Knowledge Transfer Genetic Programming with Auxiliary Population for Solving Uncertain Capacitated Arc Routing Problem

Mazhar Ansari Ardeh*, Yi Mei* and Mengjie Zhang* and Xin Yao†
*School of Engineering and Computer Science, Victoria University of Wellington
Wellington, New Zealand
†Department of Computer Science and Engineering, Southern University of Science and Technology
Shenzhen, China

***Abstract*—The uncertain capacitated arc routing problem is an NP-hard combinatorial optimisation problem with a wide range of applications in logistics domains. Genetic programming hyper-heuristic has been successfully applied to evolve routing policies to effectively handle the uncertain environment in this problem. The real world usually encounters different but related instances due to events like season change and vehicle breakdowns, and it is desirable to transfer knowledge gained from solving one instance to help solve another related one. However, the solutions found by the genetic programming process can lack diversity, and the existing methods use the transferred knowledge mainly during initialisation. Thus, they cannot sufficiently handle the change from the source to the target instance. To address this issue, we develop a novel knowledge transfer genetic programming with an auxiliary population. In addition to the main population for the target instance, we initialise an auxiliary population using the transferred knowledge and evolve it alongside the main population. We develop a novel scheme to carefully exchange the knowledge between the two populations, and a surrogate model to evaluate the auxiliary population efficiently. The experimental results confirm that the proposed method performed significantly better than the state-of-the-art genetic programming approaches for a wide range of uncertain arc routing instances, in terms of both final performance and convergence speed.**

***Index Terms*—Arc Routing, Genetic Programming, Hyper-Heuristics, Transfer Optimisation**

I. Introduction

The Uncertain Capacitated Arc Routing Problem (UCARP) is an important optimisation problem [1], [2] that models a collection of vehicles that are assigned to serve a set of required *edges* in a graph. UCARP has important real-world applications in logistic systems, such as road maintenance [3], snow removal and ice control [4], waste collection [5].

As an extension of CARP, UCARP is also NP-hard [6], [7] and, the uncertain nature of UCARP increases the difficulty even further, as the solution may become worse or infeasible when the actual environment is different from expected. Traditional optimisation algorithms such as mathematical programming and genetic algorithms cannot solve UCARP effectively [8], since the preplanned solutions are not flexible enough and may incur large recourse cost in some situations. In contrast, Genetic Programming (GP) is a promising approach to solving UCARP [8], [9]. One of the primary advantages of using GP is to evolve *routing policies* instead of actual routes themselves. Such an indirect method for routing provides much better flexibility and applicability of policies in uncertain environments. GP has shown to outperform existing proactive methods [10] and other policy learning methods including linear models and neural networks [11], due to its flexible representation and low demand on the amount of training data. GPHH has shown to outperform other robustness optimisation methods [10] and other policy learning methods including linear models and neural networks [11], due to its flexibility and low demand on the amount of training data [11].

The effectiveness of routing policies is sensitive to the characteristics of the UCARP instance (e.g., graph topology, number of vehicles and distributions of the random variables), and can dramatically deteriorate even with a slight change such as adding/removing one vehicle in the same UCARP instance [9]. Therefore, new routing policies have to be retrained after every slight change in the UCARP instance.

In reality, problems rarely exist in isolation [12], [13]. In the case of UCARP, different related instances could be encountered when expanding the fleet size or a vehicle breaks down, or changing from one season to another. Routing policies are an indirect and higher abstraction of actual routes. Policies are not dependent on any particular maps and hence are a better representation of knowledge about good routing by ignoring details specific to any particular map. Policies can capture knowledge (e.g., sub-tree structures or probability models) transferable across different problem instances better. Therefore, it is desirable to develop *transfer optimisation* techniques [13] to improve the effectiveness and efficiency by discovering and transferring the knowledge in the policies for previous related UCARP instances.

GP-based transfer learning and optimisation has been applied to a range of problems including symbolic regression [14], [15] and UCARP [16], [17]. However, previous studies have found that the GP process can lose its population diversity [16], [18], [19]. For example, the final GP population can contain many duplicated individuals with the same (good) fitness and common building blocks (i.e., sub-trees). The existing knowledge transfer methods that simply transfer knowledge from the top individuals tend to transfer many duplicated building blocks, and make the GP search in the target instance easily stuck into poor local optima. In addition, the existing GP-based transfer learning and optimisation methods are mostly limited to reusing the transferred building blocks (e.g., sub-trees or tree structure) to initialise the target population [16], [17], [19], [20], [21], [22] which leads to a trade-off to consider. On one hand, the use of the building blocks can help the GP process start from a better-than-random initial population. On the other hand, the initial population may be too restricted in the local region of the transferred individuals, especially when they have many duplicated building blocks due to the loss of diversity. As a result, the GP search for the target instance can hardly jump out of the initial local region to find better regions for the target instance.

To address the above issues, we aim to propose a novel GP-based transfer optimisation algorithm to evolve routing policies for UCARP. The new algorithm is named as GP with Auxiliary-Population for knowledge Transfer (*APTGP*). In our algorithm, all the individuals evaluated for solving the source problem form the pool of transferred knowledge. To fight off the issue of diversity loss, *APTGP* clears duplicate individuals from the knowledge pool and utilises the cleared pool to initialise the GP population for solving the target problem. Furthermore, after the initialisation, the transferred knowledge is utilised to help GP handle the issue of diversity loss during the process of solving the target problem. For this purpose, the transferred pool is preserved as a second auxiliary population that evolves alongside the main population based on a surrogate model that is learned from and update by the main population. The main purpose of the auxiliary population is to help GP maintain its population diversity and for this matter, we devise an elaborate immigrant exchange mechanism between the main and the auxiliary populations.

Hence, our research objectives in this paper are as below.

- To handle the potential presence of duplicates in the transferred knowledge, we develop a new initialisation mechanism that removes duplicates from the knowledge pool and transfers unique individuals in the GP for UCARP.
- To reuse the transferred knowledge after the initialisation phase, we propose a mechanism for adapting the transferred knowledge to the target UCARP problem and reusing it more efficiently.
- To prevent the GP for UCARP from losing its population diversity, we devise an elaborate immigrant exchange mechanism between the main and the auxiliary populations that reduces duplicates in the population.

II. Background

A. Uncertain Capacitated Arc Routing Problem

The *static* CARP [7], [6] aims to find a set of routes to serve a collection of required edges (*tasks*) in a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$. Initially, all the vehicles are stationed at a *depot* node. Each edge $e \in \mathcal{E}$ has a *demand* $d(e)$, indicating the amount of work to be done to the edge. If $d(e) > 0$, then e is also called a *task*. The task set is denoted as $\mathcal{E}_T \subseteq \mathcal{E}$. Serving a task e incurs a *serving cost* $sc(e) > 0$. Traversing an edge e without serving it will incur a *deadheading cost* $dc(e) > 0$. Each vehicle has a limited capacity of the demand q.

By taking into account the *uncertain environment*, in UCARP, the demand of each task $e \in \mathcal{E}_T$ is a random variable $D(e)$, and the deadheading cost of each edge $e \in \mathcal{E}$ is a random variable $DC(e)$. Due to the random task demands and deadheading costs, the following two types of failure can occur when executing a solution in UCARP.

1) A *route failure* occurs if the actual demand of the next served task is larger than expected, and exceeds the remaining capacity of the vehicle. The route needs to be repaired by going back to the depot to replenish.
2) An *edge failure* occurs when the next edge becomes inaccessible (deadheading cost becomes infinity). In this case, a detour needs to be found.

A *UCARP instance* can be denoted as a tuple $I = \langle \mathcal{G}, D(\cdot), sc(\cdot), DC(\cdot), q\rangle$, where $\mathcal{G}$ is the graph, $D(\cdot)$ indicates the random task demands, $sc(\cdot)$ represents the deterministic serving costs, $DC(\cdot)$ indicates the random deadheading costs, and q is the capacity.

Note that a *UCARP instance* contains a number of random variables, and can generate a large (if not infinite) number of *instance samples* by sampling a value for each random variable. Given a UCARP instance $I = \langle \mathcal{G}, D(\cdot), sc(\cdot), DC(\cdot), q\rangle$, we denote the instance sample $I_\xi = \langle \mathcal{G}, d_\xi(\cdot), sc(\cdot), dc_\xi(\cdot), q\rangle$, where $d_\xi(e)$ ($dc_\xi(e)$) is the sampled demand (deadheading cost) of e using the random seed ξ.

A UCARP instance I_ξ looks similar to a static CARP instance. The main difference is that in I_ξ, the sampled values $d_\xi(\cdot)$ and $dc_\xi(\cdot)$ are unknown during the optimisation, but gradually realised during the execution process as follows.

- The sampled demand of a task $d_\xi(e)$ is realised after the task is served.
- The sampled deadheading cost of an edge $dc_\xi(e)$ is realised after the edge is traversed.

A solution to a UCARP instance I is denoted as a mapping $h_I(\cdot)$, which can generate a *feasible* solution $S_{I_\xi} = h_I(\xi)$ for each instance sample I_ξ. Here, a solution S_{I_ξ} is feasible if it satisfies the following constraints.

(C1) Each route starts and ends at the depot.
(C2) Each task is served exactly once by one vehicle. An exception is that when route failure occurs, the failed vehicle returns to the depot to replenish in the middle of the service, and then resumes the remaining service.
(C3) The total demand served between two adjacent depot visits cannot exceed its capacity q.

Below are two examples of possible UCARP solutions.

- A UCARP solution can contain a preplanned robust solution S_I along with a recourse operator $\mathtt{RO}(\cdot)$ that transforms S_I to a feasible solution for each sample I_ξ. In this case, $h_I(\xi) = \mathtt{RO}(S_I, I_\xi)$.
- The UCARP solution can be a routing policy $\mathtt{RP}(\cdot)$ that generates a feasible solution for each sample I_ξ by making real-time reactive decisions. In this case, $h_I(\xi) = \mathtt{RP}(I_\xi)$. In this study, a UCARP solution is represented as a routing policy.

UCARP is to find a solution $h_I(\cdot)$ that can generate feasible solutions for all possible samples I_ξ, and the expected total cost is minimised. It can be stated as follows.

$$\min_{h_I(\cdot)\in\mathcal{H}_I(\cdot)} \quad E[tc(h_I(\xi))|\xi \in \Xi], \tag{1}$$

$$s.t.: \ h_I(\xi) \text{ satisfies constraints (C1)–(C3)} \tag{2}$$

where $\mathcal{H}_I(\cdot)$ is the domain of possible UCARP solutions. $h_I(\xi) = S_{I_\xi}$ is the feasible solution for I_ξ, and Ξ is a set of *unseen test samples* of the possible future situation.

It may seem that there exist some degree of similarity between UCARP and the body of dynamic optimisation problems [23], [24]. Nevertheless, the characteristic property of dynamic optimisation problems is that these problem change with time [23], [24]. UCARP, however, does not have time-dependent component and hence, it does not belong to the set of dynamic problems.

B. Related Work

1) Methods for UCARP: The existing approaches for solving UCARP can be divided into the *proactive* and *reactive* methods. The proactive algorithms [1], [25], [26], [27] focus on finding the direct routes that demonstrate the best robustness. These methods take the environmental uncertainties into account indirectly, and find the routes based on some robustness measures. The reactive approaches [8], [9], on the other hand, utilise routing policies that can create the solutions in real-time and hence, react more flexibly to the uncertainties.

A routing policy is a mathematical function that can calculate the priority of each unserved task based on the information about the state of tasks, vehicles and the environment. With the help of a routing policy, it is fairly straightforward to construct UCARP solutions in real-time. At first, all vehicles are idle and stationed at the depot. Whenever a vehicle becomes idle, it considers the set of available tasks, calculates their priorities and selects the tasks with the highest priority to serve next. As the vehicle serves the selected task, it may encounter edge and route failures which it needs to handle appropriately until the task is served. At this point, the vehicle becomes idle. If all the tasks are served, then the vehicle returns to the depot and a complete solution is obtained; otherwise, the process of selecting and serving tasks is repeated [8]. Since routing policies are heuristics that generate the final solutions, GP can be utilised as a hyper-heuristic to evolve them automatically.

2) Genetic Programming Hyper Heuristics: GP evolves a population of computer programs, abstracted as mathematical expressions, iteratively [28], [29]. It first creates an initial population of programs randomly. Then, it repeatedly creates new individuals with the crossover, mutation and reproduction operators, until some stopping criteria are met. GP is a powerful search mechanism that has been successfully applied to various problems [30], [14]. Since computer heuristics are computer programs in nature, GP can be utilised as a Hyper Heuristics (GPHH) for evolving them automatically.

