Learning Heuristics with Different Representations for Stochastic Routing

Ya-Hui Jia, *Member, IEEE,* Yi Mei, *Senior Member, IEEE,* and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Uncertainty is ubiquitous in real-world routing applications. Automated design of routing policy by hyper-heuristic methods is an effective technique to handle the uncertainty and to achieve online routing for dynamic or stochastic routing problems. Currently, the tree representation routing policy evolved by genetic programming is commonly adopted because of the remarkable flexibility. However, numeric representations have never been used. Considering the practicability of the numeric representations and the capability of the numeric optimization methods, in this paper, we investigate two numeric representations on a representative stochastic routing problem, uncertain capacitated arc routing problem. Specifically, a linear representation and an artificial neural network representation are implemented and compared with the tree representation to reveal the potential of the numeric representations and the characteristics of their optimization. Experimental results show that the tree representation is the best choice, but on a majority of the test instances, the numeric representations, especially the artificial neural network representation, can provide competitive performance. Further analyses also show that training a good artificial neural network representation policy requires more training data than the tree representation. Finally, a guideline of representation selection is given.**

***Index Terms*—Hyper-heuristic, uncertain capacitated arc routing, stochastic routing, genetic programming, artificial neural network, evolutionary learning.**

I. Introduction

ROUTING problems widely exist in the transportation and logistics industry. Many studies have thoroughly investigated the deterministic routing problems and provide fixed routing plans as the solutions. However, due to the ubiquitous uncertainty in real-world applications such as stochastic customer demand and stochastic travel time, vehicles often encounter unexpected routing failures like insufficient capacity and impassible road that make the vehicles unable to finish the tasks according to the predefined routing plan [1]–[3]. To capture such uncertainty, many traditional routing problems have been transformed into stochastic versions [4]–[7]. For stochastic routing problems, an effective routing policy instead of a fixed route is required to dynamically adjust the route based on the status of the vehicle and the environment [8]. In different methodologies, routing policies can be represented in different ways [8], [9]. One feasible approach is to take a heuristic function as the routing policy. Whenever a vehicle finishes a task and becomes idle, it can use the heuristic function to calculate the priorities of the remaining tasks and decide where to go next. For example, the Smallest-Task-First heuristic can be represented as the reciprocal of the customer demand. However, such simple heuristics are not effective enough to handle complex cases. Meanwhile, designing heuristics manually requires high expertise and is usually time-consuming. To generate effective heuristics automatically, hyper-heuristic techniques have been widely applied [10]–[12].

The representation of the heuristic highly affects the difficulty of its optimization and its effectiveness [13]–[15]. Generally, there are two categories of heuristic representations: variable-length grammar-based representation and fixed-length parametric representation [16]. The tree representation is a popular variable-length grammar-based representation [17]–[19]. The corresponding heuristic is usually evolved as a expression tree of arithmetic operators and the attributes of the problem. The fixed-length parametric representation, a.k.a. the numeric representation [20], requires a pre-defined function format and the variables to be optimized are the parameters/weights of the function.

Due to the NP-hard nature and uncertain characteristics, it is difficult to know how complex a routing policy should be in advance. Thus, most hyper-heuristic studies about stochastic or dynamic routing problems focused on genetic programming hyper-heuristic (GPHH) methods with the tree representation, since it can evolve appropriately complex functions without much priori knowledge [12], [21]. People have spent great efforts in improving the performance of GPHH through proposing new features and adopting advanced evolution schemes [22]–[24]. In contrast, to the best of our knowledge, nobody has ever tried to use hyper-heuristic algorithms with numeric representations for stochastic routing problems. Without the comparison between these two kinds of representations, it is hard to know how well the routing policy is and how mathematically complex the routing policy is.

Previously, Branke *et al.* [20] have compared three different representations of heuristics on a dynamic scheduling problem. They found that given a sufficiently high computational budget, GPHH can evolve a better tree-representation heuristic than the artificial neural network (ANN) representation and the linear representation. Nguyen *et al.* [25] however designed a linear ratio representation that showed significantly better results than the tree-representation heuristic. These comparisons

This work was supported in part by the Marsden Fund of New Zealand Government under Contracts VUW1509 and VUW1614, the Science for Technological Innovation Challenge (SfTI) fund under grant E3603/2903, and the MBIE SSIF Fund under Contract VUW RTVU1914.

Ya-Hui Jia is with School of Future Technology, South China University of Technology, Guangzhou, China and also with School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. (email: jia.yahui@foxmail.com)

Yi Mei, and Mengjie Zhang are with School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. (email:yi.mei@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz)

were made only on very few scenarios which might lead to a bias or incomplete conclusion. Also, routing problems are intrinsically different from scheduling problems because of the unique characteristics and constraints like the geographically distributed tasks and capacity limited vehicles, so that the previous observations may not be applicable to the routing problems. In addition, there are several factors that were not investigated in previous studies such as the amount of training data and the difficulty of the real-value optimization for numeric-representation heuristics.

Considering these issues, in this paper, we investigate the aforementioned three representations, i.e. 1) tree representation, 2) linear representation, and 3) ANN representation, on a representative stochastic routing problem, i.e. the uncertain capacitated arc routing problem (UCARP) [7], to cover the shortage of the research of numeric representation hyper-heuristic methods for routing problems. Specifically, during the investigation of this paper, we will focus on the following three questions.

1) Since numeric representations have never been tried for stochastic routing problems, the primary question is whether they are able to represent good routing policies, and the further question is what kind of instance each representation suits.
2) For numeric representations, once the function structure is decided, the quality of the final heuristic depends on the optimization algorithm. What the characteristics of the real-value search space of the numeric representation are is worth investigating. This question is important for choosing suitable optimization algorithms.
3) Another important question is how many simulation samples are needed to evolve a good heuristic. This question is related to the amount of data that is required when the algorithms are used in a real-world applications.

Through investigating these three questions, we want to achieve three research objectives:

1) Figuring out the difficulty of the stochastic routing problems in terms of heuristic generation;
2) Making a guideline of the three representations for stochastic routing problems;
3) Finding the research direction of the heuristic design for stochastic routing problems.

II. Background

In this paper we choose UCARP as the representative problem to study. The problem definition and the methodology of how to use hyper-heuristic method to solve the problem are introduced. The related works are also reviewed afterward.

A. Uncertain Capacitated Arc Routing Problem

Based on the capacitated arc routing problem (CARP), UCARP considers four uncertain factors: stochastic task demand, stochastic task presence, stochastic edge deadheading cost, and stochastic edge existence [7]. A UCARP instance is defined on a connected undirected graph $G(V, E)$, where V and E represent the vertex set and the edge set, respectively. Each edge $e \in E$ has a non-negative demand $d(e)$, a non-negative serving cost $s(e)$, and a positive traversal cost (deadheading cost) $t(e)$. If there is no task on the edge, $d(e) = 0$ and $s(e) = 0$. We denote the set of edges that have tasks as E_T. Due to the uncertain characteristics, these three attributes are stochastic variables. Given a fleet of homogeneous vehicles that has a max capacity Q and must depart from and return to the depot v_0, the goal is to minimize the total cost of serving all tasks.

In real-world applications, these attributes are unknown in advance and should be revealed during the vehicles traversing the edges and serving the tasks. To simulate this process, we define a *sample* of a UCARP by sampling each stochastic variable an actual value. The three attributes of a sample under a specific environment I_ξ, e.g. a random seed ξ, are denoted as $d_\xi(e)$, $s_\xi(e)$, and $t_\xi(e)$. A solution to a sample of a UCARP instance consists of two components, $S_\xi = (\mathbf{\Gamma}, \mathbf{\Pi})$. $\mathbf{\Gamma} = \{\mathit{\Gamma}^1, \dots, \mathit{\Gamma}^K\}$ represents K routes, where each $\mathit{\Gamma}^k = (\gamma_1^k, \dots, \gamma_{L_k}^k)$ is a vertex sequence. $\mathbf{\Pi} = \{\mathit{\Pi}^1, \dots, \mathit{\Pi}^K\}$ represents the serving condition of each route. Each $\mathit{\Pi}^k = (\pi_1^k, \dots, \pi_{L_k-1}^k)$ is a vector that shows the fraction of demand served at each edge. $\pi_1^k = 0.5$ means that the task of the edge (γ_1^k, γ_2^k) is served 50%. Given the above definitions, a UCARP can be formulated as follows [8]:

$$\min \mathrm{E}_{\xi \in \Xi}[\sum_{k=1}^{K} \sum_{i=1}^{L_k-1} t_\xi(\gamma_i^k, \gamma_{i+1}^k) + \sum_{e \in E_T} (s_\xi(e) - t_\xi(e))], \tag{1}$$

$$\text{s.t. } \gamma_1^k = \gamma_{L_k}^k = v_0, \forall k = 1, 2, \dots, K, \tag{2}$$

$$\sum_{k=1}^{K} \sum_{i=1}^{L_k-1} \pi_i^k \cdot z_i^k(e) = 1, \forall e \in E_T, \tag{3}$$

$$\sum_{k=1}^{K} \sum_{i=1}^{L_k-1} \pi_i^k \cdot z_i^k(e) = 0, \forall e \in E - E_T, \tag{4}$$

$$z_i^k(e) = \begin{cases} 1 & \text{if } e = (\gamma_i^k, \gamma_{i+1}^k), \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

$$\sum_{i=1}^{L_k-1} d_\xi(\gamma_i^k, \gamma_{i+1}^k) \cdot \pi_i^k \le Q, \forall k = 1, 2, \dots, K, \tag{6}$$

$$(\gamma_i^k, \gamma_{i+1}^k) \in E, \forall k = 1, \dots, K, \forall i = 1, \dots, L_k - 1, \tag{7}$$

$$\pi_i^k \in [0, 1], \forall k = 1, 2, \dots, K, \forall i = 1, 2, \dots, L_k - 1. \tag{8}$$

The objective (1) is to minimize the expectation of the total cost of all samples Ξ. (2) shows the constraint that vehicles should depart from and return to the depot. (3), (4), and (5) indicate that every task should be fully served. (6) sets the capacity constraint. (7) and (8) defines the variables.

