Genetic Programming with Niching for Uncertain Capacitated Arc Routing Problem

Shaolin Wang, *Student Member, IEEE,* Yi Mei, *Senior Member, IEEE,* Mengjie Zhang, *Fellow, IEEE* and Xin Yao, *Fellow, IEEE*

***Abstract*—The uncertain capacitated arc routing problem is an important optimisation problem with many real-world applications. Genetic programming is considered a promising hyper-heuristic technique to automatically evolve routing policies that can make effective real-time decisions in an uncertain environment. Most existing research on genetic programming hyper-heuristic for the uncertain capacitated arc routing problem only focused on the test performance aspect. As a result, the routing policies evolved by genetic programming are usually too large and complex, and hard to comprehend. To evolve effective, smaller and simpler routing policies, this paper proposes a novel genetic programming approach, which simplifies the routing policies during the evolutionary process using a niching technique. The simplified routing policies are stored in an external archive. We also developed new elitism, parent selection and breeding schemes for generating offspring from the original population and the archive. The experimental results show that the newly proposed approach can achieve significantly better test performance than the current state-of-the-art genetic programming algorithms for uncertain capacitated arc routing problem. The evolved routing policies are smaller, and thus potentially more interpretable.**

***Index Terms*—Capacitated Arc Routing, Genetic Programming, Hyper-Heuristic, Stochastic Optimisation, Program Simplification.**

I. Introduction

CAPACITATED Arc Routing Problem (CARP) [1] is a classic combinatorial optimisation problem with a number of real-world applications, such as waste collection [2] and winter gritting [3]. The main goal of CARP is to serve a set of edges in a graph using a fleet of vehicles with the minimum cost. CARP is considered as an arc routing counterpart to the well-known Vehicle Routing Problem (VRP). It has been proved to be NP-hard [4] and received much research interest. Many approaches have been proposed for dealing with CARP [4]. However, most previous studies assume that the environment is static, where all the parameters (e.g. task demand, travel cost) are fixed and known in advance. This is usually not true in the real world, where the environment is often uncertain and dynamic. For example, in waste collection, the amount of waste to be collected is not known in advance and varies from one day to another.

Uncertain CARP (UCARP) [5]–[7] was proposed to better reflect the reality. A variety of uncertainties, such as the stochastic task presence and demand, stochastic edge deadheading cost and stochastic travel time, have been considered [6]. In addition to the NP-hardness inherited from CARP, the main challenge in UCARP is the *route failure* caused by the stochastic task demand. A route failure occurs when the remaining capacity of the vehicle becomes insufficient to serve the edge, because the actual demand of an edge that needs to be served exceeds the vehicle's expected demand. Therefore, the vehicle has to go back to the depot to refill and come back to finish serving the edge again. A route failure might lead to a high recourse cost.

Traditional solution optimisation approaches for dynamic routing problem, such as mathematical programming [1], genetic algorithm [7], particle swarm optimisation [8] and Markov decision processes [9], cannot effectively handle the route failure in UCARP. On one hand, the traditional solution optimisation approaches try to optimise a solution (i.e. a solution that performs reasonably well in all possible environments) beforehand. It is repaired by the recourse operator if a route failure occurs. However, the pre-planned solution is not flexible enough, and cannot be sufficiently changed by the recourse operator to cope with the dynamic nature of the route failure situation. In addition, it may induce a large extra recourse cost in some situations. On the other hand, some solution optimisation methods [9], [10] aim to re-optimise the remaining routes when a route failure occurs. Although this can potentially obtain better solutions, the re-optimisation process by running the solution optimisation algorithm is typically too slow to respond in real time.

The routing policy-based approaches [11] are promising techniques that can effectively handle the uncertain environment in UCARP. Unlike traditional solution optimisation approaches, these approaches do not have to pre-plan any

Manuscript received XXX; revised XXX and XXX; accepted XXX.

This work is supported by the Marsden Fund of New Zealand Government under r Contract VUW1509 and Contract VUW1614; in part by by the Research Institute of Trustworthy Autonomous Systems, the Guangdong Provincial Key Laboratory (Grant No. 2020B121201001); in part by tthe Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386); and in part by Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531); and in part by the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008). The work of Shaolin Wang was also supported by the Victoria University of Wellington PhD Scholarship. (*Corresponding author: Shaolin Wang.*)

Shaolin Wang, Yi Mei, and Mengjie Zhang are with the Evolutionary Computation Research Group, School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: shaolin.wang@ecs.vuw.ac.nz; yi.mei@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

Xin Yao is with the Research Institute of Trustworthy Autonomous Systems, Department of Computer Science and Technology, Southern University of Science and Technology (SUSTech), Shenzhen, China, and CERCIA, School of Computer Science, University of Birmingham, UK (e-mail: xiny@sustc.edu.cn).

Colour versions of one or more of the figures in this article are available online at XXX.

Digital Object Identifier XXX

solution. On the other hand, they use routing policies to guide a vehicle to determine which task to serve next as soon as it becomes idle [11]. A typical manually designed routing policy is the Path Scanning [12]. However, numerous factors, such as the objective(s) and the graph topology can affect the effectiveness of the routing policy [13]. The Genetic Programming Hyper-heuristic (GPHH) is considered as a promising technique that can automatically evolve effective heuristics for dynamic problems [14], such as scheduling problems [15]–[18] and resource allocation problems [19]. Recently, GPHH has also been successfully applied to UCARP to evolve effective routing policies automatically [20]–[24].

However, the routing policies evolved by GPHH are usually too large and complex, which usually leads to poor interpretability. Evolving small and simple routing policies (without losing effectiveness) can help end users understand the evolved routing policies better so that they can use them confidently and modify them when necessary. Intuitively, the size (number of nodes) of the routing policy is an essential factor that can affect the interpretability, and a larger (smaller) routing policy tends to be harder (easier) to interpret. Usually, the tree size of the routing policy evolved by GPHH is too large. The main reason is that GP tends to continuously increase the size of its individuals, which is known as *bloat* [25]–[27]. For bloat control, one can simply limit the tree size or depth during the GP evolution (e.g., [28], [29]) or design specific genetic operators to consider both tree size and effectiveness (e.g., [26], [30], [31]). However, these approaches cannot balance the two aspects well. Their performance could be biased by either tree size or effectiveness.

GP simplification is another way to reduce the tree size by removing redundant materials from GP trees. GP produces trees that typically contain a large number of redundant components (subtrees). It is desirable to remove as much of these redundant materials as possible without sacrificing the exploration ability of GP. Manual simplification on the final GP tree has been tried in the past. However, it is more preferred to simplify GP trees during the search process [32]. Wong and Zhang et al. [33]–[36] have explored the use of algebraic tree simplification with the hashing technique and applied it to regression and classification problems. Kinzett et al. [37] and Song et al. [38] have proposed numerical tree simplification methods. Both methods are based on the local effect of a subtree.

The above simplification methods have several limitations. First, they detect redundant subtrees based on genotypic information (e.g. tree structure) rather than phenotypic information (e.g. behaviour in decision making). Thus, they may fail to detect some implicit redundancies. Second, they need predefined parameters, i.e. simplification rules and threshold. The final test performance is highly sensitive to these parameters, and it is hard to set them properly. To the best of our knowledge, there is no existing study that implements the simplification based on the phenotype of GP trees. Also, there is no existing study that applies GP simplification to GPHH for UCARP to evolve both effective and simple routing policies.

To simplify a GP tree based on phenotypic behaviour, we aim to replace a large tree with another smaller tree with the same phenotypic behaviour. An intuitive approach is to group the individuals in the population based on their phenotypic behaviour (each group contains the individuals with the same behaviour), and simplify all the individuals in each group by replacing it with the smallest individual in that group. We call this approach *niching simplification*, since it is similar to the niching techniques [39], [40] that divides the individuals in the population into different groups/niches.

The overall goal of this paper is to propose a novel GPHH that simplifies the individuals during the evolutionary process based on their phenotypic behaviour. In this way, we can obtain both more effective and smaller routing policies, which can be potentially more interpretable and generalisable. To achieve this goal, we have the following research objectives:

1) To develop a new niching simplification scheme based on phenotypic behaviour. Specifically, the individuals with the same phenotypic behaviour (i.e., fitness in this study) are grouped into a niche. In each niche, all the individuals are replaced by the individual with the smallest size (so-called *representative* of the niche);
2) To compromise the loss of population diversity due to the niching simplification, we still keep the original population, and store the niche representatives in an external archive;
3) To develop a multi-source breeding mechanism to generate offspring from the original population and the representative archive, respectively;
4) To design niching-based elitism and parent selection schemes for the breeding from the representative archive;
5) To develop a new GPHH algorithm with Niching (GPHH-N) that incorporates all the above designed components.
6) To verify the effectiveness of GPHH-N, and analyse the routing policies evolved by GPHH-N.

Note that the goal of this paper is not to address the interpretability problems explicitly, although GPHH-N is expected to evolve smaller and simpler routing policies that are potentially more interpretable.

II. Background

For CARP, there have been various studies considering different aspects of uncertainties [20], [41]–[43], such as stochastic task demand, stochastic edge deadheading cost, stochastic task presence and stochastic edge existence. In this paper, we focus on the UCARP [5] that considers all four important uncertainties.

A. Uncertain Capacitated Arc Routing Problem

A UCARP instance can be represented by a connected undirected graph $G(V, E)$, where V and E indicate the sets of vertices and edges, respectively. The vertex $v_0 \in V$ denotes the depot. Each edge $e \in E$ has a positive random deadheading cost $\bar{\varsigma}(e)$. A fleet of vehicles with a given positive capacity Q are allocated to serve all the *tasks* $T \subseteq E$. Each task $t \in T$ has a positive random demand $\bar{d}(t)$, and a positive serving cost $sc(t)$. The goal is to minimise the total cost of serving all the

tasks in T. Several constraints have to be satisfied. First, all the routes must start and end at $v_0 \in V$. Second, each task $t \in T$ must be served exactly once, but it can be traversed multiple times. Third, the total demand served by each route (an edge sequence starts and end at the depot) cannot exceed the capacity of the vehicle.

In UCARP, there are a number of random variables, each of which can have different samples. Accordingly, a UCARP instance may contain different samples. In a UCARP instance *sample*, each random variable, i.e., the task demand $\bar{d}(t)$ and deadheading cost $\bar{\varsigma}(e)$, takes a realised value. However, the realised demand of a task is unknown until it has been served, and the realised deadheading cost of an edge is unknown in advance and is revealed during the traversal over the edge. The nature of the uncertainties of the environment will lead to two kinds of failures during the serving process.

- *Route failure*: the actual demand of a task exceeds the vehicle's remaining capacity.
- *Edge failure*: an edge in the route becomes inaccessible (its deadheading cost becomes infinity).

When a route failure occurs, the vehicle will go back to the depot to refill and return to the failed task to complete the remaining service. When an edge failure occurs, the solution can be repaired by finding a shortest detour under the current situation using pathfinding algorithms (e.g., Dijkstra's algorithm [44]).

A solution to a UCARP instance is represented as $S = (S.M, S.N)$. $S.M = \{S.M^{(1)}, \ldots, S.M^{(j)}\}$ is a set of vertex sequences, where j is the number of vertex sequences (routes). $S.M^{(k)} = (S.m_1^{(k)}, \ldots, S.m_{L_k}^{(k)})$ stands for the k^{th} route, where L_k represents the number of vertices in the k^{th} route. $S.N = \{S.N^{(1)}, \ldots, S.N^{(j)}\}$ is a set of continuous vectors, and $S.N^{(k)} = (S.n_1^{(k)}, \ldots, S.n_{L_k-1}^{(k)})$ ($S.n_i^{(k)} \in [0,1]$) is the fraction of demand served along the route $S.M^{(k)}$. The problem can be formulated as follows.