UCARP routing policies are basically mathematical expressions that are used as heuristics for building UCARP solutions. Liu et al. [8] proposed a GPHH based on a tree-based GP to evolve routing policies for a single vehicle. Mei et al. [9] extended [8] to evolve routing policies for multiple vehicles. Maclachlan et al. [10] showed that allowing collaborations between the vehicles can improve the performance of the evolved policies. Wang et al. [31], [32], [33] proposed methods to evolve effective and small (thus potentially more interpretable) routing policies for UCARP. Liu et al. [34] combined the reactive and proactive approaches in which the proactive component of the algorithm evolves the sequence of vehicle routes and the reactive component evolves a routing policy that navigates the vehicle to the depot in case of route failures.

3) Evolutionary Transfer Optimisation: Problems rarely exist in isolation [13]. Most often, a family of problems share some common properties, generally referred to as *knowledge*. In the EA domain, *Transfer optimisation* methods are a set of algorithms that aim at benefiting from this fact by extracting the common knowledge from a previously-solved problem and reusing it to solve related problems more effectively [13].

In transfer optimisation, the concept of knowledge is usually defined based on the problem to solve and the algorithm to use. In early transfer optimisation attempts, the knowledge was defined as the genetic materials that were discovered during the course of solving the source problem. Consequently, transfer optimisation was performed with the *injection of genetic materials* into the population of the algorithm that searches for the solutions of the target problem. Koçer et al. [35] proposed a method, called *GATL*, that selected the best and median individuals found at each generation of the source to initialise the target EA. Dinh et al. [21] devised three injection methods that selected some genetic materials from a source population to initialise $k\%$ of the target GP. Their *FullTree* selects the best individuals of the source final population. In *BestGen*, the best individuals of each source generation are chosen. The *SubTree* method considers the final source population and selects a root subtree of individuals randomly as the transferable genetic material.

Subtrees are also effective transferable knowledge. Particularly, Iqbal et al. [36] proposed the *TLGPC* algorithm that considered the high-quality solutions of the source domain and extracted their subtrees into a pool. For initialising the target problem, the individual's root subtree was either selected created randomly or selected from the pool. Also, for mutating an individual, a randomly selected subtree of the individual was either replaced with a subtree created randomly or a subtree from the pool. While most of the aforementioned methods selected transferable subtrees randomly or based on the fitness of their individuals, other methods defined an importance measure for selecting the subtrees [37], [16].

Another approach to knowledge transfer is to learn a structure that models important information about the source problem. Such *model-based* algorithms then utilise the extracted model to guide the evolution for solving the target problem. In [38], [39], [40], the model is an autoencoder [41] that maps the search space of a source to that of a target problem. In [42], [43], the model is defined as a positive semi-definite matrix that provides a mapping between problem instances and their solutions. The probability distributions of the good source solutions are also capable models for transferring useful knowledge [18], [20], [44], [45].

Ardeh et al. [16] were the first to apply transfer optimi-

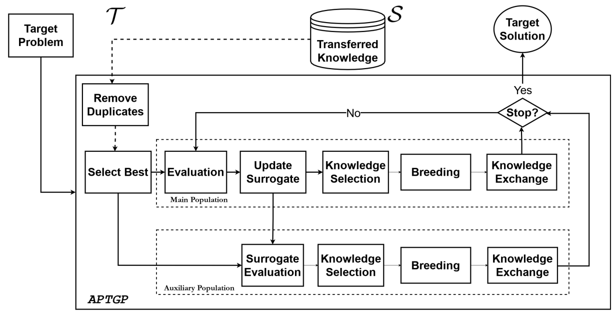


Fig. 1: The *APTGP* framework.

sation for solving UCARP who analysed the performance of several existing methods, as well as their own algorithm. They observed that the existence of duplicates and possible convergence to local optima of the source problem can have a significant negative impact on the effectiveness of knowledge transfer. This finding was later confirmed in [18], [19], [20], [46]. Consequently and to increase the diversity in the pool of transferred knowledge, Ardeh et al. [17], [47] proposed the SUFullTree algorithm that achieved a significant improvement in the effectiveness of knowledge transfer by learning a surrogate model [48], [49], [50] from the source solutions, creating a large pool of unique individuals that were created from the good transferred individuals, evaluating them with the surrogate and selecting the best ones for transfer.

The act of knowledge sharing for solving related problems is also a fundamental part of Multi-task Optimisation (MTO) [51], [13] which solves multiple related tasks together. Gupta et al. [51], [52] firstly proposed a Multi-Factorial Evolutionary Algorithm (MFEA) to solve multiple tasks with a single population, which shares knowledge implicitly through the crossover operator between individuals solving different tasks. This work was extended later by Bali et al. [53]. Zhou et al. [54] improved MFEA by proposing an adaptive knowledge sharing mechanism. Feng et al. [55] proposed an MTO algorithm for solving CARP using autoencoders. Zhong et al. [56] incorporated MFEA with GP to solve symbolic regression problems. Ardeh et. al. [57] developed a multi-task GP for solving multiple related UCARP scenarios together.

III. Proposed Algorithm

A. Overall Framework

Figure 1 presents the overall framework of *APTGP*. The inputs of the algorithm include the target UCARP instance to be solved, as well as the knowledge gained from solving the source instance. In this study, we consider that the knowledge simply includes all the examined routing policies by GP when solving the source instance. Additionally, *APTGP* is based on the assumption that the source and target problems are related.

To solve the target instance, *APTGP* first initialises the main and auxiliary populations using the transferred knowledge, in order to have a better-than-random initial population. Then, the main and auxiliary populations are evolved in parallel. Since the auxiliary population aims to assist the evolution of the main population, all the individuals in the auxiliary population are evaluated by surrogate. Here, we use the KNN surrogate [49], which will be described in Section III-D.

At each generation, the individuals in the main population are first evaluated with the full evaluation (i.e., UCARP simulations). Then, the surrogate model is updated by the newly evaluated individuals. Afterwards, the individuals in the auxiliary population are evaluated using the updated surrogate. Then, the two populations select immigrants and breed offspring using the standard tree-based crossover, mutation and reproduction operators [28]. Finally, the information is exchanged between the populations by sending the immigrants from one population to the other. As a result, the main difference between the main and auxiliary populations are the fitness evaluation, as well as the individuals within them.

The pseudocode of *APTGP* is presented in Algorithm 1. In addition to the common GP parameters, *APTGP* has two parameters η and ϑ for knowledge exchange. For initialisation, the routing policies in $\mathcal{K}_S$ are first split into the subsets of unique routing policies $\mathcal{K}_{uni}$ and duplicated policies $\mathcal{K}_{dup}$, based on their phenotypic behaviours (details are given in Section III-C). Then, the main pop_1 and auxiliary populations pop_2 are both initialised with the best $popsize$ unique routing policies. Since there are a large number of routing policies in $\mathcal{K}_S$ (e.g., 50 generations and 1024 routing policies per generation), it is safe to assume that there are always enough unique routing policies in $\mathcal{K}_S$.

At each generation, the two populations are first evaluated using the full simulations and surrogate, respectively, and the best routing policy $\texttt{rp}^*$ is updated by the best individual in pop_1. Then, the immigrants Ψ_1 and Ψ_2 for the two populations are selected (details in Section III-E). After that, the new populations are generated by the standard GP breeding process. Specifically, one or two parents are selected from the population by the size-7 tournament selection, and the tree-based crossover, mutation or reproduction operator is applied to the parents to generate the offspring. Finally, the two populations exchange knowledge with each other by the $\texttt{ExchangeImmigrants}()$ function (details in Section III-F).

B. Representation and Fitness Function

In *APTGP*, a routing policy is represented as a tree, which is essentially a priority function. Fig. 2 shows an example routing policy $10^5 * \text{CFH} - \text{DEM}/\text{SC}$, where CFH is the cost from the current location to the candidate task, DEM is the expected demand of the candidate task, and SC is the serving cost of the candidate task. This example routing policy is the well-known path-scanning heuristic [58], which selects the task closest to the current location, and selects the task with the smallest demand over serving cost to break the tie.

The fitness function of a routing policy is defined based on a set of training instance samples Ξ_{train}. Based on Eq. (1), the fitness function is defined as

$$fit(\texttt{rp}) = \frac{1}{|\Xi_{\text{train}}|} \sum_{\xi \in \Xi_{\text{train}}} tc(\texttt{rp}(I_\xi)), \tag{3}$$

where the solution $S_{I_\xi} = \texttt{rp}(I_\xi)$ is generated by Algorithm 2. It is a simulation of applying the routing policy to an instance

Algorithm 1: The proposed *APTGP*

```
Input: $\mathcal{K}_S$: routing policies examined by GP for the source instance
Input: $I_g$: the target UCARP instance to solve
Input: Parameters $\eta$ and $\vartheta$
Output: $\mathtt{rp}^*$: the best routing policy for the target instance
// Initialisation
1  $\mathcal{K}_{uni}, \mathcal{K}_{dup} = \mathtt{PhenotypicSplit}(\mathcal{K}_S)$;
2  $pop_1 = \mathcal{K}_{uni}[1 : popsize]$;            // main population
3  $pop_2 = \mathcal{K}_{uni}[1 : popsize]$;       // auxiliary population
4  $\mathtt{rp}^* = null$, $gen = 0$;
// Search loop
5  while $gen < MaxGen$ do
6      Evaluate the routing policies in $pop_1$;
7      Update $\mathtt{rp}^*$ with $pop_1$;
8      Update the surrogate model $\Upsilon$ with $pop_1$;
9      Evaluate the routing policies in $pop_2$ using the surrogate $\Upsilon$;
10     $\Psi_1 = \mathtt{Immigrants}(pop_1, \eta)$;     // Select immigrants
11     $\Psi_2 = \mathtt{Immigrants}(pop_2, \eta)$;
       /* Breeding with standard GP crossover,
          mutation and reproduction                  */
12     $pop_1 = \mathtt{Breed}(pop_1)$;
13     $pop_2 = \mathtt{Breed}(pop_2)$;
       // Exchange Knowledge
14     $pop_1 = \mathtt{ExchangeImmigrants}(pop_1, \Psi_2, \eta, \vartheta)$;
15     $pop_2 = \mathtt{ExchangeImmigrants}(pop_2, \Psi_1, \eta, \vartheta)$;
16     $gen = gen + 1$;
17 end
18 return $\mathtt{rp}^*$;
```

sample to generate a solution. Initially, all the vehicles are at the depot, and all the tasks are unserved. Once a vehicle becomes idle, the routing policy calculates the priority of each unserved task that is expected to be feasible, and selects the next task based on the priority. The simulation stops when all the tasks are served, and all the vehicles return to the depot.

Algorithm 2: $S_{I_\xi} = \mathtt{rp}(I_\xi)$

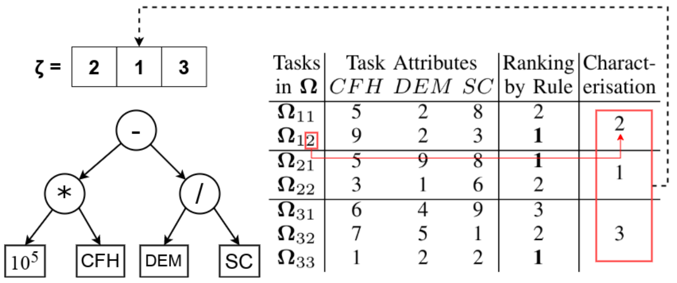
```
Input: $\mathtt{rp}$: a routing policy
Input: $I_\xi$: a UCARP instance sample
Output: $S_{I_\xi}$: a feasible solution
1  All vehicles are at the depot, all task are unserved;
2  while not all tasks are served do
3      Find the earliest idle vehicle $veh^*$ (break the tie randomly);
4      Find all the unserved tasks $\mathcal{E}_T^*$ whose expected demand do not
         exceed the remaining capacity of $veh^*$;
5      for task $e \in \mathcal{E}_T^*$ do
6          Calculate the priority value $\mathtt{rp}(e)$;
7      end
8      Select next task $e^* = \arg\min_{e \in \mathcal{E}_T^*} \mathtt{rp}(e)$;
9      Send $veh^*$ to serve $e^*$;
10     Repair if route/edge failur occurs;
11 end
12 return the generated solution;
```

C. Initialisation

The main and auxiliary populations are initialised by the top unique routing policies from the source knowledge $\mathcal{K}_S$. For this, $\mathcal{K}_S$ is first split into the subsets of *unique* and *duplicated* routing policies based on their phenotypic behaviours, by the `PhenotypicSplit()` function, described in Algorithm 3. Note that it takes any set of routing policies as input, and is used by the `ExchangeImmigrants()` function as well.

