Novel Ensemble Genetic Programming Hyper-Heuristics for Uncertain Capacitated Arc Routing Problem

Shaolin Wang
wangshao3@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

Yi Mei
yi.mei@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

Mengjie Zhang
mengjie.zhang@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

ABSTRACT

The Uncertain Capacitated Arc Routing Problem (UCARP) is an essential problem with many real-world applications. A major challenge in UCARP is to handle the uncertain environment effectively and reduce the recourse cost upon route failures. Genetic Programming Hyper-heuristic (GPHH) has been successfully applied to automatically evolve effective routing policies to make real-time decisions in the routing process. However, most existing studies obtain a single complex routing policy which is hard to interpret. In this paper, we aim to evolve an ensemble of simpler and more interpretable routing policies than a single complex policy. By considering the two critical properties of ensemble learning, i.e., the effectiveness of each ensemble element and the diversity between them, we propose two novel ensemble GP approaches namely DivBaggingGP and DivNichGP. DivBaggingGP evolves the ensemble elements sequentially, while DivNichGP evolves them simultaneously. The experimental results showed that both DivBaggingGP and DivNichGP could obtain more interpretable routing policies than the single complex routing policy. DivNichGP can achieve better test performance than DivBaggingGP as well as the single routing policy evolved by the current state-of-the-art GPHH. This demonstrates the effectiveness of evolving both effective and interpretable routing policies using ensemble learning.

CCS CONCEPTS

• **Theory of computation** → *Routing and network design problems.*

KEYWORDS

genetic programming, hyper-heuristic, ensemble learning, uncertain capacitated arc routing problem

ACM Reference Format:
Shaolin Wang, Yi Mei, and Mengjie Zhang. 2019. Novel Ensemble Genetic Programming Hyper-Heuristics for Uncertain Capacitated Arc Routing Problem. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321797

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321797

1 INTRODUCTION

The Capacitated Arc Routing Problem (CARP) [9] was first proposed by Golden and Wong in 1981. CARP is a classic combinatorial optimisation problem with broad real-world applications such as waste collection [1] and winter gritting [10, 11]. The problem aims to serve a set of edges in a graph using a fleet of vehicles with minimum cost subject to the following constraints.

- All the routes start and end at the depot;
- Each required edge must be served exactly once (but can be traversed multiple times);
- The total demand served by each route does not exceed the capacity of the vehicle.

CARP has been proven to be NP-hard [9] and has received many research interests. Many studies have been made on CARP (e.g. [5, 35]). However, there is one limitation for most of the previous studies. They assume that all the problem parameters (e.g., task demand and travel time) are fixed and known in advance. This is far away from the reality, where the problem parameters are often stochastic. For example, in waste collection, the exact amount of waste to be collected is unknown in advance and is revealed during the process of collection. The travel time can also be stochastic, depending on the real-time traffic.

This paper focus on the Uncertain CARP (UCARP) [22], which better reflects the reality. UCARP considers four types of stochastic factors, which are the presence of tasks, demands of tasks, the presence of paths and traversal costs of paths.

In addition to the NP-hardness inherited from CARP, a major extra challenge in UCARP is the *route failures* caused by the stochastic task demand. That is, the actual demand of a task can be greater than expected, and the remaining capacity of the vehicle becomes insufficient to complete the service. In this case, a route failure occurs, and the vehicle has to go back to the depot to refill in the middle of the service. This can induce huge recourse cost.

Traditional optimisation approaches such as mathematical programming [9] and evolutionary algorithms [16] are not valid when applied to solve UCARP due to the challenge of handling route failures. Specifically, they try to optimise a solution beforehand and modify the preplanned solution by a recourse operator when route failures occur. However, such modification may not be practical due to the inflexibility of the preplanned solution. On the other hand, completely reoptimising the entire remaining solution from scratch can be time-consuming.

Routing policy [29] is a promising technique to handle the uncertain environment in UCARP. Instead of pre-planning the whole solution, a routing policy constructs the solution gradually in real

time. Routing policy can respond to the environment change efficiently, and much more flexible than traditional optimisation approaches, as it may construct very different solutions in different environment realisations.

However, manually designing effective routing policies is very hard and time consuming, as the decisions depend on many factors such as the graph topology and optimised objective(s). Genetic Programming Hyper-Heuristic (GPHH) has been successfully applied to automatically evolve effective routing policies which outperform the solutions optimised by the state-of-the-art optimisation approaches [14, 18, 20].

When evolving routing policies by GPHH, the interpretability of the evolved policies is an important aspect to guarantee the confidence of the users. However, the current GPHH approaches cannot achieve satisfactory interpretability.

In this paper, we aim to evolve both effective and interpretable routing policies by GPHH. To improve the interpretability, we reduce the maximal depth of the GP individuals during the evolutionary process, since smaller GP individuals tend to be simpler and more interpretable. However, smaller GP individuals are also weaker and less effective than large ones. To compensate for the loss of effectiveness, we propose to evolve an ensemble of small and interpretable GP individuals. Based on the principle of ensemble learning [37], we expect the aggregation of these GP individuals can lead to a much more effective ensemble.

The overall research goal of this paper is to evolve effective and interpretable ensemble of routing policies for UCARP. To evolve effective and interpretable ensemble of routing policies, two critical properties of ensemble learning must be considered, i.e., the effectiveness of each ensemble element and the diversity between them. In this paper, we propose two novel ensemble GP approaches namely DivBaggingGP and DivNichGP. DivBaggingGP evolves the ensemble elements sequentially, while DivNichGP evolves them in parallel. Specifically, this paper has the following research objectives:

(1) To develop two novel ensemble GP approaches namely DivBaggingGP and DivNichGP;
(2) Evolve a group of interpretable routing policies as ensembles using two novel ensemble GP approaches, and examine their performance.
(3) Analyse the behaviour of the evolved ensembles.