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

$$s.t. \ \sum_{i=1}^{L_k-1} d_\xi\left(S_\xi.m_i^{(k)}, S_\xi.m_{i+1}^{(k)}\right) \times S_\xi.n_i^{(k)} \leq Q, \forall k = 1, \ldots, j, \tag{2}$$

$$\left(S_\xi.m_i^{(k)}, S_\xi.m_{i+1}^{(k)}\right) \in E, \tag{3}$$

$$\left(S_\xi.n_i^{(k)}\right) \in [0,1], \tag{4}$$

$$S_\xi.m_1^{(k)} = S_\xi.m_{L_k}^{(k)} = v_0, \ \ \forall k = 1, 2, \ldots, j, \tag{5}$$

$$\sum_{k=1}^{j} \sum_{i=1}^{L_k-1} S_\xi.n_i^{(k)} \times S_\xi.z_i^{(k)}(e) = 1, \ \forall e : d_\xi(e) > 0, \tag{6}$$

$$\sum_{k=1}^{j} \sum_{i=1}^{L_k-1} S_\xi.n_i^{(k)} \times (1 - S_\xi.z_i^{(k)}(e)) = 0, \ \forall e : d_\xi(e) = 0, \tag{7}$$

where Eq. (1) is the objective function. It is used to minimise the expected total cost $C(S_\xi)$ of the solution S_ξ over all samples $\xi \in \Xi$. The total cost of a solution on one sample ξ is calculated by Eq. (8), where $\varsigma_\xi(u, v)$ is the realised deadheading cost between the node u to v in ξ. For any sample ξ, a feasible solution S_ξ is generated by a routing policy which gradually builds the solution on-the-fly. Eq. (2) is the capacity constraint, and d_ξ refers to the realised task demand in ξ. Eqs. (3) and (4) are the domain constraints of $S_\xi.M$ and $S_\xi.N$. Eq. (5) indicates that all the routes start and end at the depot. Eq. (6) means that each task is served exactly once (the total demand fraction served by all vehicles is 1). $S_\xi.z_i^{(k)}(e)$ equals 1 if $\left(S_\xi.m_i^{(k)}, S_\xi.m_{i+1}^{(k)}\right) = e$, and 0 otherwise. $S_\xi.z_i^{(k)}(e)$ indicates whether the edge is a task or not. Eq. (7) ensures that any non-required edge is not served at all, i.e., its service fraction is zero everywhere in the solution.

$$C(S_\xi) = \sum_{k=1}^{j} \sum_{i=1}^{L_k-1} \left(\varsigma_\xi(S_\xi.m_i^{(k)}, S_\xi.m_{i+1}^{(k)})\right) + \sum_{t \in T}(sc(t) - \varsigma_\xi(t)) \tag{8}$$

The static CARP is a special case of UCARP, where all the variables are known in advance. Route failure is the main extra challenge of UCARP over the static CARP, and it can lead to large extra recourse cost.

B. Related Work

1) Approaches for Capacitated Arc Routing Problem: Most existing approaches focus on the static CARP. Golden and Wong [1] developed an integer linear programming model and solved CARP using the branch-and-cut algorithm. Belenguer and Benavent [45] applied a cutting plane algorithm to CARP. It is guaranteed that the obtained solutions are optimal by using *exact* approaches. However, the approach is limited to only a small number of instances due to the NP-hardness of CARP. These approaches can become time consuming as the problem becomes large.

To obtain reasonably good solutions in a limited time budget and for larger problem instances, constructive heuristics are applied to CARP [1]. Constructive heuristics generate approximated solutions from scratch. Although these approaches cannot guarantee that the obtained solutions are optimal, they are much cheaper than exact approaches. At the same time, they can usually provide good enough solutions very quickly. The path scanning heuristic, augment-merge heuristic, and Ulusoy's split heuristic are commonly used constructive heuristics for CARP [12].

To make an improvement, meta-heuristic approaches have been used in recent years. A typical meta-heuristic algorithm starts from one or more solutions, and iteratively improves them. Thus, meta-heuristic algorithms can usually obtain promising solutions in a reasonable time budget. Typically, solutions from meta-heuristic algorithms can be no worse than constructive heuristics since they can take the solution from constructive heuristics as their initial solutions. In addition, they are usually less time consuming than the exact approaches. Various meta-heuristic algorithms have been proposed for CARP, such as tabu search [46], [47], genetic algorithms [48], memetic algorithms [12], [49], [50], and ant colony optimisation [51]–[53]. Mei et al. [54] and Tang et al. [55] handle large-scale CARP with decomposition approaches.

To enhance the problem solving for CARP, Feng et al. [56] utilised Meme learning and selection to transfer useful structures or latent patterns that are captured from previous experiences of problem-solving. The captured knowledge was then applied to enhance future evolutionary search. Comparing with existing works, the work in [56] can transfer and reuse

knowledge across problems of different size, structure, or representation. To make an improvement, Feng et al. [57] proposed a new memetic computing paradigm which contains four culture-inspired operators, i.e. Learning, Selection, Variation and Imitation. Comprehensive studies on the widely studied NP-hard Capacitated Vehicle Routing Problem (CVRP) and CARP domains have been made. The experimental results showed the benefits of the proposed faster evolutionary search approach. Rather than transferring knowledge from the same problem domain, Feng et al. [58] proposed an evolutionary memetic computing paradigm that is able to learn and evolve knowledge memes that traverse two different but related problem domains, i.e. CVRP and CARP, to speed up the search efficiency. Experimental results showed that evolutionary optimization can benefit from different but related problem domains. The appropriate choice of knowledge memes is an essential factor for enhancing the evolutionary search process.

2) Approaches for Uncertain CARP: There are many different types of uncertainties in real-world CARP. Liu et al. [6] give a comprehensive review on the uncertain factors in CARP, including stochastic task demand, stochastic edge deadheading cost, stochastic task presence and stochastic edge existence, and the corresponding approaches to address them.

Many approaches aim to find solutions that can handle all possible realisations of the random variables. Typically, the output of these approaches consists of two main components, a pre-planned solution and a recourse operator. Firstly, the optimisation algorithm produces a solution based on the prediction of the environment. Then, the solution is executed while possible failures, such as route failures, could be handled by the recourse operator. The performance of these approaches is highly affected by the accuracy of predicting the stochastic environment and the effectiveness of recourse operators.

Fleury et al. [41] proposed a Memetic Algorithm (MA) that integrated an accurate approximation of the expected total cost for CARP with stochastic task demand and concluded that MA could be easily adapted to handle the stochastic task demand and obtain better solutions than other simple techniques like keeping a slack in each vehicle. Babaee Tirkolaee et al. [59] developed a mix integer programming model to handle the demand uncertainty. They also proposed a hybrid meta-heuristic approach based on simulated annealing and a constructive heuristic to obtain good solutions for CARPSD. Wang et al. [60] adapted a new MA with an integrated fitness function and a large step-size local search operator to UCARP. The new MA can find good solutions for UCARP. The integrated fitness function is used to guide the search direction, and the large step-size local search operator is used to prevent MA being trapped in local optima. Wang et al. [61] focused on finding solutions that can perform well over a set of UCARP instances and proposed an Estimation of Distribution Algorithm with Stochastic Local Search (EDASLS). The new EDASLS is based on The Edge Histogram Based Sampling Algorithm (EHBSA) [62] and a novel Stochastic Local Search (SLS) which can effectively handle the uncertainties in UCARP. Recently, Tong et al. [63] proposed a generalised meta-heuristic framework for UCARP. In addition, Tong et al. [64] also proposed a hybrid local search framework which can handle the small dynamic changes in UCARP.

Some other approaches use Genetic Programming Hyper-Heuristics (GPHH) to evolve routing policies that construct solutions gradually in real-time. It does not maintain any pre-planned solution. Weise et al. [65] first proposed a GPHH for UCARP with a single vehicle and examined its performance. The results showed that routing policies evolved by GPHH could outperform manually designed routing policies. Liu et al. [20] 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 a single-vehicle to multi-vehicle version by Mei et al. [22]. Then the solution can be generated with multiple vehicles on the road simultaneously. MacLachlan et al. [21] proposed a novel task filtering method and an effective look-ahead terminal. Then, MacLachlan et al. [66] examined the advantage of vehicle collaboration in handling the uncertain environment. Recently, Ardeh et al. [67] applied transfer learning to speed up the training process. Liu et al. [68] proposed a new predictive-reactive approach with Genetic Programming and cooperative coevolution which evolved a task sequence and a recourse policy simultaneously. The existing GPHH approaches managed to obtain effective routing policies for UCARP. However, the interpretability of the evolved routing policies has been ignored.

3) GP Tree Size Reduction Approaches: A typical tree-based GPHH approach blindly generates GP trees by randomly swapping the subtrees of the parents in crossover and replacing a subtree with a randomly generated subtree in mutation. GP search usually leads to large and complex trees that are hard to interpret without careful control of the tree size. Also, a large and complex tree tends to be less generalisable [69]. Therefore, it is essential to reduce the tree size and complexity without deteriorating the effectiveness.

There have been various studies to reduce tree size during the GP evolutionary process. The most straightforward way is to limit the number of nodes of the tree [28], [29]. However, this strategy has two main limitations. First, it limits the search space and may weaken the exploration ability of GP, and thus lead to worse effectiveness. Second, it is nontrivial to predefine a reasonable limit for the number of nodes [25]. Another commonly used strategy is to penalise large trees during the selection, known as the *parsimony pressure* [70]. However, it is hard to set a proper coefficient of the penalty term without extensive trial and error [26]. Kinzett et al. [30] point out that it is difficult to obtain a small and effective solution in some scenarios. The solution could be biased by either tree size or fitness in GP. Luke et al. [31] review a set of bloat control methods in GP and compare them with the depth limiting method on different problems. The authors argue that linear parsimony pressure [71] performed the best in the cross-problem comparison, and the double tournament selection [72] performed the second best. However, the conclusions obtained in [31] are purely empirical.

GP tree simplification methods are an alternative for tree size reduction. Hooper et al. [73] utilised the expression simplification to simplify the trees. Their simplification method employed over 200 simplification rules to simplify an ex-

pression. To speed up the simplification process, Zhang and Wong et al. [33]–[36] proposed the use of algebraic tree simplification with the hashing techniques. However, these algebraic methods cannot detect all kind of redundancies [74]. It can easily simplify the expression $A - A$ to 0. However, $A + 10^{-100}$ cannot be simplified to A directly, although 10^{-100} does not play any significant role in the final output. In addition to the algebraic approach, Kinzett [37] and Song et al. [38] proposed numerical simplification methods. These methods are based on the local effect of a subtree. Kinzett et al. [37] removed a node or a subtree if its numerical contribution is smaller than a predefined threshold. Song et al. [38] replaced the parent node with its child node if the difference between their outputs is below a predefined threshold.

In summary, most GP simplification methods can only detect redundancies based on genotypic information (contribution of the genotypic subtrees) rather than phenotypic information (output of the whole GP tree). Thus, many redundancies will still exist. Also, they require manually selected simplification rules or predefined threshold parameter. The final test performance is highly sensitive to these parameters. It is hard to set these parameters properly. This paper attempts to deal with the above issues by proposing a novel GPHH with phenotypic simplification using niching.