1) Phenotypic Characterisation: In Algorithm 3, the routing policies in $\mathcal{RP}$ are first sorted by fitness. Here $\mathcal{RP} = \mathcal{K}_S$, and we use the source fitness for the sorting. Then, we calculate the phenotypic behaviour vector $\mathbf{b}(\mathtt{rp})$ for each routing policy. This calculation is similar to the phenotypic characterisation in [49], which characterises a routing policy by its selected tasks in a range of decision situations $\mathbf{\Omega}$. Fig. 2 shows an example of the characterisation of a policy using three decision situations $\mathbf{\Omega} = \{\Omega_1, \Omega_2, \Omega_3\}$. The first situation has two candidate tasks (Ω_{11} and Ω_{12}), the second has two (Ω_{21} and Ω_{22}) and the third contains three (Ω_{31}, Ω_{32} and Ω_{33}). Based on the attributes of the tasks, we can see that the indices of the tasks selected by the routing policy in the situations are 2, 1, and 3. This forms the phenotypic vector $[2, 1, 3]$.

Knowledge transfer can be very expensive if not done carefully [59], [60]. The complexity of Algorithm 3 depends on the sorting operation (line 2) and the the main loop (lines 3–11). As a sorting algorithm, `SortByFitness` has a time complexity of $O(|\mathcal{RP})|\log(|\mathcal{RP})|))$. During the main loop, the complexity of each iteration is dominated by the phenotypic characterisation (line 4), whose complexity is $O(|\mathbf{\Omega}|)$. If $\mathcal{RP}_{uni}$ is implemented with a hash set data structure, then the uniqueness check at line 6 has a complexity of $O(1)$, the complexity of the main loop is $O(|\mathcal{RP})| \cdot |\mathbf{\Omega}|)$. Considering that usually $\log(|\mathcal{RP}|) \gg |\mathbf{\Omega}|$. Thus, the complexity of Algorithm 3 is $O(|\mathcal{RP})|\log(|\mathcal{RP})|)$.



Tasks in $\mathbf{\Omega}$	Task Attributes CFH	DEM	SC	Ranking by Rule	Charact-erisation
$\mathbf{\Omega}_{11}$	5	2	8	2	2
$\mathbf{\Omega}_{12}$	9	2	3	**1**	
$\mathbf{\Omega}_{21}$	5	9	8	**1**	1
$\mathbf{\Omega}_{22}$	3	1	6	2	
$\mathbf{\Omega}_{31}$	6	4	9	3	
$\mathbf{\Omega}_{32}$	7	5	1	2	3
$\mathbf{\Omega}_{33}$	1	2	2	**1**	

Fig. 2: An example of the path-scanning heuristic and its phenotypic characterisation.

Algorithm 3: $\mathtt{PhenotypicSplit}(\mathcal{RP})$

```
Input: $\mathcal{RP}$: a set of routing policies
Output: $\mathcal{RP}_{uni}$: the unique routing policies
Output: $\mathcal{RP}_{dup}$: the duplicated routing policies
1  $\mathcal{RP}_{uni} = \emptyset$, $\mathcal{RP}_{dup} = \emptyset$;
2  $\mathcal{RP} = \mathtt{SortByFitness}(\mathcal{RP})$;
3  for routing policy $\mathtt{rp} \in \mathcal{RP}$ do
4      Calculate the phenotypic behaviour $\mathbf{b}(\mathtt{rp})$;
5      Calculate the hash value $h(\mathtt{rp}) = \mathtt{Hash}(\mathbf{b}(\mathtt{rp}))$;
6      if $h(\mathtt{rp}) \neq h(\mathtt{rp}')$, $\forall$ $\mathtt{rp}' \in \mathcal{RP}_{uni}$ then
7          $\mathcal{RP}_{uni} = \mathcal{RP}_{uni} \cup \{\mathtt{rp}\}$;
8      else
9          $\mathcal{RP}_{dup} = \mathcal{RP}_{dup} \cup \{\mathtt{rp}\}$;
10     end
11 end
12 return $\mathcal{RP}_{uni}$, $\mathcal{RP}_{dup}$;
```

2) Hashing for Duplicate Checking: For duplicate checking, we need to compare the phenotypic vectors between routing policies, which can be time consuming if a larger number of decision situations are used to capture the behaviours of the routing policy accurately. To handle this issue, we create a hash value for each behaviour vector, and use this single hash value for the duplicate checking. Here we use the well known `LSHash` function [61], which is given in Algorithm 4. Given a behaviour vector $\mathbf{b}$, `LSHash` calculates the hash value using the formula in line 2, where $<<$ is the bitwise left-shit operator. The `Hash` function has a time complexity of $O(|\mathbf{\Omega}|)$, which depends on the number of decision situations.

```
Algorithm 4: Hash(b)
Input: A behaviour vector b
Output: The hash value of the behaviour vector
1 $h = 1$;
2 for $i = 1 \to |\mathbf{b}|$ do $h = h << 5 - h + b_i$;
3 return $h$;
```

D. Surrogate Model for Auxiliary Population

We use the KNN-based surrogate model [49] to estimate the fitness of the individuals in the auxiliary population. Specifically, the surrogate model $\varUpsilon = \{(\mathbf{b} : fit)\}$ contains the pairs of phenotypic behaviour vector and fitness. The surrogate fitness of a routing policy is assigned as the fitness corresponding to the phenotypic vector in $\varUpsilon$, which is closest to that of the routing policy based on Euclidean distance.

Initially, $\varUpsilon$ is set to empty. In each generation, after evaluating pop_1, all the phenotypically unique individuals in pop_2 will be added into $\varUpsilon$. The maximal size of $\varUpsilon$ is $2 * popsize$. If the number of elements exceeds the maximal size, $\varUpsilon$ is trimmed by removing the oldest elements.

E. Immigrants Selection

Before the breeding, for each population, we select η immigrants preparing to be transferred to the other population. Each immigrant is selected by the size-7 tournament selection. The pseudocode is shown in Algorithm 5.

```
Algorithm 5: Immigrants($pop, \eta$)
Input: $pop$: a population of routing policies
Input: $\eta$: number of immigrants
Output: $\Psi$: the immigrants from $pop$
1 $\Psi = \emptyset$;
2 while $|\Psi| < \eta$ do
      // Common tournament size for GP is 7
3     Randomly select 7 individuals from $pop$;
4     Add the best selected individual into $\Psi$;
5 end
6 return $\Psi$;
```

F. Knowledge Exchange

After breeding, the main and auxiliary populations exchange knowledge by transferring the selected immigrants to each other. When transferring, the following two factors are considered: (1) to maintain diversity, the immigrants should not be a duplicate in the goal population, and (2) the immigrants should replace the less useful individuals in the goal population.

Algorithm 6 shows the pseudocode of the knowledge exchange. First, it splits the goal population pop into the unique (uni) and duplicated individuals (dup). The individuals to be replaced ($replaced$) is first set to all the duplicated individuals, as they are expected to contribute little to the population. If there are not enough duplicates (i.e. $|replaced| < \eta$), we fill in the remaining places by the size-7 reverse tournament selection to uni. Then, we sort $replaced$ by fitness from worst to best (`ReverseSortByFitness`()). Note that `ExchangeImmigrants`() is called between the breeding and evaluation, and the newly generated individuals are not evaluated yet. In this case, we simply use the fitness inherited from the parent for the sorting.

Next, the immigrants are refined to make sure they are not duplicates of the goal population. For each immigrant, if it is a duplicate of the goal population, then we mutate it to generate a different immigrant. We keep mutating the immigrant until it becomes a non-duplicate or the maximal number of trials ϑ is reached, and Ψ is refined to Ψ'. Finally, we select the worse individuals (either duplicates or worst fitness) in pop and replace them with Ψ'.

```
Algorithm 6: ExchangeImmigrants($pop, \Psi, \eta, \vartheta$)
Input: $pop$: the goal population
Input: $\Psi$: the immigrants from the source population
Input: $\eta$: number of immigrants
Input: $\vartheta$: number of trials
Output: The new goal population $pop$
   // Select individuals to be replaced in pop
1  $uni, dup$ = PhenotypicSplit($pop$);
2  $replaced = dup$;
3  while $|replaced| < \eta$ do
       // Common tournament size for GP is 7
4      Randomly select 7 individuals from $uni$;
5      Add the worst selected individual into $replaced$;
6  end
7  $replaced$ = ReverseSortByFitness($replaced$);
   // Refine the immigrants
8  $\Psi' = \emptyset$;
9  for $i = 1 \to \eta$ do
10     for $tries = 1 \to \vartheta$ do
11         if $\exists \mathtt{rp} \in uni,\ h(\Psi[i]) = h(\mathtt{rp})$ then
12             $\Psi[i]$ = Mutate($\Psi[i]$);
13             Calculate the phenotypic behaviour $\mathbf{b}(\Psi[i])$;
14             Calculate the hash value $h(\Psi[i])$ = Hash($\mathbf{b}(\Psi[i])$);
15         else
16             $\Psi' = \Psi' \cup \{\Psi[i]\}$;
17             break;
18         end
19     end
20 end
   // Transfer immigrants
21 $replaced[1 : |\Psi'|] = \Psi'$;
22 return $pop$;
```

The time complexity of Algorithm 6 is dominated by the phenotypic split (line 1) and the operations in lines 9–20. The phenotypic split has a complexity of $O(|pop|\log(|pop|)$, and the operations in lines 9–20 have a complexity of $O(\eta\vartheta|\mathbf{\Omega}|)$. Thus, Algorithm 6 has a time complexity of $O(|pop|\log(|pop|) + \eta\vartheta|\mathbf{\Omega}|)$.

Overall, the overhead of *APTGP* for knowledge transfer is $O(|\mathcal{RP}|\log(|\mathcal{RP}|) + MaxGen \cdot (|pop|\log(|pop|) + \eta\vartheta|\mathbf{\Omega}|))$. This is much smaller than the complexity of GPHH, which is $O(MaxGen \cdot |pop| \cdot \mathtt{sim})$, where `sim` is the complexity of the training simulations, which is typically much larger than $\log(|pop|)$ and $\eta\vartheta|\mathbf{\Omega}|$.

G. Summary

The main idea of *APTGP* is to maintain two populations in the target instance and interact with each other to improve the search effectiveness. First, the main population is initialised by the unique individuals from the source knowledge, which can make the search start from a better region. Second, the auxiliary population starts from the same region as the main population, and evolves alongside the main population but towards different directions (e.g., by using the surrogate fitness). Third, the main population regularly receives non-duplicate immigrants from the auxiliary population, which can help it jump out of the initial local region and handle the concept drift from the source to the target instance. Finally, the main population migrates its best individuals to the auxiliary population to influence its search towards the best solutions that have been found so far.

It should be noted that our algorithm bears some similarities with the cultural algorithms [62], [63]. However, the cultural algorithms focuses on solving a single problem, while *APTGP* focuses on transfer optimisation and takes advantage of the source individuals during the search process in the target problem. None of the populations in *APTGP* act as a belief space like in the cultural algorithms, and the auxiliary population in *APTGP* is evolved with a surrogate.

Finally, it is important to note that, despite resembling multi-task learning algorithms, the proposed algorithm does not solve multiple problems at the same time. In this sense, the *APTGP* method in this paper is more similar to the works by Ruan et al. [59], [60] in the sense that it assumes the source task is solved first and therefore, the main and auxiliary populations of *APTGP* are focused on solving one target problem.

IV. Experiment Setup

A. Datasets

Table I shows the source and target instances used in the experiments. The UCARP instances are extended from the well-known static CARP instances (converting the deterministic variables into random ones with normal distribution), and have been commonly used in previous studies [8], [9], [16], [17]. Some of these datasets are based on real-world road network from Lancashire, UK [10]. In the table, the presence of the *dm1* (*dm2*) suffix at the end of of dataset name, e.g. Ugdb11dm2, highlights that the dataset is created by increasing the mean of the random demands by 1 (2). Each row is called a *transfer scenario* with a *source instance* and a *target* instance. The last column "Sim." (between -1 and 1) indicates the similarity between the source and target instances. To calculate the similarity, we generate 1024 unique routing policies randomly, and evaluate them on both the source and target instances (200 samples each). The similarity is then measured by the Kendall's rank correlation coefficient [64] between the fitness of the 1024 random individuals on the source and target instances. If the similarity is closer to 1, then the source and target instances are more similar. It should be noted that in practice, most UCARP instances are likely to be related to each others because all the UCARP problems aim to minimise the total cost, thus they have common desirable routing decisions such as selecting the nearest tasks. Therefore, it is unlikely to have negative or very close to zero similarity values and hence, the scenarios in Table I present a diverse range of similarity values.