There are three reasons of choosing UCARP as the representative routing problem to study:

- From the perspective of real-world applications, CARP is widely applied to model real-world applications such as road maintenance, garbage collection, and snowploughing [26]–[28]. Compared with some routing problems defined on a plain that assume all nodes are fully connected, CARP is closer to reality. In addition, CARP is a representative model that has the common underlying characteristics (e.g., objectives and constraints) of the other variants. Although CARP cannot fully represents them, it is important to start the investigation from the basic but representative problem model.

- From the theoretical perspective, a CARP can be transformed to a capacitated vehicle routing problem (CVRP) by introducing more vertices [29]. In the supplementary material, we also showed how to transform a CVRP into a CARP. Thus, the experiments made on CARP should be suitable for CVRP as well to a certain extent.
- From the perspective of uncertainty, UCARP belongs to the category of stochastic routing. It has concerned four stochastic factors of both task demand and traversing cost, which has basically covered the stochastic characteristics of the real-world applications.

Overall, based on these three reasons, our study on UCARP is expected to be extendable to some other stochastic routing problems [1], such as stochastic CARP [28], vehicle routing problem (VRP) with stochastic demand [30], and stochastic VRP with random travel times [6].

B. Routing Policy

Due to the uncertainty of UCARP, it is hard to apply a static solution to all possible situations. Routing policy (dispatching rule), as a heuristic, is an effective mechanism to handle the uncertainty since it does not rely on any pre-planned solution [31], [32]. During the serving process, a routing policy is used to give instructions to the vehicles, telling each vehicle which task to serve next based on the status of the vehicle and the attributes of the remaining tasks. Although any method that can give instructions to the vehicles can be taken as a routing policy, in this paper, we specifically use the routing policy as a function to calculate priority values for candidate tasks. Whenever a vehicle becomes idle and there are still unassigned tasks, the task with the highest priority is chosen to be served by the vehicle.

In this paper, we use simulation to evaluate a routing policy [33]. The simulation process of applying a routing policy to a UCARP instance sample is shown in Algorithm I. Since the true values of the attributes are unknown in advance, we usually use the information of historical distributions of the attributes to calculate the priorities of the tasks like the expectation value. Due to the online routing process, a vehicle may encounter two kinds of failure:

- Route failure: the actual demand of the task is beyond the remaining capacity of the vehicle.
- Edge failure: the edge ahead of the vehicle is impassable.

The route failure can be repaired by returning depot and refill. The edge failure can be repaired by making a detour using Dijkstra's algorithm. Also, to obtain the objective (1), we usually use several samples rather than a single sample to evaluate a routing policy.

C. Learning Routing Policies

Taking a routing policy as a heuristic, a hyper-heuristic algorithm searches the space of heuristics rather than the space of solutions [34]–[36]. Mathematically, a routing policy is a function that maps several attributes to a priority value, $f(\mathbf{x}) = y$. Thus, if the priority is defined as a real value $y \in \mathbb{R}$, the search space of the hyper-heuristic is affected by the other two components, attributes $\mathbf{x}$ and representation f.

- Attributes are the information that can be used during the simulation. They can be extracted from task-related information, vehicle-related information, and environmental information. These attributes can be either static like the maximum capacity of vehicles or dynamic like the distance between the vehicle and the task. How many attributes are extracted defines the dimensionality of the attributes space $\mathbf{x} \in S$. Suppose ns attributes are extracted, $S \subseteq \mathbb{R}^{ns}$.
- A representation of a routing policy f consists of two parts: 1) the function class $f \in F$ that defines the structure of the function (e.g. linear combination of the attributes, an expression tree with attributes as terminals) and 2) the parameter space Ψ which specifies the parameters ψ that can be tuned to optimize the routing policy.

Assisted by the simulation, the overall approach to generating the routing policy for a UCAPR is depicted in Fig. 1. Given the attribute set and function class, the optimization algorithm generates candidate routing policies and sends them to the simulation process to evaluate. According to the simulation results, the optimization algorithm would gradually adjust the parameters to generate better policies. Whenever the stopping criterion is met, e.g. the maximum number of fitness evaluations (simulations), the optimization algorithm stops and returns the best routing policy.

Algorithm 1 Simulation

```
Input: a instance sample $I_\xi$, a routing policy $h(\cdot)$, number of vehicles $K$.
Output: a solution $S_\xi$.
 1: Initialize an empty event sequence $\Upsilon$;
 2: initialize an empty solution $S_\xi$;
 3: unassigned task set $U_T = E_T$;
 4: for $i = 1, \ldots, K$ do
 5:     add an event of the $i$th vehicle into $\Upsilon$;
 6: end for
 7: while $\Gamma$ is not empty do
 8:     retrieve the next event $\epsilon$ and its corresponding vehicle;
 9:     if $U_T$ is not empty then
10:         calculate the priority values of $e \in U_T$ by $h(\cdot)$;
11:         assign $e_h$ with the highest priority to the vehicle;
12:         remove $e_h$ from $U_T$; remove $\epsilon$ from $\Upsilon$;
13:         update the event sequence;
14:     else
15:         the vehicle returns to depot;
16:     end if
17:     update the solution $S_\xi$;
18: end while
19: return $S_\xi$;
```

D. Related Work

Hyper-heuristic methods can be classified into two categories: generative hyper-heuristic and selective hyper-heuristic [34]. Although some selective hyper-heuristic methods can

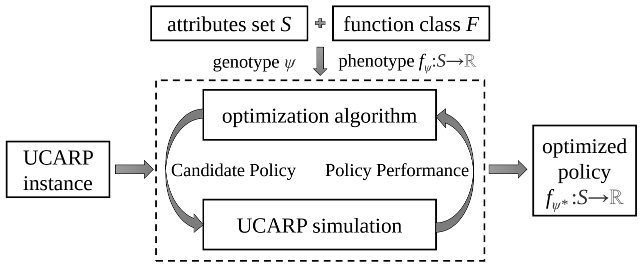


Fig. 1. Overview of finding the best routing policy by hyper-heuristic.

also achieve online decision making [37]–[39], their performance depends on the quality of the low-level heuristics. Designing effective low-level heuristic is a problem-dependent job that requires high expertise [40], [41]. Generative hyper-heuristic methods usually do not rely on low-level heuristics [34], that is flexible to different problems. In this paper, the heuristic representations are investigated under the generative hyper-heuristic category.

Broadly, hyper-heuristic algorithms are also related to automatic algorithm design since the product is a heuristic that can be considered as an algorithm [42]. The similarity between hyper-heuristic and algorithm portfolio construction [43], [44] is that they both pay attention to the generalization ability of the generated algorithm. The difference is that hyper-heuristic considered in this paper takes charge of not only tuning the parameters of the algorithm but also build the algorithm from scratch if the algorithm's structure is not determined like the tree representation. A hyper-heuristic algorithm is essentially a knowledge learning process that discovers knowledge through evolution to achieve the goal. However, different from the evolutionary transfer optimization algorithms [45], [46], the learned heuristic in a hyper-heuristic algorithm is used to directly build solutions for a case instead of re-optimizing existing solutions or transferring knowledge between different cases.

Regarding the generative hyper-heuristic methods that were proposed for routing problems, Weise *et al.* [47] first applied GPHH to CARP with stochastic task demand. After that, Liu *et al.* [22] and MacLachlan *et al.* [8] extracted more informative attributes and proposed better GPHH algorithms for UCARP. Wang *et al.* [23], [48] adopted multi-objective optimization and niching techniques in GPHH to evolve more explainable tree-representation heuristics. Ardeh *et al.* [24], [49] proposed several transfer learning methods to investigate whether a routing policy evolved for a case can be used for the other cases. Besides arc routing problems, GPHH methods are also applied to vehicle routing problems (VRPs). Benyahia and Potvin [12] proposed a GPHH method to help the vehicle dispatching in a courier service application. Lewczuk *et al.* [50] used GP for a time-dependent VRP. Sim and Hart [51] proposed a hybrid algorithm that combined GPHH with a selective hyper-heuristic for VRP. Recently, Jacobsen-Grocott *et al.* [52] tried GPHH for dynamic VRP with time window and achieved better results than manually designed heuristics. All of the above works have demonstrated the effectiveness of GPHH, but so far, there has not been a generative hyper-heuristic algorithm that uses numeric representations for dynamic or stochastic routing problems.

Numeric representation is currently not the mainstream, but several works have shown its practicality for scheduling problems although GPHH is the most widely used generative hyper-heuristic method for scheduling problems as well [53]–[57]. The linear representation has been used in the early days [58]–[60]. Among these works, Kuczapski *et al.* [60] proposed an interesting linear representation as the weighted sum of different dispatching rules. Eguchi *et al.* [61] proposed a hyper-heuristic method to evolve an ANN representation dispatching rule by simulated annealing for a job-shop scheduling problem with dynamically changing environment. Recently, Nguyen and Zhang [25] have successfully evolved a linear-ratio-representation heuristic using particle swarm optimization that showed significantly better results than GPHH. Branke *et al.* [20] made a comparison among the tree representation, the linear representation, and the ANN representation on a dynamic scheduling problem. They have found that 1) in terms of the effectiveness, given sufficiently high computational budget, the tree representation can provide better performance than the ANN representation and the linear representation. The linear representation is far worse than the other two representations. 2) In terms of the computational budget, the linear representation requires the lowest budget to converge, followed by the ANN representation. GP requires the highest budget to evolve a good tree-representation dispatching rule.

However, this comparison was only made under a single scenario of scheduling which might be biased or incomplete. Also, whether the conclusion suits routing problems is questionable since routing problems have intrinsic difference with scheduling problems. Besides, there are two aspects that have not been investigated in any literature.

- Researchers have found several characteristics of the search space of the tree representation heuristic like the duplication phenomenon where different heuristics lead to the same result [20], [62]. However, there is no study about the real-value search space of the numeric representations.
- Most of the studies conducted their experiments under a same experimental setting. Although several of them have investigated the computational budget requirement [20], [25], the requirement of training data is not investigated.