III. The Proposed GPHH-N

A. Overall Framework

The overall framework of GPHH-N is shown in Fig. 1. Firstly, a population of routing policies (each represented as a GP tree) is initialised randomly. Then, the routing policies are evaluated at each generation based on their simulation performance (total cost). If the stopping criteria are not reached, the population undergoes the evolutionary process. In the evolutionary process, routing policies are first simplified by the niching simplification. Details of the niching simplification will be given in Section III-D. After that, the smallest tree in each niche is considered the niche representative and is stored in a representative archive. The niching elitism and niching tournament selection are then applied to both the original population and representative archive to produce offspring. The niching elitism selects diverse trees for the next generation. It will be described in Section III-E. The newly developed niching tournament selection can ensure that diverse parents are selected from the simplified population. Details will be discussed in Section III-F. The multi-source breeding process selects parents from both the original and simplified populations so that the simplified blocks are not completely lost. We will describe its details in Section III-G. The relationship between the population and representative archive and how the newly proposed operations update them are shown in Fig. 2.

B. Individual Representation

In this paper, each routing policy is represented as a priority function. The priority function is a combination of the state features, such as the cost from the current location to the candidate task (CFH) and the cost from the candidate task

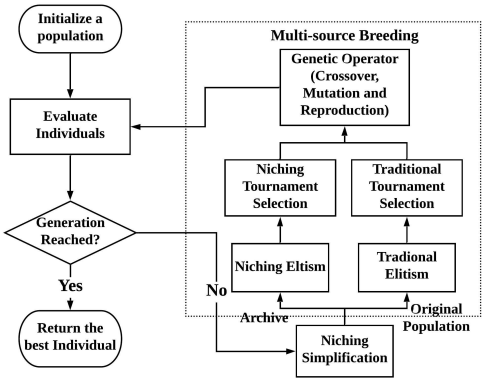


Fig. 1. The overall Framework of GPHH-N

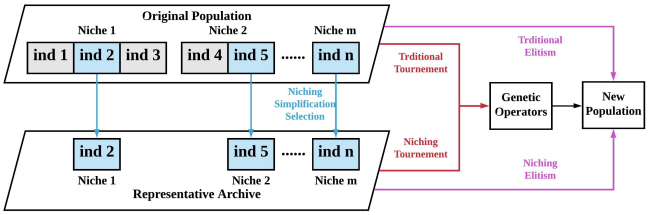


Fig. 2. The illustration of the offspring generation in GPHH-N.

to the depot (CTD). For example, if the priority function is "CFH + CTD", as shown in Fig. 3, the priority function tends to select tasks that are closer to the current location and also closer to the depot. The routing policy works as a decision-maker during the solution construction process. Typically, each vehicle can only serve one task at a time. The routing policy will be applied to each unserved task to determine each unserved task's priority once a vehicle completes its current task and is ready to serve the next task. The task with the best priority value will be served next. The decision process finishes when all the tasks have been served and the routes returned as the solution constructed by the routing policy.

C. Fitness Evaluation

Given a routing policy rp, the fitness is defined based on the average quality (i.e. total cost) of the routes that it generates. Specifically,

$$\text{fit}(rp) = \frac{1}{|\mathcal{S}'|} \sum_{s \in \mathcal{S}'} tc(rp, s), \tag{9}$$

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 instance sample s. The solution is constructed based on a simulation process that is commonly used in UCARP literature [23], [29]. Algorithm 1 shows the fitness evaluation in GPHH-N.

One can see that each routing policy is applied as a priority (heuristic) function. When a vehicle is ready (at a decision-making point), a routing policy is applied to determine which candidate task to serve next. The quality of each routing policy can be evaluated by applying it to a set of training samples

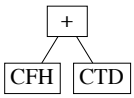


Fig. 3. An example of GP tree.

Algorithm 1: The fitness evaluation in GPHH-N.

Input: Training set $\mathcal{S}_{train}$, An unevaluated population *pop*
Output: An evaluated population *pop*

1 Randomly sample a training subset $\mathcal{S}' \subseteq \mathcal{S}_{train}$;
`// Evaluate pop using` $\mathcal{S}'$
2 **for** *each routing policy* $rp \in pop$ **do**
`// Evaluate` rp
3 **for** *each training sample* $s \in \mathcal{S}'$ **do**
4 **while** *tasks in queue is not empty* **do**
5 **if** *vehicle is ready* **then**
6 Select which task to serve using rp;
7 **end**
8 **end**
9 Get the total cost of the generated solution, i.e. $tc(rp, s)$;
10 **end**
11 Calculate $fit(rp)$ using Eq. (9);
12 **end**
13 **return** evaluated population *pop*;

Algorithm 2: The niching simplification.

Input: Original Population *pop*
Output: Original Population *pop*, Simplified Population pop', Representative Archive $archive$

1 Initialise $archive = \emptyset$;
2 **for** *each routing policy* $rp \in pop$ **do**
3 $inNiche(rp) = false$;
4 **end**
5 **for** *each routing policy* $rp \in pop$ **do**
6 **if** $inNiche(rp) == true$ **then**
7 **continue ;**
8 **end**
9 Create a new $niche = \{rp\}$;
10 Set $inNiche(rp) = true$;
11 **for** *each routing policy* $rp' \in pop$ **do**
12 **if** $fit(rp')$ *equals to* $fit(rp)$ **then**
13 Add rp' to $niche$;
14 Set $inNiche(rp') = true$;
15 **end**
16 **end**
17 Find out the individual with smallest tree size ind_s in $niche$;
18 Add ind_s to $archive$;
19 Replace all other individuals in the niche with ind_s;
20 **end**
21 **return** *pop*, pop' and $archive$;

$\mathcal{S}'$. A solution can be constructed using the routing policy on a training sample $s \in \mathcal{S}'$.

We randomly re-sample a subset of training samples for the fitness evaluation (line 1). Such sample rotation has been commonly used in other studies [20], [75] to improve the generalisation of the evolved heuristics.

D. Niching Simplification

The niching simplification method is shown in Algorithm 2. It firstly places the trees with the same phenotype into the same niche. Then, each niche identifies the tree with the smallest tree size (number of nodes in the tree) in the niche as the niche representative, and replaces all the other trees with this representative. If there are multiple trees with the smallest tree size, the first identified tree is chosen as the representative. In this way, each niche is represented by a tree with the smallest tree size under the same phenotype. Fig. 4 shows two trees with the same phenotype but different tree sizes. Tree A can be hardly simplified to Tree B by the algebraic or the numerical simplification method. However, Tree A can be easily simplified to Tree B by the niching simplification approach.

Note that fitness is used as a high-level phenotypic behaviour of a GP individual. This is a specific design for GPHH for UCARP, since our preliminary study has shown that many routing policies have different tree structures but the same fitness value. Although we can use other more precise phenotypic characterisations such as the structure of the obtained routes for a given instance, or the phenotypic vector [76], the fitness-based characterisation is shown to be effective enough empirically, and is very efficient to compute.

E. Niching Elitism

Elitism is used in GP to avoid losing the previously found best individuals. In the traditional elitism scheme, we directly copy the best individuals (e.g. the top 10 individuals in terms of fitness) from the population to the next generation. However, this will lead to a significant loss of diversity after the niching simplification since all the best individuals might come from the same niche. We propose a niching elitism scheme to address this issue, which selects the elitists only from the representative archive to maintain diversity. Firstly, the archive is sorted based on the fitness value. Then, top individuals are returned, and they can survive to the next generation.

F. Niching Tournament Selection

In GP, the parents are typically selected by tournament selection. In GPHH-N, each niche can contain many individuals with identical (good) fitness, and traditional tournament selection tends to select identical parents. To avoid such a loss of diversity, we modified the tournament selection operator. The new niching tournament selection selects only from the representative archive so that the parents for crossover are more likely to be different. We set the probability of selecting a representative proportional to its niche size using the following equation,

$$P(rep_i) = \frac{H_i^\alpha}{\sum_{i=1}^{\mathcal{N}} H_i^\alpha}, \alpha \in [0, 1] \tag{10}$$

where H_i stands for the number of individuals in $niche_i$, $\mathcal{N}$ refers to the total number of niches. α $(0 \leq \alpha \leq 1)$ is a parameter to control the balance between exploration and exploitation. When α equals 1, the niching tournament selection becomes the traditional tournament selection, which selects parents directly from the simplified population. On the other hand, when α equals 0, each representative will have the same probability of selection regardless of the number of individuals in its niche. A parameter sensitivity analysis on α will be given in Section IV-C1.

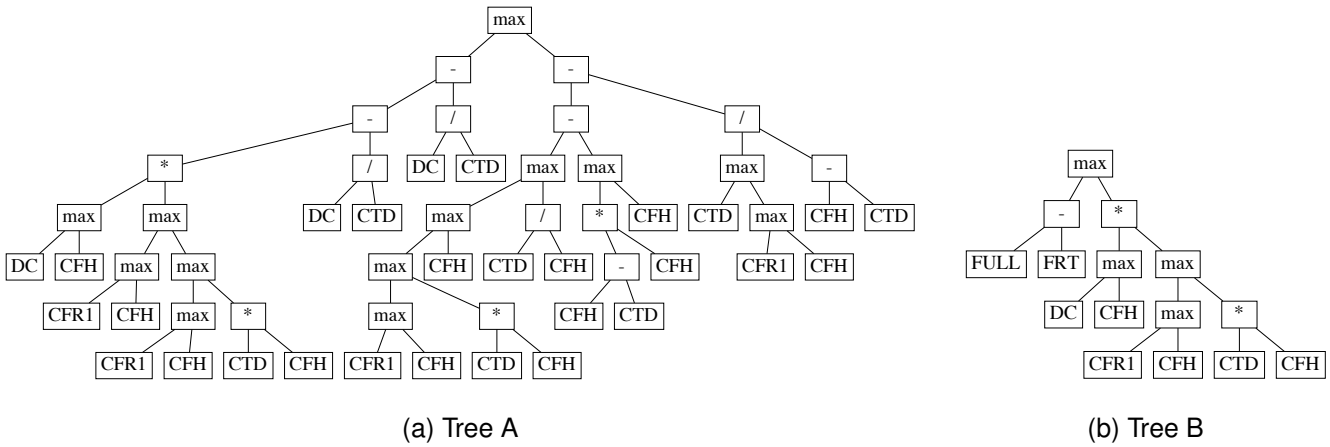


Fig. 4. Two trees with the same fitness but different tree sizes.

G. Multi-source Breeding

The niching simplification process may lead to the loss of the potential useful building blocks in the large trees. To compensate for such loss, we design a multi-source breeding scheme. The multi-source breeding produces offspring from two sources, i.e. the original population and the representative archive. Each source is used to produce half of the offspring population. The pseudocode of the multi-source breeding is shown in Algorithm 3. We breed offspring from two sources so that each source will produce half of the population in the next generation.