As shown in Table I, the experiments include source and target instances with varying similarities (from 0.46 to 0.99). Note that we have calculated a large number of source and target instances, and most of them show decent similarities with each other. A similarity of 0.46 is already a weak relatedness between different UCARP instances.

TABLE I: The transfer scenarios used in the experiments

Scn.	Source Instance	Target Instance	Sim.
1	Uval4A with 2 vehicles	Ugdb17 with 3 vehicles	0.46
2	Ugdb17 with 5 vehicles	Ugdb12 with 5 vehicles	0.46
3	Ugdb17 with 5 vehicles	Ugdb12 with 7 vehicles	0.48
4	Ugdb17 with 5 vehicles	Ugdb12 with 8 vehicles	0.49
5	Uval6B with 5 vehicles	Ugdb15 with 3 vehicles	0.56
6	Ugdb17 with 5 vehicles	Ugdb11 with 3 vehicles	0.57
7	Ugdb17 with 5 vehicles	Ugdb11 with 4 vehicles	0.59
8	Ugdb17 with 5 vehicles	Ugdb11 with 5 vehicles	0.61
9	Uegl-e1-C with 8 vehicles	Ugdb13 with 5 vehicles	0.65
10	Uval4A with 2 vehicles	Ugdb6 with 5 vehicles	0.65
11	Uval4A with 2 vehicles	Ugdb6 with 4 vehicles	0.66
12	Ugdb23 with 10 vehicles	Ugdb12 with 5 vehicles	0.67
13	Ugdb23 with 10 vehicles	Ugdb12 with 7 vehicles	0.69
14	Ugdb23 with 10 vehicles	Ugdb12 with 8 vehicles	0.7
15	Uval6B with 5 vehicles	Ugdb6 with 5 vehicles	0.7
16	Ugdb21 with 5 vehicles	Ugdb5 with 4 vehicles	0.76
17	Ugdb4 with 4 vehicles	Ugdb4 with 2 vehicles	0.89
18	Ugdb7 with 5 vehicles	Ugdb1 with 6 vehicles	0.9
19	Ugdb4 with 4 vehicles	Ugdb4 with 6 vehicles	0.9
20	Ugdb3 with 5 vehicles	Ugdb3 with 7 vehicles	0.91
21	Ugdb4 with 4 vehicles	Ugdb4 with 3 vehicles	0.91
22	Ugdb1 with 5 vehicles	Ugdb1 with 3 vehicles	0.92
23	Ugdb6 with 5 vehicles	Ugdb7 with 6 vehicles	0.92
24	Ugdb3 with 5 vehicles	Ugdb3 with 3 vehicles	0.92
25	Ugdb7 with 5 vehicles	Ugdb7 with 7 vehicles	0.92
26	Ugdb7 with 5 vehicles	Ugdb7 with 3 vehicles	0.93
27	Ugdb6 with 5 vehicles	Ugdb6 with 3 vehicles	0.93
28	Ugdb1 with 5 vehicles	Ugdb1 with 7 vehicles	0.93
29	Ugdb1 with 5 vehicles	Ugdb2 with 7 vehicles	0.93
30	Ugdb2 with 6 vehicles	Ugdb6 with 6 vehicles	0.94
31	Ugdb3 with 5 vehicles	Ugdb3 with 6 vehicles	0.94
32	Ugdb5 with 6 vehicles	Ugdb5 with 4 vehicles	0.94
33	Ugdb5 with 6 vehicles	Ugdb5 with 8 vehicles	0.94
34	Ugdb2 with 6 vehicles	Ugdb2 with 4 vehicles	0.94
35	Ugdb6 with 5 vehicles	Ugdb6 with 7 vehicles	0.94
36	Ugdb4 with 4 vehicles	Ugdb4 with 5 vehicles	0.94
37	Ugdb21 with 6 vehicles	Ugdb21 with 4 vehicles	0.95
38	Ugdb7 with 5 vehicles	Ugdb7 with 4 vehicles	0.95
39	Ugdb2 with 6 vehicles	Ugdb2 with 8 vehicles	0.95
40	Ugdb6 with 5 vehicles	Ugdb6 with 4 vehicles	0.95
41	Ugdb6 with 5 vehicles	Ugdb6 with 6 vehicles	0.96
42	Ugdb21 with 6 vehicles	Ugdb21 with 5 vehicles	0.97
43	Ugdb11dm2 with 5 vehicles	Ugdb11dm1 with 5 vehicles	0.98
44	Uval8A with 3 vehicles	Uval8Adm1 with 3 vehicles	0.99
45	Uval5Adm1 with 3 vehicles	Uval5Adm2 with 3 vehicles	0.99

B. Parameter Settings

Table II gives the terminal, function and GP parameter sets which are commonly used in literature [8], [9], [16], [17]. In the function set, the protected division returns 1 if divided by zero. All the parameter settings in Table II, except η and ϑ, are commonly used in literature of using GPHH for solving UCARP [8], [9], [16], [17] and to remain consistent with the previous studies, the values of these parameters are the same as the previous studies. We conducted parameter sensitivity analysis on the two new η and ϑ parameters, and achieved the best results with $\eta = 300$ and $\vartheta = 10$. In Algorithm 3, a set of decision situations is required for calculating the phenotypic behaviour. For each scenario (a row in Table I), we apply the path scanning heuristic to a target instance sample and arbitrarily select 20 decision situations during the process.

For each transfer scenario, the source instance is first solved by running GP once with the setting in Table II. This produces $1024 \times 50 = 51200$ examined routing policies as the transferred knowledge. Then, we run *APTGP* (or the compared GP with knowledge transfer) based on the transferred knowledge on the target instance to train the routing policy. During the training, we use 5 instance samples, and rotate the samples at each generation to reduce overfitting [8]. The trained routing policy is then tested on 500 unseen samples. Each algorithm is run 30 times independently.

TABLE II: The GP parameter settings.

Terminal	**Description**
CFH	Cost From Here
CFR1	Cost From the closest alternative Route
CR	Cost to Refill
CTD	Cost To Depot
CTT1	Cost To the closest Task
DEM	DEMand
DEM1	DEMand of the closest unserved task
FRT	Fraction of Remaining Tasks
FUT	Fraction of Unassigned Tasks
FULL	FULLness (vehicle load over capacity)
RQ	Remaining Capacity
RQ1	Remaining Capacity of closest alternative route
SC	Serving Cost
ERC	Ephemeral Random Constant number
DC	Deadheading Cost

Parameter	**Value**	**Function**	**Description**
Population	1024	$+$	Addition
Crossover rate	80%	$-$	Subtraction
Mutation rate	15%	$*$	Multiplication
Reproduction rate	5%	/	Protected Division
Number of generations	50	min	Minimum
Number of Elitists	10	max	Maximum
Max depth	8		
η (for *APTGP*)	300		
ϑ (for *APTGP*)	10		

C. Compared Algorithms

The comparison is made with GPHH, since it has been demonstrated to be a state-of-the-art approach for solving UCARP, even when considering non-GP methods [10]. Table III shows the state-of-the-art GP with knowledge transfer compared in the experiments. It also briefly describes the knowledge transfer mechanism of each algorithm. In the papers we compared with, they had compared these algorithms against other methods from other groups [16], [21], [36] and found them to be generally less effective. Hence we did not compare against them again in this paper.

TABLE III: The compared GP with knowledge transfer.

Algorithm	**Knowledge transfer mechanism**
GATL [35]	Select the best and median trees of each generation into a pool. Choose randomly from the pool to initialise the target GP population.
TLGPC [36]	Select random subtrees of the better-than-average final individuals into a pool. During initialisation and mutation, create a root or subtree randomly or select randomly from the pool.
SUFullTree [47]	Select the best individuals of all the generations into a pool. Expand the pool with policies created from the good individuals and evaluate them with a surrogate. Choose the best from the pool to initialise the target GP population.

V. Results and Discussions

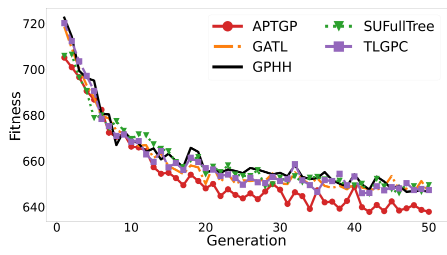
In this section, we compare different aspects of the *APTGP* performance against the set of existing methods. In order to investigate how the proposed algorithm performs on test datasets, we conduct experiments in Section V-A. To analyse how the proposed algorithm affects the complexity of the evolved policies, we investigate the GP tree sizes in Section V-B. Finally, to the overhead of proposed knowledge transfer we examine the training time of the algorithms in Section V-C.

A. Test Performance

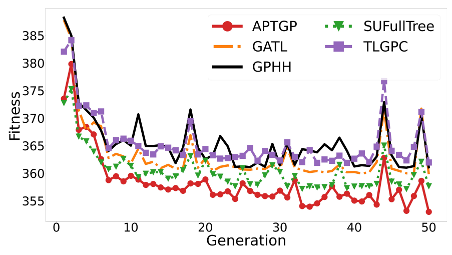
Table IV presents the test performance of the compared algorithms, where the best results are highlighted in boldface. As is evident, in almost all cases, *APTGP* had a best test performance among the compared ones. The Friedman test is also conducted to verify statistical significance. The calculated rank and p-value are given in the bottom rows of Table IV. We can see that *APTGP* has the best rank and the p-value shows that the difference between the results is significant. *APTGP* obtained the best mean test performance for all the scenarios in the experiments. To pinpoint the difference, the Conover post-hoc analysis [65] is performed and the p-value of the pairwise comparisons are given in Table V after being adjusted with the Benjamini-Hochberg method [66]. The very small p-values in the last column of the table show that the difference between *APTGP* and all other algorithm is significant and, considering its rank, this indicates its superiority to the other methods.

Upon investigating Table IV, it could be noted that *APTGP* has a larger variance on some problems which weakens the justification for using the mean value to compare the algorithms' performances. Hence, to check the existence of significant difference between the algorithm performances on each problem, we performed a Wilcoxon rank-sum test on the results of each problem and confirmed that *APTGP* is superior to all the compared algorithms in the majority of the scenarios.

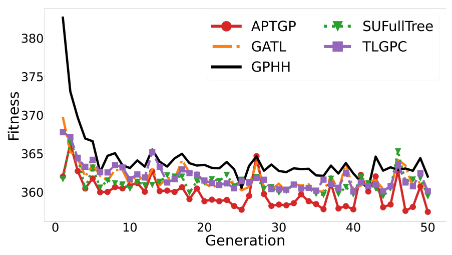
Figure 3 presents the convergence curve of the algorithms for a few scenarios. As is evident, *APTGP* starts with a better initial state and manages to maintain a better performance throughout the entire evolutionary process. We observed similar patterns in almost all other scenarios. In Figure 4, the distribution of the fitness value of the solutions obtained with each algorithm is given as violin plots for a few scenarios. It can be seen that *APTGP* obtained the best (lowest) distributions. Similar patterns were observed in other scenarios too.