All of the aforementioned issues and questions motivate us to conduct the comparison of different heuristic representations for routing problems.

III. Policy Representations

From the perspective of evolutionary computation (EC), representation is the bridge to connect the genotype and the phenotype of the solution. Genetic or evolutionary operators are directly applied to genotypes. Phenotypes represent the actual solutions that are transformed from genotypes through the representation [13]. The representation of routing policy highly affects the search space of the optimization algorithm. Intuitively, a small and smooth search space that contains the optimal solution is always desirable. However, due to the

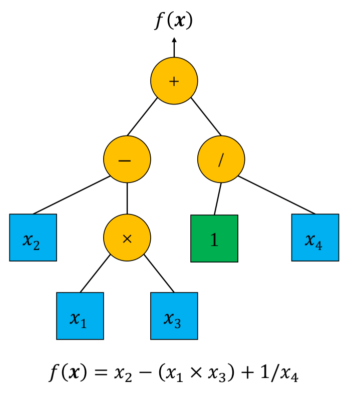


Fig. 2. Tree representation.

complexity of UCARP, it is hard to design a representation that can lead to such a search space. This problem leads to the significance of this paper that it would be helpful in understanding the advantages and disadvantages of different representation alternatives through an empirical comparison, thus to guide the future research.

A. Tree Representation

A routing policy with tree representation is a valid arithmetic expression [20]. As Fig. 2 shows, the inner nodes, a.k.a. non-terminal nodes, of the expression tree are operators, and the leaf nodes, a.k.a. terminal nodes, are the attributes or constant numbers. Corresponding to the type of the operator, the number of child nodes of each inner node is fixed. Which operators can be used as non-terminal nodes and which auxiliary data or values can be used as terminal nodes besides the attributes should be specified before the evolution process.

In the example of Fig. 2, there are four attributes (x_1, x_2, x_3, x_4) used in the routing policy. Each time the priority of a task is calculated, we replace the terminal nodes with their real values and evaluate the expression tree.

GP is an evolutionary algorithm that can evolve flexibly complex arithmetic expressions using the variable-length tree representation. The search space contains all valid expression trees that each non-terminal node of a tree satisfies the arity-constraint.

B. Linear Representation

Linear representation may be the simplest numeric representation. A routing policy with linear representation is a linear combination of the attributes:

$$f_\psi(\mathbf{x}) = \sum_{i=1}^{n} \psi_i x_i. \tag{9}$$

where the genotype ψ equals to a vector containing the weights of attributes. Since it has the same number of elements as the attribute vector, the search space is $\mathbb{R}^n$ where n is the number of attributes. Simple heuristics like nearest-first can be taken as a special case of linear representation where the weights of other attributes except for the distance are zero.

When a linear-representation routing policy is applied, we should be aware of that the attributes that are identical for different tasks are useless, i.e. the attributes that are not task-related. These attributes only change the priority values but do not change the priority rank.

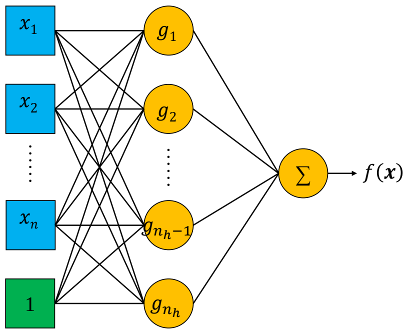


Fig. 3. Artificial neural network representation.

C. Artificial Neural Network Representation

Artificial neural network techniques, especially deep neural networks, have been thoroughly investigated in the research field of machine learning [63]. However, their potential to act as routing policies has not been widely studied.

Although the theoretical study in [64] shows that any continuous function can be approximated by a feed-forward neural network with a single hidden layer, researchers tend to use deeper structures to approximate more complex situations nowadays [63]. As a preliminary study, we will not use very complex or deep neural network structures in this paper, because it is both hard to gain insights into the problem and hard to be interpret or understood. Also, it runs counter to our original intention of real-time decision making. Thus, we start with a simple feed-forward neural network first. The input layer of the neural network contains all attributes and a bias node. The output layer has only one neuron to calculate the priority value. The linear function is used as the activation function of the neuron in the output layer. For each neuron in the hidden layers, the leaky rectified linear unit function, i.e. Leaky ReLU, is used as the activation function [65]. Thus, an ANN with single hidden layer can be represented as:

$$f_\psi(\mathbf{x}) = \sum_{i=1}^{n_h} \psi_i^o \cdot \max(0.1 \cdot f_{\psi_i^h}(\mathbf{x}), f_{\psi_i^h}(\mathbf{x})), \tag{10}$$

$$f_{\psi_i^h}(\mathbf{x}) = \sum_{j=1}^{n} \psi_{i,j}^h x_j + \psi_{i,n+1}^h. \tag{11}$$

where $\psi = (\psi^h, \psi^o)$ and n_h represents the number of hidden nodes in the hidden layer. $\psi^h = (\psi_1^h, \dots, \psi_{n_h}^h)$ represents the weight matrix of the links between the input layer and the hidden layer in which each ψ_i^h represents a weight vector of between the ith hidden node and input layer. ψ^o represents the weight vector of the links between the hidden layer and the output layer. Since Leaky ReLU is a non-linear activation function, the ANN representation is a non-linear representation.

An ANN structure with only one hidden layer is shown in Fig. 3. The upper bound of the complexity of the function

TABLE I
INFORMATION OF THE UCARP INSTANCES

name	#vertex	#task	#vehicle	demand
Ugdb1	12	22	5	identical
Ugdb2	12	26	6	identical
Ugdb3	12	22	5	identical
Ugdb4	11	19	4	identical
Ugdb5	13	26	6	identical
Ugdb6	12	22	5	identical
Ugdb7	12	22	5	identical
Ugdb8	27	46	10	different
Ugdb9	27	51	10	different
Ugdb10	12	25	4	different
Ugdb11	22	45	5	different
Ugdb12	13	23	7	different
Ugdb13	10	28	6	different
Ugdb14	7	21	5	different
Ugdb15	7	21	4	different
Ugdb16	8	28	5	different
Ugdb17	8	28	5	different
Ugdb18	9	36	5	different
Ugdb19	8	11	3	different
Ugdb20	11	22	4	different
Ugdb21	11	33	6	different
Ugdb22	11	44	8	different
Ugdb23	11	55	10	different

TABLE II
ATTRIBUTE INFORMATION

Name	Description	Type
CFH	Cost From Here (vehicle position) to the task	1
CFD	Cost From Depot to the head node of the task	1
CFR1	Cost From the closest other Route to the task	1
CTT1	Cost from the Task to its closest remaining Task	1
CTD	Cost from the Task to Depot	1
RQ1	Remaining Capacity of the alternative vehicle	1
SC	Serving Cost of the task	1
DEM	expected DEMand of the task	1
DEM1	expected DEMand of the closet remaining task	1
DC	Deadheading cost of the task	1
FULL	FULLness of the vehicle	2
FRT	Fraction of Remaining Tasks	2
FUT	Fraction of unassigned Tasks	2
RQ	Remaining Capacity of the vehicle	2
CR	Cost to Refill	2

Type 1 means task-related. Type 2 means task-unrelated. A task-related attribute takes different values for different tasks when a vehicle makes a decision.

that an ANN can approximate is determined by the number of hidden layers and the number of neurons in each hidden layer. More neurons and layers mean higher upper bound of the complexity, also bring higher pressure to its optimization.

IV. EXPERIMENTAL SETUP

A. Benchmark Problem and Simulation Settings

In the experiment, a commonly used test set Ugdb is adopted [7]. It contains 23 UCARP instance. The information of each instance is shown in Table I.

To simulate the uncertain characteristics, the demand of each task and the traversal cost of each edge of each UCARP instance are assumed to follow the two truncated normal distributions [8], [23]:

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

where $d(\bar{e})$ and $t(\bar{e})$ represent the original values taken from the static instances. If a sampled task demand is smaller than 0, it is set to 0. If a sampled traversal cost is smaller than 0, it is set to ∞, representing a broken edge.

Following the experimental settings in [8], [23], for each algorithm on each UCARP instance, the experiment is divided into two parts, training and test. The batch simulation method is used in the training process. N_s samples of the instance are generated to evaluate the routing policies. Each generation uses N_s/N_g samples, where N_g is the maximum generation number. In the test phase, N_s unseen samples different from the training samples will be generated to evaluate the final routing policy. The average solution quality of that routing policy on the N_s test samples is taken as the test performance. In the following experiment, we set N_s to 2000.

B. Attribute Set and Representation Parameters

A set of suitable attributes is important to routing policy generation. Theoretically, more relevant attributes would provide more information about the problem, but the search space also grows exponentially along with the number of attributes. In this paper, the frequently-used attributes by GPHH are adopted [23]. They are shown in Table II.

- For the tree representation, besides these attributes, it also takes random constant values as terminals. Also, we need to define the operators before optimization. Following [8], [23], $\{+, -, \times, \div, \min, \max\}$ are adopted.
- For the linear representation, we have made it clear that only task-related attributes are useful. According to Table II, there are 10 task-related attributes. Thus, the genotype of a linear-representation routing policy is a 10-dimension real-value vector.
- For the neural network representation, since the activation function of the neurons in the hidden layer can provide non-linear mapping, all attributes are useful. However, if the attributes are normalized, we can find that FULL and RQ have a simple relationship that 1-FULL=RQ. Since the attributes form a linear combination before the activation function in a neuron, it is easy to represent the relationship between FULL and RQ. Thus, in the following experiments, we omit FULL for the neural network presentation. Given 14 attributes, the dimensionality of the search space depends on the number of hidden layers and the number of neurons in each hidden layer. In our experiment, starting from a basic neural network structure with only one hidden layer containing 14 neurons equal to the number of attributes [20], we increase the size of the neural network in two ways, i.e. one with twice of the neurons in the single hidden layer and another with two hidden layers. The purpose is to check whether it is easy to improve the performance by simply increasing the number of neurons or hidden layers.