Algorithm 3: The multi-source breeding

Input: Original population pop, Representative archive $archive$, population size S
Output: New population $newpop$

$traditional\ elitism(pop)$;
while $size(newpop) < S/2$ **do**
  Select parents from pop using $traditional\ tournament\ selection$;
  Generate an offspring by applying (crossover/mutation/reproduction operators);
  Add the offspring to $newpop$;
end
$niching\ elitism(archive)$;
while $size(newpop) < S$ **do**
  Select parents from $archive$ using $niching\ tournament\ selection$;
  Generate an offspring by applying (crossover/mutation/reproduction operators);
  Add the offspring to $newpop$;
end
return $newpop$;

IV. Experimental Studies

To verify the effectiveness of the proposed GPHH-N, we compare it with the baseline GPHH (which evolves routing policies using GPHH without any simplification methods) [20], GPHH with the algebraic simplification (GPHH-A) [33]. Bloat control approaches are most commonly used to consider both test performance and tree size in GP. In this case, we compare our GPHH-N with three representative bloat control approaches, i.e. Tarpeian, linear parametric parsimony pressure (LPPP), double tournament (DT) in the literature [31]. The comparisons are conducted within the scope of GP approaches. First, the purpose of the experiments is to verify whether our new GP algorithm can obtain both effective and small routing policies. The tree size can only be compared with other GP approaches. Second, the previous study [66] has already shown that GPHH is a competitive approach for UCARP, which is no worse than the state-of-the-art non-GP approaches such as EDASLS [61].

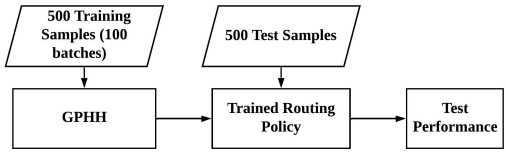


Fig. 5. The training and test phases of a GP Run on a UCARP Instance.

For each UCARP instance, we randomly generate 500 training samples and 500 test samples by randomly sampling the stochastic task demands and deadheading costs. The samples share the same graph topology of the UCARP instance but different realised values for the stochastic demands and deadheading costs.

A GP run on a UCARP instance contains training and test phases. In the training phase, routing policies are trained on 500 training samples. These samples are split into 100 batches, each containing 5 samples ($\mathcal{S}'$ in Eq. 9). A different batch is used for the training in each of the 100 generations. The best routing policy in the final population is returned as the trained policy. In the test phase, the trained routing policy is tested on the 500 test samples. The test performance of a routing policy on the UCARP instance is defined as the average total cost over the 500 test samples, calculated as follows.

$$\text{TestPerf}(rp) = \frac{1}{|\mathcal{S}_{\text{test}}|} \sum_{s \in \mathcal{S}_{\text{test}}} tc(rp, s), \tag{11}$$

where $\mathcal{S}_{\text{test}}$ is the test set, and $|\mathcal{S}_{\text{test}}| = 500$ is the size of the test set. Fig. 5 shows the training and test phases of one GP run on one UCARP instance.

TABLE I
THE TERMINAL SET IN THE EXPERIMENTS.

Terminal	Description
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	cost from the candidate to its closest remaining task
CTD	cost from the depot to the candidate task
CR	cost from the depot to the current location
DEM	expected demand of the candidate task
DEM1	demand of the closest unserved task to the candidate task
DC	deadheading cost of 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 alternative route
SC	cost of serving the candidate task
ERC	a random constant value

TABLE II
THE PARAMETER SETTING FOR THE COMPARED APPROACHES.

Parameter	Value	Parameter	Value
Population size	1000	Crossover rate	80%
Generations	100	Mutation rate	15%
Tournament size	7	Reproduction rate	5%
Maximal depth	8	Training sample size ($\|\mathcal{S}'\|$)	5 each generation
Elitism size	10	Test sample size	500

A. Dataset

In this paper, we use the *Ugdb* and *Uval* datasets, which are commonly used in UCARP literature [20]–[23]. They are extended from *gdb* and *val* which are well known static CARP datasets. The instances in the *gdb* dataset are mostly small. They contain at most 55 tasks that need to be served by at most 5 vehicles. The *val* dataset contains instances with the number of tasks ranging from 34 to 97 by at most 10 vehicles.

For each UCARP instance, the task demand $\bar{d}(t)$ and traversal cost $\bar{\varsigma}(e)$ are random variables. They are transformed from the original task demand $d(t)$ and traversal cost $\varsigma(e)$ from the static CARP instance. The random variables are assumed to follow the truncated normal distribution [20]–[22].

$$\bar{d}(t) \sim \mathcal{N}(d(t), \frac{d(t)}{5}), \bar{\varsigma}(e) \sim \mathcal{N}(\varsigma(e), \frac{\varsigma}{5}). \quad (12)$$

Any negative sampled task demand is set to 0, and any negative sampled traversal cost is set to ∞, which means that the arc becomes inaccessible.

B. Parameter Setting

The function set is set to $\{+, -, \times, /, \min, \max\}$, where the "/" operator returns 1 if divided by 0. The terminal set is shown in Table I. Table II shows the parameter settings of the GP algorithms compared in the experiments. The initialisation method of the population for all compared algorithms is Ramped Half-and-Half. All these parameter settings are commonly used in the GPHH literature [20], [22], [66], [77]–[79].

For the compared bloat control methods, the parameters are set according to the literature [31]. The simplification rules for GPHH-A are set according to the literature [33].

TABLE III
THE WIN-DRAW-LOSE TABLE FOR THE PAIRWISE COMPARISONS BETWEEN DIFFERENT α VALUES IN TERMS OF **TEST PERFORMANCE**.

Approach	GPHH-N-1.0	GPHH-N-0.5	GPHH-N-0.0
GPHH-N-1.0		0-29-28	1-20-36
GPHH-N-0.5	28-29-0		2-50-5
GPHH-N-0.0	36-20-1	5-50-2	

TABLE IV
THE WIN-DRAW-LOSE TABLE FOR THE PAIRWISE COMPARISONS BETWEEN DIFFERENT α VALUES IN TERMS OF **TREE SIZE**.

Approach	GPHH-N-1.0	GPHH-N-0.5	GPHH-N-0.0
GPHH-N-1.0		57-0-0	57-0-0
GPHH-N-0.5	0-0-57		55-0-2
GPHH-N-0.0	0-0-57	2-0-55	

We use Evolutionary Computation Java (ECJ) package [80] to implement all the algorithms. For each UCARP instance, each compared algorithm is run 30 times independently (each run trains a routing policy on the 500 training samples, and then test it on the 500 test samples).

C. Results and Discussions

We compared the algorithms using the Wilcoxon rank sum test with the significance level of 0.05. In the tables, "+", "-" or "=" next to each compared algorithm indicates that the compared algorithm performed statistically significantly better than, worse than, or comparable to GPHH-N.

1) Parameter Sensitive analysis on α: The parameter α is important in balancing exploration and exploitation in the niching tournament selection. To set the α value, we compared the GPHH-N with 3 different α values, i.e. 0, 0.5 and 1.

Tables III and IV show the results of the pairwise comparison among GPHH-N-0.0, GPHH-N-0.5 and GPHH-N-1.0. Each entry is represented in the Win-Draw-Lose format. Win (Lose) indicates the number of instances where the row approach performs significantly better (worse) than the column approach. Draw indicates the number of instances where the two approaches show no significant difference. Table III shows the results in terms of the test performance, and Table IV shows the results in terms of the tree size. From Table III, we can see that both GPHH-N-0.0 and GPHH-N-0.5 achieved the best test performance among the compared algorithms. From Table IV, GPHH-N-0.5 achieved significantly smaller tree size than GPHH-N-0.0. Overall, GPHH-N-0.5 is the best in terms of the test performance and the tree size.

Fig. 6 shows the scatter plot of GPHH-N with different α values on a representative instance *Uval2B*. Each shape represents the mean tree size and test performance of the 30 independent runs. We can see that GPHH-N-0.5 and GPHH-N-0.0 achieved much better test performance than GPHH-N-1.0. In addition, GPHH-N-0.5 obtained a much smaller tree size than GPHH-N-0.0. Therefore, $\alpha = 0.5$ is used in the subsequent experiments.

2) Test Performance: Tables V and VI show the mean and standard deviation for the test performance over the 30 independent runs on *Ugdb* and *Uval* instances.

TABLE V
THE MEAN AND STANDARD DEVIATION FOR **TEST PERFORMANCE** OF COMPARED APPROACHES ON THE **UGDB** INSTANCES OF 30 RUNS.

Instance	GPHH	GPHH-A	Tarpeian	DT	LPPP	GPHH-N
Ugdb1	351.25(14.66)(-)	352.1(12.22)(-)	360.34(15.85)(-)	359.03(24.37)(-)	360.89(8.73)(-)	344.12(6.23)
Ugdb2	367.73(4.77)(=)	369.51(12.19)(=)	374.35(9.26)(-)	369.88(6.38)(-)	383.72(10.26)(-)	367.49(13.31)
Ugdb3	307.03(2.87)(=)	306.35(3.1)(=)	308.69(2.55)(=)	307.03(4.3)(=)	312.42(5.27)(-)	307.63(3.64)
Ugdb4	324.9(6.42)(=)	321.95(3.08)(+)	325.91(3.15)(-)	322.87(4.33)(=)	329.18(10.06)(-)	323.43(4.13)
Ugdb5	422.17(6.09)(=)	426.57(15.3)(=)	429.59(8.93)(-)	425.49(9.32)(-)	450.11(14.66)(-)	422.22(13.32)
Ugdb6	344.3(8.37)(-)	344.9(9.11)(-)	359.26(7.17)(-)	342.51(6.46)(-)	362.24(0.0)(-)	337.6(5.27)
Ugdb7	353.24(4.41)(-)	352.58(3.67)(=)	359.59(0.68)(-)	354.95(4.05)(-)	359.85(0.0)(-)	351.68(4.68)
Ugdb8	430.79(8.16)(-)	436.28(37.62)(-)	439.64(11.37)(-)	427.71(7.06)(=)	451.64(13.39)(-)	425.96(5.95)
Ugdb9	389.02(9.86)(-)	387.8(9.17)(-)	397.22(10.41)(-)	392.48(14.55)(-)	408.01(13.95)(-)	380.47(10.02)
Ugdb10	293.0(7.22)(=)	293.75(7.29)(=)	298.22(5.56)(-)	293.79(7.58)(=)	298.7(2.09)(-)	291.47(7.01)
Ugdb11	433.32(8.52)(-)	435.88(6.31)(-)	446.76(7.19)(-)	437.03(6.03)(-)	446.48(4.31)(-)	429.82(11.32)
Ugdb12	604.8(17.36)(=)	606.66(15.48)(=)	618.57(13.27)(-)	604.82(18.0)(=)	622.06(15.8)(-)	604.19(13.94)
Ugdb13	577.11(8.53)(=)	580.12(9.02)(=)	586.62(8.97)(-)	578.01(6.93)(=)	599.42(15.69)(-)	574.93(7.15)
Ugdb14	107.02(1.31)(=)	110.24(12.68)(=)	108.25(1.59)(-)	107.75(2.58)(=)	117.81(2.88)(-)	106.8(1.27)
Ugdb15	58.34(0.81)(=)	58.26(0.19)(-)	58.26(0.24)(-)	58.27(0.41)(=)	62.01(0.0)(-)	58.11(0.09)
Ugdb16	134.52(0.55)(=)	134.64(0.09)(=)	134.51(0.07)(+)	134.61(0.11)(=)	134.47(0.0)(+)	134.91(1.92)
Ugdb17	91.47(1.9)(-)	91.08(0.12)(+)	91.23(0.09)(=)	91.18(0.46)(+)	93.82(0.48)(-)	91.29(0.35)
Ugdb18	167.49(1.77)(-)	167.62(3.06)(-)	168.81(2.19)(-)	168.18(5.61)(-)	180.78(5.52)(-)	166.06(0.92)
Ugdb19	63.29(1.61)(=)	64.16(1.47)(=)	63.95(1.61)(=)	63.66(1.56)(=)	67.81(1.07)(-)	63.63(1.41)
Ugdb20	127.09(1.57)(=)	127.19(1.53)(=)	128.85(2.05)(-)	128.0(4.57)(=)	137.31(0.0)(-)	126.67(2.16)
Ugdb21	164.74(2.27)(-)	167.54(17.58)(-)	166.29(2.96)(-)	165.35(2.82)(-)	181.52(5.49)(-)	163.25(1.58)
Ugdb22	210.11(2.22)(-)	209.26(1.38)(=)	211.01(3.61)(-)	209.28(1.69)(=)	221.46(2.44)(-)	209.09(1.94)
Ugdb23	250.62(2.98)(-)	251.07(4.16)(-)	251.66(2.49)(-)	249.52(1.86)(-)	258.34(6.08)(-)	247.73(2.13)
Average	285.80	286.76	290.76	286.58	297.39	283.85

TABLE VI
THE MEAN AND STANDARD DEVIATION FOR **TEST PERFORMANCE** OF COMPARED APPROACHES ON THE **UVAL** INSTANCES OF 30 RUNS.