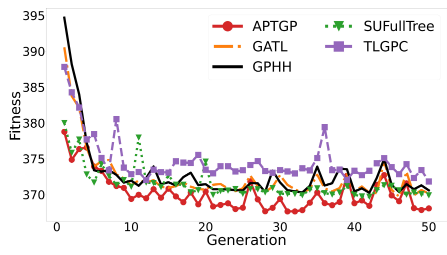
(a) from *Ugdb23*, from 10 to *Ugdb12* with 8 vehicles (Scn. 14)



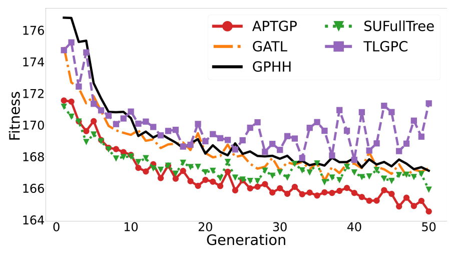
(b) from *Ugdb7* with 5 vehicles to *Ugdb1*, 6 vehicles (Scn. 18)



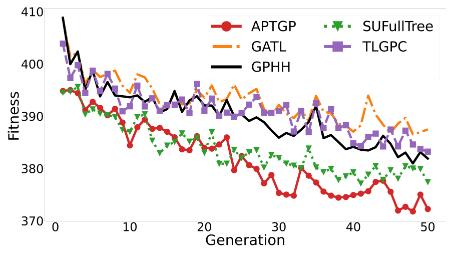
(c) *Ugdb3*, from 5 to 7 vehicles (Scn. 20)



(d) *Ugdb6*, from 5 to 7 vehicles (Scn. 35)

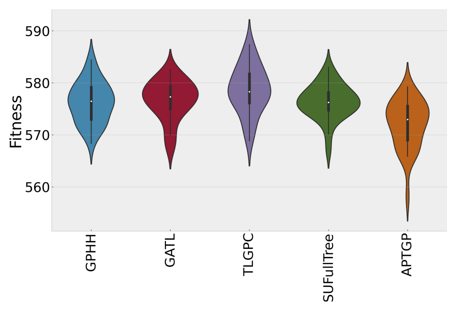


(e) *Ugdb21*, from 6 to 4 vehicles (Scn. 37)

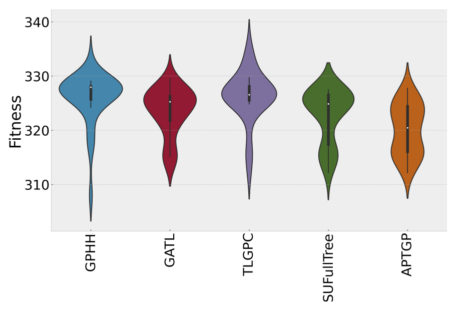


(f) *Ugdb7*, from 5 to 4 vehicles (Scn. 38)

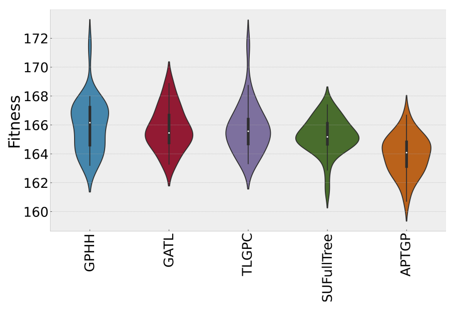
Fig. 3: Convergence curve of *APTGP* and some existing transfer methods.



(a) from *Uegl-1-C* with 8 vehicles to *Ugdb13*, 5 vehicles (Scn. 9)



(b) *Ugdb3*, from 5 to 6 vehicles (Scn. 31)



(c) *Ugdb21*, from 6 to 5 vehicles (Scn. 42)

Fig. 4: Performance violin plots of *APTGP* and the compared transfer algorithms.

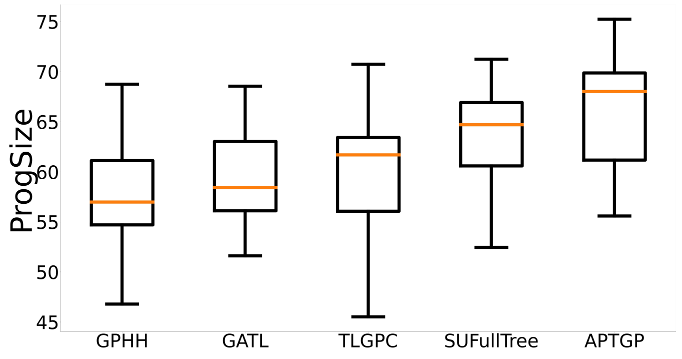


Fig. 5: Size of the programs evolved with the compared algorithms.

B. Program Size

Figure 5 presents the distribution of the program sizes (i.e., number of nodes in the tree) in the final population obtained by the compared algorithms, which indicate that *APTGP* evolved larger trees (65 nodes on average versus 58 nodes by GPHH). A possible reason for this could be that *APTGP* removes duplicates from the GP population, which prevents the computational resources from being wasted on evaluating the duplicates. As a result, *APTGP* can spend the saved computational resources on discovering more effective policies, which tend to be larger and more complex.

Figure 6 presents an example of the evolved routing policies. As is evident, the policy does not have a trivial size which makes it difficult to interpret but at the same time, it represents the complexity of the processing it performs on the state of the environment. Nevertheless, it is possible to gain some high-level insights from this policy. In this policy, the most frequent terminals is CR (Cost to Refill). This indicates that this policy considers this information about the environmental state to be more important. This is consistent with our domain expertise that during the course of serving tasks, a majority of the total cost pertains to the cost returning to depot to refill.

C. Training Time

Figure 7 presents the summary of the training time for each of the examined algorithms. According to this figure, the train-

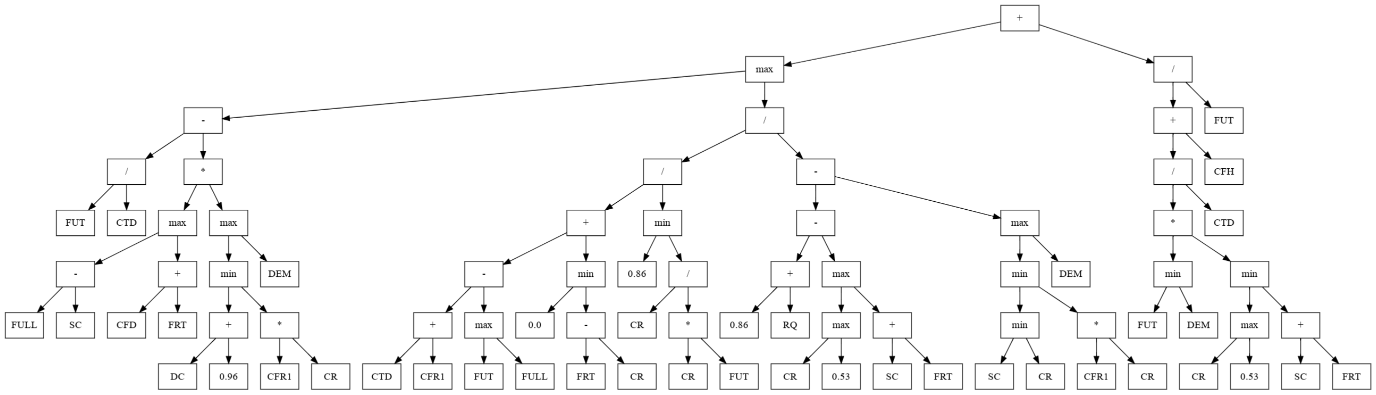


Fig. 6: A programs evolved with the compared algorithms.

TABLE IV: Test performance of 30 independent runs of the compared algorithms (mean ± std).

Scn.	GPHH	GATL [35]	TLGPC [36]	SUFullTree [47]	APTGP
1	91.2±0.1	91.2±0.1	91.2±0.1	91.2±0.1	**91.1±0.01**
2	551.0±10.3	550.8±8.1	552.7±9.0	551.5±9.4	**542.9±10.7**
3	598.6±8.8	599.6±9.5	603.5±10.3	601.9±11.0	**595.0±9.7**
4	639.5±11.3	636.0±10.7	638.5±13.1	644.1±15.8	**632.6±7.3**
5	58.2±0.1	58.3±0.1	58.2±0.1	58.2±0.1	**58.1±0.1**
6	424.8±8.5	424.5±8.1	426.8±9.1	423.3±8.5	**417.6±7.2**
7	432.1±7.1	431.0±6.7	432.9±7.2	431.8±8.3	**427.6±5.8**
8	432.6±5.5	432.2±5.2	431.8±6.3	430.0±6.1	**423.3±5.5**
9	576.2±3.9	576.7±3.8	578.6±4.6	576.4±3.3	**572.0±4.6**
10	340.5±4.7	338.1±4.2	338.7±3.1	337.2±2.1	**335.1±4.0**
11	347.2±6.1	347.1±6.0	349.3±6.4	344.4±5.1	**340.6±4.1**
12	551.0±10.3	553.7±10.5	554.4±11.4	552.7±8.9	**544.8±8.0**
13	598.6±8.8	598.5±7.5	608.7±11.2	600.2±10.3	**595.4±6.2**
14	639.5±11.3	640.1±12.2	640.0±11.5	640.8±14.8	**630.9±5.7**
15	340.5±4.7	339.8±3.5	340.4±5.1	337.0±3.3	**334.4±3.2**
16	444.4±4.7	444.5±5.6	446.6±6.4	444.1±4.9	**441.4±5.6**
17	324.3±6.2	322.4±5.7	325.0±5.2	319.8±4.6	**317.1±3.8**
18	360.3±3.1	359.4±4.5	360.9±3.0	356.2±4.2	**352.2±2.4**
19	358.3±2.6	358.2±3.4	358.5±3.7	357.6±5.3	**354.8±2.6**
20	359.0±1.8	359.0±1.3	359.2±1.6	358.3±1.1	**356.6±1.2**
21	340.8±4.4	340.9±4.6	341.0±3.2	337.6±4.6	**332.6±5.9**
22	351.9±3.5	353.1±4.8	352.1±3.8	350.2±3.3	**349.7±2.4**
23	356.6±1.6	356.7±1.7	356.7±1.7	356.3±1.4	**355.7±0.7**
24	310.9±0.5	311.0±0.5	310.7±2.3	308.8±2.8	**307.9±4.0**
25	389.2±0.2	389.1±0.2	389.1±0.2	389.1±0.2	**389.0±0.1**
26	363.1±2.8	363.2±4.1	363.3±4.0	362.2±3.1	**357.8±4.9**
27	342.1±6.2	341.9±5.1	343.9±6.8	338.6±4.7	**335.1±4.6**
28	382.0±5.5	380.2±6.0	381.2±7.5	384.7±6.0	**378.3±6.5**
29	382.8±3.3	382.6±2.2	383.7±6.4	382.1±3.2	**381.0±2.7**
30	351.5±2.5	351.4±1.1	351.5±1.3	351.0±1.6	**349.9±3.2**
31	326.0±4.7	323.5±4.4	326.2±4.8	322.8±4.9	**320.1±4.7**
32	444.4±4.7	445.2±6.8	446.2±7.6	441.2±5.8	**439.9±6.4**
33	448.2±0.5	448.4±0.8	448.8±1.5	448.9±1.8	**448.0±0.6**
34	384.6±4.4	385.8±6.5	386.2±5.2	384.3±5.1	**382.9±4.6**
35	369.3±1.8	369.2±2.8	369.5±2.7	368.4±2.6	**367.1±1.9**
36	321.4±5.2	323.7±5.5	325.2±5.7	321.1±2.5	**319.5±1.7**
37	166.2±2.0	166.0±1.4	165.8±2.2	165.1±2.0	**163.8±1.4**
38	376.1±7.6	379.8±7.8	378.7±5.6	372.0±9.2	**366.6±5.9**
39	415.7±9.2	416.6±8.4	417.2±6.3	412.3±7.4	**409.0±8.3**
40	347.2±6.1	347.1±6.7	350.3±11.5	344.5±5.0	**340.8±4.9**
41	351.5±2.5	351.7±2.3	351.9±3.9	351.2±1.9	**350.3±3.1**
42	165.9±1.8	165.7±1.5	165.7±1.7	165.2±1.2	**163.9±1.4**
43	462.6±6.0	460.8±5.3	460.3±6.6	456.7±6.5	**451.8±7.3**
44	426.6±3.3	427.1±2.5	427.3±1.6	425.8±3.8	**424.3±4.3**
45	499.0±3.9	498.8±4.5	499.4±4.3	497.5±3.3	**496.5±3.4**
Rank	3.58	3.48	4.48	2.47	**1**
Friedman's p-value					**3.42e-25**

TABLE V: Post-hoc comparison of the compared existing algorithms with adjusted p-values

	GATL	TLGPC	SUFullTree	APTGP
GPHH	0.76	**7.8e-03**	**1.0e-03**	**9.2e-13**
GATL	–	**3.2e-03**	**2.8e-03**	**5.2e-12**
TLGPC	–	–	**1.04e-08**	**4.9e-20**
SUFullTree	–	–	–	**2.0e-05**

ing time of *APTGP* is slightly longer than the existing methods. In *APTGP*, the initialisation operator and the knowledge exchange mechanism incur an additional computational cost and hence, the increased training time is expected. However, the increment in time does not hinder the applicability of the algorithm and, considering its superior performance, the additional cost can be considered acceptable.