The rest of this paper is organised as follows. Section 2 introduces the background. Section 3 describes the ensemble approaches proposed for UCARP. Section 4 gives the experimental studies. Finally, Section 5 gives the conclusions and possible future directions.

2 BACKGROUND

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

2.1 Problem Description

A UCARP instance can be represented by a connected graph $G(V, E)$, where V is the set of vertices, E is the set of edges. Each edge $e \in E$ has a positive random deadheading cost $\tilde{dc}(e)$, indicating the cost of traversing the edge. A set of edge *tasks* $E_R \subseteq E$ are required to be served by the vehicles. Each task $e_R \in E_R$ has positive random demand $\tilde{d}(e_R)$ which represents the demand to serve, and a positive serving cost $sc(e_R)$. A set of vehicles with capacity Q are located at the depot $v_0 \in V$. The goal is to minimise the total cost of serving all the tasks in R. A vehicle must start at v_0 and finish at v_0, and the total served demand for each route cannot exceed the capacity of the vehicle.

Note that a UCARP instance contains many random variables, each of which can have different samples. Correspondingly, a UCARP instance contains different samples. In a UCARP instance *sample*, each random variable takes a realised value. However, the actual demand of a task is unknown until it has been served, and the actual deadheading cost of an edge is unknown in advance and is revealed during the vehicle traversing over the edge.

The nature of the uncertainty of the environment will lead to failures during execution. There are two kinds of failures may occur during the process.

- *Route failure*: the actual demand of a task exceeds the remaining capacity of the vehicle.
- *Edge failure*: the edge in the route become infeasible.

When a route failure occurs, the solution is repaired by a recourse operator. A typical recourse operator is that the vehicle goes back to the depot to refill and return to the failed task to complete the remaining service. When an edge failure occurs, a detour is found by shortest path finding (e.g., Dijkstra's algorithm) under the current situation.

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

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

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

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

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

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

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

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

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

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

2.2 Related Work

The static CARP is a classic combinatorial optimisation problem and has received extensive research interest. Golden and Tong [9] developed an integer linear programming model and solved it using the branch-and-cut algorithm. However, the approach is limited to only small instances. To obtain reasonably good solutions in limited time budget and for larger problem instances, various approximate solution methods such as tabu search [2, 7, 12, 21], genetic algorithm [15], memetic algorithm [16, 23, 31], and ant colony optimisation [4, 17] have been proposed. Although these approaches showed good performance on CARP, they are not directly applicable to UCARP, because they cannot handle the route failures effectively.

For solving UCARP, two types of approaches have been proposed. The proactive approaches [8, 32, 33] attempt to find robust solutions that can handle all the possible realisations of the random variables. In contrast, reactive approaches mainly use GPHH [18, 20, 24] to evolve routing policies which are used to make real-time decisions. Weise et al. [34] first proposed a GPHH for UCARP with a single vehicle, and examined its performance. Liu et al. [18] designed a novel and effective meta-algorithm which filters irrelevant candidate task during the decision-making process and achieved better performance than the GPHH in [34]. After that, Mei et al. [24] extended the model from single-vehicle to multiple-vehicle version, so that the solution can be generated with multiple vehicles on the road simultaneously. A novel task filtering method and an effective look-ahead terminal were proposed by MacLachlan et al. [20] to improve the GPHH further.

Ensemble learning [37], as a popular machine learning method, has been widely used in classification problems [25, 30, 36]. It can improve performance effectively. Due to its effectiveness, various algorithms, such as decision tree [26], active example selection algorithm [25], neural network [36], SVM and Naive Bayes algorithm [30], and genetic programming [3, 6, 27] have been combined with ensemble learning.

There are two main challenges when combining ensemble learning with GPHH. First, compared with traditional ensemble learning such as decision tree learning, it is much more time consuming to evolve the routing policies in GPHH (involves many fitness evaluations). Second, it is very time-consuming to examine the performance of a routing policy on a sample (requires a simulation). This limits the number of samples GPHH can use during the evaluation. There has been no study in evolve ensembles of routing policies for UCARP so far. The most related works are the BaggingGP and BoostingGP [6] and the Cooperative Co-evolution GP (CCGP) [27] for evolving ensembles of dispatching rules for dynamic job shop scheduling. However, the BaggingGP and BoostingGP are based on unrealistic assumptions that there are already a set of candidate rules existing. CCGP, on the other hand, evolves the ensemble as a whole without considering the collaborations between the elements in the ensemble.

In this paper, we propose novel ensemble GPHH approaches that overcome the existing approaches by explicitly considering both the quality of each policy, and the collaboration between them in the ensemble.

3 PROPOSED APPROACHES

In this section, we propose two ensemble GPHH approaches, named Diverse Bagging GP (DivBaggingGP) and Diverse Niching Genetic Programming (DivNichGP), respectively. Both approaches explicitly consider the quality of the ensemble elements as well as the diversity between them. DivBaggingGP follows the standard bagging procedure in ensemble learning and evolves the ensemble elements sequentially. DivNichGP, on the other hand, takes advantage of the population in GP and evolves the ensemble elements in parallel.

In the following, we first describe the individual representation, which is common in the proposed algorithms. Then, we describe DivBaggingGP and DivNichGP one by one.

3.1 Individual Representation

During the GP process, a routing policy is represented as a Lisp tree, which is essentially an arithmetic priority function to determine which task from the pool of candidate tasks to serve next. Once a vehicle is ready to serve the next task, the routing policy is used to calculate the priorities of all the unserved tasks in the pool. Then, the task with the best (lowest) priority will be selected as the next task. The priority function is a combination of the state features, such as the cost from the current location to the candidate task, and the current remaining capacity of the vehicle. For example, if the priority function is "CFH + DEM", where CFH is the cost from the current location, and DEM is the expected demand of the candidate task, then the priority function tends to select the tasks that are closer to the current location and have smaller demand. An ensemble $\mathcal{RP}$ is represented as a set of routing policies.