C. Normalization

For the tree representation, normalization is an optional choice [20]. Through concatenating related attributes by the operators, each subtree (sub-expression) can have their prac-

TABLE III
PARAMETERS OF THE GP ALGORITHM

parameter	value	parameter	value
population size	1024	generation	100
selection method	tournament (7)	max tree depth	8
crossover rate	0.8	mutation rate	0.15
reproduction rate	0.05	elitism size	10

tical meaning like RQ-DEM. Also, the dividing operator '÷' can play the role of normalization.

Linear representation also does not rely on normalization that much. However, since different attributes have different value ranges and the genotype of the linear-representation routing policy is the real-value weights, normalization is at least helpful for the optimization algorithm to set uniform boundaries of the search space. Although we have used Leaky ReLU as the activation function of the neural network representation, it is still helpful to apply normalization to the attributes due to the same reason of using normalization for linear-representation. Thus, for the two numeric representations, we use normalized attributes. All the attributes in Table II have a natural lower bound zero. The upper bound of each attribute is calculated by using the static CARP instance in the experiment. We normalize each attribute by dividing its corresponding upper bound value. Due to the uncertainty, sometimes, the normalized attribute can be larger than one, but in most cases, the normalized attribute value is within [0, 1]. Corresponding to the normalized attribute value, the value range of each variable of ψ is bounded in $[-1, 1]$.

D. Optimization Algorithms and Algorithm Parameters

In the following experiment, the tree-representation routing policies are evolved by the GP algorithm. The numeric-representation routing policies are evolved by the covariance matrix adaptation evolution strategy (CMA-ES) algorithm [66].

1) Genetic Programming: A standard GP is used to evolve the tree-representation routing policies. The ramped half and half initialization method is adopted. The other parameters are shown in the Table III.

Since the elitism strategy will save the best 10 individuals directly to the next generation each time, we can directly take the best individual of the last generation as the overall best one. This strategy is commonly used in GPHH methods [8], [23].

2) CMA-ES: CMA-ES is used to optimize the weights of the two numeric-representation routing policies. There are two reasons we choose CMA-ES rather than the other evolutionary algorithms. First, it is one of the state-of-the-art algorithms for continuous optimization. Second, using CMA-ES can avoid re-evaluation. For example, if a particle swarm optimization (PSO) algorithm is applied, in each generation, not only the newly-generated solutions should be evaluated, but the personal best solutions should be re-evaluated because of the batch simulation process. Since CMA-ES generates new solutions based on a distribution, there is no need to re-evaluate any other solutions. However, this scheme brings another problem that which solution should be chosen as the final solution. Because of the solution generation scheme, i.e. sampling a distribution, and the step-size control method, CMA-ES may generate a whole population of solutions where no one is better than the best solution of the previous generation. Since the focus of this paper is not to solve the re-calculation problem or propose a new optimization method, we use a simple strategy that the solution with the best training fitness value that the algorithm ever found is taken as the final solution.

Regarding the parameters of CMA-ES, there are mainly two parameters that we need to adjust, the population size λ and the initial deviation of the distribution σ_0. In this experiment, we set $\sigma_0 = 0.4$. Empirical study in [67] showed that for different problems, using different population sizes would greatly affect the performance. Generally, for a unimodal problem, the original setting $\lambda = 4 + \lfloor 3\ln(D) \rfloor$ is good enough, where D represents the dimension of the problem. However, for multi-modal problems, it is better to set a big λ value that is larger than D. Currently, there has not been any work that reveals the characteristic of the search space of numeric-representation routing policies, but it is easy to know that the search space of the neural network representation is multimodal since the neurons of the hidden layer do not distinguish from each other. For the linear representation, the search space may be instance-related. Considering both the effectiveness of CMA-ES and the computational burden, we set $\lambda = 2 \cdot D$ [68]. The maximal generation number is still set to 100 to keep a same number of training samples with GP. In order to show the advantage of CMA-ES, we also compared CMA-ES with another state-of-the-art evolutionary large-scale optimization algorithm called competitive swarm optimizer in the supplementary material, and the results showed that CMA-ES is better.

Each algorithm is executed on each instance 30 independent times to get the statistic information. All algorithm are implemented based on the Evolutionary Computation Java (ECJ) package [69]. The CPU of the computing platform is Core i7-6700 3.40GHz.

V. EXPERIMENTS AND ANALYSES

In this section, we first test the ANN representation with different structure configurations. Then, the three representations are compared in different aspects, including effectiveness, converging speed, and the requirement of the quantity of training samples. Also, the characteristic of the real-value search space of the linear representation is investigated. To gain more insight into the problem and the representations, we have made more analyses in the supplementary material to show when the tree representation can significantly outperform the linear representation and how different attributes contribute to the success of the heuristic for different instances.

A. Artificial Neural Network Representation

Before conducting the overall comparison among the three representations, we test the ANN representation under the three different structures first. The comparison is shown in Table IV including the mean value and the standard deviation. Wilcoxon rank-sum tests are conducted between the smallest

TABLE IV
COMPARISON OF THE OBJECTIVE VALUES OF ANN REPRESENTATION UNDER DIFFERENT STRUCTURES.

Instance	14x1	28x1	14x2
Ugdb1	344.93(1.34)	345.24(1.23)	344.95(2.24)
Ugdb2	364.72(2.3)	364.77(2.69)	364.98(3.28)
Ugdb3	307.77(1.07)	308.01(1.54)	307.77(1.09)
Ugdb4	319.53(1.01)	319.25(0.56)	319.79(0.98)
Ugdb5	419.66(5.91)	420.45(6.99)	420.06(5.63)
Ugdb6	333.24(4.37)	333.15(4.3)	332.83(3.01)
Ugdb7	350.89(5.59)	348.28(2.28)	354.31(7.61)
Ugdb8	432.53(6.57)	433.62(5.47)	439.66(4.71)(−)
Ugdb9	393.34(5.09)	394.73(5.11)	394.74(4.75)
Ugdb10	296.43(3.79)	295.85(3.72)	297.81(2.75)
Ugdb11	426.59(4.81)	427.41(4.53)	430.17(5.35)(−)
Ugdb12	623.5(6.68)	624.16(8.13)	627.75(10.53)
Ugdb13	581.26(3.53)	580.93(2.99)	581.26(3.7)
Ugdb14	106.36(0.49)	106.51(0.4)(−)	106.45(0.78)
Ugdb15	58.13(0.11)	58.13(0.09)	58.2(0.23)
Ugdb16	134.76(1.5)	133.85(0.75)(+)	134.42(1.4)
Ugdb17	91.11(0.11)	91.15(0.13)	91.2(0.11)(−)
Ugdb18	166.18(0.99)	166.64(0.93)(−)	166.38(1.09)
Ugdb19	61.43(0.72)	61.09(0.42)	61.41(0.77)
Ugdb20	126.94(1.07)	126.63(0.78)	127.23(1.15)
Ugdb21	163.06(0.56)	162.88(0.47)	163.12(0.49)
Ugdb22	209.11(1.24)	209.06(0.91)	210.25(1.15)(−)
Ugdb23	248.51(1.26)	249.39(1.1)(−)	249.4(0.93)(−)

[1] 14x1 represents the basic structure with a single hidden layer containing 14 neurons. 28x1 represents the structure with 28 neurons in the single hidden layer. 14x2 represents the structure with two hidden layers, each containing 14 neurons.
[2] '+' means the larger ANN structure gets significantly better results than the basic ANN structure. '-' means the larger ANN structure gets significantly worse results than the basic ANN structure.

TABLE V
COMPARISON RESULTS OF FRIEDMAN TEST

Instance	Tree	Linear	ANN
Ugdb1	# / = / =	= / # / =	= / = / #
Ugdb2	# / + / =	− / # / −	= / + / #
Ugdb3	# / + / +	− / # / =	− / = / #
Ugdb4	# / = / =	= / # / =	= / = / #
Ugdb5	# / + / =	− / # / −	= / + / #
Ugdb6	# / + / −	− / # / −	+ / + / #
Ugdb7	# / + / =	− / # / −	= / + / #
Ugdb8	# / = / +	= / # / =	− / = / #
Ugdb9	# / + / +	− / # / =	− / = / #
Ugdb10	# / + / +	− / # / −	− / + / #
Ugdb11	# / = / −	− / # / −	+ / + / #
Ugdb12	# / + / +	− / # / =	− / = / #
Ugdb13	# / = / =	= / # / =	= / = / #
Ugdb14	# / + / =	− / # / −	= / + / #
Ugdb15	# / + / =	− / # / −	= / + / #
Ugdb16	# / + / =	− / # / −	= / + / #
Ugdb17	# / = / +	= / # / =	− / = / #
Ugdb18	# / = / =	= / # / =	= / = / #
Ugdb19	# / + / +	− / # / −	− / + / #
Ugdb20	# / = / =	= / # / =	= / = / #
Ugdb21	# / − / −	+ / # / =	+ / = / #
Ugdb22	# / = / =	= / # / +	= / − / #
Ugdb23	# / − / −	+ / # / =	+ / = / #
w/t/l	12/9/2(linear) 7/12/4(ann)	2/9/12(tree) 1/12/10(ann)	4/12/7(tree) 10/12/1(linear)

[1] '#' means not available. '+' means the algorithm gets significantly better results than the compared algorithm. '-' means the algorithm gets significantly worse results than the compared algorithm.
[2] 'w/t/l' shows on how many instances the algorithm wins, ties, and loses the competitions against the algorithm in the bracket.

network structure and the other two structures. The significance level is set to 0.05 with Bonferroni correction.

Generally, the results show that adding more layers or neurons did not improve the performance. Only on Ugdb16, the Wilcoxon rank-sum test shows that the performance of using 28 neurons is better. On the other instances, the performance either keeps the same level or become worse. There are two possible reasons of this phenomenon.