Instance	GPHH	GPHH-A	Tarpeian	DT	LPPP	GPHH-N
Uval1A	175.43(2.73)(=)	176.47(3.33)(-)	180.75(2.32)(-)	176.51(6.15)(=)	188.73(6.65)(-)	174.37(1.71)
Uval1B	184.2(2.73)(=)	183.87(1.46)(=)	185.65(2.44)(-)	183.74(1.3)(=)	190.04(3.09)(-)	183.69(1.29)
Uval1C	311.93(10.22)(-)	313.69(10.77)(-)	318.29(5.53)(-)	314.01(9.08)(-)	328.39(5.82)(-)	306.5(12.67)
Uval2A	228.68(1.7)(=)	229.37(3.36)(=)	230.97(3.46)(=)	229.06(2.77)(=)	237.65(4.18)(-)	229.25(2.48)
Uval2B	276.49(4.06)(-)	277.95(4.12)(-)	279.74(5.18)(-)	278.63(3.25)(-)	293.33(15.04)(-)	274.2(3.36)
Uval2C	593.34(23.87)(-)	592.11(16.74)(-)	618.37(26.69)(-)	593.6(22.97)(-)	615.61(14.72)(-)	585.09(32.83)
Uval3A	82.18(1.47)(=)	82.44(1.32)(=)	82.36(0.91)(-)	83.63(4.19)(=)	88.37(2.15)(-)	81.97(0.66)
Uval3B	96.04(2.22)(-)	97.13(2.39)(-)	100.0(3.71)(-)	95.91(1.7)(-)	101.36(0.67)(-)	94.09(1.45)
Uval3C	176.45(7.56)(-)	176.18(5.65)(-)	177.49(7.76)(-)	175.66(5.63)(-)	191.54(14.79)(-)	171.31(4.22)
Uval4A	420.29(9.25)(-)	419.32(6.39)(-)	426.99(18.41)(-)	418.89(6.68)(-)	430.38(9.0)(-)	415.05(3.25)
Uval4B	440.85(5.92)(-)	440.07(5.77)(=)	449.25(12.83)(-)	441.81(7.28)(-)	452.09(10.8)(-)	437.64(4.84)
Uval4C	490.73(12.69)(-)	487.45(10.77)(-)	496.52(11.98)(-)	488.95(12.07)(-)	502.05(11.07)(-)	481.42(7.47)
Uval4D	699.52(32.49)(-)	699.17(41.57)(-)	723.26(32.38)(-)	694.53(30.51)(-)	726.66(39.43)(-)	679.62(24.65)
Uval5A	440.36(3.98)(-)	441.58(4.16)(-)	444.51(4.6)(-)	439.69(5.3)(=)	450.26(4.75)(-)	437.46(4.75)
Uval5B	469.81(5.51)(-)	469.55(5.51)(-)	477.66(12.84)(-)	473.99(20.39)(-)	480.97(11.09)(-)	465.81(4.25)
Uval5C	513.04(5.46)(-)	514.82(8.29)(-)	518.32(7.1)(-)	514.65(7.88)(-)	531.0(12.04)(-)	508.33(4.32)
Uval5D	723.04(14.21)(=)	725.36(27.47)(=)	750.36(24.91)(-)	724.44(16.5)(=)	749.13(25.04)(-)	720.0(14.98)
Uval6A	229.0(2.36)(=)	230.67(10.98)(=)	229.39(2.19)(-)	228.69(2.95)(=)	232.31(5.11)(-)	228.36(2.89)
Uval6B	257.32(3.71)(=)	257.53(4.5)(=)	259.23(3.15)(-)	256.1(3.5)(=)	266.56(9.88)(-)	255.71(4.6)
Uval6C	400.99(11.4)(-)	403.55(12.32)(-)	422.21(22.31)(-)	402.19(11.42)(-)	428.03(23.79)(-)	395.1(8.26)
Uval7A	289.74(15.08)(=)	294.17(20.28)(-)	289.73(5.6)(-)	288.03(9.94)(=)	298.57(6.06)(-)	287.14(10.39)
Uval7B	293.51(8.59)(=)	298.33(20.17)(-)	297.86(8.31)(-)	293.96(6.33)(-)	308.95(9.0)(-)	289.62(5.3)
Uval7C	405.06(6.32)(=)	404.95(8.67)(=)	415.7(18.46)(-)	405.42(8.09)(=)	414.11(11.49)(-)	402.77(5.21)
Uval8A	398.75(8.67)(-)	400.7(18.68)(-)	398.24(2.34)(-)	397.08(1.84)(-)	404.15(4.65)(-)	396.17(4.7)
Uval8B	426.22(6.28)(-)	425.1(5.71)(-)	432.02(10.94)(-)	425.21(6.16)(-)	437.98(6.99)(-)	421.27(4.98)
Uval8C	664.62(19.93)(-)	662.56(18.46)(-)	682.25(15.4)(-)	657.38(14.87)(=)	688.07(15.69)(-)	651.12(18.18)
Uval9A	333.75(3.75)(-)	335.02(6.79)(-)	336.69(3.47)(-)	333.52(2.04)(-)	342.35(6.31)(-)	330.88(2.36)
Uval9B	348.89(4.77)(-)	349.08(4.68)(-)	351.79(4.28)(-)	349.35(4.07)(-)	356.58(5.8)(-)	345.14(4.46)
Uval9C	363.89(6.27)(-)	364.33(4.45)(-)	369.69(6.37)(-)	361.15(4.06)(=)	373.94(5.01)(-)	360.36(5.16)
Uval9D	476.86(10.82)(-)	475.31(9.89)(=)	486.36(18.25)(-)	478.3(12.22)(-)	491.21(9.41)(-)	469.8(12.01)
Uval10A	439.33(4.74)(=)	441.52(19.54)(-)	441.37(4.74)(-)	438.55(3.63)(=)	458.51(20.73)(-)	436.58(1.09)
Uval10B	458.65(7.58)(-)	459.46(5.03)(-)	461.27(4.45)(-)	458.54(4.98)(-)	471.01(17.59)(-)	453.76(4.0)
Uval10C	479.37(6.92)(-)	478.5(6.38)(-)	482.29(6.73)(-)	478.85(5.93)(-)	485.69(10.49)(-)	474.86(6.0)
Uval10D	622.17(16.77)(=)	619.15(6.68)(=)	623.39(23.78)(=)	619.56(7.05)(=)	628.2(10.06)(-)	620.74(15.32)
Average	388.54	389.01	395.29	388.22	401.29	384.27

On the *Ugdb* dataset, we can see that GPHH-N significantly outperformed all the compared algorithms. It performed significantly better than GPHH on 11 out of 23 instances and never performed worse than GPHH. Besides, GPHH-N significantly outperformed GPHH-A on 9 out of 23 instances while was defeated by GPHH-A on only 2 instances. GPHH-N performed significantly better than Tarpeian on 19 out of 23 instances and slightly worse on only 1 instance. GPHH-N was defeated

TABLE VII
THE MEAN AND STANDARD DEVIATION FOR **TREE SIZE** OF ROUTING POLICIES OF THE COMPARED ALGORITHMS ON **UGDB** INSTANCES OF 30 RUNS.

Instance	GPHH	GPHH-A	Tarpeian	DT	LPPP	GPHH-N
Ugdb1	83.0(21.64)(-)	74.8(19.29)(-)	21.4(12.68)(+)	83.2(25.28)(-)	7.47(3.14)(+)	50.0(16.13)
Ugdb2	88.2(23.16)(-)	62.87(14.11)(-)	24.13(10.46)(+)	81.67(21.61)(-)	8.67(3.45)(+)	51.33(20.92)
Ugdb3	69.13(23.69)(-)	63.67(19.22)(-)	18.8(10.18)(+)	73.93(25.56)(-)	6.53(1.25)(+)	34.13(17.46)
Ugdb4	66.0(28.63)(-)	65.6(30.39)(-)	16.27(7.6)(+)	64.8(21.08)(-)	8.47(3.86)(+)	45.2(16.53)
Ugdb5	80.6(19.27)(-)	70.13(20.0)(-)	25.13(11.31)(+)	91.87(23.2)(-)	7.0(3.06)(+)	49.47(23.74)
Ugdb6	66.13(16.19)(-)	57.67(18.81)(-)	8.8(19.42)(+)	72.47(24.81)(-)	1.0(0.0)(+)	41.73(17.74)
Ugdb7	43.4(17.63)(-)	35.07(20.06)(=)	2.13(2.81)(+)	44.13(17.97)(-)	1.0(0.0)(+)	32.73(14.35)
Ugdb8	62.33(19.67)(=)	65.33(16.77)(=)	26.27(15.83)(+)	79.8(27.69)(-)	8.27(4.15)(+)	65.0(21.69)
Ugdb9	66.53(21.91)(=)	67.4(20.17)(=)	24.27(16.46)(+)	68.2(18.71)(=)	8.4(3.9)(+)	69.27(20.43)
Ugdb10	64.6(21.4)(-)	60.87(20.74)(-)	14.4(13.52)(+)	63.13(22.69)(-)	5.13(0.51)(+)	29.93(18.77)
Ugdb11	51.27(31.21)(=)	54.6(17.94)(-)	8.73(12.02)(+)	54.53(18.23)(-)	1.73(2.32)(+)	38.67(15.1)
Ugdb12	75.0(20.6)(-)	66.33(17.07)(-)	25.13(11.87)(+)	85.53(28.04)(-)	14.0(4.54)(+)	48.47(17.08)
Ugdb13	69.4(22.05)(-)	66.0(19.76)(-)	17.13(6.39)(+)	64.07(20.08)(=)	8.47(2.97)(+)	54.27(20.55)
Ugdb14	80.4(26.39)(-)	62.13(16.87)(-)	20.33(9.6)(+)	75.53(22.11)(-)	1.53(1.38)(+)	42.53(24.73)
Ugdb15	96.0(30.83)(-)	71.07(32.92)(-)	21.07(12.3)(=)	91.33(39.31)(-)	1.0(0.0)(+)	22.8(16.04)
Ugdb16	19.0(17.62)(=)	10.27(5.52)(=)	2.67(4.07)(+)	28.0(14.86)(-)	1.0(0.0)(+)	20.6(22.37)
Ugdb17	85.73(35.77)(-)	55.67(20.89)(-)	10.87(18.69)(+)	80.47(41.31)(-)	1.07(0.37)(+)	31.53(36.35)
Ugdb18	80.07(29.18)(-)	75.4(25.14)(-)	25.0(13.54)(+)	92.87(36.2)(-)	5.33(2.58)(+)	47.2(23.31)
Ugdb19	71.8(27.87)(-)	71.87(33.84)(-)	16.27(14.05)(+)	74.33(31.86)(-)	1.4(0.81)(+)	26.47(17.58)
Ugdb20	78.93(28.74)(-)	63.53(20.6)(-)	19.8(14.16)(+)	85.53(31.95)(-)	1.0(0.0)(+)	41.27(19.22)
Ugdb21	81.07(32.49)(-)	67.53(19.86)(-)	20.6(9.53)(+)	79.8(31.31)(-)	2.73(2.77)(+)	53.6(25.61)
Ugdb22	80.87(26.72)(-)	67.33(15.09)(-)	23.07(9.24)(+)	78.87(22.34)(-)	1.4(1.22)(+)	55.47(17.49)
Ugdb23	81.33(28.09)(=)	71.13(20.14)(=)	24.73(12.65)(+)	81.2(20.73)(=)	5.4(2.54)(+)	73.2(26.88)
Average	71.33	62.01	18.13	73.70	4.70	44.56

TABLE VIII
THE MEAN AND STANDARD DEVIATION FOR **TREE SIZE** OF ROUTING POLICIES OF THE COMPARED ALGORITHMS ON **UVAL** INSTANCES OF 30 RUNS.