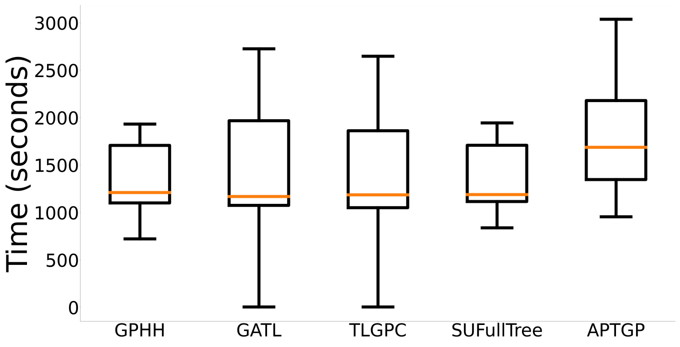


Fig. 7: Training time of the compared algorithms.

VI. Further Analysis

After confirming the superior performance of *APTGP* in Section V, we conduct further analysis in this section. First, to investigate the effect of each new component in *APTGP* on its overall performance, we consider different variants of *APTGP* and examine their performances in Section VI-A. One of the main goals of this work is to maintain the population diversity and we investigate this in Sections VI-B and VI-C. Finally, Section VI-D investigates the quality of the knowledge source in terms of the number of unique individuals it contains and the effect it can have on the quality of knowledge transfer.

A. Component Analysis

APTGP consists of the following main novel components:

1) Initialisation with the transferred knowledge;
2) An auxiliary population that is evolved alongside the main population;
3) A knowledge exchange mechanism that promotes diversity and quality of the populations;
4) A surrogate model that allows the algorithm to evaluate more individuals;
5) A duplicate removal mechanism that removes redundant individuals when exchanging immigrants.

In order to investigate the contributions of each component to the effectiveness of *APTGP*, we conducted additional experiments with the following versions of *APTGP* in which one or more of the components were disabled.

- *APTGP*-NT: the *APTGP* with No knowledge Transfer in initialisation. It randomly generates the initial GP population for the target instance.
- *APTGP*-NA: the *APTGP* with No Auxiliary population. After the initialisation with transfer, it runs the traditional GP for the target instance.
- *APTGP*-SE: the *APTGP* with a Simple Exchange scheme, which replaces all the low-quality individuals with the immigrants from the other population;
- *APTGP*-ND: the *APTGP* with No Duplicate removal when initialising *APTGP*;
- *APTGP*-SA: the Surrogate Assisted *APTGP*, which has no explicit auxiliary population, but generates an extra 1024 offspring from the main population in each generation as the auxiliary population.

All the variants of *APTGP* except *APTGP*-NA generates 2000 offspring in each generation, evaluate 1024 offspring by the actual evaluation, while the other 1024 offspring by the KNN surrogate model. *APTGP*-NA uses a traditional GPHH process in the target instance, which generates and evaluates 1024 offspring in each generation.

Table VI presents the test performance of *APTGP* and its variants. For each scenario (row), the entry with the best mean value is highlighted in boldface. We also conducted the Friedman test, and show the ranks of each compared algorithm and the p-value at the bottom of the table. As can be seen, *APTGP* has the best test performance (lowest rank of 1.68) and the p-value is close to zero, indicating that the differences among the compared algorithms are statistically significant. For post-hoc pairwise comparison, the adjusted p-values of the Conover pairwise comparisons [65] are given in Table VII.

From the table, it can be seen that *APTGP* significantly outperformed *APTGP*-NA, *APTGP*-SE and *APTGP*-ND. The advantage of *APTGP* over *APTGP*-NA indicates that without using the auxiliary population in the search process, the performance of *APTGP* degrades significantly. This is an indicator of the important role that the auxiliary population plays in the algorithm. On the other hand, using an auxiliary population with a simple knowledge exchange mechanism is not enough for achieving significant improvements, since *APTGP*-SE and *APTGP*-NA are statistically similar and *APTGP*-SE was ranked even worse than *APTGP*-NA.

The advantage of *APTGP* over *APTGP*-ND indicates that duplicate removal increases the amount of useful knowledge that is inherited from the source which in turn, highlights the negative impact that the presence of duplicates in the source knowledge can have on the performance on the target instance.

Compared with *APTGP*-NT, the p-value was 0.02, which is very close to the significance level of the Bonferroni correction ($0.05/4 = 0.0125$ for the four comparisons between APTGP and other four algorithms). From Table VI, we see that the rank of *APTGP*-NT is 2.62, closer to the rank of *APTGP* than *APTGP*-NA, *APTGP*-SE and *APTGP*-ND, and obtained the best test performance on 10 scenarios. This shows that even without the initial transfer, *APTGP* can still be quite effective due to the use of auxiliary population for knowledge transfer during the search process. Meanwhile, the considerable difference between *APTGP* and *APTGP*-NT (rank 1.68 vs rank 2.62) shows the effectiveness of the knowledge transfer in the initial population of the target instance.

APTGP-SA showed the closest test performance to *APTGP*. Its rank was 2.6, and the p-value was 0.02, slightly larger than the corrected significance level of 0.0125. It achieved the best test performance on 11 scenarios. However, we can still see the advantage of *APTGP* over *APTGP*-SA, as *APTGP* still has a much better rank (1.68 vs 2.6) and performed the best on 21 scenarios. This verifies the effectiveness of using the auxiliary population instead of directly generating extra offspring and evaluate using the surrogate model.

It should also be noted that *APTGP*-NA and *APTGP*-SE had similar performances. Both algorithms transfer unique individuals, but *APTGP*-SE additionally performs a simple knowledge exchange that does not consider the possible presence of duplicates. Hence, not only the algorithm does not benefit from the exchange of knowledge, its slightly inferior performance indicates that the exchange was even harmful. This could be attributed to possibility that the exchange mechanism may have increased duplicates in the main population. The damage of this effect is to the point that *APTGP*-NT, which does not utilise any transferred knowledge, has a significantly better performance than *APTGP*-SE. On the other hand, *APTGP*-NT performs rather similarly to *APTGP*-NA.

B. Phenotypic Diversity

The GPHH method for solving UCARP is known to suffer from the loss of population diversity during the evolutionary process [16], [19]. This property can negatively impact the quality of knowledge transfer since the transfer of duplicates reduces the amount of useful knowledge that can be transferred and may run into the risk of trapping the search process in poor local optima. This understanding was one of the key considerations for designing *APTGP*. In this subsection, we investigate how effective *APTGP* was in maintaining and increasing the diversity during the search process. For this purpose, we employ the entropy measure [67] for calculating the population diversity. To compute the entropy of the population Pop, we grouped the similar individuals into a set of clusters $\mathcal{C}$ using the DBScan clustering algorithm [68], where each individual is characterised by the phenotypic vector [49] and the cluster

TABLE VI: Test performance of 30 independent runs of *APTGP* and its variants (mean ± std).

Scn.	APTGP-NT	APTGP-NA	APTGP-SE	APTGP-ND	APTGP-SA	APTGP
1	91.1±0.1	91.1±0.1	91.1±0.1	91.1±0.1	91.1±0.1	**91.1±0.0**
2	544.7±5.2	554.3±9.4	556.8±10.2	547.4±7.2	546.7±9.0	**542.9±10.7**
3	595.8±6.4	604.6±9.2	604.2±9.4	**594.6±8.4**	596.0±10.1	595.0±9.7
4	**627.8±5.4**	647.8±13.8	648.4±16.6	634.0±7.7	633.0±9.8	632.6±7.3
5	58.1±0.1	58.2±0.1	58.3±0.2	58.1±0.1	58.1±0.1	**58.1±0.1**
6	417.7±7.0	424.3±9.5	426.0±8.9	419.8±7.6	418.4±7.6	**417.6±7.2**
7	**425.0±5.5**	430.6±7.7	432.3±7.4	427.8±5.5	428.2±5.6	427.6±5.8
8	425.8±4.6	431.8±5.5	431.0±6.8	423.6±7.6	425.1±7.0	**423.3±5.5**
9	572.6±3.0	576.3±3.0	578.1±3.8	573.4±3.9	574.7±11.0	**572.0±4.6**
10	**335.0±4.4**	337.4±2.0	338.0±1.8	335.3±3.4	335.3±3.5	335.1±4.0
11	341.1±3.9	343.1±3.9	345.4±4.4	341.8±5.1	**339.9±4.1**	340.6±4.1
12	**544.7±5.2**	550.6±8.2	557.4±11.8	549.0±7.0	545.0±7.6	544.8±8.0
13	595.8±6.4	604.0±9.3	604.7±11.2	596.1±9.8	595.6±9.2	**595.4±6.2**
14	**627.8±5.4**	642.0±11.6	649.8±13.1	634.5±8.3	631.4±9.1	630.9±5.7
15	335.0±4.4	337.5±2.1	337.9±3.3	334.5±4.0	335.2±3.1	**334.4±3.2**
16	442.2±4.0	444.7±5.6	443.5±7.6	442.5±4.3	**439.7±8.5**	441.4±5.6
17	**317.0±3.4**	321.0±5.1	321.8±5.0	318.0±4.3	317.6±4.3	317.1±3.8
18	354.5±3.8	356.9±4.2	356.4±3.8	352.3±3.0	**352.0±4.0**	352.2±2.4
19	355.9±2.9	359.1±5.5	358.5±5.0	357.3±3.8	355.9±4.1	**354.8±2.6**
20	356.8±1.6	358.0±1.3	358.3±1.1	356.9±1.5	**356.6±2.3**	356.6±1.2
21	337.2±5.3	338.1±5.0	340.5±3.4	336.7±5.8	334.2±5.6	**332.6±5.9**
22	349.8±1.8	350.6±3.2	350.9±2.3	350.8±2.9	**349.0±2.9**	349.7±2.4
23	355.6±0.1	356.4±1.5	356.7±1.7	**355.6±0.1**	355.8±1.0	355.7±0.7
24	310.9±0.8	308.8±2.9	309.7±1.3	309.1±2.6	**307.8±3.1**	307.9±4.0
25	389.0±0.1	389.1±0.1	389.1±0.2	**389.0±0.1**	389.1±0.1	389.0±0.1
26	359.0±3.1	362.0±2.6	362.8±2.6	361.1±3.1	359.5±3.9	**357.8±4.9**
27	335.7±3.9	339.9±6.8	340.3±6.1	337.2±8.5	335.5±4.6	**335.1±4.6**
28	379.8±5.2	382.9±6.6	385.1±5.3	379.9±8.8	379.7±6.2	**378.3±6.5**
29	382.1±2.0	381.3±4.0	383.8±3.5	381.4±3.9	**378.9±5.2**	381.0±2.7
30	350.7±2.3	351.1±1.1	351.0±1.6	350.8±1.6	350.8±2.2	**349.9±3.2**
31	323.5±4.9	324.0±4.6	323.9±4.5	320.6±5.6	321.2±5.2	**320.1±4.7**
32	442.2±4.0	439.6±6.3	442.6±4.4	439.3±6.3	**438.9±6.7**	439.9±6.4
33	448.0±0.5	448.5±1.3	448.6±1.4	448.2±1.4	**447.8±0.5**	448.0±0.6
34	382.7±3.4	384.6±5.2	385.6±4.9	384.6±5.5	**381.1±4.1**	382.9±4.6
35	367.8±0.9	369.2±2.9	369.1±2.5	367.8±2.2	367.7±2.0	**367.1±1.9**
36	**318.5±1.2**	322.5±3.8	322.9±4.2	320.3±2.2	320.2±1.9	319.5±1.7
37	163.8±1.3	165.1±1.4	165.6±1.7	164.5±1.3	**163.7±1.4**	163.8±1.4
38	366.6±5.7	374.7±7.6	375.5±7.3	369.5±6.0	369.8±6.4	**366.6±5.9**
39	**407.7±7.1**	412.4±7.1	415.7±7.1	411.6±5.6	409.6±6.8	409.0±8.3
40	341.1±3.9	346.8±5.9	346.7±5.1	341.9±4.9	341.4±4.8	**340.8±4.9**
41	350.7±2.3	351.6±2.6	351.7±2.5	350.5±4.2	351.1±1.7	**350.3±3.1**
42	164.6±1.0	164.7±1.6	165.4±1.6	164.5±1.5	164.0±1.4	**163.9±1.4**
43	457.4±7.4	456.9±7.8	458.9±6.2	453.8±7.4	453.8±6.8	**451.8±7.3**
44	**424.2±4.3**	426.0±2.9	426.1±3.3	425.9±2.3	424.7±3.9	424.3±4.3
45	**496.2±3.4**	497.7±3.4	497.3±3.9	496.5±3.7	496.8±2.7	496.5±3.4
Rank	2.62	5.1	5.72	3.37	2.6	1.68
Friedman's p-value						**1.11e-16**

TABLE VII: Post-hoc comparison of the *APTGP* and its variants.