- The optimizer, i.e. CMA-ES, is not capable to exhibit the best performance of the neural network presentation. It is easy to understand that the neural network with 28 neurons in the hidden layer is more powerful than the neural network with 14 neurons. However, the results show that there are three instances on which the 28x1 neural network performs worse. This situation is even severe for the network with more layers. This fact proves that the growth of the neural network size indeed brings high pressure to its optimization.
- It is hard to estimate how many neurons are needed for a UCARP instance. Although theoretical analysis states that the single hidden layer structure is able to approximate any continuous function, it may use a large number of neurons, which exceeds our computational capacity.

Based on the results, we can partially answer the second question raised in the introduction section that it is not an easy job to design a good function format/structure for the numeric representation. At least, simply adding more neurons or layers for an ANN representation is not a good choice. In the following experiments, we use the neural network representation with 14 neurons in the single hidden layer as the representative to compare with other algorithms.

B. Comparison of Different Representations

The comparison results among the three representations are shown in Fig. 4 and Table V. Table V shows result of the nonparametric statistical test. To compare these three algorithms together, we use the Friedman test with Holm as the post-hoc multiple comparison. Fig. 4 displays the performance of each representation in the form of box-plot. The upper labels of each figure show the mean objective values. Meanwhile, we have tested the five path scanning heuristics [70] on each instance, the best performance of the five path scanning heuristics is taken as the baseline that is shown as the dashed line in the figures.

The non-parametric statistical test results in Table V show the overall performance of each representation in a comparing way. The results are affected not only by the representations but also the optimizers. The results in Fig. 4 show more detailed information, which can be used to judge whether the difference is caused by the representations or the optimizers. In the following analysis, we take the best case performance, i.e. the minimum objective value, and the overall performance, i.e. the statistical significance test results, of each representation on each instance among 30 runs as two measurements to analyze the results. According to these two measurements, the results can be classified into four situations:

- The first situation is that the tree representation has better best case performance and its overall performance is also

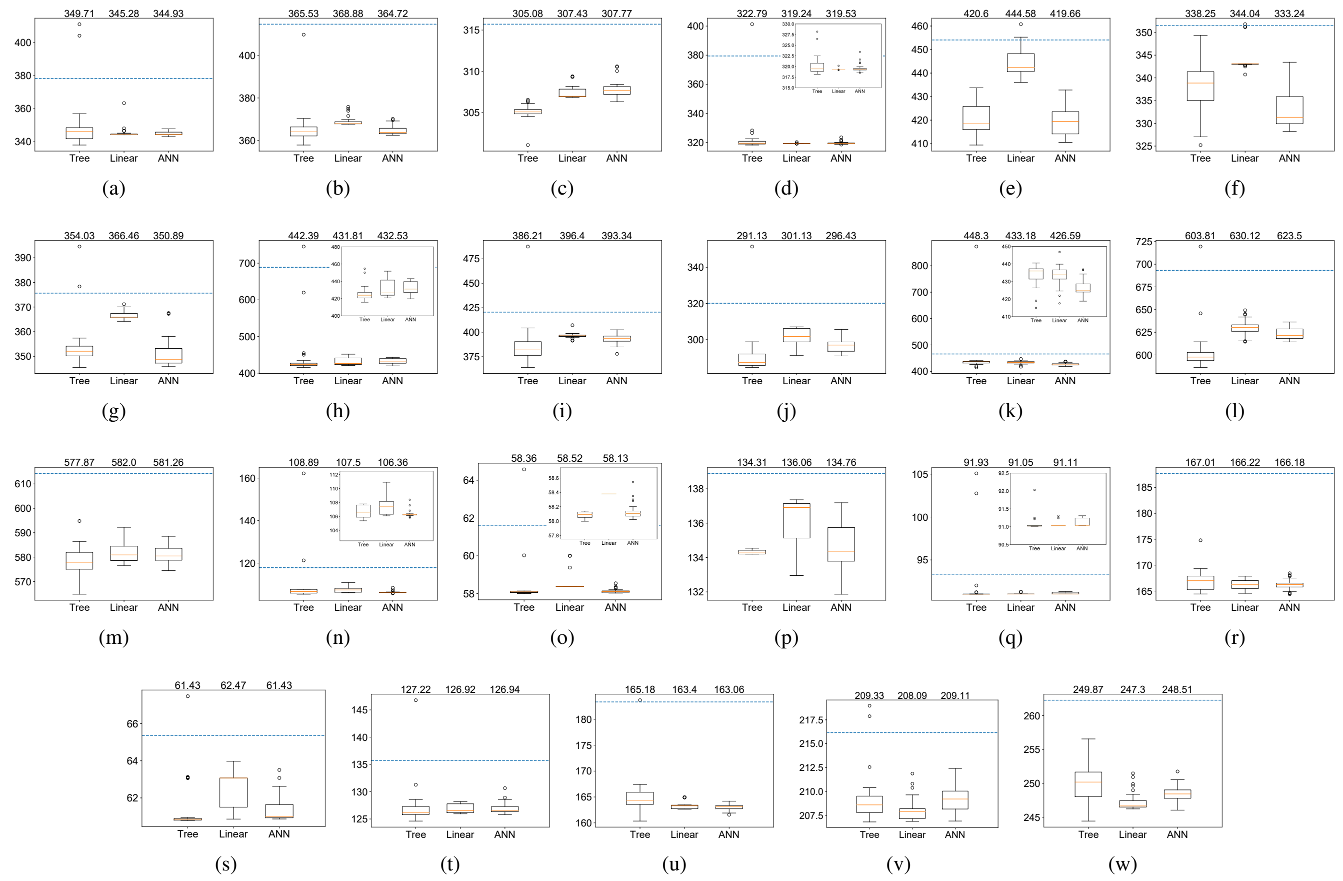


Fig. 4. Experimental results of the three representations on the Ugdb instances. (a)-(w) represent Ugdb1-Ugdb23. The x-axis shows the representations. The y-axis represents the test performance. The upper labels are the mean objective values that are not shown in the box-plots.

better than the two numeric representations. This situation happened on four instances, Ugdb3, Ugdb9, Ugdb10, and Ugdb12. On Ugdb3 and Ugdb9, the ANN representation showed better best case performance than the linear representation but it is not capable enough to compete with the tree representation. On Ugdb10 and Ugdb12, the ANN representation showed similar performance with the linear representation. This situation indicates that the complexity of the problem is superlinear, but the ANN representation can only partially characterize the non-linear relationship between the attributes of a task and its priority.

- The second situation is that the tree representation has better best case performance but its overall performance is not significantly better than the two numeric representations. This situation happened on seven instances, Ugdb1, 2, 8, 13, 20, 21, and 23. Among these instances, the representative cases are Ugdb13, Ugdb21, and Ugdb23. On Ugdb13, the tree representation is clearly more capable to represent better routing policies than the linear representation and the ANN representation, but the Friedman test shows that the three representations have similar performance. On Ugdb21 and Ugdb23, the Friedman test shows that both the linear representation and the ANN representation have significantly outperformed the tree representation. This situation indicates that although the tree representation is powerful to represent very good routing policies, its optimization is not easy. Sometimes, it is harder than doing real-value optimization for a numeric representation that is not as capable as the tree representation.
- The third situation is that the tree representation does not have better best case performance than the other two numeric representations but it has achieved better overall results. This situation only happened once on Ugdb19. Fig. 4(s) shows that the minimum objective values of the three representations are basically identical but GP evolved the best tree representation routing policy more stably than CMA-ES optimizing routing policies with the other two representations.
- The last situation is that the tree representation does not have better best case performance than the other two numeric representations and its overall performance is not better either. This situation happened on the rest eleven instances. Among these eleven instances, ten of them have a similar pattern that the tree representation has similar best case performance with the ANN representation or with both ANN and linear. The only exception is Ugdb16 corresponding to Fig. 4(p) on which the best test performance of the ANN representation is better than the tree representation.

Then, the results shown in Fig. 4 are analysed in a comparing way with the results conducted on the dynamic scheduling problem in [20], [25].

- In the previous studies of the scheduling problems, the experiment showed a clear pattern that the tree representation was significantly better than the ANN representation and both of them were significantly better than the linear representation. However, from Fig. 4, we never see this pattern on a specific UCARP instance. In most cases, the ANN representation either reached the same level of the tree representation like Fig. 4(e)(f)(g)(o) showing or degenerated to the level of the linear representation like Fig. 4(c)(i)(l). It seldom locates in the middle of the other two representations.
- Also, unlike the observations on the dynamic scheduling problems that the tree representation could always achieve significantly better results than the linear representation, on UCARP, it has been defeated by the linear representation twice, even on the largest instance Ugdb23.

In general, the tree representation has shown a better capability to approximate suitable routing policies on different instances than the other two numeric representations, which is a big advantage. Although the capability of the ANN representation is not as powerful as the tree representation, it provides competitive performance because of the stability of real-value optimization. Based on the experimental analysis, if there is a new UCARP instance, the tree representation routing policy evolved by GP is still the first choice, and the ANN representation can be considered as a potential alternative.

C. Comparison of Convergence Speed

From the perspective of evolutionary optimization, we care not only the quality of the final results, but also the convergence speed of each algorithm. The convergence curves of the algorithms are drawn by using the generation number as the x-axis. The results on six representative instances are displayed in Fig. 5. However, different algorithms generate different numbers of solutions in each generation. Thus, combined with the curves, how long each algorithm takes to generate and evaluate solutions in each generation is reported in Table VI.

According to the results in Fig. 5 and Table VI, we can get the following observations.