During the decision-making process, the routing policy or ensemble is applied to select the next task to serve. For a single routing policy rp, the task with the best (lowest) priority will be selected, i.e. $t^* = \arg\min_{t \in pool}\{prio(t, rp)\}$. However, for an ensemble, the decision-making requires an aggregation method to combine the decisions made by its elements. Here, the decision of an ensemble $\mathcal{RP}$ is made by the following aggregation method.

(1) For each routing policy in the ensemble $rp \in \mathcal{RP}$, calculate the priority of each candidate task $prio(t, rp)$, $\forall t \in pool$.
(2) For each routing policy $rp \in \mathcal{RP}$, normalise the priority values of all the candidate tasks:

$$\overline{prio}(t, rp) = \frac{prio(t, rp) - \min\{prio(t', rp)\}}{\max\{prio(t', rp)\} - \min\{prio(t', rp)\}}. \quad (8)$$

(3) For each task $t \in pool$, calculate the aggregated priority: $prio_{agg}(t, \mathcal{RP}) = \sum_{rp \in \mathcal{RP}} \overline{prio}(t, rp)$.
(4) Select the next task $t^* = \arg\min_{t \in pool}\{prio_{agg}(t, \mathcal{RP})\}$.

The normalisation on the priority values is to avoid a single routing policy dominating the others in the ensemble.

3.2 DivBaggingGP

The DivBaggingGP algorithm evolves the ensemble elements sequentially. It consists of a number of *cycles*, each of which aims to evolve one ensemble element and add into the current ensemble. Algorithm 1 describes the pseudocode of DivBaggingGP. Initially, the ensemble $\mathcal{RP}$ is empty. Then, in each cycle, a new routing policy rp^* is evolved and added into the ensemble. At the beginning of each cycle, a training subset is randomly resampled from the training set (the same as bagging in ensemble learning), and the population is randomly reinitialised. Then, the population is evolved for *cycleLength* generations. In each generation, each individual rp in the population is first evaluated. Then, a new population is generated by the evolve() method, which consists of crossover, mutation and reproduction operators.

Algorithm 1: The DivBaggingGP algorithm

Input: Training set T_{train}, number of cycles *cycles*, cycle length *cycleLength*
Output: An ensemble of routing polices $\mathcal{RP}$

1 initialise an empty ensemble $\mathcal{RP} = \emptyset$;
2 **for** $c = 1 \rightarrow cycles$ **do**
  `// evolve the ensemble element for this cycle`
3 randomly sample a training subset $T' \subseteq T_{train}$;
4 initialise the population *pop*;
5 **for** $g = 0 \rightarrow cycleLength$ **do**
6 **for** $rp \in pop$ **do**
7 evaluate($rp, \mathcal{RP}$);
8 **end**
9 pop = evolve(pop);
10 **end**
11 find the best individual $rp^* \in pop$;
12 $\mathcal{RP} = \mathcal{RP} \cup rp^*$;
13 **end**
14 **return** $\mathcal{RP}$

In line 7 of Algorithm 1, the evaluation of a routing policy rp depends on the current $\mathcal{RP}$. First, the difference between rp and each ensemble element in $\mathcal{RP}$ is calculated. If rp behaves the same as some ensemble element, then it is ignored, and its fitness is set to infinity. Otherwise, it is added into the ensemble to form a temporary ensemble. Then, its fitness is set to the fitness of the temporary ensemble. The details of the evaluation are given in Algorithm 2.

To calculate the difference $\delta(rp, rp', \mathcal{RP}')$ between two routing policies in an ensemble, we employ the phenotypic characterisation for dispatching rules [13], which is an effective characterisation of the behaviours of heuristics. Instead of comparing the tree structure (genotype), the phenotypic characterisation compares the decisions made by the heuristics in different decision-making situations. In the context of UCARP, a decision-making situation cosists of an idle vehicle, a set of candidate tasks to be served, and the state features (e.g. the current routes). Given a set of decision-making

Algorithm 2: The evaluation in DivBaggingGP

Input: A routing policy rp, the current ensemble $\mathcal{RP}$
Output: The fitness of rp

1 form a temporary ensemble $\mathcal{RP}' = \mathcal{RP} \cup rp$;
  `// check the difference`
2 $\Delta = \infty$;
3 **for** $rp' \in \mathcal{RP}$ **do**
4 calculate the difference $\delta(rp, rp', \mathcal{RP}')$;
5 **if** $\delta(rp, rp', \mathcal{RP}') < \Delta$ **then**
6 $\Delta = \delta(rp, rp', \mathcal{RP}')$;
7 **end**
8 **end**
9 **if** $\Delta > 0$ **then**
10 $fit(rp) = \infty$; `// ignore duplicates`
11 **return** ;
12 **end**
  `// evaluate the temporary ensemble`
13 **for** *each training sample* $S \in T'$ **do**
14 apply $\mathcal{RP}'$ to S and calculate the total cost of the generated solution $tc(\mathcal{RP}', S)$;
15 **end**
16 $fit(rp) = \frac{1}{|T'|} \sum_{S \in T'} tc(\mathcal{RP}', S)$;

situations *DMS*, the phenotypic characterisation of a routing policy or an ensemble is a vector, in which each dimension is the task id selected in the corresponding decision-making situation.

The difference calculation is given in Algorithm 3. First, the phenotypic characterisation of the ensemble is calculated. Then, the difference between two routing policies in the ensemble is calculated as the Euclidean distance between the normalised priorities of the task selected by the ensemble by the two routing policies.

