size [13]. The multi-objective optimisation approach does not have to predefine any aggregation functions to combine those objective values and find a set of solutions that each solution can satisfy different objectives. There are multiple multi-objective optimization approaches such as SPEA2 [13] and NSGA-II [31] that can be applied to control tree size. However, there is one problem that is using a small tree size as an objective. It can result in premature convergence to small individuals [14], [32]. This problem can be partially addressed by Preference-Based EMO approach [33], [34]. However, to our knowledge, Preference-Based EMO only focuses on some specific regions of the Pareto front, such as the region with good performance and large size. As a result of using Preference-Based EMO, we may still get a small portion of the Pareto front in the preferred region. However, our goal is still to cover the entire Pareto front as much as possible, which in this case cannot be easily achieved by the traditional EMO algorithms. Bleuler [16] summaries two basic strategies that can deal with this problem using MOEA. First, using a bias against small individuals [35], [36]. Second, increase the diversity in the population [14], [37], [38]. Bleuler [13] also applies Pareto-based strategies to GP to deal with this issue on a even-parity problem.

In this paper, we aim to propose a novel TS-GPHH approach that uses performance and size as two objectives. Besides that, TS-GPHH is expected to avoid premature convergence to small individuals so that we can have compact and effective routing policies.

III. Two-Stage Multi-Objective Genetic Programming

A. Overall Framework

The overall framework of TS-GPHH is given in Algorithm 1. It consists of two stages. The first stage (lines 3–12) is from generation 0 to generation $G/2$, i.e. half of the entire search process. It is a standard single-objective GP, where the elitism (line 6) and parent selection (line 8) are based on the single objective value (i.e. the total cost (performance)).

The second stage (lines 13–22) is a multi-objective GP. It is almost the same as the first stage, except the elitism (line 16) and parent selection (line 18) are based on both objective values (performance and size). However, it is quite different from NSGA-II which uses random selection to select parents, and combines two populations and select the individuals by non-dominated sorting. NSGA-II is too greedy for our problem. So we still keep the selection of GP, a simple multi-objective tournament selection. Specifically, given k randomly selected individuals, the non-dominated individual is selected to be the parent. If there are multiple non-dominated individuals, the first one identified during the comparison is selected. The multi-objective elitism mechanism of our approach will be described in more detail in Section III-D. Finally, the non-dominated routing policies in the population is returned.

B. Individual Representation

In TS-GPHH, each routing policy is essentially an arithmetic priority function. The routing policy is applied in a solution construction process, which is modelled as a decision

Algorithm 1: The overall framework of TS-GPHH.

```
Input: Training set $\mathcal{I}_{train}$, number of generations $G$,
       number of elites $k$
Output: A set of routing policies $\mathcal{RP}$
1  initialise the population $pop$;
2  $g = 0$;
   // stage 1: single-objective
3  while $g < G/2$ do
4      randomly sample a training subset $\mathcal{I}' \subseteq \mathcal{I}_{train}$;
5      evaluate $pop$ using $\mathcal{I}'$;
6      the next population $pop' = $ so-elites$(pop, k)$;
7      while $|pop'| < popsize$ do
8          Breed $pop'$ using single-objective parent
             selection and genetic operators;
9          $g = g + 1$;
10     end
11     $pop = pop'$;
12 end
   // stage 2: multi-objective
13 while $g < G$ do
14     randomly sample a training subset $\mathcal{I}' \subseteq \mathcal{I}_{train}$;
15     evaluate $pop$ using $\mathcal{I}'$;
16     the next population $pop' = $ mo-elites$(pop, k)$;
17     while $|pop'| < popsize$ do
18         Breed $pop'$ using multi-objective parent selection
             and genetic operators;
19         $g = g + 1$;
20     end
21     $pop = pop'$;
22 end
23 return the non-dominated routing policies in $pop$;
```

making process. During the process, the vehicles serve a task at a time. Once a vehicle is idle, the routing policy is applied to all the unserved tasks to calculate the priority of each unserved task. Then, the most prior feasible task will be selected to serve next. For example, a priority function is “SC + DEM”, then the routing policy prefers tasks that require less serving cost and demand. The process completes when all the tasks have been served, and the routes of the vehicles are returned as the solution constructed by the routing policy.