- Checking the results from the perspective of the generation number, the convergence speed of GP evolving tree-representation routing policies is faster than both CMA-ES evolving linear-representation routing policies and neural-network-representation routing polices. Since we give the same number of training samples in each generation, the results show that GP can use less training samples to get a good routing policy than CMA-ES.
- Checking the results from the perspective of the execution time, definitely the linear representation is the fastest. As to the ANN representation and the tree representation, it is hard to say that which one is faster since it is related to both the optimizer and the instance. Basically, most of the execution time comes from the simulation process that is related to the number of solutions generated in each generation. GP usually needs a large number of individuals in the population to keep the diversity. The large number of parameters of an ANN representation also requires a large number of individuals in an EC optimizer. Thus, their converging speed in terms of execution time is slow.
- Fig. 5(a) and Fig. 5(c) show a pattern that although the starting points are different, but when the algorithms finally converged, they got similar results. Fig. 5(b) and Fig. 5(d) show another pattern that the complex of the tree-representation routing policy is naturally higher than the linear representation and the ANN representation. Finally, the tree-representation routing policy gets better performance than the other two. Between the linear representation and the ANN representation, since ANN can approximate more complex functions, it gets better results. Fig. 5(e) shows an interesting result that the training performance of the tree representation is quite stable, but its test performance is not stable. The reason may come from the search space of the arithmetic expression that a small change of the tree may influence the performance a lot. Compared with it, the numeric representation is more stable. Fig. 5(f) shows a parameter sensitive case that the linear representation get the best result. We believe that the tree-representation routing policy must has approximated the basic relationship between the priority of task and the attributes. However, it is hard for GP to do more real-value parameter refinement.

TABLE VI
EXECUTION TIME OF EACH GENERATION OF THE COMPARED ALGORITHMS

instance	Tree	Linear	ANN
Ugdb1	9.02	0.18	4.82
Ugdb5	10.21	0.25	6.97
Ugdb11	21.47	0.65	17.47
Ugdb15	5.29	0.11	2.84
Ugdb17	8.35	0.19	5.54
Ugdb23	44.35	1.16	30.05

The unit of time is second.

Overall, evolving a good linear or ANN representation routing policy is computationally cheaper than evolving a tree-representation routing policy currently. However, in the future, a more powerful numeric-representation routing policy may have more parameters/weights that needs to be optimized. Correspondingly, the EC optimizer may require more computational budget to optimize a high-dimensional problem.

D. Investigation of the Number of Training Samples

From the perspective of evolutionary learning, how many training samples are required by each algorithm is also critical since it affects how many data the algorithm needs in real-world applications. According to [71], we have theoretically analyzed the relationship between the number of training samples and the accuracy of the estimation in the supplementary material. Although it is shown that 2000 is a reasonable number but it is still hard to give a universal setting that fits all algorithms and problems. To further test the sensitivity of each algorithm to the number of training samples, we conduct an empirical comparison by setting another two N_s values, 500 and 1000, for training. Still, 2000 test samples are used for test. The experimental results are shown in Table VII.

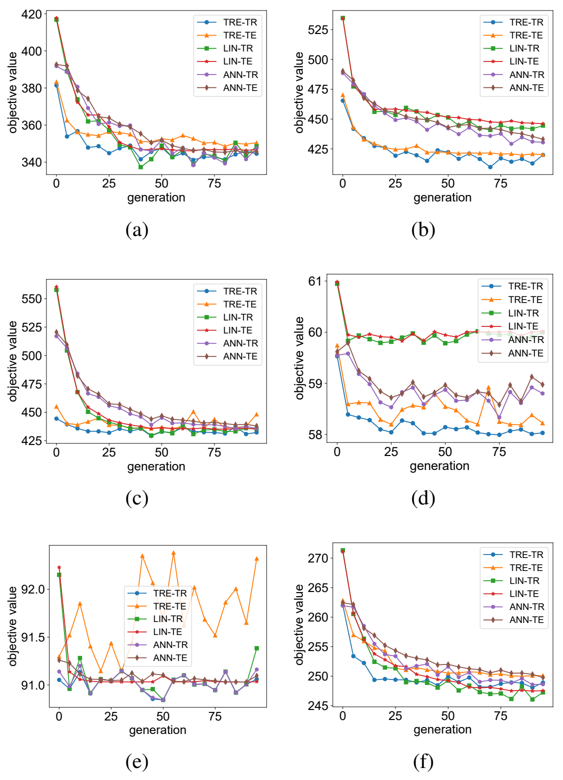


Fig. 5. Convergence curves of the compared algorithms. (a) Ugdb1, (b) Ugdb5, (c) Ugdb11, (d) Ugdb15, (e) Ugdb17, (f) Ugdb23. 'TRE', 'LIN', and 'ANN' represent the tree representation, the linear representation, and the artificial neural network representation, respectively. 'TR' means training; 'TE' means test.

- For the GP algorithm, there is not a general pattern showing whether it prefers more training samples. On Ugdb2, Ugdb4, Ugdb5, Ugdb6, and Ugdb20, the results show that the performance is getting better and better with the increase of the number of training samples. However, On Ugdb8, Ugdb9, Ugdb10, and Ugdb12, the results show that GP evolved better routing policies under 1000 training samples than 500 and 2000 samples. Finally, the results on Ugdb11 and Ugdb14 show that GP even performs worse and worse along with the increase of the number of training samples.
- For the linear-representation routing policies evolved by CMA-ES, the performance generally gets better along with the increase of the training samples. However, since its complexity is low, the improvement is limited. The negative effect of increasing the number of training samples only shows on Ugdb12.
- For the ANN representation, the pattern is stable that the performance is kept improved with the increase of the training samples. The improvement is significant on some instances, such as Ugdb5, Ugdb6, Ugdb8, and Ugdb11.

Overall, the results show that the number of training samples can affect both the training of the tree-representation routing policy and the numeric-representation routing policy. The pattern how the number of training samples affect the numeric representation is clear that more training samples lead to better results. However, the pattern how it affect the tree representation is not very clear. On the one hand, the results reveal that training a tree-representation routing policy does not require much training data as the ANN representation, which is the advantage of the tree representation. On the other hand, it gives pressure to the best training configuration that increasing the amount of training data may lead to worse performance, which is a disadvantage.

E. Discussion

The experiments show that the capability and the potential of the tree representation are better than the numeric representations. However, using GPHH to train an effective tree-structure heuristic is more computationally expensive than the other two representations. Meanwhile, the tree representation seems easier to overfit. Increasing the amount of training data in many cases is not helpful to overcome the overfitting problem or to improve the performance. On the contrary, the capability and the potential of the two numeric representations are weaker than the tree representation, but they are easier to be trained. Meanwhile, increasing the amount of training data is helpful to improve their performance in most cases.

There are two main reasons leading to the above results. The first one is that the search space of the tree representation is much larger than the numeric search space. Theoretically, GPHH can evolve any arithmetic expression consisting of the attributes and the operators with the tree representation. The upper limit of capability of the tree representation is very high. However, for the two numeric representations, once the function structure like the network architecture of the ANN representation and the value range of the weights are determined, the upper limit of capability is defined. Compared to the tree representation, the search space of a numeric-representation heuristic with a fixed structure is much smaller. Thus, the tree representation has better potential and capability, but it is harder to evolve an effective tree-representation heuristic than numeric-representation heuristics since searching a larger space is more difficult.

The second reason is that the characteristics of the search spaces of different representations are different. For a tree-representation heuristic, a small change of the genotype like replacing a small sub-tree with another sub-tree may lead to a huge change of its phenotype. However, for a numeric-representation heuristic, although we have found that the search space is multimodal, usually a small change of the coefficients will not make a huge difference. It indicates that the search space of a numeric-representation heuristic is smoother. Thus, the performance of GPHH evolving a tree-representation heuristic is less stable than CMA-ES evolving a numeric-representation heuristic.

The overfitting problem of GPHH may relate to both reasons. If GPHH finds a heuristic that fits a part of the training data, due to the two reasons, it would be hard to guide the algorithm to find a more general one in such a huge search space. As to the numeric representation, the search space is much smaller and smoother than the search space of a tree-representation heuristic. Thus, the optimization algorithm, i.e. CMA-ES, can search the space more thoroughly than GPHH.