Algorithm 3: The difference calculation $\delta(rp_1, rp_2, \mathcal{RP})$

Input: An ensemble $\mathcal{RP}$, two routing policies $rp_1, rp_2 \in \mathcal{RP}$
Output: The difference between rp_1 and rp_2

1 set $\delta = 0$;
2 generate a set of decision-making situations *DMS*;
3 calculate the phenotypic characterisation for the ensemble $pc(\mathcal{RP})$;
4 **for** $i = 1 \rightarrow |DMS|$ **do**
5 get the task selected by $\mathcal{RP}$: $t_i = pc_i(\mathcal{RP})$;
6 calculate the normalised priority of t_i by rp_1: $\overline{prio}(t_i, rp_1)$;
7 calculate the normalised priority of t_i by rp_2: $\overline{prio}(t_i, rp_2)$;
8 $\delta = \delta + \left(\overline{prio}(t_i, rp_1) - \overline{prio}(t_i, rp_2)\right)^2$;
9 **end**
10 $\delta = \sqrt{\delta}$;
11 **return** δ;

The proposed DivBaggingGP is different from the existing bagging GP [6]. First, the existing bagging GP [6] evolves the combination of a set of existing elements, while DivBaggingGP evolves the

elements from scratch. Second, DivBaggingGP considers the difference between elements when forming the ensemble. This way, we can prevent having duplicate elements into the ensemble, leading to a biased ensemble.

3.3 DivNichGP

DivBaggingGP evolves the routing policies one by one. On the other hand, GP has a natural advantage of maintaining a population (set) of routing policies during the evolutionary process. Therefore, we can evolve the ensemble elements simultaneously in the population.

To evolve a diverse set of routing policies for the ensemble, we combine GP with the niching technique. Specifically, we employ the clearing method [28], which is a simple yet effective niching algorithm.

The pseudocode of DivNichGP is described in Algorithm 4. In each generation, the routing policies are first evaluated individually. Then, the clearing method is employed to penalise the routing policies that are similar to each other (line 10). This way, the subsequent evolution in evolve(*pop*) tends to select more diverse individuals as parents, and generate more diverse offsprings. At the end of the evolution, an ensemble is selected from the final population by a greedy strategy (Algorithm 6).

Algorithm 4: The DivNichGP algorithm

Input: Training set T_{train}, number of generations G
Output: An ensemble of routing polices $\mathcal{RP}$

1 initialise the population *pop*;
2 **for** $g = 1 \rightarrow G$ **do**
3 randomly sample a training subset $T' \subseteq T_{train}$;
4 **for** $rp \in pop$ **do**
  `// evaluate the routing policy rp`
5 **for** *each training sample* $S \in T'$ **do**
6 apply rp to S and calculate the total cost of the generated solution $tc(rp, S)$;
7 **end**
8 $fit(rp) = \frac{1}{|T'|} \sum_{S \in T'} tc(rp, S)$;
9 **end**
10 clearing(*pop*);
11 *pop* = evolve(*pop*);
12 **end**
13 $\mathcal{RP}$ = greedyEnsemble(*pop*);
14 **return** $\mathcal{RP}$;

The clearing method is described in Algorithm 5. First, the individuals in the population *pop* are sorted according to their fitness. Then, the sorted individuals are scanned one by one from the best to the worst. For each individual, if it is not penalised (fitness is smaller than ∞), a new niche is created around it. Then, for each remaining individual, if it is within the radius of the niche, then it is included into the niche, and the size n of the niche is incremented. Once the niche becomes full, all the subsequent individuals within the radius are penalised, and their fitness values are set to ∞. After the clearing, the penalised individuals become the worst ones in the population and are much less likely to be selected as parents for crossover and mutation.

Algorithm 5: The clearing method

Input: A population *pop*, a radius γ, and a capacity N
Output: Modified fitness for *pop*

1 sortFitness(*pop*);
2 **for** $i = 1 \rightarrow |pop|$ **do**
3 **if** $fit(pop_i) < \infty$ **then**
4 $n = 1$;
5 **for** $j = i + 1 \rightarrow |pop|$ **do**
6 **if** $fit(pop_j) < \infty$ *and* $\delta(pop_i, pop_j) < \gamma$ **then**
7 **if** $n < N$ **then**
8 $n = n + 1$;
9 **else**
10 $fit(pop_j) = \infty$;
11 **end**
12 **end**
13 **end**
14 **end**
15 **end**

The clearing method requires the calculation of the difference between two routing policies (line 6). Note that the difference calculation in Algorithm 3 cannot be directly used here, since there is no ensemble until the evolutionary process is finished. Therefore, we slightly modify the difference calculation. First, we calculate the phenotypic characterisations $pc(rp_1)$ and $pc(rp_2)$. Then, the difference between rp_1 and rp_2 is defined as follows.

$$\delta(rp_1, rp_2) = \sqrt{\sum_{i=1}^{|DMS|} (pc_i(rp_1) - pc_i(rp_2))^2}.$$

After the evolutionary process, the ensemble is selected from the final population greedily. The ensemble selection procedure is given in Algorithm 6. First, the routing policies are sorted by fitness. When considering each routing policy, if it is too similar to any of the elements in the current ensemble, then it is skipped. Otherwise, a temporary ensemble $\mathcal{RP}'$ is formed and evaluated on the validation set. If $\mathcal{RP}'$ is better than the current ensemble $\mathcal{RP}$, then the routing policy is included into the ensemble.

4 EXPERIMENTAL STUDIES

To evaluate the performance of the proposed DivBaggingGP and DivNichGP, we test them on a number of UCARP instances and compare with the SimpleGP [24], which evolves a single complex routing policy using GPHH and the CCGP [27] that evolves an ensemble by cooperative co-evolution.

4.1 Experiment Settings

We select 8 representative UCARP instances from the commonly used Ugdb and Uval datasets [18, 20, 24, 32, 33] in the experiments. The problem size of these instances varies from small (22 tasks and 5 vehicles) to large (97 tasks and 10 vehicles). Therefore, we can

Algorithm 6: The ensemble selection greedyEnsemble(*pop*)

Input: A population *pop*
Output: An ensemble $\mathcal{RP}$

```
sortFitness(pop);
RP = ∅, fit(RP) = ∞;
generate a validation set T_valid;
for i = 1 → |pop| do
    inNiche = false;
    for each rp ∈ RP do
        if δ(rp, pop_i) ≤ γ then
            inNiche = true;
            break;
        end
    end
    if inNiche = false then
        RP' = RP ∪ pop_i;
        fit(RP') = 1/|T_valid| Σ_{S∈T_valid} tc(RP', S);
        if fit(RP') < fit(RP) then
            RP = RP';
            fit(RP) = fit(RP');
        end
    end
    return RP;
end
return the ensemble RP;
```

evaluate the effectiveness of the proposed algorithms in different problem scenarios.