C. Fitness Evaluation

The fitness evaluation requires to calculate the size and performance of a routing policy. Given a routing policy x, the $size(x)$ can be simply calculated by counting the number of nodes in the tree. The performance measure is defined as the average solution quality (i.e. total cost) generated by x over a set of instance samples $\mathcal{I}$. Specifically,

$$perf(x) = \frac{1}{|\mathcal{I}|}\sum_{I \in \mathcal{I}} tc(x, I), \tag{8}$$

where $|\mathcal{I}|$ is the number of instance samples. $tc(x, I)$ stands for the total cost of the solution obtained by x on sample I.

In Algorithm 1, we randomly re-sample a subset of training instances for the fitness evaluation (lines 4 and 14). Such instance rotation has been commonly used in other studies [9], [39] and has been demonstrated to be able to improve the generalisation of the evolved solutions.

D. Elitism

Elitism is very important for multi-objective optimisation. A typical elitism mechanism in evolutionary multi-objective optimisation is to combine the parent and offspring populations and select the next population from the combined population (e.g. NSGA-II [31]). However, our preliminary study shows that such selection is too greedy for GP, making it easier to get stuck in poor local optima. To maintain a good tradeoff between exploration and exploitation in the second stage of TS-GPHH, we still inherit the standard single-objective GP process, which copies only a small portion of elites from the parent population.

1) so-elites: In the single-objective case, the elitism in GP copies the top n ($n \ll popsize$) individuals to the next generation in terms of the single objective value.

2) mo-elites: The elitism in the multi-objective case is designed based on the dominance relation. Specifically, we sort the individuals using the non-dominated sorting [31] and select the top fronts one by one. When copying each front, we consider two options: (1) with duplicates and (2) without duplicates. It simply adds the fronts into the elite set one by one until the elite set is full. The only difference between with duplicates and without duplicates is that before adding each front, it removes the duplicates (the individuals with the same performance and size) from the front.

IV. Experimental Studies

To evaluate the performance of the proposed approach, we test them on a number of UCARP instances. For the sake of convenience, we denote the TS-GPHH with different elitism mechanisms as follows: TS-GPHH-d (using so-elites$(performance)$ in stage one and mo-elites with duplicates in stage two), TS-GPHH-n (using so-elites$(performance)$ in stage one and mo-elites without duplicates in stage two) and TS-GPHH-s (using so-elites$(performance)$ in both stage one and stage two). We compare these three algorithms with the SimpleGP [11], which evolves a single routing policy using GPHH, traditional NSGA-II which simply use performance and size as two objectives, and Two-Stage NSGA-II (TS-NSGA-II) which optimises single objective (performance) stage one and optimises both objectives in stage two.

A. Experiment Setup

8 representative UCARP instances are selected from the commonly used Ugdb and Uval datasets [9]–[11] to evaluate the performance of the proposed approach. For each instance, the problem size may vary from 22 tasks and 5 vehicles (small) to 97 tasks and 10 vehicles (large). In this way, the performance of the proposed approach can be evaluated in

TABLE I: The terminal set in the experiments.

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

different scenarios. There are two phases for each UCARP instance and each algorithm in the experiment, training and test phases. In the training phase, a routing policy is generated based on the training set T_{train}. During the training phase, all the algorithms use 5 training samples during the evaluation. The 5 training samples will be re-sampled in each generation.. In the test phase, the routing policy will be tested on an unseen test set, which contains 500 test samples that can avoid testing bias.

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

The total number of generations for all the compared algorithms is 50. The population size for all the compared algorithms is 1000. 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 NSGA-II, the tournament selection size is 2. For all other algorithms, the tournament selection size is 7. The maximal depth is 8, which has been commonly used in previous studies (e.g., [9]). Elitism size for all compared approaches is 10 except NSGA-II. There is no elitism parameter for NSGA-II approach.