Instance	GPHH	GPHH-A	Tarpeian	DT	LPPP	GPHH-N
Uval1A	68.2(24.45)(-)	66.0(26.61)(-)	11.33(6.1)(+)	77.47(27.33)(-)	3.73(2.55)(+)	29.0(19.81)
Uval1B	67.0(20.1)(-)	58.8(15.63)(-)	18.73(9.26)(=)	67.6(22.71)(-)	6.53(2.21)(+)	23.2(10.11)
Uval1C	69.13(20.58)(=)	65.27(25.22)(=)	24.4(12.68)(+)	82.67(29.53)(=)	8.4(3.45)(+)	69.73(14.58)
Uval2A	73.07(29.08)(-)	69.67(21.98)(-)	19.87(8.27)(=)	76.4(29.6)(-)	7.53(1.66)(+)	31.27(27.17)
Uval2B	73.2(23.7)(-)	72.67(22.92)(-)	19.67(9.93)(+)	76.27(30.29)(-)	6.93(4.05)(+)	33.67(23.4)
Uval2C	81.73(31.01)(=)	64.2(24.4)(=)	23.0(11.72)(+)	68.67(20.69)(=)	13.33(6.06)(+)	72.2(19.26)
Uval3A	72.2(25.59)(-)	65.6(22.63)(-)	27.6(12.87)(+)	77.33(24.2)(-)	5.6(1.19)(+)	36.73(15.87)
Uval3B	68.87(32.24)(-)	67.13(25.94)(-)	11.87(8.92)(+)	72.47(19.43)(-)	3.13(0.51)(+)	41.07(19.93)
Uval3C	75.6(19.26)(-)	69.67(17.05)(=)	27.13(12.94)(+)	84.67(36.86)(-)	8.0(3.35)(+)	63.6(18.07)
Uval4A	68.27(22.76)(-)	68.33(22.54)(-)	22.27(11.54)(+)	76.73(25.47)(-)	8.13(2.45)(+)	47.2(16.13)
Uval4B	72.53(20.41)(-)	60.93(19.11)(=)	25.67(13.97)(+)	70.6(20.15)(-)	11.47(3.95)(+)	54.33(19.05)
Uval4C	68.8(19.02)(=)	66.87(21.05)(=)	26.6(10.54)(+)	77.2(23.41)(-)	12.6(3.12)(+)	64.33(19.84)
Uval4D	66.8(22.98)(=)	66.6(19.79)(=)	29.2(12.65)(+)	70.67(23.35)(=)	16.8(8.78)(+)	70.4(18.52)
Uval5A	68.33(20.16)(-)	68.4(20.92)(-)	21.87(10.14)(+)	68.53(19.31)(-)	6.87(2.4)(+)	40.47(16.47)
Uval5B	84.8(33.43)(-)	70.07(23.32)(-)	28.27(14.21)(+)	80.53(29.0)(-)	10.4(3.57)(+)	52.67(17.61)
Uval5C	63.67(18.48)(-)	60.67(24.65)(-)	21.07(10.3)(+)	64.8(21.97)(-)	11.0(4.39)(+)	46.87(17.08)
Uval5D	70.93(20.65)(=)	64.8(19.2)(=)	19.73(9.73)(+)	72.6(23.76)(=)	12.47(5.12)(+)	66.33(15.85)
Uval6A	77.87(25.58)(-)	73.87(28.8)(-)	23.47(11.79)(+)	74.33(24.76)(-)	9.53(3.6)(+)	39.87(14.09)
Uval6B	74.27(23.92)(-)	62.0(18.58)(-)	22.8(10.28)(+)	71.6(17.94)(-)	9.67(3.8)(+)	47.13(17.4)
Uval6C	66.67(21.32)(=)	63.93(19.06)(=)	26.27(12.54)(+)	74.07(20.5)(-)	9.73(6.23)(+)	63.67(16.71)
Uval7A	73.07(25.02)(-)	67.4(22.51)(-)	27.27(16.71)(+)	76.93(17.78)(-)	10.13(2.91)(+)	40.07(22.85)
Uval7B	74.6(22.3)(-)	66.87(22.17)(-)	22.0(8.55)(+)	80.07(25.13)(-)	10.33(4.28)(+)	37.67(15.57)
Uval7C	73.67(21.13)(=)	64.6(15.24)(=)	25.53(10.94)(+)	75.47(21.46)(-)	14.4(5.07)(+)	63.8(23.58)
Uval8A	70.07(18.06)(-)	65.13(18.99)(-)	24.87(21.78)(+)	66.6(18.44)(-)	7.47(1.87)(+)	43.33(17.63)
Uval8B	68.33(22.62)(-)	63.13(20.36)(-)	18.4(12.87)(+)	66.33(20.56)(-)	7.0(3.28)(+)	45.87(15.9)
Uval8C	73.6(18.66)(-)	68.53(19.25)(=)	27.73(14.53)(+)	71.2(21.27)(=)	14.93(4.65)(+)	62.07(16.31)
Uval9A	68.67(19.55)(-)	70.33(21.3)(-)	22.2(10.99)(+)	69.47(22.59)(-)	8.27(3.3)(+)	48.67(20.53)
Uval9B	60.4(17.84)(=)	64.67(20.89)(=)	25.33(12.55)(+)	69.2(21.97)(-)	7.87(2.27)(+)	53.8(21.86)
Uval9C	69.2(30.56)(=)	55.73(15.96)(=)	24.33(14.18)(+)	84.13(29.38)(-)	8.2(3.18)(+)	61.0(17.9)
Uval9D	80.73(30.09)(-)	76.2(22.81)(-)	29.33(12.95)(+)	77.8(24.92)(-)	14.87(3.48)(+)	62.07(15.83)
Uval10A	62.73(18.84)(-)	58.0(19.28)(-)	22.13(12.39)(+)	62.13(23.1)(-)	5.87(3.14)(+)	38.2(19.19)
Uval10B	70.87(21.73)(-)	60.73(21.42)(=)	21.73(11.71)(+)	68.4(22.21)(-)	8.73(4.06)(+)	56.27(20.32)
Uval10C	65.73(18.13)(-)	63.2(17.34)(=)	26.27(12.64)(+)	70.4(18.58)(-)	8.53(2.81)(+)	55.07(15.14)
Uval10D	67.8(20.45)(=)	66.07(22.53)(=)	31.27(14.69)(+)	77.4(21.98)(-)	16.33(4.4)(+)	63.93(21.05)
Average	70.93	65.77	23.51	73.55	9.55	50.76

by DT on only 1 instance. However, it outperformed DT on 10 out of 23 instances. GPHH-N outperformed LPPP on all the instances except *Ugdb16*. On average, GPHH-N (283.85) performed the best among all the compared approaches on the *Ugdb* dataset.

The same pattern can be observed for the *Uval* dataset. GPHH-N significantly outperformed GPHH on 22 out of 34 instances while never performed worse. GPHH-N outperformed

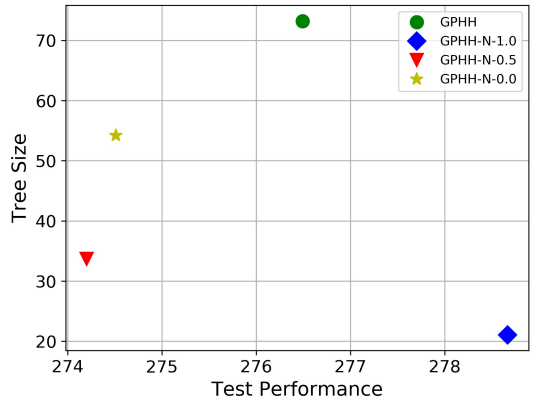


Fig. 6. Scatter plot of GPHH-N with different α value on *Uval2B*

GPHH-A on 24 instances while showed comparable results on the remaining 10 instances. GPHH-N beat Tarpeian on 32 out of 34 instances but never showed significantly worse test performance. GPHH-N also significantly outperformed DT on 20 out of 34 instances and showed comparable test performance on the other instances. GPHH-N beat LPPP on all the instances. The average test performance of GPHH-N (384.27) is better than all the other approaches.

3) Tree Size: Tables VII and VIII show the mean and standard deviation for the tree size of routing policies of the compared algorithms on *Ugdb* and *Uval* instances.

In Table VII, we can see that GPHH-N can evolve much smaller routing policies than GPHH on 18 out of 23 instances on the $Ugdb$ dataset. GPHH-N can also obtain smaller routing policies than GPHH-A on 18 out of 23 instances. In Table VIII, the results are consistent with that in Table VII. GPHH-N can evolve much smaller routing policies than GPHH on 24 out of 34 instances on the $Uval$ dataset. It can evolve much smaller routing policies than GPHH-A on 19 out of 34 instances.

Note that Tarpeian and LPPP achieved much smaller routing policies on both datasets. However, their test performance is much worse than GPHH-N in most instances. The tree size of routing policies evolved by DT is comparable with that in GPHH on both datasets. This is because DT firstly selects the parents based on the fitness and then considers the tree size. Thus, if all the good individuals selected from the first tournament selection are large, there is no chance to select small individuals in the second tournament selection process. Overall, GPHH-N performed well on the tree size, which is expected, as it can effectively remove redundant components in GP trees.

4) Test Performance vs Tree Size: To show the results more clearly, we also plot the results in a scatter map, where the x-axis is mean test performance on 30 independent runs and the y-axis is the mean tree size. Fig. 7 shows the scatter plot on Ugdb1 and Uval2B. Each compared algorithm is represented as a single dot with a specific shape in each figure.

From Fig. 7, we can see that GPHH-N dominates all the other methods except Tarpeian and LPPP, which achieve smaller tree size but much worse test performance. Consider that the test performance is relatively more important than the tree size, we can see that GPNN-N is better than all the other compared algorithms.

TABLE IX
THE MEAN **TRAINING TIME** (SECONDS) OF THE COMPARED ALGORITHMS ON **UGDB** AND **UVAL** DATASETS.