For each UCARP instance and each algorithm, the experiment consists of the training and test phases. First, a routing policy or an ensemble is obtained from the training phase based on the training set T_{train}. Then, the obtained policy or ensemble is tested on an unseen test set. In the experiments, we generate 500 test samples to avoid testing bias. The test performance is defined as the average total cost of the generated solutions over the 500 test samples, i.e.

$$\text{Perf}_{\text{test}}(rp) = \frac{1}{|T_{test}|} \sum_{S \in T_{test}} tc(rp, S).$$

During the training, all the algorithms use 5 samples during the evaluation. In SimpleGP, CCGP, and DivNichGP, the 5 training samples are re-sampled in each generation. In DivBaggingGP, the 5 training samples are re-sampled at the beginning of each cycle. In DivNichGP, the validation set for selecting the ensemble consists of 100 randomly generated samples.

In the experiments, the terminal set is given in Table 1. The function set includes $+, -, \times, /$, and binary operators min and max. The "/" operator is protected and returns 1 if divided by 0.

The population size is set to 1000 for all the compared algorithms. For CCGP, we set to 5 sub-populations to evolve 5 routing policies in the ensemble. Each sub-population has a population size of 200. The total number of generations is set to 100 for all the compared algorithms. For DivBaggingGP, there are 5 cycles to evolve 5 routing policies in the ensemble. The cycle length is set to 20 generations. This way, the total number of generations is 100 for DivBaggingGP, which is the same as other algorithms. All the methods use ramp-half-and-half initialisation and size-7 tournament selection. The crossover, mutation and reproduction rates are 80%, 15% and 5%, respectively. The maximal depth of CCGP, DivBaggingGP, and DivNichGP is 5 so that they can evolve smaller and more interpretable routing policies in the ensemble. The maximal depth for the SimpleGP is 8, which has been commonly used in previous studies (e.g., [18]).

Table 1: The terminal set in the experiments.

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

In DivNichGP, the clearing method requires two parameters: the radius γ and the capacity N. From preliminary experiments, we set the parameter values as $\gamma = 0$ and $N = 1$. In other words, the duplicate routing policies with exactly the same behaviour are ignored.

All the algorithms are implemented by the Evolutionary Computation Java (ECJ) package [19]. Each algorithm was run 30 times independently for each UCARP instance.

4.2 Experiment Results

Table 2 shows the mean and standard deviation of test performance of the compared algorithms. Wilcoxon rank sum test with significance level of 0.05 is used to compare each ensemble method with SimpleGP. For each ensemble method, if its test performance is statistically significantly lower (better) than, higher (worse) than, and comparable with that of SimpleGP, the corresponding entry is marked with (+), (-), or (=), respectively.

Table 2: The test performance of the compared algorithms. For each ensemble method, (+), (-) and (=) indicates it is significantly lower (better) than, higher (worse) than, and comparable with SimpleGP.

Instance	SimpleGP	CCGP	DivBaggingGP	DivNichGP
Ugdb1	354.8(15.0)	360.1(15.5)(-)	351.4(5.8)(=)	348.4(4.7)(+)
Ugdb2	377.1(26.6)	372.3(14.9)(=)	370.2(7.1)(=)	364.8(3.4)(+)
Ugdb8	499.8(35.5)	467.4(22.8)(+)	444.6(8.5)(+)	467.4(22.4)(+)
Ugdb23	252.1(3.3)	255.8(4.0)(-)	252.3(2.3)(=)	250.9(1.5)(=)
Uval9A	340.3(13.3)	338.8(13.1)(=)	337.2(5.1)(=)	333.4(2.3)(+)
Uval9D	478.3(18.7)	486.9(18.7)(-)	489.6(9.7)(-)	504.5(11.6)(-)
Uval10A	440.6(5.6)	448.2(16.6)(-)	440.8(5.5)(=)	439.4(3.3)(+)
Uval10D	630.2(62.7)	641.3(10.9)(-)	634.9(10.6)(-)	636.8(9.9)(-)

It can be seen from Table 2 that the test performance of CCGP is worse than SimpleGP. It performed worse than SimpleGP on 5 out of the total 8 instances, while outperformed SimpleGP on only 1 instance (Ugdb8). DivBaggingGP performed much better than CCGP, but still slightly worse than SimpleGP. It showed significantly better performance than SimpleGP on Ugdb8, while significantly worse performance on Uval9D and Uval10D. There was no statistical significance between DivBaggingGP and SimpleGP on the remaining 5 instances. DivNichGP, on the other hand, performed the best among the three compared ensemble methods and performed better than SimpleGP. It significantly outperformed SimpleGP on 5 instances (Ugdb1, Ugdb2, Ugdb8, Uval9A and Uval10A), and performed significantly worse on only 2 instances (Uval9D and Uval10D).

The reasons for the above observations can be as follows. CCGP evolves the ensemble as a whole without considering the diversity of the elements in the ensemble. Therefore, it is likely to evolve a biased ensemble in which one element dominates the decisions. In this case, the ensemble is essentially the same as the dominating element, which is weaker than the single routing policy evolved by SimpleGP (depth 5 versus depth 8). DivBaggingGP considers both the ensemble performance and diversity between the elements in the ensemble. However, it evolves the elements sequentially. The resource (20 generations) allocated for each element may be limited, and the evolution of different elements are independent and cannot help each other. Thus, the evolved elements may not be effective enough. DivNichGP also considers both the ensemble performance and diversity between the ensemble elements. In contrast with DivBaggingGP, DivNichGP takes advantage of the population and evolves the ensemble elements simultaneously. The evolution of different elements (niches) can help each other to evolve more effective ensemble elements. Therefore, DivNichGP outperformed the other ensemble algorithms as well as SimpleGP.