Evolutionary Computation Java (ECJ) package [40] is applied to implement all the algorithms. Each algorithm was run 30 times independently for each UCARP instance.

B. Results and Discussions

For the value of HV, the larger the better. The Wilcoxon rank sum test with a significance level of 0.05 is used to verify the performance of the proposed approaches. For each $+$, $-$ and $=$ in parentheses refer the Wilcoxon rank sum test significance between each approach and NSGA-II.

Table II shows mean and standard deviation of HV of compared approaches. It can be seen that TS-NSGA-II significantly outperforms NSGA-II on all instances. TS-GPHH-d significantly outperforms NSGA-II on 7 out of the total 8 instances. Both TS-GPHH-n and TS-GPHH-s significantly outperforms NSGA-II on all instances. This indicates the effectiveness of the two-stage mechanism.

Table III shows the pairwise comparison results between the algorithms. In the table, each entry is represented in W-D-L format. W (L) indicates the number of instances where the row approach performs significantly better (worse) than the column approach. D indicates the number of instances where the two approaches showed no significant difference. From the table, we can see that TS-GPHH-n obtained the best HV over all the compared algorithms on almost all the instances. Both TS-GPHH-n and TS-GPHH-s can generate better Pareto fronts than NSGA-II and TS-NSGA-II.

Note that IGD is another important EMO performance measure. Due to the page limit, we omit the comparison of IGD here, as we observed consistent patterns with HV.

In this study, performance is the primary objective. Thus, we took the routing policies with the best performance obtained by the compared algorithms and compare their performance and size. The results are shown in Tables IV and V, respectively.

From Table IV, one can see that NSGA-II performs worse than SimpleGP on all instance. This is as expected since NSGA-II cannot handle the challenge of premature convergence on smaller individuals. Both TS-NSGA-II and TS-GPHH-d perform better than NSGA-II. However, they are still slightly worse than SimpleGP on all instances. TS-GPHH-n performs slightly worse than SimpleGP on 6 out of total 8 instances and achieves comparable performance with SimpleGP on Ugdb2 and Ugdb8. TS-GPHH-s performs best among all compared algorithms, and it obtains comparable performance with SimpleGP on all instances. This indicates that TS-GPHH-s can evolve smaller (shorter) routing policies without losing performance.

Table V shows that all the compared approaches can evolve much smaller routing policies than SimpleGP. This indicates that using size as an objective can reduce routing policy size effectively. NSGA-II achieves the smallest size on all instances, although it has the problem of premature convergence to small individuals (shown in Table IV). TS-GPHH-d evolves individuals of similar size with TS-NSGA-II. However, TS-GPHH-d obtains better performance than TS-NSGA-II. This indicates that the TS-GPHH approaches have the ability to find good individuals with smaller size. Both TS-GPHH-n and TS-GPHH-s obtain better mean performance than TS-GPHH-d, and their mean size is also larger than TS-GPHH-d. TS-GPHH-s obtains the greatest mean size among all three proposed approaches. This is as expected, TS-GPHH-s regards individual with better performance as elites. Usually, larger individuals are more likely to achieve better performance. So there will be much larger individuals in the population than the other two approaches. In this case, TS-GPHH-s obtains better performance than other compared approaches.

Overall, the proposed TS-GPHH can generate better Pareto fronts than NSGA-II. Besides that, with a proper elitism mechanism, it can achieve much smaller rule size than SimpleGP with comparable performance.

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.