TABLE VII
TEST PERFORMANCE UNDER DIFFERENT NUMBER OF TRAINING SAMPLES

	Tree			Linear			ANN		
instance	500	1000	2000	500	1000	2000	500	1000	2000
Ugdb1	350.08(7.58)	357.32(23.42)	**349.71(16.48)**	348.35(6.34)	347.56(6.46)	**345.28(3.54)**	347.06(5.06)	345.97(1.68)	**344.93(1.34)**
Ugdb2	367.25(5.04)	366.2(6.52)	**365.53(8.94)**	370.29(3.89)	371.36(4.24)	**368.88(2.21)**	368.09(3.45)	365.66(2.37)	**364.72(2.3)**
Ugdb3	305.7(1.59)	311.48(27.12)	**305.08(0.94)**	308.27(1.43)	308.01(1.12)	**307.43(0.79)**	312.8(4.47)	308.32(1.41)	**307.77(1.07)**
Ugdb4	325.5(16.97)	323.23(14.88)	**322.79(14.99)**	319.84(1.34)	319.44(0.21)	**319.24(0.18)**	321.63(4.18)	320.8(2.16)	**319.53(1.01)**
Ugdb5	429.68(21.76)	427.72(23.39)	**420.6(6.19)**	449.29(8.11)	449.29(5.71)	**444.58(5.81)**	428.17(11.49)	422.51(8.73)	**419.66(5.91)**
Ugdb6	344.07(6.22)	339.59(4.35)	**338.25(5.17)**	344.78(4.44)	345.14(3.49)	**344.04(2.96)**	340.92(8.03)	336.12(6.15)	**333.24(4.37)**
Ugdb7	**352.9(6.13)**	354.46(5.96)	354.03(9.65)	370.39(2.04)	369.3(2.3)	**366.46(1.75)**	357.55(9.7)	**350.73(5.13)**	350.89(5.59)
Ugdb8	441.3(46.18)	**426.61(6.08)**	442.39(67.85)	437.03(9.22)	437.34(13.01)	**431.81(10.4)**	444.14(9.05)	438.55(6.83)	**432.53(6.57)**
Ugdb9	388.23(11.08)	**384.4(8.53)**	386.21(21.71)	400.95(4.92)	398.25(3.28)	**396.4(2.6)**	397.87(6.05)	395.49(6.55)	**393.34(5.09)**
Ugdb10	293.21(8.1)	**289.54(3.24)**	291.13(12.07)	302.56(5.01)	301.55(5.41)	**301.13(5.14)**	299.8(6.02)	298.47(5.42)	**296.43(3.79)**
Ugdb11	**436.51(5.17)**	437.93(16.15)	448.3(80.38)	434.87(7.03)	433.55(5.47)	**433.18(5.59)**	434.02(7.95)	431.03(7.93)	**426.59(4.81)**
Ugdb12	609.85(26.62)	**601.24(8.25)**	603.81(24.3)	**626.3(9.28)**	628.26(10.16)	630.12(8.63)	639.86(13.11)	625.97(9.79)	**623.5(6.68)**
Ugdb13	580.07(7.44)	580.93(12.37)	**577.87(6.46)**	585.63(3.79)	582.4(5.57)	**582.0(4.5)**	585.78(5.61)	582.79(4.12)	**581.26(3.53)**
Ugdb14	107.33(1.92)	**107.04(1.55)**	108.89(10.46)	108.2(1.96)	108.04(2.26)	**107.5(1.25)**	107.22(0.97)	106.6(0.58)	**106.36(0.49)**
Ugdb15	58.7(2.16)	58.51(1.99)	**58.36(1.23)**	58.53(0.5)	**58.51(0.43)**	58.52(0.44)	58.34(0.29)	58.25(0.22)	**58.13(0.11)**
Ugdb16	134.61(0.64)	134.43(0.15)	**134.31(0.11)**	136.37(1.25)	**135.79(1.32)**	136.06(1.38)	136.5(2.1)	**134.37(1.14)**	134.76(1.5)
Ugdb17	92.42(3.41)	**91.82(2.88)**	91.93(3.28)	**91.04(0.04)**	91.13(0.1)	91.05(0.06)	**91.05(0.06)**	91.23(0.17)	91.11(0.11)
Ugdb18	167.71(4.48)	167.32(1.53)	**167.01(2.04)**	167.73(2.27)	166.68(1.27)	**166.22(1.04)**	167.93(2.62)	166.83(0.8)	**166.18(0.99)**
Ugdb19	63.8(2.19)	61.8(1.05)	**61.43(1.43)**	63.82(2.16)	63.07(1.17)	**62.47(1.0)**	62.98(1.98)	61.68(1.25)	**61.43(0.72)**
Ugdb20	130.66(13.15)	128.15(6.27)	**127.22(3.92)**	128.35(1.82)	127.32(1.17)	**126.92(0.81)**	128.84(1.23)	127.45(1.22)	**126.94(1.07)**
Ugdb21	165.36(2.42)	165.36(2.82)	**165.18(3.89)**	163.88(1.48)	163.84(0.97)	**163.4(0.83)**	163.83(1.53)	163.45(0.78)	**163.06(0.56)**
Ugdb22	209.06(1.61)	**208.65(1.31)**	209.33(2.75)	209.81(1.89)	209.17(1.4)	**208.09(1.2)**	211.12(1.54)	210.03(1.45)	**209.11(1.24)**
Ugdb23	250.62(2.73)	**249.68(2.35)**	249.87(2.7)	248.3(1.71)	247.64(1.12)	**247.3(1.44)**	250.48(1.61)	249.58(1.81)	**248.51(1.26)**

VI. CONCLUSION AND FUTURE WORK

The goal of this paper was to cover the research shortage of hyper-heuristics with numeric representations for routing problems by answering the following three questions. 1) Whether are the numeric representations able to represent good routing policies, and what kind of instance each representation suits? 2) How difficult is it to optimize the parameters of a numeric-representation routing policy? 3) How much training data is required by each representation? Through thoroughly investigation, we have successfully obtained the answers of these three questions.

1) For the first question, the tree representation was still more adaptive to represent a good routing policy than the other two representations, but benefit from the stability of the real-value optimization, the ANN representation showed competitive performance on a majority of the instances. This result indicates that the numeric representations have good potential to represent good routing policies. Meanwhile, we have found that the problems having sparsely connected networks or densely connected networks with low variance of edge cost are usually simple that a linear representation heuristic can handle well. The moderately connected road networks that do not have a neat structure and clear partitions are complex, which require more complex representations.
2) For the second question, through investigating the characteristics of the search space of the linear representation, we have found that the search space in terms of continuous real-value optimization is multimodal and contains flat fitness areas. Thus, it is not easy to evolve/learn a good numeric-representation routing policy.
3) For the last question, the two numeric representations have a constant preference for more training data that the performance was always improved with the growth of the number of training samples. As to the tree representation, although it requires less training data than the ANN representation, its preference is hard to predict. Sometimes the increase of the training data lead to better performance. Sometimes the increase of the training data affects the algorithm in an opposite way.

Based on these answers, we can have a rough guideline:

1) When the problem is simple that meets the description in Section I of the supplemental material and the computational budget is low, the linear representation is recommended.
2) When the problem is complex and the computational budget is sufficient, the tree representation is recommended. If the road network has a neat structure like clustering structure, the ANN representation is also recommended.
3) When there are a small number of training samples, the tree representation is recommended, but when there are a lot of training samples, the GP algorithm should be carefully designed to avoid the overfitting problem.

It is worth noting that the experiments conducted in this paper are based on the elementary tree and numeric representations without auxiliary techniques such as multi-objective optimization, niching, and neural architecture search. These techniques can make up for the defects of the representations to a certain extent. Considering the limitations of our research, there are three promising directions to continue the study of hyper-heuristic methods for routing problems in the future. 1) For the numeric representations, we have seen that increasing the complexity of the representation indeed improves the effectiveness. However, how complex the representation should be is a question. A straightforward idea is to use evolutionary neural network techniques to decide the structure of the ANN representation automatically, but this may cause extremely high computational cost. Another promising way is to design a specific representation that follows the 'situation-decision' form. Since on some instances we have witnessed the success of the linear representation, if we think the linear representation only fitting one situation of the whole routing process, the whole routing process actually only consists of several different situations. For situation recognition, some famous algorithms including radial basis function network and gradient boost can be applied. For decision making, the linear representation or the ANN representation can be applied. Nevertheless, it still requires high expertise to design the structure but once it is made, we think it is both effective and explainable. 2) For the tree representation, it naturally has the ability to recognize different situations because of the operators it involved. Although it has shown remarkable performance in the experiments, we have found that it is not very stable and it encounters the overfitting problem frequently even with more training data. Combining numeric optimization methods with GP is good way to stabilize its performance. However, due to the uncertainty of the problem and the non-differentiable operators, advanced optimization algorithms are required rather than traditional mathematical optimization methods based on gradient. Meanwhile, the overfitting problem may require new genetic operators to maintain the diversity of the population and new selection methods to choose the individual with high generalization ability. 3) From the perspective of application, in the future research, more constraints and characteristics of routing problems can be considered, and the evolutionary transfer optimization techniques can be developed to improve the potential generalization ability of each representation.

References

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

[2] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *Eur. J. Oper. Res.*, vol. 225, no. 1, pp. 1–11, 2013.

[3] Á. Corberán, R. Eglese, G. Hasle, I. Plana, and J. M. Sanchis, "Arc routing problems: A review of the past, present, and future," *Networks*, vol. 77, no. 1, pp. 88–115, 2021.

[4] S. Pelletier, O. Jabali, and G. Laporte, "The electric vehicle routing problem with energy consumption uncertainty," *Transport. Res. B Meth.*, vol. 126, pp. 225–255, 2019.

[5] C. E. Gounaris, W. Wiesemann, and C. A. Floudas, "The robust capacitated vehicle routing problem under demand uncertainty," *Oper. Res.*, vol. 61, no. 3, pp. 677–693, 2013.

[6] J. Zhang, W. H. Lam, and B. Y. Chen, "A stochastic vehicle routing problem with travel time uncertainty: trade-off between cost and customer service," *Netw. Spat. Econ.*, vol. 13, no. 4, pp. 471–496, 2013.

[7] Y. Mei, K. Tang, and X. Yao, "Capacitated arc routing problem in uncertain environments," in *Proc. CEC2010*. IEEE, 2010, pp. 1–8.

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

[9] M. W. Ulmer, D. C. Mattfeld, M. Hennig, and J. C. Goodson, "A rollout algorithm for vehicle routing with stochastic customer requests," in *Logistics management*. Springer, 2016, pp. 217–227.

[10] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *J. Oper. Res. Soc.*, vol. 64, no. 12, pp. 1695–1724, 2013.

[11] N. Pillay and R. Qu, *Hyper-heuristics: Theory and applications*. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer, 2018.

[12] I. Benyahia and J.-Y. Potvin, "Decision support for vehicle dispatching using genetic programming," *IEEE Trans. Syst. Man Cybern. A Syst. Humans*, vol. 28, no. 3, pp. 306–314, 1998.

[13] F. Rothlauf, *Representations for Genetic and Evolutionary Algorithms*. Berlin, Heidelberg: Springer, 2006.

[14] G. R. Raidl and J. Gottlieb, "Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem," *Evol. Comput.*, vol. 13, no. 4, pp. 441–475, 2005.

[15] F. Rothlauf, "On the locality of representations," in *Proc. GECCO2003*, 2003, pp. 1608–1609.

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

[17] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'neill, "Grammar-based genetic programming: a survey," *Genet. Program. Evol. Mach.*, vol. 11, no. 3, pp. 365–396, 2010.

[18] Y. Mei and M. Zhang, "Genetic programming hyper-heuristic for multi-vehicle uncertain capacitated arc routing problem," in *Proc. GECCO2018 Companion*, 2018, pp. 141–142.

[19] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling," *IEEE Trans. Cybern.*, 2020.

[20] J. Branke, T. Hildebrandt, and B. Scholz-Reiter, "Hyper-heuristic evolution of dispatching rules: A comparison of rule representations," *Evol. Comput.*, vol. 23, no. 2, pp. 249–277, 2015.

[21] 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," *Evol. Comput.*, vol. 28, no. 2, pp. 289–316, 2020.

[22] ——, "Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem," in *Proc. GECCO2017*, 2017, pp. 290–297.

[23] S. Wang, Y. Mei, and M. Zhang, "A multi-objective genetic programming hyper-heuristic approach to uncertain capacitated arc routing problems," in *Proc. CEC2020*. IEEE, 2020, pp. 1–8.