Table 3 shows the mean and standard deviation of the number of nodes in the routing policies or routing policies in the ensembles evolved by the compared algorithms. It can be seen from the table that all the ensemble algorithms evolved much smaller routing policies than SimpleGP. This is as expected, as they used a maximal depth of 5, while SimpleGP used a maximal depth of 8. A more interesting observation is that with the same maximal depth, different ensemble methods evolved different sizes of routing policies. The routing policies evolved by CCGP were the smallest. DivBaggingGP evolved larger routing policies, and DivNichGP obtained the largest routing policies (but still much smaller than the routing policies evolved by SimpleGP). This indicates that DivNichGP can evolve more effective ensemble elements than CCGP and DivBaggingGP. Note that a full tree with a depth of 5 should have $1 + 2 + 4 + 8 + 16 = 31$ nodes. From Table 3, DivNichGP seemed to make the use of the maximal depth better.

4.3 Further Analysis

Both DivBaggingGP and DivNichGP consider the difference between the ensemble elements explicitly when forming the ensemble. To verify the effectiveness of the consideration of the difference, we conduct two sets of comparisons. First, we compare DivBaggingGP with the BaggingGP without checking the difference (i.e. without lines 2–12 in Algorithm 2).

Table 3: The mean and standard deviation of the number of nodes in the routing policies or routing polices in the ensembles evolved by the compared algorithms.

Instance	SimpleGP	CCGP	DivBaggingGP	DivNichGP
Ugdb1	75.7(19.3)	9.5(6.5)	16.4(8.4)	21.3(5.1)
Ugdb2	68.3(16.3)	11.2(7.3)	15.4(8.4)	21.4(5.5)
Ugdb8	67.5(22.0)	11.4(6.2)	14.5(7.3)	23.4(3.7)
Ugdb23	68.3(16.9)	9.2(5.8)	15.6(7.3)	19.9(6.5)
Uval9A	65.3(21.6)	12.2(6.8)	14.3(7.4)	21.5(4.9)
Uval9D	68.3(11.9)	11.7(6.5)	16.1(7.3)	25.1(3.5)
Uval10A	58.1(17.7)	11.5(6.8)	14.1(6.9)	21.6(5.5)
Uval10D	65.7(13.3)	10.4(6.0)	15.4(7.2)	24.8(5.0)

Table 4 shows the mean and standard deviation of the test performance of BaggingGP and DivBaggingGP. From the table, we can see that the two algorithms showed similar performance on all the instances. This indicates that the removing duplicates from the ensemble cannot significantly improve the performance. This interesting observation may be due to the sequential evolution of ensemble elements. If the previous element(s) are stuck into local optima, it becomes hard to find other elements that are both effective and collaborate well with the existing local optimal elements.

Table 4: The test performance of BaggingGP and DivBaggingGP. (+), (-) and (=) indicates DivBaggingGP is significantly lower (better), higher (worse) and comparable with BaggingGP.

Instance	BaggingGP	DivBaggingGP
Ugdb1	353.0(7.6)	351.4(5.8)(=)
Ugdb2	374.1(11.4)	370.2(7.1)(=)
Ugdb8	445.2(9.4)	444.6(8.5)(=)
Ugdb23	252.8(3.1	252.3(2.3)(=)
Uval9A	337.8(5.7)	337.2(5.1)(=)
Uval9D	488.0(9.8)	489.6(9.7)(=)
Uval10A	441.1(5.2)	440.8(5.5)(=)
Uval10D	637.3(16.0)	634.9(10.6)(=)

Then, we compare DivNichGP with a simpleGP with Ensemble Selection (ESSimpleGP). ESSimpleGP is the same as DivNichGP, except that it does not have the clearing step (line 10 in Algorithm 4) to increase the diversity of the population.

The comparative results are shown in Table 5. It is clear that DivNichGP performed better than ESSimpleGP. This indicates that considering the diversity of the population is effective for DivNichGP.

Instead of predefined ensemble size (5 for CCGP and DivBaggingGP), the ensemble size of ESSimpleGP and DivNichGP depends on the ensemble selection method (Algorithm 6). Table 6 shows the mean and standard deviation of the ensemble size of the ESSimpleGP and DivNichGP. From the table, it can be seen that DivNichGP usually obtains a much larger ensemble size than ESSimpleGP. This is because ESSimpleGP does not consider the diversity of the population. As a result, the individuals are very similar in the final population, making it hard to find complementary ensemble elements. On the other hand, DivNichGP maintains a more diverse population, making it easier to form larger ensembles with complementary elements.

To gain further understanding of the behaviours of the routing policies in the ensemble, we select an effective ensemble evolved by

Table 5: The test performance of the ESSimpleGP and DivNichGP. (+), (-) and (=) indicates DivNichGP is significantly lower (better), higher (worse) and comparable with ESSimpleGP.

Instance	ESSimpleGP	DivNichGP
Ugdb1	355.4(9.7)	348.4(4.7)(+)
Ugdb2	372.1(5.4)	364.8(3.4)(+)
Ugdb8	473.8(27.9)	467.4(22.4)(=)
Ugdb23	253.7(3.1)	250.9(1.5)(+)
Uval9A	336.7(3.1)	333.4(2.3)(=)
Uval9D	507.5(12.5)	504.5(11.6)(=)
Uval10A	439.8(5.7)	439.4(3.3)(=)
Uval10D	646.0(13.6)	636.8(9.9)(=)

Table 6: The mean and standard deviation of the ensemble size of the ESSimpleGP and DivNichGP.