	APTGP-NA	APTGP-SE	APTGP-ND	APTGP-SA	APTGP
APTGP-NT	**1.7e-08**	**7.8e-13**	0.07	0.95	**0.02**
APTGP-NA	–	0.079	**7.8e-05**	**1.5e-08**	**3.7e-14**
APTGP-SE	–	–	**2.2e-08**	**7.40e-13**	**4.1e-19**
APTGP-ND	–	–	–	0.067	5.6e-05
APTGP-SA	–	–	–	–	**0.02**

radius was set to zero. Then, the entropy of the population is calculated as $entropy(Pop) = -\sum_{c\in\mathcal{C}} \frac{|c|}{|Pop|} \log \frac{|c|}{|Pop|}$.

Figure 8 presents the distribution of the entropy of the populations during the evolutionary process of the compared algorithms. For each algorithm, the entropy of the populations in all the generations of all the 30 runs are taken into account. From Figure 8, we can clearly see that *APTGP* managed to reach a much higher entropy (better diversity) of the populations than the other algorithms.

To understand how the population entropy changed over the course of evolution, the average population entropy over the 30 runs is plotted in Figure 9 against GP generation for a few scenarios (we observed similar patterns for other scenarios as well). As can be easily seen from the figure, all the compared algorithms started with a high degree of diversity. However, as the evolution proceeded, all the other algorithms except *APTGP* lost their diversity in the population rather quickly. On the other hand, although *APTGP* also lost some of its diversity, it managed to maintain a high entropy (diversity) over time. Another interesting observation in Figure 9 is that the initial entropy of *APTGP* and SUFullTree is higher that other methods. This can be explained by considering the fact that both *APTGP* and SUFullTree algorithms place a high emphasis on creating a very diverse initial population.

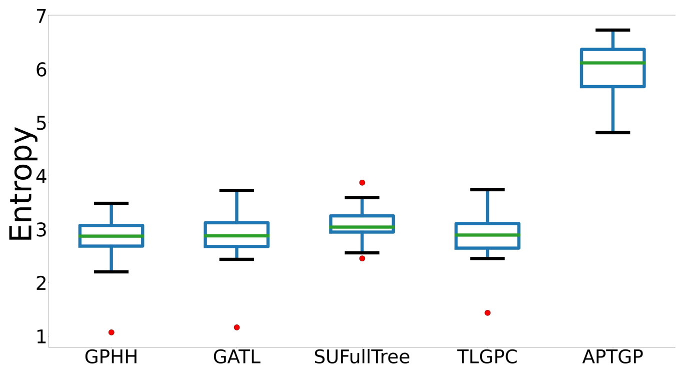


Fig. 8: Population entropy of the compared algorithms.

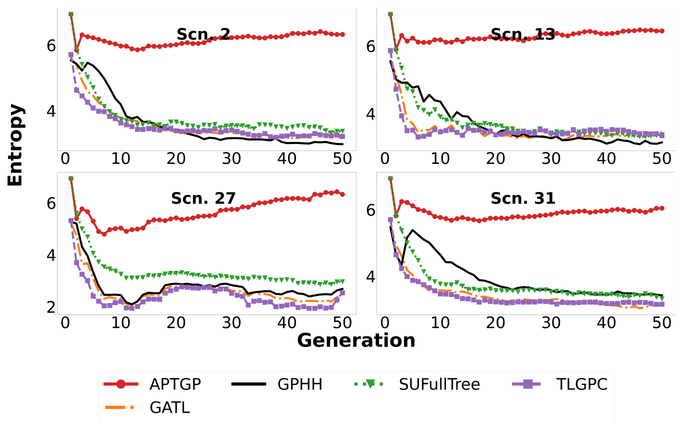


Fig. 9: The curve of the population entropy of the compared algorithms for some representative scenarios.

C. Duplicate Removal

In the knowledge exchange phase of *APTGP* (i.e. Algorithm 6), when selecting individuals in a population to be replaced with the incoming immigrants from the other population, the duplicates in the population are first considered for replacement. If the population does not contain enough duplicates, then low-quality unique individuals are selected to be replaced. Figure 10a presents the number of duplicates selected and replaced in each population over the generations of *APTGP*, averaged over all runs on all the scenarios. As can be seen, throughout the evolutionary process, there exist a large number of duplicates in the population. The number of duplicates decrease slowly throughout the evolution as is suggested by the downward trend of the plot and the increased standard deviation. On average, 298.67 duplicates were replaced in each generation of *APTGP*, with a standard deviation of 7.14. Overall, we can see that there are a large amount of duplicates in the population of *APTGP*, despite the pressure of the duplicate removal mechanism in *APTGP*.

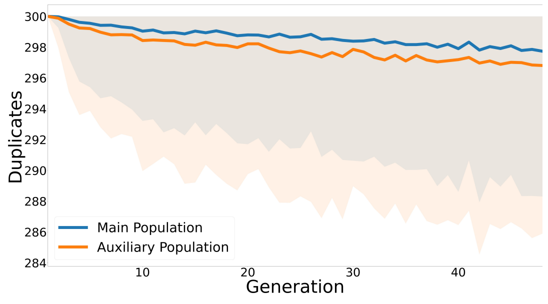
According to Algorithm 1, the duplicate replacement is performed after the breeding operation. Since the population diversity is maintained at a high level at the end of each generation, as discussed in Subsection VI-B, the breeding operator is the main reason for introducing duplicates. This observation indicates the need for designing more effective breeding operators for solving UCARP.

Figure 10a suggests that one may remove more duplicates and achieve better performance by increasing the value of the η parameter. However, when we investigated this possibility, we did not observe any significant improvement in the results. On the contrary, there were cases in which the performance became slightly worse. One possible reason for this phenomenon is that as the value of η increases, more unique individuals are likely to be discarded in favour of the transferred immigrants. In such cases, *APTGP* does benefit from the possible improvement in diversity that immigrants could provide and it can just benefit from the exchange of the knowledge that immigrants contain. However, as the GP evolution proceeds, the quality of the population also increases. As a result, when unique individuals are replaced with the immigrants, it is more likely that the population replaces good individuals with the immigrants which are not guaranteed to be good (considering that the immigrants may go through mutation before being transferred).

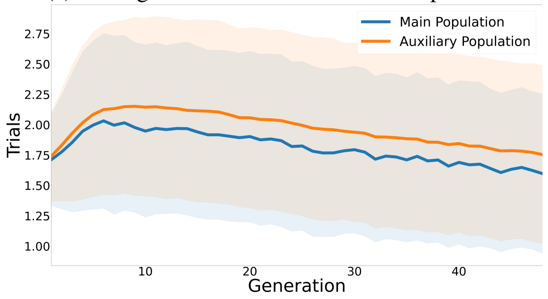
Figure 10b presents the average number of trials (mutations) for *APTGP* to obtain a unique individual as an immigrant for knowledge exchange. As we can see from the figure, in the early generations, the algorithm is is more likely to find out that the candidate immigrant is redundant in the target. However, as the algorithm proceeds, this likelihood decreases slightly because the diversity in both populations increases (as is seen in the entropy plots in Figure 9).

D. Transferred Unique Individuals

Previous studies [16], [18], [19] have indicated that GP may create numerous phenotypic duplicates when it solves UCARP, which can reduce the effectiveness of knowledge transfer. This finding motivated the design of the `RemoveDuplicate` method in *APTGP*. To verify the effectiveness of the `RemoveDuplicate` method, we recorded the number of unique individuals that could be transferred to the target problem. Figure 11 presents the distribution of the unique individuals that were found in each knowledge source over the 30 runs. It should be noted that some scenarios share the same knowledge source. However, since the decision situations needed for phenotypic characterisation are obtained from the target problem, the number of duplicates varies even when the knowledge source is the same.



(a) Average number of the removed duplicates



(b) Average number of trials to find a unique immigrant

Fig. 10: Dynamics of the duplicate removal mechanism.

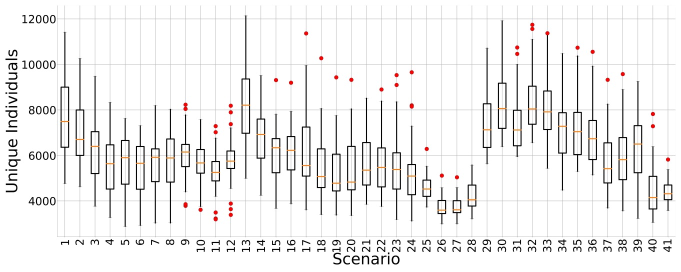


Fig. 11: Number of the transferred unique individuals for each scenario

According to our experiment settings, 1024×50 individuals have been examined in the GP process for solving the source problem. As can be seen in Figure 11, there were at most roughly around 12000 unique individuals in the knowledge source (scenario 13). There are other scenarios in which this number falls even below 2000 individuals (e.g., scenarios 5, 6, 7, 8). Consequently, if the presence of this large number of duplicates is not considered, then the quality of initial knowledge transfer will degrade. The main reason for this effect is that the duplicates which initialise GP will limit the pool of useful information that the search can begin with. As a consequence, GP will require more effort for investigating the search space. To verify this, we considered *APTGP*-NA that does not utilise the auxiliary population, and compared its performance against the results obtained with the FullTree and GPHH algorithms with Friedman's test. The test ranked *APTGP*-NA as the best and the post-hoc analysis revealed it to be significantly better than both other methods. On the other hand, the analysis revealed GPHH, which does not utilise any transferred knowledge, and FullTree to be statistically similar.

For further illustration, an example of the unique transferred individuals by *APTGP* is shown in Section S-III of the supplementary file.

VII. Conclusions

In this work, a novel transfer optimisation algorithm for GP, called *APTGP*, was proposed to evolve routing policies for UCARP. In *APTGP*, all the routing policies that GP has examined for solving a source problem are considered as the transferable knowledge. First, we propose to remove duplicates before using the source individuals to initialise the GP population for the target problem. Then, during the search process, the transferred knowledge is retained and evolved in a separate auxiliary population alongside the main population. The auxiliary population is evolved with a surrogate model that is learned from the main population. The main purpose of the auxiliary population is to help GP address the issue of losing its population diversity and increase its exploration capabilities. To achieve this, an elaborate knowledge exchange mechanism is devised to share high-quality and unique individuals between the main and auxiliary populations.

The effectiveness of *APTGP* has been verified by comparing with the state-of-the-art algorithms on a wide range of transfer scenarios. Our analysis demonstrated that *APTGP* managed to significantly outperform all the state-of-the-art GP algorithms with knowledge transfer. Additionally, we demonstrated that *APTGP* helped GP overcome the limitation of losing population diversity. Furthermore, we conducted a detailed set of control experiments to verify the effectiveness and the contribution of each novel component of *APTGP*.

There are several possible directions for future works. First, the knowledge exchange mechanism of *APTGP* searches the receiving population to ensure the candidate individual for knowledge transfer is not present there. However, our mechanism only considers the current population and does not check if the candidate has been seen in earlier generations or not. We intend to address this shortcoming in future. Also, the knowledge exchange mechanism of *APTGP* does not consider the possibility that the set of immigrants may contain duplicates that, through transfer, may introduce duplicates in the receiving population. Furthermore, *APTGP* is designed based on the assumption that the source and target problems are related and hence, does not have any measures to prevent negative transfer when this assumption is not valid. Finally, the policies evolved by *APTGP* tend to be large and difficult to interpret. In future, we will improve *APTGP* to produce more interpretable policies and also, we plan to test our algorithm on larger real-world datasets, such as the real-world waste collection datasets in [69].

References

[1] Y. Mei, K. Tang, and X. Yao, "Capacitated arc routing problem in uncertain environments," in *Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[2] J. Liu, K. Tang, and X. Yao, "Robust Optimization in Uncertain Capacitated Arc Routing Problems: Progresses and Perspectives," *IEEE Computational Intelligence Magazine*, vol. 16, no. 1, pp. 63–82, 2021.

[3] L. Chen, M. Gendreau, M. H. Hà, and A. Langevin, "A robust optimization approach for the road network daily maintenance routing problem with uncertain service time," *Transportation Research Part E: Logistics and Transportation Review*, vol. 85, pp. 40–51, 2016.