Dataset	GPHH	GPHH-A	GPHH-N
Ugdb (Average)	1027.86	901.05	863.07
Uval (Average)	7794.32	7385.92	7178.14

TABLE X
THE WIN-DRAW-LOSE TABLE FOR THE CONTROLLED EXPERIMENTS BETWEEN THE COMPARED ALGORITHMS AND GPHH-N IN TERMS OF **TEST PERFORMANCE**.

	v.s. no niching elitism	**v.s. no multi-source breeding**	**v.s. no niching tournament selection**
W-D-L	1-55-1	15-42-0	28-29-0

5) Training Time: Table IX shows the mean training time of GPHH, GPHH-A and GPHH-N on *Ugdb* and *Uval* instances. As expected, both GPHH-A and GPHH-N can significantly reduce the training time on most instances. This is mainly because the simplification operation can remove the redundant components. The simplified routing policies need less time to be evaluated. We can see that GPHH-N can further reduce training time comparing with GPHH-A. This is mainly because GPHH-N can remove more redundant components than GPHH-A and makes evolved trees smaller.

Overall, we can see obvious advantage of GPHH-N over GPHH, GPHH-A and the compared bloat control methods. GPHH-N can obtain better and smaller routing policies in a shorter training time.

V. FURTHER ANALYSIS

A. Effect of Each Component

It has been shown that GPHH-N can evolve both better and smaller routing policies. There are four main components in GPHH-N. They are the niching simplification (Section III-D), niching elitism scheme (Section III-E), niching tournament selection (Section III-F) and multi-source breeding (Section III-G). In order to verify the effectiveness of each component of GPHH-N, we designed some controlled experiments. Due to the page limit, we will present the experimental results in the Win-Draw-Lose format. Win (Lose) indicates that GPHH-N can significantly perform better (worse) than the compared algorithm. Draw indicates that GPHH-N can achieve comparable results with the compared algorithm. Table X shows the results in terms of test performance and Table XI shows the results in terms of the tree size of the evolved routing policies.

From Table X, one can see that the niching tournament selection made the most contribution to GPHH-N in terms of test performance. The test performance decreases on 28 out of 57 instances when niching tournament selection is not used (using traditional tournament selection on the simplified population instead). The multi-source breeding method can make some contributions. The niching elitism does not play a major role in improving the test performance.

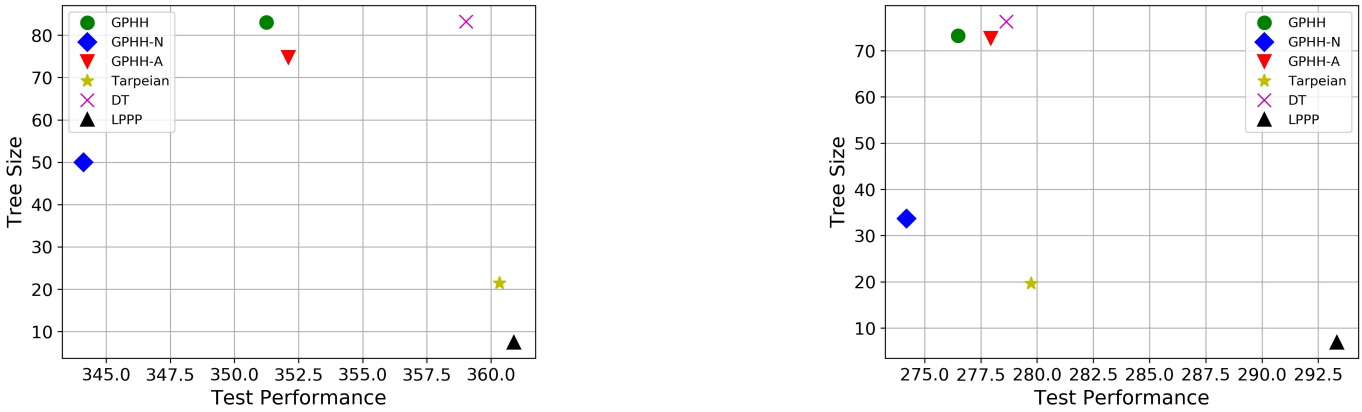


Fig. 7. Scatter map of compared approaches on representative instances.

TABLE XI
THE WIN-DRAW-LOSE TABLE FOR THE CONTROLLED EXPERIMENTS BETWEEN THE COMPARED ALGORITHMS AND GPHH-N IN TERMS OF **TREE SIZE**.

	v.s. no niching elitism	**v.s. no multi-source breeding**	**v.s no niching tournament selection**
W-D-L	33-0-24	42-0-15	0-0-57

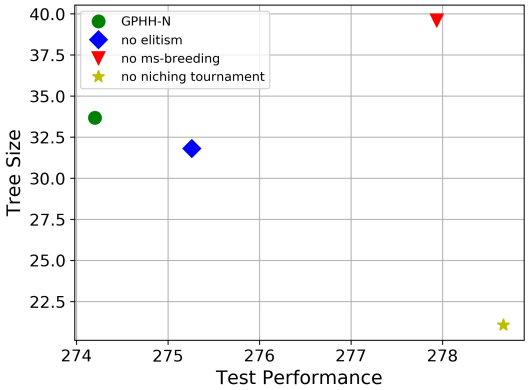


Fig. 8. Scatter map of controlled experiment on a representative instance *Uval2B*

From Table XI, one can observe that the niching tournament selection tends to increase the tree size. The niching tournament selection can improve the diversity of the parent selection process and lead to better test performance. However, it will also increase the tree size.

Fig. 8 shows the scatter plot of the GPHH-N with and without different components. We can see that all the three components are important in terms of test performance. We can see that the dot of GPHH-N without niching tournament selection is located at the bottom right area of the plot. This indicates that niching tournament selection can improve the test performance at the cost of larger tree size.

B. Semantic Analysis of Evolved Policies

To gain further understanding of the behaviour of the routing policies, a representative routing policy is selected. Eqs. (13) – (16) show a selected policy evolved by GPHH-N for the Ugdb19 instance. The policy has a promising test performance (63.39, while the mean test performance of GPHH-N is 63.63). In addition, it has 21 nodes, which is much smaller than the routing policies evolved by other algorithms with the similar test performance.

$$RP = S_1 + \max(S_2, S_3) \tag{13}$$

where

$$S_1 = 2DEM + CFH - CTD \tag{14}$$

$$S_2 = DEM + CFH - CTD \tag{15}$$

$$S_3 = FUT + RQ - \max(CFR1, CTT1) \tag{16}$$

To make it easier to understand, we can also transform RP to the following IF-ELSE format rule set.

if $S_2 \geq S_3$ **then**
  $RP = 3DEM + 2CFH - 2CTD$
else
  if $CFR1 > CTT1$ **then**
    $RP = 2DEM + CFH - CTD + FUT + RQ - CFR1$
  else
    $RP = 2DEM + CFH - CTD + FUT + RQ - CTT1$
  end if
end if

We can identify the following patterns and interpretations from the above rule set.

- When $S_2 \geq S_3$, the RP becomes $RP = 3DEM + 2CFH - 2CTD$. There are two possible cases for $S_2 \geq S_3$ to happen:
  - S_2 is large. This indicates that all the remaining tasks have large demand, far away from the current location of the vehicle and close to the depot;
  - S_3 is small. This indicates that there are not many remaining tasks, the vehicle is almost full, and all the remaining tasks have a large $CFR1$ and $CTT1$

(they are far away from the other vehicles and each other);

In these two cases, RP prefers the tasks with small demands, close to the current location of the vehicle and far away from the depot.

- Otherwise, $S_2 < S_3$ indicates that the remaining tasks can have small demands, are close to the current location of the vehicle, and far away from the depot, and the vehicle is relatively empty. It also has two possible cases:
  - $CFR1 > CTT1$, this indicates that the task is close to some other tasks, but far away from other vehicles. In this case, in addition to small demands, close to the current location of the vehicle and far away from the depot, RP also prefers the tasks with larger $CFR1$, i.e. farther away from other vehicles;
  - $CFR1 < CTT1$, this indicates the task is far away from other tasks, but can be close to some other vehicles. In this case, in addition to small demands, close to the current location of the vehicle and far away from the depot, RP also prefers the tasks with larger $CTT1$, i.e. far away from other tasks. In other words, RP prefers the isolated tasks towards the beginning of the routes.

VI. Conclusions and Future Work

The goal of this work was to evolve both effective and smaller/simpler routing policies for UCARP. This goal has been successfully achieved by the newly proposed novel GPHH with a simplification approach using a niching technique (GPHH-N). GPHH-N was examined and compared with the basic GPHH approach without simplification (GPHH), the basic GPHH approach with algebraic simplification (GPHH-A) and three representative bloat control methods on 57 UCARP instances. The results suggest that GPHH-N can outperform all the compared approaches in terms of test performance. GPHH-N can also outperform GPHH and GPHH-A in terms of tree size and training time. We also analysed the effect of the newly proposed components by a set of controlled experiments. The results showed that all the three new components could contribute to evolve smaller and better routing policies. The niching tournament selection and multi-source breeding components are more effective than the niching elitism component. Overall, GPHH-N can obtain better test performance and smaller and potentially more interpretable routing policies than the current state-of-the-art GPHH approach.

In the future, we will consider combining our approach with other simplification approaches. In addition, we will consider some other ways that can improve the interpretability of evolved routing policies, such as Genetic programming visualisation techniques and grammar-based Genetic Programming. GPHH-N was developed for UCARP. However, it could also be extended to evolve effective and simple heuristics for other dynamic and uncertain problems such as dynamic job shop scheduling by changing the fitness evaluation and phenotypic characterisation for niching if necessary. We will extend GPHH-N to other problems in our future work.

References

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

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

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

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

[5] 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.

[6] J. Liu, K. Tang, and X. Yao, "Robust optimization in uncertain capacitated arc routing problems: Progresses and perspectives," *IEEE Computational Intelligence Magazine*, vol. 16, no. 1, pp. 63–82, 2021.

[7] H. Tong, L. L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "Towards novel meta-heuristic algorithms for dynamic capacitated arc routing problems," in *Proceedings of the International Conference on Parallel Problem Solving from Nature*. Springer, 2020, pp. 428–440.

[8] M. R. Khouadjia, B. Sarasola, E. Alba, L. Jourdan, and E.-G. Talbi, "A comparative study between dynamic adapted pso and vns for the vehicle routing problem with dynamic requests," *Applied Soft Computing*, vol. 12, no. 4, pp. 1426–1439, 2012.

[9] N. Secomandi and F. Margot, "Reoptimization approaches for the vehicle-routing problem with stochastic demands," *Operations research*, vol. 57, no. 1, pp. 214–230, 2009.

[10] G. Chryssolouris and V. Subramaniam, "Dynamic scheduling of manufacturing job shops using genetic algorithms," *Journal of Intelligent Manufacturing*, vol. 12, no. 3, pp. 281–293, 2001.

[11] 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.

[12] 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.

[13] 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.

[14] H. Al-Sahaf, Y. Bi, Q. Chen, A. Lensen, Y. Mei, Y. Sun, B. Tran, B. Xue, and M. Zhang, "A survey on evolutionary machine learning," *J. Roy. Soc. New Zeal.*, vol. 49, no. 2, pp. 205–228, 2019.

[15] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 193–208, 2013.

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

[17] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Collaborative multi-fidelity based surrogate models for genetic programming in dynamic flexible job shop scheduling," *IEEE Transactions on Cybernetics*, 2021. Doi: 10.1109/TCYB.2021.3050141.

[18] ——, "Correlation coefficient based recombinative guidance for genetic programming hyper-heuristics in dynamic flexible job shop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 3, pp. 552–566, 2021.