Instance	NSGA-II	TS-NSGA-II	TS-GPHH-d	TS-GPHH-n	TS-GPHH-s
Ugdb1	0.67(0.09)	0.73(0.08)(+)	0.66(0.14)(-)	0.79(0.08)(+)	0.77(0.09)(+)
Ugdb2	0.81(0.03)	0.85(0.05)(+)	0.84(0.05)(+)	0.88(0.04)(+)	0.86(0.05)(+)
Ugdb8	0.81(0.05)	0.83(0.06)(+)	0.82(0.07)(+)	0.86(0.05)(+)	0.86(0.08)(+)
Ugdb23	0.84(0.03)	0.86(0.04)(+)	0.87(0.04)(+)	0.89(0.03)(+)	0.9(0.03)(+)
Uval9A	0.67(0.07)	0.78(0.07)(+)	0.8(0.07)(+)	0.81(0.04)(+)	0.78(0.05)(+)
Uval9D	0.66(0.08)	0.76(0.06)(+)	0.76(0.11)(+)	0.81(0.06)(+)	0.8(0.06)(+)
Uval10A	0.65(0.09)	0.75(0.08)(+)	0.86(0.06)(+)	0.87(0.06)(+)	0.81(0.13)(+)
Uval10D	0.58(0.15)	0.72(0.11)(+)	0.73(0.1)(+)	0.78(0.09)(+)	0.77(0.1)(+)

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

Approach	NSGA-II	TS-NSGA-II	TS-GPHH-d	TS-GPHH-n	TS-GPHH-s
NSGA-II		0-0-8	1-0-7	0-0-8	0-0-8
TS-NSGA-II	8-0-0		4-0-4	0-0-8	0-0-8
TS-GPHH-d	7-0-1	4-0-4		0-0-8	2-0-6
TS-GPHH-n	8-0-0	8-0-0	8-0-0		7-0-1
TS-GPHH-s	8-0-0	8-0-0	6-0-2	1-0-7	

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

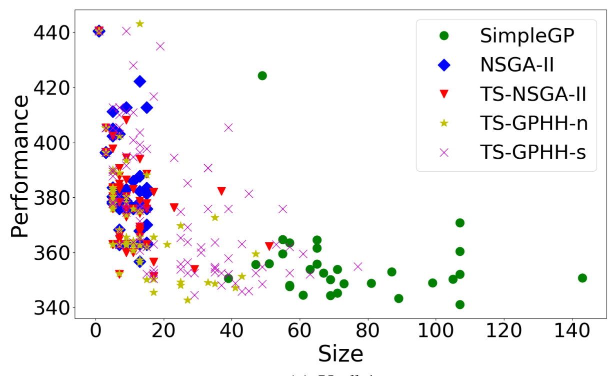
Instance	SimpleGP	NSGA-II	TS-NSGA-II	TS-GPHH-d	TS-GPHH-n	TS-GPHH-s
Ugdb1	355.4(14.8)	373.2(9.8)(-)	366.5(9.0)(-)	372.6(16.9)(-)	359.4(9.7)(-)	356.4(10.7)(=)
Ugdb2	371.7(7.6)	392.8(6.5)(-)	384.0(11.5)(-)	382.7(13.6)(-)	375.9(8.9)(=)	372.0(9.1)(=)
Ugdb8	463.3(54.3)	476.3(15.1)(-)	468.6(19.1)(-)	471.0(21.0)(-)	457.9(15.7)(=)	452.0(23.0)(=)
Ugdb23	252.4(3.1)	260.2(2.9)(-)	258.2(4.1)(-)	257.0(4.0)(-)	254.9(3.5)(-)	253.0(3.3)(=)
Uval9A	335.1(3.8)	351.3(6.0)(-)	341.1(6.8)(-)	338.7(5.2)(-)	337.9(3.4)(-)	336.7(4.6)(=)
Uval9D	478.1(16.6)	522.7(17.4)(-)	500.7(14.7)(-)	498.7(25.8)(-)	489.4(15.5)(-)	480.7(10.1)(=)
Uval10A	439.4(5.9)	460.0(7.0)(-)	451.1(6.9)(-)	441.2(5.8)(-)	441.0(4.5)(-)	442.0(10.9)(=)
Uval10D	620.9(7.9)	668.9(24.0)(-)	644.7(19.0)(-)	638.0(16.9)(-)	631.5(16.4)(-)	624.2(8.6)(=)