[4] J. F. Campbell and A. Langevin, *Roadway Snow and Ice Control*. Boston, MA: Springer US, 2000, pp. 389–418.

[5] E. B. Tirkolaee, I. Mahdavi, M. M. S. Esfahani, and G. W. Weber, "A hybrid augmented ant colony optimization for the multi-trip capacitated arc routing problem under fuzzy demands for urban solid waste management," *Waste Management and Research*, vol. 38, pp. 156–172, 2020.

[6] S. Wøhlk, *A Decade of Capacitated Arc Routing*. Springer, 2008, pp. 29–48.

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

[8] Y. Liu and Y. Mei, "Automated Heuristic Design Using Genetic Programming Hyper-Heuristic for Uncertain Capacitated Arc Routing Problem," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 290–297.

[9] Y. Mei and M. Zhang, "Genetic Programming Hyper-heuristic for Multi-vehicle Uncertain Capacitated Arc Routing Problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2018, pp. 141–142.

[10] J. MacLachlan, Y. Mei, J. Branke, and M. Zhang, "Genetic Programming Hyper-Heuristics with Vehicle Collaboration for Uncertain Capacitated Arc Routing Problems," *Evolutionary Computation*, pp. 563–593, 2019.

[11] J. Branke, T. Hildebrandt, and B. Scholz-Reiter, "Hyper-heuristic Evolution of Dispatching Rules: A Comparison of Rule Representations," *Evolutionary Computation*, vol. 23, no. 2, pp. 249–277, 06 2015.

[12] S. J. Louis and J. McDonnell, "Learning with case-injected genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 4, pp. 316–328, 2004.

[13] A. Gupta, Y. S. Ong, and L. Feng, "Insights on Transfer Optimization: Because Experience is the Best Teacher," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 51 – 64, 2018.

[14] B. Al-Helali, Q. Chen, B. Xue, and M. Zhang, "A new imputation method based on genetic programming and weighted KNN for symbolic regression with incomplete data," *Soft Computing*, vol. 25, no. 8, pp. 5993–6012, apr 2021.

[15] ——, "Multi-tree genetic programming with new operators for transfer learning in symbolic regression with incomplete data," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 6, pp. 1049–1063, 2021.

[16] M. A. Ardeh, Y. Mei, and M. Zhang, "Transfer Learning in Genetic Programming Hyper-heuristic for Solving Uncertain Capacitated Arc Routing Problem," in *Congress on Evolutionary Computation*, Wellington, New Zealand, 2019, pp. 49–56.

[17] ——, "A GPHH with Surrogate-assisted Knowledge Transfer for Uncertain Capacitated Arc Routing Problem," in *Proceedings of the Symposium Series on Computational Intelligence*. IEEE, 2021, pp. 2786–2793.

[18] ——, "Genetic Programming Hyper-heuristic with Knowledge Transfer for Uncertain Capacitated Arc Routing Problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2019, pp. 334–335.

[19] ——, "A Novel Genetic Programming Algorithm with Knowledge Transfer for Uncertain Capacitated Arc Routing Problem," in *PRICAI 2019: Trends in Artificial Intelligence*. Springer, 2019, pp. 196–200.

[20] ——, "Genetic Programming Hyper-Heuristics with Probabilistic Prototype Tree Knowledge Transfer for Uncertain Capacitated Arc Routing Problems," in *Congress on Evolutionary Computation*. Glasgow, UK: IEEE, 2020, pp. 1–8.

[21] T. T. H. Dinh, T. H. Chu, and Q. U. Nguyen, "Transfer learning in genetic programming," in *Congress on Evolutionary Computation*, 2015, pp. 1145–1151.

[22] E. Haslam, B. Xue, and M. Zhang, "Further investigation on genetic programming with transfer learning for symbolic regression," in *Congress on Evolutionary Computation*. IEEE, 2016, pp. 3598–3605.

[23] K. Deb, *Two Approaches for Single and Multi-Objective Dynamic Optimization*. Berlin, Heidelberg: Springer, 2013, pp. 99–116.

[24] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary Dynamic Optimization: A Survey of the State of the Art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, oct 2012.

[25] J. Wang, K. Tang, J. A. Lozano, and X. Yao, "Estimation of the Distribution Algorithm With a Stochastic Local Search for Uncertain Capacitated Arc Routing Problems," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 96–109, 2016.

[26] J. Wang, K. Tang, and X. Yao, "A memetic algorithm for uncertain Capacitated Arc Routing Problems," in *Proceedings of the Workshop on Memetic Computing*. IEEE, 2013, pp. 72–79.

[27] H. Tong, L. L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "Towards novel meta-heuristic algorithms for dynamic capacitated arc routing problems," in *Parallel Problem Solving from Nature – PPSN XVI*,

T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, and H. Trautmann, Eds. Cham: Springer, 2020, pp. 428–440.

[28] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, USA: MIT, 1992.

[29] A. De Lorenzo, A. Bartoli, M. Castelli, E. Medvet, and B. Xue, "Genetic programming in the twenty-first century: a bibliometric and content-based analysis from both sides of the fence," *Genetic Programming and Evolvable Machines*, pp. 181–204, 2019.

[30] 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 Transactions on Cybernetics*, 2020.

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

[32] ——, "A Multi-Objective Genetic Programming Hyper-Heuristic Approach to Uncertain Capacitated Arc Routing Problems," in *IEEE Congress on Evolutionary Computation*, 2020, pp. 1–8.

[33] S. Wang, Y. Mei, M. Zhang, and X. Yao, "Genetic programming with niching for uncertain capacitated arc routing problem," *IEEE Transactions on Evolutionary Computation*, 2021, doi:10.1109/TEVC.2021.3095261.

[34] Y. Liu, Y. Mei, M. Zhang, and Z. Zhang, "A predictive-reactive approach with genetic programming and cooperative coevolution for the uncertain capacitated arc routing problem," *Evolutionary Computation*, vol. 28, no. 2, pp. 289–316, 2019.

[35] B. Koçer and A. Arslan, "Genetic transfer learning," *Expert Systems with Applications*, vol. 37, no. 10, pp. 6997–7002, 2010.

[36] M. Iqbal, B. Xue, H. Al-Sahaf, and M. Zhang, "Cross-Domain Reuse of Extracted Knowledge in Genetic Programming for Image Classification," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 4, pp. 569–587, 2017.

[37] D. O'Neill, H. Al-Sahaf, B. Xue, and M. Zhang, "Common subtrees in related problems: A novel transfer learning approach for genetic programming," in *Proceedings of IEEE Congress on Evolutionary Computation*, 2017, pp. 1287–1294.

[38] L. Feng, Y. S. Ong, S. Jiang, and A. Gupta, "Autoencoding evolutionary search with learning across heterogeneous problems," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 5, pp. 760–772, oct 2017.

[39] L. Feng, L. Zhou, J. Zhong, A. Gupta, Y. S. Ong, K. C. Tan, and A. K. Qin, "Evolutionary Multitasking via Explicit Autoencoding," *IEEE Transactions on Cybernetics*, vol. 49, no. 9, pp. 3457–3470, sep 2019.

[40] L. Feng, W. Zhou, W. Liu, Y.-S. Ong, and K. C. Tan, "Solving Dynamic Multiobjective Problem via Autoencoding Evolutionary Search," *IEEE Transactions on Cybernetics*, pp. 1–14, oct 2020.

[41] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol, "Stacked denoising autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[42] L. Feng, Y. S. Ong, M. H. Lim, and I. W. Tsang, "Memetic Search with Interdomain Learning: A Realization between CVRP and CARP," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 644–658, 2015.

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

[44] M. Pelikan and M. W. Hauschild, "Distance-based bias in model-directed optimization of additively decomposable problems," in *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation*. New York, USA: ACM Press, 2012, pp. 273–280.

[45] M. W. Hauschild and M. Pelikan, "Intelligent bias of network structures in the hierarchical BOA," in *Proceedings of the 11th Annual Genetic and Evolutionary Computation Conference*. New York, USA: ACM Press, 2009, pp. 413–420.

[46] M. A. Ardeh, Y. Mei, and M. Zhang, "Diversity-driven Knowledge Transfer for GPHH to Solve Uncertain Capacitated Arc Routing Problem," in *Proceedings of the Symposium Series on Computational Intelligence*. IEEE, 2020, pp. 2407–2414.

[47] ——, "Surrogate-Assisted Genetic Programming with Diverse Transfer for the Uncertain Capacitated Arc Routing Problem," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2021, pp. 628–635.

[48] R. Cheng, C. He, Y. Jin, and X. Yao, "Model-based evolutionary algorithms: a short survey," *Complex & Intelligent Systems*, vol. 4, no. 4, pp. 283–292, 2018.

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

[50] H. Tong, C. Huang, L. L. Minku, and X. Yao, "Surrogate models in evolutionary single-objective optimization: A new taxonomy and experimental study," *Information Sciences*, vol. 562, pp. 414–437, 2021.

[51] A. Gupta, Y. S. Ong, and L. Feng, "Multifactorial Evolution: Toward Evolutionary Multitasking," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 343–357, 2016.

[52] L. Feng, L. Zhou, A. Gupta, J. Zhong, Z. Zhu, K.-C. Tan, and K. Qin, "Solving Generalized Vehicle Routing Problem With Occasional Drivers via Evolutionary Multitasking," *IEEE Transactions on Cybernetics*, vol. PP, pp. 1–14, 2019.

[53] K. K. Bali, Y. S. Ong, A. Gupta, and P. S. Tan, "Multifactorial Evolutionary Algorithm with Online Transfer Parameter Estimation: MFEA-II," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 69–83, feb 2020.

[54] L. Zhou, L. Feng, K. C. Tan, J. Zhong, Z. Zhu, K. Liu, and C. Chen, "Toward Adaptive Knowledge Transfer in Multifactorial Evolutionary Computation," *IEEE Transactions on Cybernetics*, pp. 1–14, mar 2020.

[55] 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 Transactions on Cybernetics*, mar 2020.

[56] J. Zhong, L. Feng, W. Cai, and Y. Ong, "Multifactorial Genetic Programming for Symbolic Regression Problems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, jul 2018.

[57] M. A. Ardeh, Y. Mei, and M. Zhang, "A Novel Multi-Task Genetic Programming Approach to Uncertain Capacitated Arc Routing Problem," in *Proceedings of the ACM Genetic and Evolutionary Computation Conference*. ACM, 2021, pp. 759—-767.

[58] R. Kendy Arakaki and F. Luiz Usberti, "An efficiency-based path-scanning heuristic for the capacitated arc routing problem," *Computers and Operations Research*, vol. 103, pp. 288–295, mar 2019.

[59] G. Ruan, L. L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "When and How to Transfer Knowledge in Dynamic Multi-objective Optimization," in *IEEE Symposium Series on Computational Intelligence*, 2019, pp. 2034–2041.

[60] ——, "Computational Study on Effectiveness of Knowledge Transfer in Dynamic Multi-objective Optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2020, pp. 1–8.

[61] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, in *Introduction to Algorithms*, 3rd ed. MIT Press and McGraw-Hill, 2009, ch. Hash Table, p. 1320.

[62] R. G. Reynolds, "An introduction to cultural algorithms," in *Proceedings of the Third Annual Conference on Evolutionary Programming*. USA: World Scientific Press, 1994, pp. 131–139.

[63] T.-T. Nguyen and X. Yao, "An experimental study of hybridizing cultural algorithms and local search," *International journal of neural systems*, vol. 18 1, pp. 1–17, 2008.

[64] M. G. Kendall, "A New Measure of Rank Correlation," *Biometrika*, vol. 30, no. 1-2, pp. 81–93, jun 1938.

[65] W. J. Conover, *Practical Nonparametric Statistics*. John Wiley & Sons, 1998, vol. 350.

[66] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.

[67] E. K. Burke, S. Gustafson, and G. Kendall, "Diversity in Genetic Programming: An Analysis of Measures and Correlation with Fitness," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 47–62, feb 2004.

[68] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996, pp. 226–231.

[69] W. Lan, Z. Ye, P. Ruan, J. Liu, P. Yang, and X. Yao, "Region-focused Memetic Algorithms with Smart Initialisation for Real-world Large-scale Waste Collection Problems," *IEEE Transactions on Evolutionary Computation*, 2021, doi: 10.1109/TEVC.2021.3123960.