[24] M. A. Ardeh, Y. Mei, and M. Zhang, "A novel multi-task genetic programming approach to uncertain capacitated arc routing problem," in *Proc. GECCO2021*, 2021, pp. 759–767.

[25] S. Nguyen and M. Zhang, "A pso-based hyper-heuristic for evolving dispatching rules in job shop scheduling," in *Proc. CEC2017*. IEEE, 2017, pp. 882–889.

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

[27] E. B. Tirkolaee, I. Mahdavi, and M. M. S. Esfahani, "A robust periodic capacitated arc routing problem for urban waste collection considering drivers and crew's working time," *Waste Manag.*, vol. 76, pp. 138–146, 2018.

[28] S. Gonzalez-Martin, A. A. Juan, D. Riera, M. G. Elizondo, and J. J. Ramos, "A simheuristic algorithm for solving the arc routing problem with stochastic demands," *J. Simul.*, vol. 12, no. 1, pp. 53–66, 2018.

[29] H. Longo, M. P. De Aragao, and E. Uchoa, "Solving capacitated arc routing problems using a transformation to the cvrp," *Comput. Oper. Res.*, vol. 33, no. 6, pp. 1823–1837, 2006.

[30] D. J. Bertsimas, "A vehicle routing problem with stochastic demand," *Oper. Res.*, vol. 40, no. 3, pp. 574–585, 1992.

[31] T. Le-Anh, J. R. van der Meer *et al.*, "Testing and classifying vehicle dispatching rules in three real-world settings," *J. Oper. Manag.*, vol. 22, no. 4, pp. 369–386, 2004.

[32] G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno, "Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies," *Eur. J. Oper. Res.*, vol. 151, no. 1, pp. 1–11, 2003.

[33] M. C. Fu, F. W. Glover, and J. April, "Simulation optimization: a review, new developments, and applications," in *Proc. WSC2005*. IEEE, 2005, pp. 83–95.

[34] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches: revisited," in *Handbook of metaheuristics*. Springer, 2019, pp. 453–477.

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

[36] G. L. Pappa, G. Ochoa, M. R. Hyde, A. A. Freitas, J. Woodward, and J. Swan, "Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms," *Genet. Program. Evol. Mach.*, vol. 15, no. 1, pp. 3–35, 2014.

[37] S. Luo, "Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning," *Appl. Soft Comput.*, vol. 91, p. 106208, 2020.

[38] P. Garrido and M. C. Riff, "Dvrp: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic," *J. Heuristics*, vol. 16, no. 6, pp. 795–834, 2010.

[39] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "Grammatical evolution hyper-heuristic for combinatorial optimization problems," *IEEE Trans. Evol. Comput.*, vol. 17, no. 6, pp. 840–861, 2013.

[40] C. Moustakas, *Heuristic research: Design, methodology, and applications*. Newbury Park, California, USA: Sage Publications, 1990.

[41] N. A. El-Sherbeny, "Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods," *J. King Saud University-Sci.*, vol. 22, no. 3, pp. 123–131, 2010.

[42] S. Liu, K. Tang, and X. Yao, "Automatic construction of parallel portfolios via explicit instance grouping," in *Proc. AAAI2019*, vol. 33, no. 01, 2019, pp. 1560–1567.

[43] ——, "Generative adversarial construction of parallel portfolios," *IEEE Trans. Cybern.*, 2020.

[44] K. Tang, S. Liu, P. Yang, and X. Yao, "Few-shots parallel algorithm portfolio construction via co-evolution," *IEEE Trans. Evol. Comput.*, vol. 25, no. 3, pp. 595–607, 2021.

[45] K. C. Tan, L. Feng, and M. Jiang, "Evolutionary transfer optimization-a new frontier in evolutionary computation research," *IEEE Computational Intelligence Magazine*, vol. 16, no. 1, pp. 22–33, 2021.

[46] L. Feng, Y. Huang, L. Zhou, J. Zhong, A. Gupta, K. Tang, and K. C. Tan, "Explicit evolutionary multitasking for combinatorial optimization: A case study on capacitated vehicle routing problem," *IEEE Trans. Cybern.*, vol. 51, no. 6, pp. 3143–3156, 2020.

[47] T. Weise, A. Devert, and K. Tang, "A developmental solution to (dynamic) capacitated arc routing problems using genetic programming," in *Proc. GECCO2012*, 2012, pp. 831–838.

[48] S. Wang, Y. Mei, M. Zhang, and X. Yao, "Genetic programming with niching for uncertain capacitated arc routing problem," *IEEE Trans. Evol. Comput.*, 2021.

[49] M. A. Ardeh, Y. Mei, and M. Zhang, "Genetic programming with knowledge transfer and guided search for uncertain capacitated arc routing problem," *IEEE Trans. Evol. Comput.*, 2021.

[50] K. Lewczuk, J. Żak, D. Pyza, and I. Jacyna-Gołda, "Vehicle routing in an urban area: Environmental and technological determinants," *WIT Trans. Built Envir.*, vol. 130, pp. 373–384, 2013.

[51] K. Sim and E. Hart, "A combined generative and selective hyper-heuristic for the vehicle routing problem," in *Proc. GECCO2016*, 2016, pp. 1093–1100.

[52] J. Jacobsen-Grocott, Y. Mei, G. Chen, and M. Zhang, "Evolving heuristics for dynamic vehicle routing with time windows using genetic programming," in *Proc. CEC2017*. IEEE, 2017, pp. 1948–1955.

[53] 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 Trans. Evol. Comput.*, vol. 18, no. 2, pp. 193–208, 2013.

[54] ——, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Trans. Evol. Comput.*, vol. 17, no. 5, pp. 621–639, 2012.

[55] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Comput. Ind. Eng.*, vol. 54, no. 3, pp. 453–473, 2008.

[56] D. Jakobović and L. Budin, "Dynamic scheduling with genetic programming," in *Proc. EuroGP2006*. Springer, 2006, pp. 73–84.

[57] Y. Zhou, J.-j. Yang, and Z. Huang, "Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming," *Int. J. Prod. Res.*, vol. 58, no. 9, pp. 2561–2580, 2020.

[58] J. C. Hershauer and R. J. Ebert, "Search and simulation selection of a job-shop sequencing rule," *Manag. Sci.*, vol. 21, no. 7, pp. 833–843, 1975.

[59] P. O'Grady and C. Harrison, "Search-based job shop scheduling and sequencing: Extensions to the search sequencing rule," *Math. Comput. Model.*, vol. 13, no. 3, pp. 13–27, 1990.

[60] A. M. Kuczapski, M. V. Micea, L. A. Maniu, and V. I. Cretu, "Efficient generation of near optimal initial populations to enhance genetic algorithms for job-shop scheduling," *Inf. Technol. Control*, vol. 39, no. 1, 2010.

[61] T. Eguchi, F. Oba, and S. Toyooka, "A robust scheduling rule using a neural network in dynamically changing job-shop environments," *Int. J. Manuf. Technol. Manag.*, vol. 14, no. 3-4, pp. 266–288, 2008.

[62] T. Hildebrandt and J. Branke, "On using surrogates with genetic programming," *Evol. Comput.*, vol. 23, no. 3, pp. 343–367, 2015.

[63] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[64] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[65] J. Yu, M. Tan, H. Zhang, D. Tao, and Y. Rui, "Hierarchical deep click feature prediction for fine-grained image recognition," *IEEE Trans. Pattern Anal. Mach Intell.*, 2019.

[66] N. Hansen, "The cma evolution strategy: a comparing review," *Towards a new evolutionary computation*, pp. 75–102, 2006.

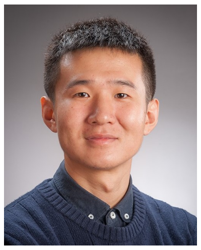
[67] N. Hansen and S. Kern, "Evaluating the cma evolution strategy on multimodal test functions," in *Proc. PPSN2004*. Springer, 2004, pp. 282–291.

[68] Y.-H. Jia, Y. Mei, and M. Zhang, "Contribution-based cooperative co-evolution for nonseparable large-scale problems with overlapping subcomponents," *IEEE Trans. Cybern.*, 2020.

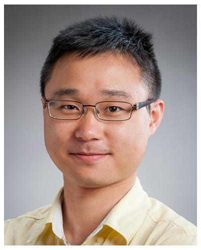
[69] E. O. Scott and S. Luke, "Ecj at 20: toward a general metaheuristics toolkit," in *Proc. GECCO2019 Companion*, 2019, pp. 1391–1398.

[70] B. L. Golden, J. S. DeArmon, and E. K. Baker, "Computational experiments with algorithms for a class of routing problems," *Comput. Oper. Res.*, vol. 10, no. 1, pp. 47–59, 1983.

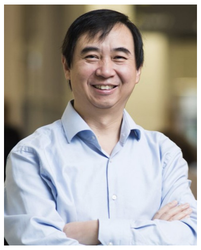
[71] S. Liu, K. Tang, Y. Lei, and X. Yao, "On performance estimation in automatic algorithm configuration," in *Proc. AAAI2020*, vol. 34, no. 03, 2020, pp. 2384–2391.



Ya-Hui Jia (M'20) received the B.Eng. and Ph.D. degrees from Sun Yat-sen University, China, in 2013 and 2019, respectively. He is currently an Assistant Professor at the School of Future Technology, South China University of Technology, Guangzhou, China. His research interests include evolutionary computation, combinatorial optimization, software engineering, cloud computing, and intelligent transportation.



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 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. He has over 150 fully referred publications in EC and Operations Research. He is the Chair of IEEE Taskforce on Evolutionary Scheduling and Combinatorial Optimisation. He is a reviewer of 50+ international journals.



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 Professor of Computer Science, Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) in the Faculty of Engineering. His current research interests include evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of image analysis, multi-objective optimization, feature selection and reduction, job shop scheduling, and transfer learning. Prof. Zhang is a Fellow of Royal Society of New Zealand, a Fellow of IEEE, and an IEEE Distinguished Lecturer.