[19] B. Tan, H. Ma, Y. Mei, and M. Zhang, "A cooperative coevolution genetic programming hyper-heuristic approach for on-line resource allocation in container-based clouds," *IEEE Transactions on Cloud Computing*, 2020.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] G. Gao, Y. Mei, B. Xin, Y.-H. Jia, and W. N. Browne, "Automated coordination strategy design using genetic programming for dynamic multipoint dynamic aggregation," *IEEE Transactions on Cybernetics*, 2021.

[25] 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.

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

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

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

[29] 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*, 2019, pp. 1093–1101.

[30] D. Kinzett, M. Zhang, and M. Johnston, "Investigation of simplification threshold and noise level of input data in numerical simplification of genetic programs," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[31] S. Luke and L. Panait, "A comparison of bloat control methods for genetic programming," *Evolutionary Computation*, vol. 14, no. 3, pp. 309–344, 2006.

[32] D. Kinzett, M. Johnston, and M. Zhang, "How online simplification affects building blocks in genetic programming," in *Proceedings of the conference on Genetic and evolutionary computation*, 2009, pp. 979–986.

[33] M. Zhang, P. Wong, and D. Qian, "Online program simplification in genetic programming," in *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer, 2006, pp. 592–600.

[34] M. Zhang and P. Wong, "Genetic programming for medical classification: a program simplification approach," *Genetic Programming and Evolvable Machines*, vol. 9, no. 3, pp. 229–255, 2008.

[35] M. Zhang, Y. Zhang, and W. Smart, "Program simplification in genetic programming for object classification," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 2005, pp. 988–996.

[36] P. Wong and M. Zhang, "Algebraic simplification of gp programs during evolution," in *Proceedings of the conference on Genetic and evolutionary computation*, 2006, pp. 927–934.

[37] D. Kinzett, M. Zhang, and M. Johnston, "Using numerical simplification to control bloat in genetic programming," in *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer, 2008, pp. 493–502.

[38] A. Song, D. Chen, and M. Zhang, "Contribution based bloat control in genetic programming," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[39] S. W. Mahfoud, "Niching methods for genetic algorithms," Ph.D. dissertation, Citeseer, 1995.

[40] T. Huang, Y.-J. Gong, S. Kwong, H. Wang, and J. Zhang, "A niching memetic algorithm for multi-solution traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, 2019.

[41] 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.

[42] H. Handa, L. Chapman, and X. Yao, "Robust salting route optimization using evolutionary algorithms," in *Evolutionary Computation in Dynamic and Uncertain Environments*. Springer, 2007, pp. 497–517.

[43] M. Fadzli, N. Najwa, and M. Luis, "Capacitated arc routing problem and its extensions in waste collection," in *AIP Conference Proceedings*, vol. 1660, no. 1. AIP Publishing LLC, 2015, p. 050001.

[44] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[45] J. M. Belenguer and E. Benavent, "A cutting plane algorithm for the capacitated arc routing problem," *Computers & Operations Research*, vol. 30, no. 5, pp. 705–728, 2003.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] L.-N. Xing, P. Rohlfshagen, Y.-W. Chen, and X. Yao, "A hybrid ant colony optimization algorithm for the extended capacitated arc routing problem," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 41, no. 4, pp. 1110–1123, 2011.

[53] Y.-H. Jia, Y. Mei, and M. Zhang, "A bilevel ant colony optimization algorithm for capacitated electric vehicle routing problem," *IEEE Transactions on Cybernetics*, 2021.

[54] Y. Mei, X. Li, and X. Yao, "Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 435–449, 2013.

[55] K. Tang, J. Wang, X. Li, and X. Yao, "A scalable approach to capacitated arc routing problems based on hierarchical decomposition," *IEEE transactions on cybernetics*, vol. 47, no. 11, pp. 3928–3940, 2016.

[56] L. Feng, Y.-S. Ong, I. W.-H. Tsang, and A.-H. Tan, "An evolutionary search paradigm that learns with past experiences," in *2012 IEEE Congress on Evolutionary Computation*. IEEE, 2012, pp. 1–8.

[57] L. Feng, Y.-S. Ong, A.-H. Tan, and I. W. Tsang, "Memes as building blocks: a case study on evolutionary optimization+ transfer learning for routing problems," *Memetic Computing*, vol. 7, no. 3, pp. 159–180, 2015.

[58] L. Feng, Y. Ong, M. Lim, and I. W. Tsang, "Memetic search with interdomain learning: A realization between cvrp and carp," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 644–658, 2015.

[59] E. Babaee Tirkolaee, M. Alinaghian, M. Bakhshi Sasi, and M. Seyyed Esfahani, "Solving a robust capacitated arc routing problem using a hybrid simulated annealing algorithm: a waste collection application," *Journal of Industrial Engineering and Management Studies*, vol. 3, no. 1, pp. 61–76, 2016.

[60] 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.

[61] 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.

[62] S. Tsutsui, M. Pelikan, and D. E. Goldberg, "Using edge histogram models to solve permutation problems with probabilistic model-building genetic algorithms," *IlliGAL Report*, vol. 2003022, 2003.

[63] H. Tong, L. L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "A novel generalised meta-heuristic framework for dynamic capacitated arc routing problems," *arXiv preprint arXiv:2104.06585*, 2021.

[64] H. Tong, L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "A hybrid local search framework for the dynamic capacitated arc routing problem," in *Genetic and Evolutionary Computation Conference*, 2021.

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

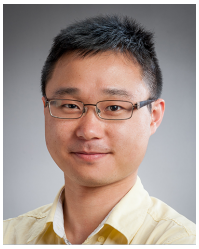
[66] J. MacLachlan, Y. Mei, J. Branke, and M. Zhang, "Genetic programming hyper-heuristics with vehicle collaboration for uncertain capacitated arc routing problems," *Evolutionary computation*, vol. 28, no. 4, pp. 563–593, 2020.

[67] M. A. Ardeh, Y. Mei, and M. Zhang, "Genetic programming hyper-heuristic with knowledge transfer for uncertain capacitated arc routing problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019, pp. 334–335.

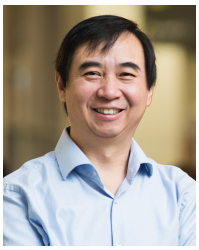
[68] Y. Liu, Y. Mei, M. Zhang, and Z. Zhang, "A predictive-reactive approach with genetic programming and cooperative coevolution for the uncertain

capacitated arc routing problem," *Evolutionary computation*, vol. 28, no. 2, pp. 289–316, 2020.
[69] Q. Chen, B. Xue, and M. Zhang, "Rademacher complexity for enhancing the generalization of genetic programming for symbolic regression," *IEEE Transactions on Cybernetics*, 2020.
[70] J. R. Koza and J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.
[71] M. J. Cavaretta and K. Chellapilla, "Data mining using genetic programming: The implications of parsimony on generalization error," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 2. IEEE, 1999, pp. 1330–1337.
[72] S. Luke and L. Panait, "Fighting bloat with nonparametric parsimony pressure," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2002, pp. 411–421.
[73] D. C. Hooper and N. S. Flann, "Improving the accuracy and robustness of genetic programming through expression simplification," in *Proceedings of the conference on genetic programming*, 1996, pp. 428–428.
[74] D. Kinzett, M. Johnston, and M. Zhang, "Numerical simplification for bloat control and analysis of building blocks in genetic programming," *Evolutionary Intelligence*, vol. 2, no. 4, p. 151, 2009.
[75] 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.
[76] T. Hildebrandt and J. Branke, "On using surrogates with genetic programming," *Evolutionary computation*, vol. 23, no. 3, pp. 343–367, 2015.
[77] 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.
[78] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward, "A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 942–958, 2010.
[79] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, "Automating the packing heuristic design process with genetic programming," *Evolutionary computation*, vol. 20, no. 1, pp. 63–89, 2012.
[80] 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.

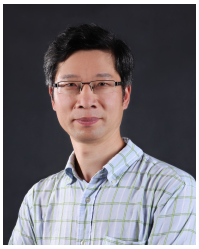
Shaolin Wang Shaolin Wang is a second-year PhD student at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. He holds a Master of Computer Science (2019) from Victoria University of Wellington, a Graduate Diploma in Computer Science (2016) from University of Canterbury, Postgraduate Diploma in Geographic Information System (2014) from University of Otago and a BSc in Geographic Information System (2012) from Nanning Normal University. His research interests are Artificial Intelligence, Evolutionary Computation, Hyper-Heuristic, Routing and Real-World Optimisation.



Yi Mei Dr. Yi Mei (M'09-SM'18) received the BSc and PhD degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively. He is currently a Senior Lecturer at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research interests include evolutionary scheduling and combinatorial optimisation, machine learning, genetic programming, and hyper-heuristics. He has over 130 fully referred publications, including the top journals in EC and Operations Research such as IEEE TEVC, IEEE TCYB, Evolutionary Computation Journal, European Journal of Operational Research, ACM Transactions on Mathematical Software. He serves as a Vice-Chair of the IEEE CIS Emergent Technologies Technical Committee, and a member of Intelligent Systems Applications Technical Committee. He is an Editorial Board Member/Associate Editor of four International Journals, and a guest editor of a special issue of the Genetic Programming Evolvable Machine journal. He serves as a reviewer of over 30 international journals.



Mengjie Zhang Professor Mengjie Zhang (M'04-SM'10-F'19) received the B.E. and M.E. degrees from Artificial Intelligence Research Center, Agricultural University of Hebei, Baoding, China, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 1989, 1992, and 2000, respectively. He is currently a Professor of Computer Science, the Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) with the Faculty of Engineering, Victoria University of Wellington, Wellington, New Zealand. His current research interests include evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of image analysis, multiobjective optimization, feature selection and reduction, job-shop scheduling, and transfer learning. She has published over 500 research papers in refereed international journals and conferences. Prof. Zhang was the Chair of the IEEE CIS Intelligent Systems and Applications Technical Committee, the IEEE CIS Emergent Technologies Technical Committee, and the Evolutionary Computation Technical Committee, and a member of the IEEE CIS Award Committee. He is a Vice-Chair of the Task Force on Evolutionary Computer Vision and Image Processing and the Founding Chair of the IEEE Computational Intelligence Chapter in New Zealand. He is also a Committee Member of the IEEE NZ Central Section. He is a Fellow of the Royal Society of New Zealand and an IEEE Distinguished Lecturer.



Xin Yao Professor Xin Yao obtained his Ph.D. in 1990 from the University of Science and Technology of China (USTC), MSc in 1985 from North China Institute of Computing Technologies, and BSc in 1982 from USTC. He is a Chair Professor of Computer Science at the Southern University of Science and Technology (SUSTech), Shenzhen, China, and a part-time Professor of Computer Science at the University of Birmingham, UK. He is an IEEE Fellow and was a Distinguished Lecturer of the IEEE Computational Intelligence Society (CIS). He served as the President (2014-15) of IEEE CIS and the Editor-in-Chief (2003-08) of IEEE Transactions on Evolutionary Computation. His major research interests include evolutionary computation, ensemble learning, and their applications to software engineering. His work won the 2001 IEEE Donald G. Fink Prize Paper Award; 2010, 2016 and 2017 IEEE Transactions on Evolutionary Computation Outstanding Paper Awards; 2011 IEEE Transactions on Neural Networks Outstanding Paper Award; 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist); and many other best paper awards at conferences. He received a 2012 Royal Society Wolfson Research Merit Award, the 2013 IEEE CIS Evolutionary Computation Pioneer Award and the 2020 IEEE Frank Rosenblatt Award.