Instance	ESSimpleGP	DivNichGP
Ugdb1	2.9(2.2)	5.7(4.7)
Ugdb2	3.7(2.5)	9.2(5.5)
Ugdb8	1.3(1.8)	1.6(3.1)
Ugdb23	2.5(2.6)	7.2(7.3)
Uval9A	2.7(2.2)	8.9(5.2)
Uval9D	1.1(0.5)	1.2(1.1)
Uval10A	2.2(1.4)	6.0(3.7)
Uval10D	1.7(3.3)	1.8(3.6)

DivNichGP for Ugdb1. It contains 7 routing policies, and achieved a test performance of 348.69.

We applied the ensemble to a test sample and investigated the behaviour of the ensemble by tracing the rank of the tasks selected by the ensemble in each routing policy during the decision-making process. Table 7 shows such ranks. In the title row, there are 20 decision situations (DMS). For each decision situation, the corresponding column shows the rank of the task selected by the ensemble in each routing policy. For example, in decision situation 11, the task selected by the ensemble was ranked 0 by rp_2, rp_3, rp_6 and rp_7, while ranked 1 by rp_1, rp_4 and rp_5. A rank 0 indicates the routing policy makes consistent with the ensemble. A higher rank implies that the decision made by the routing policy is more distant from the ensemble decision.

Table 7: The rank of the tasks selected by the ensemble in each routing policy during a decision-making process.

DMS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
rp_1	2	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0
rp_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
rp_3	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
rp_4	1	0	0	1	0	3	0	0	0	3	1	0	1	0	0	0	0	0	2	0
rp_5	7	6	3	1	2	3	2	2	2	0	1	0	1	0	0	1	0	0	0	1
rp_6	1	1	0	1	0	1	0	0	1	0	0	0	1	0	4	0	3	0	0	1
rp_7	1	1	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1

From Table 7, one can see that the first three routing policies have the most consistent decisions with the ensemble (they have the most 0 ranks). The last four routing policies seemed not so important and may be removed from the ensemble without affecting the behaviour of the ensemble. In addition, the routing policies compliment with each other. For example, rp_2 dominates the decisions in all the decision situations except the 16th one, whose decision was determined by rp_3, rp_4 and rp_6.

Since the first three routing policies seem to be the most important ones in the ensemble, we focus on analysing the structure of them. The structure of the first three routing policies is given as follows.

$$rp_1 = \max\{\text{DEM1}, \text{CFH}\} * (\text{SC} + \text{CTD}) * \left(\frac{\text{CFH}}{\text{DC}} - \min\{\text{RQ}, 0.32\}\right),$$

$$rp_2 = \min\{\text{CFH}, \text{DC}\} * \text{CTD} - \frac{\text{DC}}{\min\{\text{CFH}, \text{DC}\}},$$

$$rp_3 = \max\{\text{FRT}, \text{CFH}\} * \text{DEM} * \text{CTD} * \left(\frac{\text{CFH}}{\text{DC}} - \min\{\text{FUT}, 0.32\}\right).$$

It is clear that all the three routing policies use the terminal CFH, which is known to be a very important terminal. rp_2 strongly emphasise on the importance of CFH and almost always select the tasks with small CFH. For example, if a task is adjacent to the current location, i.e. CFH = 0, then $rp_2 = 0 - 1 = -1$ for that task, which is smaller than any other task with CFH > DC. Such emphasis dominates the decisions in many decision situations. However, it is by no means the best option to always prefer closer tasks to the current location. Therefore, rp_1 and rp_3 considers other terminals as well. In particular, the sign of rp_1 and rp_3 depends on the relationship between CFH/DC and RQ and FUT, which represent the routing stage. Towards the end of the routing stage (RQ and FUT become smaller), the sign of rp_1 and rp_3 are more likely to be positive (CFH/DC becomes larger than $\min\{\text{RQ}, 0.32\}$ in rp_1 and $\min\{\text{FUT}, 0.32\}$ in rp_3), and the tasks with smaller DEM, SC and CTD will become more preferred. This is consistent with our intuition, as selecting the tasks with smaller DEM, SC and CTD tend to reduce the recourse cost upon route failures. For example, serving a task with a smaller expected demand will be less likely to encounter a route failure. When a route failure occurs, a task with smaller serving cost and cost to the depot will lead to smaller refill cost. Overall, we can see that rp_2 is able to make dominating decisions in most "easy" decision situations, while rp_1 and rp_3 may correct the decisions of rp_2 for some more complex decision situations.

5 CONCLUSION

This paper proposes two ensemble GPHH algorithms, namely DivBaggingGP and DivNichGP, to evolve an effective ensemble of interpretable routing policies for solving UCARP. The experimental results showed that the proposed ensemble algorithms are able to evolve smaller and thus more interpretable routing policies without losing the test performance. DivNichGP can obtain ensembles with both better test performance and interpretability than the single routing policies evolved by the GPHH. This demonstrates the effectiveness of ensemble learning in obtaining both effective and interpretable routing policies.

In the future, we will consider improving the search mechanisms to enhance the effectiveness of the evolved ensembles further. In addition, we will consider different aggregation methods in ensemble learning. We will also explore the hybridisation of ensemble methods and grammar-based GP to improve the interpretability of the evolved routing policies further.

REFERENCES

[1] S.K. Amponsah and S. Salhi. 2004. The Investigation of a Class of Capacitated Arc Routing Problems: The Collection of Garbage in Developing Countries. *Waste Management* 24, 7 (2004), 711–721.

[2] José Brandão and Richard Eglese. 2008. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research* 35, 4 (2008), 1112–1126.

[3] Grant Dick, Caitlin A Owen, and Peter A Whigham. 2018. Evolving bagging ensembles using a spatially-structured niching method. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 418–425.

[4] Karl F Doerner, Richard F Hartl, Vittorio Maniezzo, and Marc Reimann. 2004. Applying ant colony optimization to the capacitated arc routing problem. In *International Workshop on Ant Colony Optimization and Swarm Intelligence*. Springer, 420–421.