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

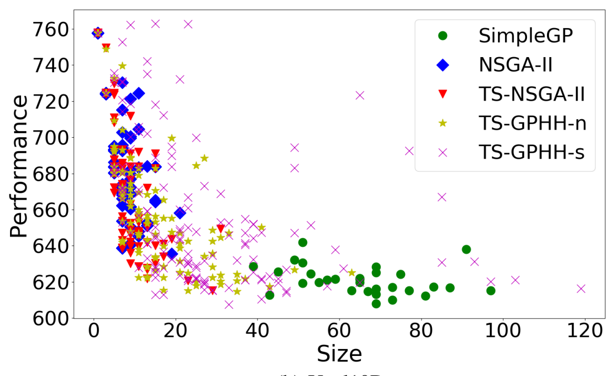
Instance	SimpleGP	NSGA-II	TS-NSGA-II	TS-GPHH-d	TS-GPHH-n	TS-GPHH-s
Ugdb1	74.6(23.84)	10.0(3.99)	12.13(9.41)	11.47(10.85)	15.6(10.45)	30.53(18.02)
Ugdb2	71.93(23.79)	6.93(3.13)	16.27(12.83)	17.93(15.27)	22.27(10.78)	38.33(19.8)
Ugdb8	65.47(24.33)	7.07(3.08)	16.07(20.08)	15.93(24.01)	22.6(15.94)	42.27(36.62)
Ugdb23	71.8(25.22)	8.27(3.66)	14.4(13.93)	14.8(12.15)	21.67(13.8)	31.13(19.5)
Uval9A	56.93(18.27)	9.73(4.65)	22.53(15.61)	24.53(17.88)	26.07(13.24)	30.6(13.72)
Uval9D	69.27(29.46)	10.33(4.85)	20.33(15.16)	23.73(24.32)	26.2(20.01)	42.67(19.31)
Uval10A	60.47(18.59)	8.07(3.47)	15.27(10.42)	16.53(8.06)	18.67(7.93)	24.07(10.86)
Uval10D	65.33(14.35)	8.93(3.46)	11.4(5.39)	19.33(15.34)	23.0(11.5)	35.0(21.22)

C. Further Analysis

In Fig. 1, we plot the routing policies evolved by SimpleGP and all the non-dominated routing policies from each compared approach for Ugdb1 and Uval10D. It can be seen that all routing policies from SimpleGP locate on the right-bottom corner. This indicates that SimpleGP can only evolve large and complex routing policies with good performance. All the routing policies evolved by NSGA-II are located on the left-top corner on all instances. This indicates that NSGA-II has the problem of premature convergence to small but poor routing policies. The three versions of the proposed TS-GPHH algorithm have better distributions than both SimpleGP and NSGA-II. It can be seen that TS-NSGA-II has better distribution than NSGA-II. However, its distribution is still not as good as the TS-GPHH approaches since there is still a limitation of TS-NSGA-II that it is hard to maintain a good tradeoff between exploration and exploitation in the second stage. TS-GPHH-n and TS-GPHH-s have a similar distribution and evolve smaller routing policies with similar performance with SimpleGP. This is because both approaches can inherit routing policies with good performance from generation to generation. To gain further understanding of the compared approaches, we plot the movement of the non-dominated routing policies every 10 generations for compared algorithms on Ugdb1. It is shown in Fig. 2. From Fig 2a, it can be seen that NSGA-II rarely evolve large routing policies during its evolution process. Since the size is much easier to be optimised than performance, all non-dominated routing policies have small sizes. The evolution process converges to small size at the beginning. Although TS-NSGA-II also converges to small

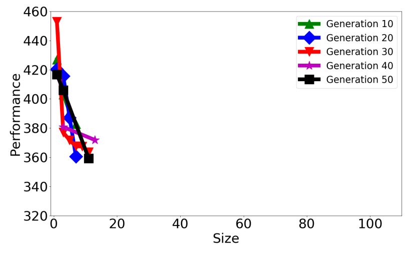


(a) Ugdb1

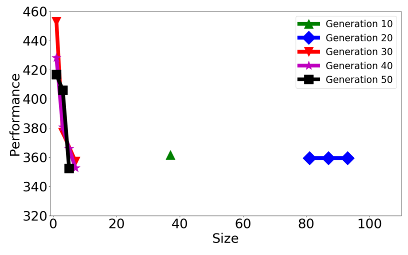


(b) Uval10D

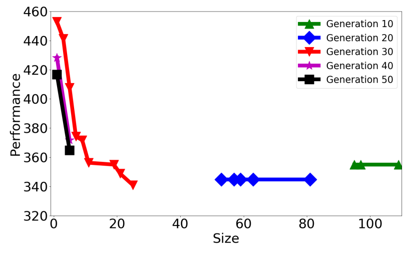
Fig. 1: The plot map of non-dominated routing policies from each independent run for all compared approaches.



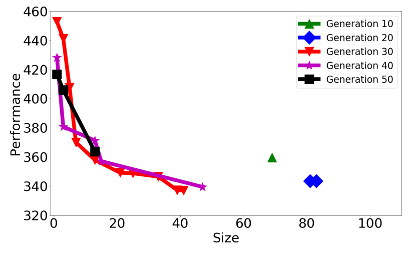
(a) NSGA-II



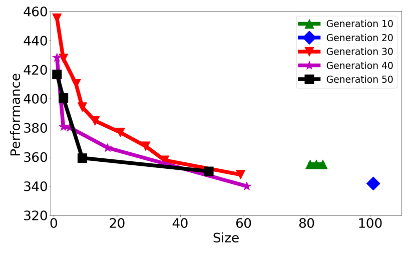
(b) TS-NSGA-II



(c) TS-GPHH-d



(d) TS-GPHH-n



(e) TS-GPHH-s

Fig. 2: The movement of Pareto front every 10 generations for all compared multi-objective approaches on Ugdb1 for a single run.

size, it still inherits some useful subtrees that can increase performance from stage one. However, TS-NSGA-II loses some important routing policies which can achieve better performance since the evolution process of the second stage of TS-NSGA-II is too greedy. One interesting observation is that all three TS-GPHH approaches make non-dominated routing policies diverse and cover more area on the map. This is as expected. Different elitism mechanisms lead to different inheritance relationships from the previous generation. It can be seen from Fig. 2c, Fig. 2d and Fig. 2e that the non-dominated routing policies lose performance which inherits from stage one gradually in stage two in TS-GPHH-d. This is improved in TS-GPHH-n since we removed duplicates from the elites. Many duplicates of small individuals make it harder to inherit good performance from the previous generation. We can see that TS-GPHH-s rarely lose performance in the whole evolution process. This is because it will inherit all routing policies with good performance which means that individuals with good performance will not be lost.

V. Conclusions and Future Work

This paper proposes a simple yet effective Two-Stage GPHH algorithm, and apply it with different elitism mechanisms, to evolve effective and interpretable routing policies for solving UCARP. The experimental results showed that with a more sophisticated control on the balance between the two objectives, the proposed TS-GPHH algorithm is able to evolve much smaller and thus more interpretable routing policies without losing the test performance. This demonstrates that our proposed approach can handle the challenge in the multi-objective optimisation that when the size is much easier to be optimised than the performance. In the future, we will consider improving elitism mechanisms to enhance the effectiveness of the evolved routing policies further. In addition, we will consider different dominance criteria that can better balance the tradeoff between performance and size.

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

[4] 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.

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

[6] 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.

[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] 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.

[13] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler, "Multiobjective genetic programming: Reducing bloat using spea2," in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, vol. 1. IEEE, 2001, pp. 536–543.

[14] 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.

[15] R. Poli, N. F. McPhee, and L. Vanneschi, "Elitism reduces bloat in genetic programming," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. Citeseer, 2008, pp. 1343–1344.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[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] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] C. Gathercole and P. Ross, "An adverse interaction between crossover and restricted tree depth in genetic programming," in *Proceedings of the 1st annual conference on genetic programming*. MIT Press, 1996, pp. 291–296.

[30] T. Soule and J. A. Foster, "Effects of code growth and parsimony pressure on populations in genetic programming," *Evolutionary computation*, vol. 6, no. 4, pp. 293–309, 1998.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.