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

[6] Marko Durasević and Domagoj Jakobović. 2018. Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment. *Genetic Programming and Evolvable Machines* 19, 1-2 (2018), 53–92.

[7] Richard W Eglese and Leon YO Li. 1996. A tabu search based heuristic for arc routing with a capacity constraint and time deadline. In *Meta-Heuristics*. Springer, 633–649.

[8] Gérard Fleury, Philippe Lacomme, and Christian Prins. 2004. Evolutionary algorithms for stochastic arc routing problems. In *Workshops on Applications of Evolutionary Computation*. Springer, 501–512.

[9] Bruce L Golden and Richard T Wong. 1981. Capacitated arc routing problems. *Networks* 11, 3 (1981), 305–315.

[10] H. Handa, L. Chapman, and Xin Yao. 2005. Dynamic salting route optimisation using evolutionary computation. In *IEEE Congress on Evolutionary Computation*. 158–165.

[11] H. Handa, L. Chapman, and Xin Yao. 2006. Robust route optimization for gritting/salting trucks: a CERCIA experience. *IEEE Computational Intelligence Magazine* 1, 1 (2006), 6–9.

[12] Alain Hertz, Gilbert Laporte, and Michel Mittaz. 2000. A tabu search heuristic for the capacitated arc routing problem. *Operations research* 48, 1 (2000), 129–135.

[13] Torsten Hildebrandt and Jürgen Branke. 2015. On using surrogates with genetic programming. *Evolutionary computation* 23, 3 (2015), 343–367.

[14] Josiah Jacobsen-Grocott, Yi Mei, Gang Chen, and Mengjie Zhang. 2017. Evolving heuristics for Dynamic Vehicle Routing with Time Windows using genetic programming. In *IEEE Congress on Evolutionary Computation*. IEEE, 1948–1955.

[15] Philippe Lacomme, Christian Prins, and Wahiba Ramdane-Chérif. 2001. A genetic algorithm for the capacitated arc routing problem and its extensions. In *Workshops on Applications of Evolutionary Computation*. Springer, 473–483.

[16] Philippe Lacomme, Christian Prins, and Wahiba Ramdane-Cherif. 2004. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research* 131, 1-4 (2004), 159–185.

[17] Philippe Lacomme, Christian Prins, and Alain Tanguy. 2004. First competitive ant colony scheme for the CARP. In *International Workshop on Ant Colony Optimization and Swarm Intelligence*. Springer, 426–427.

[18] Yuxin Liu, Yi Mei, Mengjie Zhang, and Zili Zhang. 2017. Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 290–297.

[19] Sean Luke, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Jeff Bassett, Robert Hubley, and A Chircop. 2006. 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).

[20] Jordan MacLachlan, Yi Mei, Juergen Branke, and Mengjie Zhang. 2018. An Improved Genetic Programming Hyper-Heuristic for the Uncertain Capacitated Arc Routing Problem. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 432–444.

[21] Yi Mei, Ke Tang, and Xin Yao. 2009. A global repair operator for capacitated arc routing problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39, 3 (2009), 723–734.

[22] Yi Mei, Ke Tang, and Xin Yao. 2010. Capacitated arc routing problem in uncertain environments. In *IEEE Congress on Evolutionary Computation*. IEEE, 1–8.

[23] Yi Mei, Ke Tang, and Xin Yao. 2011. Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation* 15, 2 (2011), 151–165.

[24] Yi Mei and Mengjie Zhang. 2018. Genetic Programming Hyper-heuristic for Multi-vehicle Uncertain Capacitated Arc Routing Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '18)*. ACM, New York, NY, USA, 141–142. https://doi.org/10.1145/3205651.3205661

[25] Sangyoon Oh, Min Su Lee, and Byoung-Tak Zhang. 2011. Ensemble learning with active example selection for imbalanced biomedical data classification. *IEEE/ACM transactions on computational biology and bioinformatics* 8, 2 (2011), 316–325.

[26] Mahesh Pal. 2005. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing* 26, 1 (2005), 217–222.

[27] John Park, Su Nguyen, Mengjie Zhang, and Mark Johnston. 2015. Evolving ensembles of dispatching rules using genetic programming for job shop scheduling. In *European Conference on Genetic Programming*. Springer, 92–104.

[28] Alain Pétrowski. 1996. A clearing procedure as a niching method for genetic algorithms. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. IEEE, 798–803.

[29] Ulrike Ritzinger, Jakob Puchinger, and Richard F Hartl. 2016. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research* 54, 1 (2016), 215–231.

[30] Lei Shi, Xinming Ma, Lei Xi, Qiguo Duan, and Jingying Zhao. 2011. Rough set and ensemble learning based semi-supervised algorithm for text classification. *Expert Systems with Applications* 38, 5 (2011), 6300–6306.

[31] Ke Tang, Yi Mei, and Xin Yao. 2009. Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation* 13, 5 (2009), 1151–1166.

[32] Juan Wang, Ke Tang, Jose A Lozano, and Xin Yao. 2016. Estimation of the distribution algorithm with a stochastic local search for uncertain capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation* 20, 1 (2016), 96–109.

[33] Juan Wang, Ke Tang, and Xin Yao. 2013. A memetic algorithm for uncertain capacitated arc routing problems. In *Memetic Computing (MC), 2013 IEEE Workshop on*. IEEE, 72–79.

[34] Thomas Weise, Alexandre Devert, and Ke Tang. 2012. A developmental solution to (dynamic) capacitated arc routing problems using genetic programming. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM, 831–838.

[35] Sanne Wøhlk. 2008. A decade of capacitated arc routing. In *The vehicle routing problem: latest advances and new challenges*. Springer, 29–48.

[36] Lean Yu, Shouyang Wang, and Kin Keung Lai. 2008. Credit risk assessment with a multistage neural network ensemble learning approach. *Expert systems with applications* 34, 2 (2008), 1434–1444.

[37] Zhi-Hua Zhou. 2012. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC.