Transfer Learning in Genetic Programming Hyper-heuristic for Solving Uncertain Capacitated Arc Routing Problem

Mazhar Ansari Ardeh
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
mazhar.ansariardeh@ecs.vuw.ac.nz

Yi Mei
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
yi.mei@ecs.vuw.ac.nz

Mengjie Zhang
School of Engineering and *Computer Science*
Victoria University of Wellington
Wellington, New Zealand
mengjie.zhang@ecs.vuw.ac.nz

***Abstract*—Uncertain Capacitated Arc Routing Problem (UCARP) is a combinatorial optimization problem that has many important real-world applications. Genetic programming (GP) is a powerful machine learning technique that has been successfully used to automatically evolve routing policies for UCARP. Generalisation is an open issue in the field of UCARP and in this direction, an open challenge is the case of changes in number of vehicles which currently leads to new training procedures to be initiated. Considering the expensive training cost of evolving routing policies for UCARP, a promising strategy is to learn and reuse knowledge from a previous problem solving process to improve the effectiveness and efficiency of solving a new related problem, i.e. transfer learning. Since none of the existing GP transfer methods have been used as a hyper-heuristic in solving UCARP, we conduct a comprehensive study to investigate the behaviour of the existing GP transfer methods for evolving routing policy in UCARP, and identify the potentials of existing methods. The results suggest that the existing methods applying subtree transfer cannot scale well to environment changes and cannot be adapted for this purpose. However, applying GP transfer methods is a good option for creating a better initial populations on target domain and though this effect does not last, we can obtain comparable results in the target domain in a much shorter time. Overall, we conclude that UCARP needs stronger and more effective transfer learning methods.**

I. Introduction

Capacitated Arc Routing Problem (CARP), first proposed by Golden and Wong [1], is an optimization problem that is modelled in a connected undirected graph $G(V, E)$ in which nodes represent locations and edges represent routes between locations with non-negative costs and demands on edges. Given a number of identical vehicles each with a capacity, a CARP aims to determine a set of routes to serve tasks with minimal total costs and satisfying some predefined constraints [2]. Many real-world applications can be modelled as CARP such as street watering and snow removal [3] and waste collection [4]. Many variants of CARP have been proposed by scholars [2], [5]. However, most existing studies focused only on static CARP. It is difficult to apply the approaches to the static CARP to real-world and practical cases, in which problem parameters are often stochastic and unknown before edges are served or traversed. To overcome this limitation, Mei et al. proposed an uncertain model of CARP in (denoted as UCARP) with four stochastic parameters, i.e., presence of tasks, demands of tasks, presence of paths and traversal costs of paths [6] and Liu et. al [7] devised a few novel evolutionary methods for solving this model of CARP. A more detailed overview of UCARP and its characteristics is given in Subsections II-A. There are several methods for solving UCARP. One of the most flexible methods is the use of routing policy. A routing policy is a priority function that vehicles use it to determine priority of available tasks to select and serve. It has been shown that routing policy is a powerful tool for solving UCARP that can handle the dynamic and uncertain of this problem very effectively and is considered the state-of-the-art method for solving UCARP. Routing policies for UCARP can be designed manually but doing so requires extensive domain expertise and is time consuming. On the other hand, GP can be utilized to design routing heuristics effective more efficiently. After a routing policy has been trained, vehicles can use it to guide their serving tasks. An interesting feature of the routing policy approach is that even if characteristics of the problem for which the policy is trained change, i.e number of vehicles or nodes, the policy is still able to guide vehicles on the new problem but it has been shown that it may not perform optimally. Consequently, it is more desired to optimize a policy for the new problem which can be time-consuming, especially if the new problem is large. However, since the new problem has emerged from a previously solved problem, it is reasonable to believe that the knowledge in solving the old problem can be exploited for training of new policies for the new problem. This goal falls into the realm of transfer learning. By salvaging the knowledge in a solved problem, transfer learning techniques promise an easier path to better solutions for related but unsolved solution. Consequently, transfer learning is an ideal candidate for tackling the mentioned scalability problem. There are a plethora of transfer learning techniques available in machine learning literature [8], however, none of

them has been considered as a hyper-heuristic approach in evolving routing policies of UCARP. Although some works have been done for similar problems [9], [10], to the best of our knowledge, this is the first effort in addressing the scalability issue of UCARP with transfer learning. This paper aims to achieve the following research goals:

- Review the literature of transfer learning techniques that are designed for GP.
- Proposing a new GP transfer learning method.
- Investigate the applicability of subtree-based transfer learning methods to handling scalability in UCARP.

The majority of current GP transfer methods mainly transfer subtrees and to the best of our knowledge, no GP transfer learning method has been proposed or verified in the context of GP hyper-heuristic and a major contribution of this paper is to use UCARP as a benchmark problem and explore the potentials and shortcomings of subtree-based GP transfer learning methods. By doing so, we intend to answer the main research question of whether (sub)tree-based GP transfer methods are effective for UCARP and what the shortcomings and benefits of using these methods are.

This paper is organized as follows. In Section II, a comprehensive background review is given. In Section III we propose our new transfer learning method. Experiments and results are presented in Section IV. Finally, we give the conclusions and future directions in Section V.

II. Background

A. UCARP

A UCARP problem is a graph $G(V, E)$ in which V is the set of nodes of the graph and E is the set of edges. In UCARP, each vehicle has a capacity Q and at the beginning, all vehicles are located at the depot $v_0 \in V$. Each edge $e \in E$ represents a route and with each route two cost values are associated: (1) a positive stochastic deadheading cost $dc(e)$ and, (2) a positive deterministic serving cost $sc(e)$. Each edge also has a non-negative stochastic demand $d(e)$. In UCARP's terminology, an edge such that $d(e) > 0$ is called a task which should be be served. To solve a UCARP means to find routes for vehicles to serve all tasks so that total cost, which is the total sum of all serving and deadheading costs in each routes, is minimised. In solving a UCARP, following constraints must be respected:

- All vehicles need to start their routes from the depot and end it at depot. However, vehicles are allowed to visit the depot multiple times to replenish their capacity.
- Total amount of demand that each vehicle serves cannot be more than its capacity without returning to the depot to replenish capacity.
- Each task must be served exactly once.

A sampled UCARP instance is a UCARP instance in which stochastic variables are sampled from the distribution function of the corresponding stochastic variable. For demand of tasks, the actual value is revealed after the vehicle serving the task finishes serving it. For deadheading costs, the sampled value is revealed after the vehicle has finished traversing it.

Generally speaking, there are two approaches to solving UCARP: proactive methods that take a robust solution and optimize it and reactive methods that utilize Genetic Programming Hyper-Heuristic (GPHH) to search for routing policies that generate solution for the problem. The work of Wang et al. in [11] and [12] and the work of Mei et al. in [6] are interesting techniques in the category of proactive methods.

Although proactive approaches have been proposed for solving UCARP, they are not flexible enough for real-time and real-world scenarios [7]. To overcome this shortcoming, Liu et al. proposed a Genetic Programming-based Hyper Heuristic method for solving UCARP [7] and evaluated the performance of the proposed method on benchmark instances described in [6]. In this work, a routing policy is utilized to model a decision making process for helping vehicles select their next tasks after serving the current one. The optimal routing policies are evolved with GP. Mei et al. extended the approach in [7] to a general scenario that can process multiple vehicles for serving tasks [13]. MacLachlan et al. took advantage of GPHH for solving a modified and improved UCARP by adding a new feature into the feature set of UCARP [14].

B. GPHH for UCARP

GP is an extension of the famous Genetic Algorithm in which the members of the algorithm's population are computer programs. Popularized by Koza, it utilizes evolutionary operations to evolve computer programs [15] and has been used extensively for solving a vast variety of problems [16], [17], [18]. Genetic Programming Hyper-Heuristic is a standard GP that is employed to search the space of heuristic functions. In the context of UCARP, the heuristics are the routing policies and GPHH can be exploited to evolve routing policies for vehicles. A standard GPHH has four general steps:

1) Create an initial population of routing policies, represented with GP trees.
2) Evaluate each routing policy in the population and find its fitness, which is defined over a set of training instances.
3) Use standard genetic operators of selection, crossover, mutation and reproduction to create new population from current population.
4) Repeat the steps (2) through (3) until stopping criteria is met.

In this setting, a GP tree is routing policy that guides decision making of vehicles. When a vehicle is idle, all available tasks are considered first. To improve efficiency and reduce size of the set of available task, the collection of available tasks are filtered out by a filtering method. For each task, the routing policy calculates a priority value for each filtered task based on environment and task properties and finally, a task with best priority is selected to be served. Because of the dynamic nature of the problem, it is likely that selected routes lead to failure. As it was mentioned earlier, demand of tasks have a stochastic nature and therefore, when a vehicle arrives at a task, it may find the actual demand greater than available capacity of the vehicle. This situation leads to a *route failure*.

Also, again, because of the random nature of the problem, it is likely that a vehicle arrives at a route e to find it inaccessible (i.e. $dc(e) = \infty$) which leads to an *edge failure* during a vehicle's course of action. It is because of this nature of UCARP that traditional and exact optimization methods that try to find robust solutions will fail to provide a solution that can address these failures effectively. In case of a route failure, the vehicle returns to depot, refills and goes back to serve the failed task. Edge failures are also handled by taking a detour around the failed edge. The fitness function of a policy is defined as the average total cost of the solutions it generates on the predefined training instances.

Compared with manual design of routing policies, which can be time expensive and requires high level of domain knowledge, GPHH is a very viable alternative that does not require extensive domain knowledge or any preplanned solutions, and studies have shown it to be the state-of-the-art for tackling this problem [14], [7], [13]. However, a shortcoming of this method is that, because of the nature of GP, it requires model training. The training phase, though done by machines, can be non-trivial and time consuming. On the other hand, the problem scenario can change over time in reality, e.g. in transportation systems, it is very likely that fleet size ~~be~~ expanded or reduced. It has been shown that when the scenario changes, even if one vehicle is added or removed, the performance of the existing routing policies deteriorates greatly [13] and therefore, it is needed to retrain new routing policies. Intuitively, we can train new routing policies from scratch. However, due to the high computational cost of the training process, it is desirable to have methods that alleviate the training cost by reusing the knowledge gained from previous problem solving to retrain the routing policies for the new problem more efficiently. In the literature of machine learning, this scenario is a prime candidate for transfer learning [19], [8].

C. GP Transfer Learning

Learning is an inherent capability of human beings and is a source of inspiration and motivation for empowering machines with learning capabilities. An interesting feature of the learning ability in humans is the ability of transferring the knowledge learned in one domain to another related or similar (new) domains which can boost the learning efficiency in the new domains. This feature, as useful as it is, is not present in traditional machine learning techniques and consequently, they need to learn new problems from scratch even if similar and related problems are already solved. As a result, they do not benefit from any potential boost in performance or reduction in training and computational efforts from learned knowledge in previous experiences [20], [21].

Transfer learning tries to address this shortcoming and it can be described as "the ability of a system to recognise and apply knowledge and skills learned in previous tasks to novel tasks" [22]. With a focus on GP, transfer learning methods can help GP in three aspects: it can help GP with better initial population and genetic operations, boost the convergence speed of GP, and finally, result in better final solutions [20]. Despite being a rather new concept compared with the main body of machine learning literature, extensive efforts have been done in this area and consequently, and there exists a magnitude of research in this area [8], [19]. Because of the vast extent of the work in transfer learning, the focus of this work will be only on transfer learning methods that have been designed specifically for GP techniques.

One of the early works in applying concepts of transfer learning to an evolutionary algorithm was given in [21]. The paper implements transfer learning as transferring individuals from population of the algorithm solving the source problem to the initial population of the algorithm solving the target problem. In their implementation, the authors selected two sample individuals I_1 and I_2 at each iteration of the source problem, one of which is the best individual in the population and the other is the median individual in terms of fitness, and saved into an "individual pool". For initializing the population of the target problem, 30% of the initial population of the target problem were chosen randomly from this pool [21]. In this paper, we refer to this method as *GTLKnow*.

Dinh et al. [23] present three new algorithms for transfer learning in GP In *FullTree* method, k% best of individuals in the final generation source problem are used as initial individuals. In their second algorithm, named *SubTree*, they considered the idea that there is a good subtree structure that is shared between related problems. In their implementation of this idea, the authors focused on last generation of source problem. To transfer, a random subtree of each individual of final population is selected and added to a pool and then moved to the target problem as initial individuals. The third method, *BestGen*, it is assumed that good knowledge to transfer may appear, not only in the final generation but also during the evolutionary process of source problem. Consequently, in each generation of GP on source problem, k best individuals are sampled into a pool that are finally used as initial population on target.

Iqbal et al. [24] modified the initialization and mutation phases of GP to utilize transfer learning and dubbed the modification as *TLGPCriptor*. The authors considered children of the root node of each individual in the population as a transferable segment of knowledge and called them "code fragment". After training a GP on source task, average fitness of the population in the final generation is calculated and code fragments are extracted from individuals with better than average fitness. These code fragments are stored to be reused on target problem and on a target problem, they are utilized in the initialization phase as well as the mutation phase. For initialization, children of a root node are either selected from the storage with a probability of 0.5 or from terminal or function set of GP. For mutation, a subtree of individual to be mutated will be selected and and replaced with a tree that is created the same way described for initialization.

O'Neil et al. [22] also proposed a novel transfer learning method for solving symbolic regression methods. In their algorithm, dubbed as *Common SubTree in Related Problems*

(CSRP), they first identify the common subtrees that repeat in population of source domain and then the subtrees are converted to functions and added to the function set of GP for solving the target domain.

Considering n source domains, the Fu et al. first train GP populations to learn knowledge from the source domains. To use the knowledge, for each source domain i, p individuals are selected randomly from the final population of the trained GP. For each instance in the target domain, each of the selected individuals are used on the instance to find its class. Output of the individuals are then summed and if the sum is greater than $0.5 * p * n$ it is classified as class 1 else it is classified as class 0 [25].

III. Transfer Learning as Frequent Sub-Tree Transfer

As it is obvious from Section II-C, a large body of work has been dedicated to implementing GP transfer learning as transfer of subtrees. Consequently, in this section, we also propose a novel transfer learning method based on subtree transfer.

It is reasonable to assume that if an individual has a good fitness on source domain, it tends to be a good candidate for extracting good subtrees to be used in target domain as new individuals. Furthermore, if a subtree is repeated many times in good individuals of the source domain, it must be important in the source domain and it should also be more likely to be important in target domain. For this purpose, we consider all the individuals in the final GP population during the training process of the source domain, and extract subtrees from top 50% individuals in terms of their test performance in the source domain using the following three methods:

1) **All**: All the possible subtrees of an individual (dubbed as *FreqSub-root*).
2) **Root Subtree**: Immediate children of the root of the considered individuals (dubbed as *FreqSub-sub*).
3) **Entire**: The entire tree is transferred (dubbed as *FreqSub-root*).

After all subtrees are extracted from the final population of the source domain, they are sorted based on the number of times that they were repeated in the source domain and then, all the most frequent subtrees were transferred as individuals until 50% of the initial population of GP on target problem is filled.

In our experiments, we noticed that size of transferred subtrees are important. Large subtrees usually had a better fitness on both source and target domains but also they tended to afflict GP with code bloats and also, affect diversity of population. On the other hand, small subtrees were also not beneficial as they usually disappeared very early in evolutionary generations. Therefore, we consider a limit on minimum and maximum size of subtrees and only transfer the ones that are within the size limit.

It is worth to point out that our work has some similarities with the work in [22] that extracts subtrees and uses them as functions on target domain. Although the authors achieved good results with their approach on some non-dynamic problems, in our experiments, we noticed that this approach, in the context of UCARP, will lead to hidden code bloat and unacceptable computational cost, because such a function node in a GP tree is indeed a subtree of its own, disguised as single functional node that when is needed to be evaluated, the whole subtree needs to be evaluated. It is can be rather trivial to optimize this procedure for static problems but doing so for a dynamic problem like UCARP can be more challenging.

TABLE I
The algorithms examined in the experiments

Algorithm	Subtree selection mechanism
BestGen	Select best individuals of each generation
GTLKnow	Select best and median individual from each generation
TLGPCriptor	Select random subtrees of final individuals that are better than average.
FreqSub	Select most frequent subtrees in the final population
Fulltree	Select k% of best of individuals of the final poplulation
Subtree	Select a random subtree of each individual of final population

IV. Experimental Studies

A. Experiment Design

To probe the effectiveness of GP-based transfer learning methods for solving UCARP, we consider the scenario in which the number of vehicles is changed from a source problem in a target problem and explore the potentials of some existing transfer methods described in Section II-C and also the method that we proposed in Section III. To be specific, our experiments have three stages. First we consider n vehicles in a UCARP problem and train an appropriate routing policy (GP tree) for it and then evaluate its performance. This is labeled as the source domain. Then we consider a change in the number of vehicles to either $n-1$ or $n+1$ to form a target domain and then use a transfer learning method to extract the knowledge in solving the problem in source domain in the form of subtrees and use it to initialize the GP population for training routing policies in the target domain and finally, performance of the trained routing policy is evaluated.

We will focus on *SubTree*, *FullTree*, *BestGen*, *TLGPCriptor* and *GTLKnow* and we compare their performance on transferring a learned knowledge on the *Ugdb* and *Uval* instances [6]. The subtree selection mechanism of the algorithms are described in Table I. For the *SubTree* and *FullTree* methods, we consider four settings in which 25%, 50%, 75% and 100% of the final population of source domain is considered for transfer and are designated *SubTree25*, *SubTree50*, *SubTree75* and *SubTree100* respectively for the *SubTree* method and *FullTree25*, *FullTree50*, *FullTree75* and *FullTree100* for the *FullTree* method in tables and plots. For the *BestGen* method, we considered two settings in which one and two individuals from each population is considered for transferring and we refer to them respectively as *BestGen1* and *BestGen2* in tables

TABLE II
THE NUMBER OF VEHICLES OF SOURCE AND TARGET DOMAINS IN THE EXPERIMENTS.

SmallDataset	gdb1	gdb1	gdb2	gdb2	gdb21	gdb21
Source	5	5	6	6	6	6
Target	4	6	5	7	5	7
Medium Dataset	gdb8	gdb8	gdb9	gdb9	gdb23	gdb23
Source	10	10	10	10	10	10
Target	9	11	9	11	9	11
Large Dataset	val9C	val9C		val9D		val9D
Source	5	5		10		10
Target	4	6		9		11

and plots. For the *FreqSub* method, we limit the size of transferable subtrees to be within the depth limit of 2 and 5.

We have categorized the experiments into small-sized, medium-sized and large-sized problems, as given in Table II. In each configuration, the source and target domains share the same UCARP instance, but have different numbers of vehicles.

In our experiments, we have selected truncated normal distribution for stochastic variables wherein value of the static instance from dataset are given to μ and $\sigma = 0.2\mu$. For training the priority functions, a training set of 5 samples instances are considered for each generation. The number of test samples was set to 500.

TABLE III
TERMINAL SET

GP 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
GP Setting	**Value**
Population	1024
Crossover rate	0.8
Mutation rate	0.15
Reproduction rate	0.05
Number of generations	50
Elitism	10
Max depth	8

The GP settings and terminal set used in our experiments are based on the work in [13] and is presented in Table III. The function set is $\{+, -, \times, /, min, max\}$ in which the division function $/$ is protected and returns 1 if the denominator is zero. All algorithms are run 30 times independently. Wilcoxon rank sum test with a 0.05 significance level was applied for pairwise comparisons between transfer learning methods and the ~~ease of~~ no knowledge transfer.

TABLE IV
TEST RESULTS OF THE COMPARED ALGORITHMS - DATASET **GDB1**

Scenario	From 5 to 4 vehicles			From 5 to 6 vehicles		
Algorithm	Best	Mean(Stdev)	PVal	Best	Mean(Stdev)	PVal
No Transfer	345.52	357.45 (6.26)	–	352.17	361.08(3.09)	–
BestGen1	348.14	359.36 (10.27)	0.47	351.28	357.46(4.37)	**0.00**
BestGen2	344.47	360.96 (16.71)	0.71	349.25	356.58(4.48)	**0.00**
GTLKnow	350.19	362.25 (8.25)	**0.01**	351.49	357.78(4.38)	**0.00**
TLGPCriptor	349.06	357.43 (5.49)	0.82	351.82	359.56(4.49)	0.16
FreqSub-root	349.77	359.81 (10.55)	0.31	350.11	357.96(4.11)	**0.00**
FreqSub-all	349.28	357.58 (4.74)	0.55	348.79	360.76(6.00)	0.66
FreqSub-sub	350.90	358.60 (4.96)	0.35	351.94	359.27(3.89)	0.11
Fulltree25	350.77	359.85 (6.85)	0.18	349.82	357.91(5.38)	**0.00**
Fulltree50	350.75	362.43 (8.97)	**0.04**	345.69	359.32(6.77)	0.18
Fulltree75	342.34	358.74 (9.73)	0.78	349.87	358.01(4.63)	**0.00**
Fulltree100	350.60	358.95 (5.03)	0.14	349.25	358.03(4.72)	**0.00**
Subtree25	345.33	357.81 (6.79)	0.74	352.51	361.28(3.87)	1.0
Subtree50	345.21	358.95 (6.67)	0.49	350.89	359.76(4.31)	0.22
Subtree75	349.45	358.98 (5.09)	0.37	350.81	362.09(15.21)	0.11
Subtree100	348.72	362.55 (10.09)	**0.01**	350.50	359.77(4.27)	0.05

TABLE V
TEST RESULTS OF THE COMPARED ALGORITHMS - DATASET **GDB2**

Scenario	From 6 to 5 vehicles			From 6 to 7 vehicles		
Algorithm	Best	Mean(Stdev)	PVal	Best	Mean(Stdev)	PVal
No Transfer	378.97	385.80(4.78)	–	373.24	385.49(9.16)	–
BestGen	377.78	388.78(16.41)	0.68	368.32	381.70(4.09)	**0.00**
BestGen	380.90	388.47(19.46)	0.99	371.36	383.39(4.07)	0.46
GTLKnow	382.26	386.36(4.52)	0.26	377.36	383.01(2.39)	0.07
TLGPCriptor	381.22	388.81(9.40)	**0.03**	374.44	383.21(3.88)	0.27
FreqSub-root	382.89	386.37(3.72)	0.57	367.94	382.20(5.84)	**0.03**
FreqSub-all	384.04	389.80(8.02)	**0.01**	381.17	384.00(3.49)	0.26
FreqSub-sub	380.91	385.90(3.85)	0.99	370.60	382.84(3.61)	0.10
Fulltree25	377.42	385.85(3.68)	0.68	371.57	383.13(4.16)	0.12
Fulltree50	381.72	387.16(4.33)	**0.04**	367.98	383.13(6.94)	0.17
Fulltree75	378.95	388.54(12.03)	0.51	371.57	383.95(6.53)	0.01
Fulltree100	379.15	387.14(6.03)	0.59	371.30	383.84(12.17)	0.02
Subtree25	377.78	386.26(4.43)	0.44	377.01	384.47(4.28)	0.88
Subtree50	377.08	387.11(6.29)	0.76	378.89	384.20(3.53)	0.38
Subtree75	379.27	387.37(10.28)	0.92	378.78	384.61(3.65)	0.24
Subtree100	380.97	386.67(6.17)	0.73	368.78	384.27(5.28)	0.58

B. Results

To investigate the potentials of GP-based transfer learning methods for UCARP, a large body of experiments were conducted. Tables IV through to XI present the best, mean and standard deviation of the total cost of the compared algorithms for the experiments in the Table II when a knowledge from a solved problem with a number of vehicles specified in the source row is transferred to an unsolved problem with number of vehicles in target row.

A quick examination of these tables reveal that in almost all cases, none of the transfer learning methods were able to produce a statistically significant improvement compared with the case in which no knowledge transfer is applied. Of course there are cases in which the result is statistically better (e.g. *BestGen1* and *GTLKnow* for *gdb1* in Table IV) but these cases are a minority. Furthermore, one or more of the methods produced better minimum total cost value but nevertheless, statistical test dismiss them. This is also true about our proposed method. There is a case (*gdb1*) in which the *FreqSub-root* method had a statistically significant performance and a few cases that the best value was better than the case of not transferring anything, but overall, this method, like the other ones, does not perform well. We notice that, the methods *FreqSub-all* and *FreqSub-sub* performed slightly

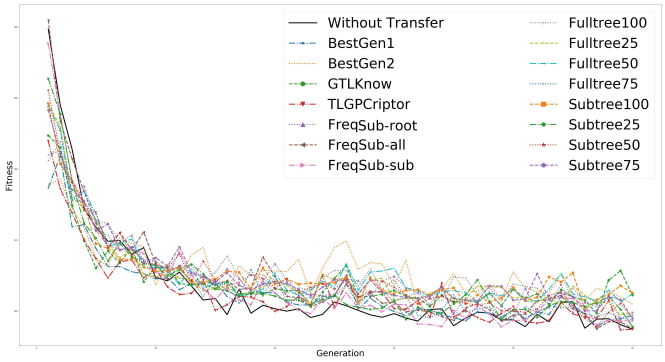

Fig. 1. Means of 30 runs of best of GP generations for dataset gdb1, from 5 to 4 vehicles

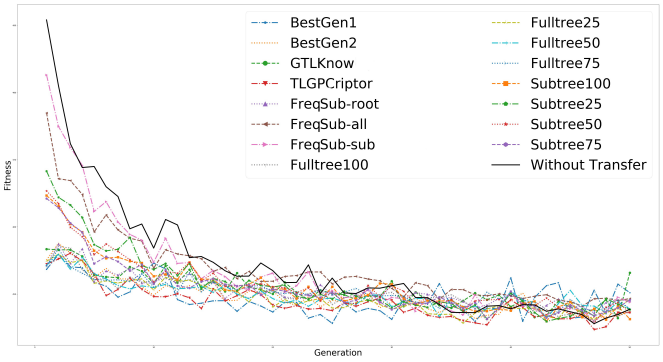

Fig. 2. Means of 30 runs of best of GP generations for dataset gdb8, from 10 to 11 vehicles

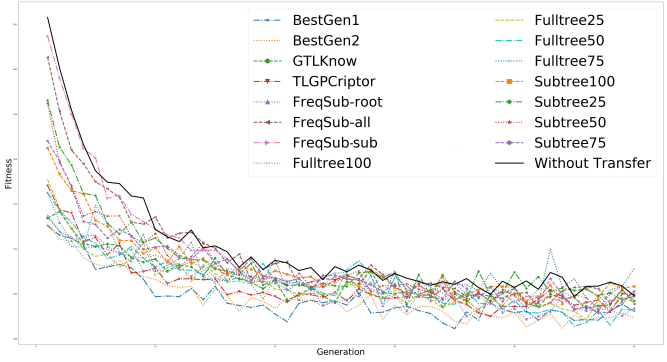

Fig. 3. Means of 30 runs of best of GP generations for dataset gdb23, from 10 to 11 vehicles

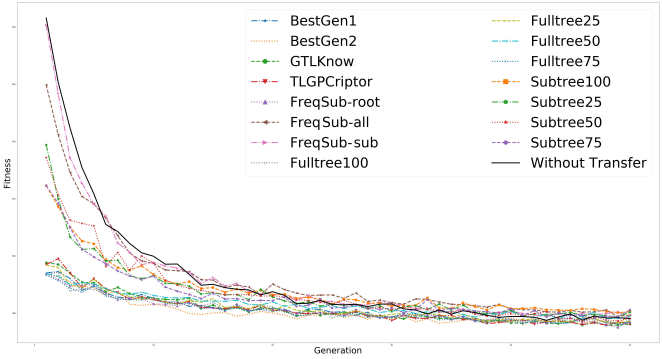

Fig. 4. Means of 30 runs of best of GP generations for dataset val9D, from 10 to 11 vehicles

better than *FreqSub-root* which can be due to more diversity of transfered subtrees. More discussion of this method is given shortly with an example of a transferred frequent subtree.

Considering the dynamic nature of problem, this is not surprising. The current transfer methods try to create a good initial population but they do not consider future generations and hence, are unable to address the environment changes that occur in problem in future generations. It is obvious that any transfer learning method that needs to work for this problem has to consider the dynamic nature of this particular problem. This is further confirmed with the fact that the *BestGen* methods, that samples from all generations of source domain, performs relatively better that other methods which can be attributed to the diversity of its repository. Based on these experiments, we conclude that transfer learning methods that focus only on creating a good initial population are likely be successful in doing so but any initial advantage is likely to be lost on future generations. This is of course in terms of final fitness of optimum solution but even so, the benefit of a good initial population cannot be ignored as it saves computational effort on a problem that is inherently expensive computationally.

To gain a better understanding of the results, mean of best value of each generation of GP over 30 runs of four of the experiments are presented in Fig. 1 through to 4 that depict interesting features of our experiments more clearly. In these plots, it is confirmed that in almost all cases, the use of transfer learning methods has created a better initial population for GP. This property is repeated in all experiments and depicts that the use of sub-tree transfer methods was partially successful but their effect is lost in later generations. However, in almost all test cases, transfer methods were able to produce better initial populations for GP on target domain. Considering the fact that in all the transfer methods in this paper, the computational cost of knowledge transfer is insignificant, these approaches can be considered as viable methods for initializing GP population on target domain and can lead to relatively significant reduction in computational cost of training GP in target domain. For example, a quick inspection of the figures reveals that the *BestGen* method can reduce training time of GP more than 50% which is an important achievement. Our inspections of other results also confirm this finding.

To gain a better understanding we examined to consider the subtrees that were transferred. A quick investigation of experiments revealed interesting findings. For example, in one run of the *FreqSub-sub* algorithm, with a frequency of 500, the tree $(min(0.52, 0.83) - (FRT - CFH))$ is detected as the most frequent item in the final population of source domain. This indicates that the best half of the final population on source domain contained this subtree and further investigation

TABLE VI
TEST RESULTS OF THE COMPARED ALGORITHMS - DATASET **GDB8**

Scenario	From 10 to 9 vehicles			From 10 to 11 vehicles		
Algorithm	Best	Mean(Stdev)	PVal	Best	Mean(Stdev)	PVal
No Transfer	414.27	428.28(8.67)	–	430.62	447.74(6.76)	–
BestGen1	413.39	438.16(20.01)	**0.00**	432.93	450.21(21.38)	0.53
BestGen2	409.77	435.75(22.44)	0.11	431.12	447.60(6.56)	0.95
GTLKnow	420.31	436.58(16.21)	0.07	434.36	453.17(29.95)	0.82
TLGPCriptor	402.20	430.95(13.69)	0.27	432.13	447.23(6.44)	0.95
FreqSub-root	409.69	441.40(16.98)	**0.00**	435.17	449.06(5.81)	0.38
FreqSub-all	405.57	431.73(17.24)	0.50	441.59	449.33(4.84)	0.44
FreqSub-sub	405.18	432.01(15.62)	0.15	431.41	448.79(7.97)	0.49
Fulltree25	414.20	438.21(18.02)	**0.02**	436.13	449.31(6.44)	0.44
Fulltree50	411.03	438.07(15.66)	**0.00**	437.02	447.79(5.02)	0.81
Fulltree75	407.58	436.42(16.95)	**0.02**	435.26	447.80(6.87)	0.94
Fulltree100	417.79	437.66(14.94)	**0.00**	434.34	447.41(6.96)	0.89
Subtree25	409.81	435.46(17.36)	0.07	429.58	447.69(6.95)	0.90
Subtree50	411.56	437.44(16.54)	**0.01**	429.89	447.28(7.42)	0.89
Subtree75	413.67	439.29(17.07)	**0.00**	437.38	449.17(5.18)	0.55
Subtree100	409.46	433.20(12.90)	**0.04**	435.70	446.27(5.90)	0.30

TABLE VII
TEST RESULTS OF THE COMPARED ALGORITHMS - DATASET **GDB9**

Scenario	From 10 to 9 vehicles			From 10 to 11 vehicles		
Algorithm	Best	Mean(Stdev)	PVal	Best	Mean(Stdev)	PVal
No Transfer	372.86	383.99(8.79)	–	378.75	392.20(7.62)	–
BestGen1	373.12	382.90(6.22)	0.86	376.97	394.26(10.35)	0.61
BestGen2	369.12	381.78(5.81)	0.46	378.26	393.01(10.17)	0.92
GTLKnow	373.12	383.53(7.37)	0.99	374.0	391.39(9.35)	0.58
TLGPCriptor	372.46	384.99(6.62)	0.36	376.61	393.36(9.94)	0.63
FreqSub-root	367.14	385.03(9.79)	0.53	378.16	395.60(10.29)	0.49
FreqSub-all	375.96	385.20(6.90)	0.53	372.14	393.91(11.40)	0.90
FreqSub-sub	371.49	383.24(5.87)	0.98	373.03	393.59(9.96)	0.89
Fulltree25	369.69	384.51(8.92)	0.74	379.86	397.36(12.83)	0.09
Fulltree50	374.14	383.74(5.96)	0.74	379.43	397.06(12.37)	0.11
Fulltree75	373.33	383.23(7.01)	0.95	373.89	396.48(13.61)	0.25
Fulltree100	377.83	384.87(7.78)	0.73	380.94	395.94(10.59)	0.21
Subtree25	373.69	383.28(7.93)	0.61	379.50	393.04(11.57)	0.64
Subtree50	373.98	382.46(5.75)	0.73	375.61	396.62(9.94)	**0.02**
Subtree75	368.02	384.43(8.14)	0.67	378.83	395.91(11.47)	0.24
Subtree100	371.65	383.38(8.47)	0.67	380.40	397.44(11.12)	0.06

TABLE VIII
TEST RESULTS OF THE COMPARED ALGORITHMS - DATASET **GDB21**

Scenario	From 6 to 5 vehicles			From 6 to 7 vehicles		
Algorithm	Best	Mean(Stdev)	PVal	Best	Mean(Stdev)	PVal
No Transfer	163.39	167.11(3.20)	–	160.74	164.08(1.41)	–
BestGen1	162.55	166.97(3.57)	0.76	161.64	164.88(2.26)	0.09
BestGen2	162.50	167.53(3.16)	0.37	161.50	164.39(1.91)	0.54
GTLKnow	163.41	166.57(2.07)	0.94	161.84	164.40(1.51)	0.39
TLGPCriptor	161.72	165.61(1.93)	0.05	162.36	164.46(1.33)	0.17
FreqSub-root	162.17	166.58(2.64)	0.64	161.74	164.82(1.53)	0.06
FreqSub-all	162.98	167.32(2.61)	0.24	161.74	164.95(2.26)	0.10
FreqSub-sub	161.09	166.84(2.13)	0.94	162.50	164.26(1.24)	0.92
Fulltree25	163.02	168.41(8.70)	0.58	162.42	164.81(1.89)	0.05
Fulltree50	162.50	166.96(2.39)	0.54	162.26	164.87(2.22)	0.15
Fulltree75	163.28	166.80(2.63)	0.99	161.68	164.25(1.69)	0.62
Fulltree100	161.73	167.19(2.83)	0.59	162.38	164.47(1.42)	0.20
Subtree25	162.18	167.03(3.05)	0.86	162.03	164.82(1.41)	**0.03**
Subtree50	163.63	166.60(2.15)	0.95	161.62	164.83(1.78)	**0.02**
Subtree75	163.27	167.49(2.64)	0.33	162.48	164.37(1.19)	0.19
Subtree100	161.02	167.18(3.16)	0.89	162.00	164.66(2.03)	0.10

TABLE IX
TEST RESULTS OF THE COMPARED ALGORITHMS - DATASET **GDB23**

Scenario	From 10 to 9 vehicles			From 10 to 11 vehicles		
Algorithm	Best	Mean(Stdev)	PVal	Best	Mean(Stdev)	PVal
No Transfer	247.35	251.82(3.48)	–	245.22	249.91(2.59)	–
BestGen1	245.92	249.52(1.81)	**0.00**	244.85	249.22(2.58)	0.33
BestGen2	245.22	250.05(2.54)	**0.01**	242.78	248.92(2.38)	**0.04**
GTLKnow	246.80	250.99(2.39)	0.37	245.57	249.68(1.70)	0.92
TLGPCriptor	246.74	250.97(2.46)	0.34	246.01	249.96(2.56)	0.74
FreqSub-root	246.10	250.50(2.34)	0.12	246.02	249.36(1.71)	0.53
FreqSub-all	246.56	251.43(2.70)	0.61	245.09	249.54(2.87)	0.61
FreqSub-sub	245.45	250.23(2.69)	0.11	245.34	249.87(2.17)	0.75
Fulltree25	246.92	250.71(2.27)	0.24	245.28	249.63(2.36)	0.32
Fulltree50	246.38	251.06(2.57)	0.36	244.54	249.27(2.46)	0.44
Fulltree75	245.66	250.52(2.89)	0.05	245.92	251.11(7.96)	0.81
Fulltree100	247.36	251.29(2.51)	0.41	245.36	250.12(3.40)	0.97
Subtree25	247.18	251.04(2.75)	0.22	244.96	249.90(2.59)	0.94
Subtree50	246.16	251.27(3.66)	0.15	246.93	249.93(2.41)	0.79
Subtree75	246.73	250.75(2.72)	0.25	246.23	250.11(2.70)	0.97
Subtree100	245.63	250.82(3.19)	0.41	246.05	250.32(2.38)	0.55

revealed that in this case, the final population had converged to a very uniform population in which one single individual had dominated the entire population and this deficit in the source domain was translated into a weak and negative transfer to target domain. This simple example illuminates a few weaknesses of (sub)tree transfer methods. First, it reveals that negative transfer is also an issue in this method and if, for any reason, population of source domain does not contain beneficial information, lacks diversity or is stuck in a local optima, the transfer will be very likely to be ineffective. Second, it is obvious that the transferred tree contains redundancy. GP is known to be susceptible to redundancy and simple transfer methods will transfer redundancy too and therefore, effective transfer methods need to address this issue. Finally, the transferred subtree has a small size and therefore, it contains only two non-ERC terminals which indicates that it does not consider other environmental information. This disregard of information is detrimental to performance of tree and most probably, the selection mechanism of GP will discard the transferred tree, making the transfer futile. These issues are not just relevant to this particular tree and can be generalized easily and our examinations of other results confirms this. Another issue is the transfer of code bloats. Again it is obvious that if source domain is afflicted with large and ineffective trees, it is very likely that this negative property will also be transferred with a (sub)tree. For example, a tree *((max(max(max(min(DC * (CTD + 0.89)), RQ), (min(0.52, DC) - (FRT - CFH))), (((CTD - CFR1) + (DEM - DEM)) + CTD)), ((((FUT * CR) + (DC * DEM1)) * ((min(0.52, 0.83) - (SC - CTD)) - min(0.52, 0.83))) + (RQ * (SC - CTD)))) * (max(min(0.52, 0.83), ((CFR1 / (DEM + (SC - CTD))) + (DC + CFH))) - (FRT - CFH)))* shows effectiveness in the source domain, but it is hard to directly extract compact and useful subtrees from it.

V. CONCLUSIONS AND FUTURE WORK

The ultimate goal of this paper was to investigate the applicability of GP transfer learning methods for alleviating the training cost of UCARP when a change in the number of vehicles is inflicted to a solved instance. To fulfill this goal, a number of experiments were performed to cover a vast variety of scenarios. All the considered methods are based on transfer of GP (sub)trees from source to target domain, and in all cases, they endeavoured to create a better initial population for GP in target domain. Although these methods has produced a few results that are better than the case of no transfer, in most cases, none of the algorithms was able to achieve better results. However, a good effect of almost all algorithms was that they created better initial population

TABLE X
TEST RESULTS OF THE COMPARED ALGORITHMS - DATASET VAL9C

Scenario	From 5 to 4 vehicles			From 5 to 6 vehicles		
Algorithm	Best	Mean(Stdev)	PVal	Best	Mean(Stdev)	PVal
No Transfer	379.31	390.71(10.32)	–	357.76	364.87(8.59)	–
BestGen1	380.48	390.35(4.62)	0.57	351.11	363.83(10.60)	0.64
BestGen2	381.95	389.22(5.14)	0.76	353.51	363.24(5.50)	0.82
GTLKnow	379.76	389.45(8.04)	0.65	352.352	363.01(6.53)	0.64
TLGPCriptor	380.68	390.47(5.46)	0.57	351.41	364.17(8.06)	0.86
FreqSub-root	382.98	391.55(5.54)	0.34	352.05	362.62(5.78)	0.28
FreqSub-all	376.60	391.06(5.81)	0.33	351.51	363.11(5.91)	0.557
FreqSub-sub	377.46	388.59(4.56)	0.38	352.66	363.53(5.56)	0.49
Fulltree25	381.64	390.99(7.75)	0.61	353.25	363.71(6.51)	0.67
Fulltree50	380.99	390.60(4.84)	0.41	353.40	364.69(9.80)	0.94
Fulltree75	383.41	391.35(4.79)	0.32	354.25	365.55(6.96)	0.59
Fulltree100	381.06	388.90(6.35)	0.37	352.61	365.83(9.50)	0.28
Subtree25	375.67	389.03(7.18)	0.76	354.47	363.65(4.19)	0.84
Subtree50	383.76	392.99(8.20)	0.08	352.32	364.18(7.14)	0.70
Subtree75	373.36	389.62(6.35)	0.97	352.86	365.01(5.55)	0.37
Subtree100	379.85	389.50(4.80)	0.73	355.32	364.21(5.62)	0.87

TABLE XI
TEST RESULTS OF THE COMPARED ALGORITHMS - DATASET VAL9D

Scenario	From 10 to 9 vehicles			From 10 to 11 vehicles		
Algorithm	Best	Mean(Stdev)	PVal	Best	Mean(Stdev)	PVal
No Transfer	465.56	483.30(15.14)	–	463.52	478.11(8.81)	–
BestGen1	458.75	480.39(12.54)	0.53	456.79	476.69(10.00)	0.44
BestGen2	463.94	481.58(13.18)	0.92	460.47	479.19(12.53)	0.73
GTLKnow	462.44	483.62(14.00)	0.84	460.57	476.15(8.52)	0.38
TLGPCriptor	464.38	484.03(14.44)	0.34	458.68	476.08(9.30)	0.49
FreqSub-root	463.28	483.40(14.07)	0.73	457.81	477.10(9.14)	0.79
FreqSub-all	462.78	485.48(12.44)	0.45	468.58	479.33(7.15)	0.51
FreqSub-sub	466.08	480.28(10.09)	0.42	465.90	478.08(7.17)	0.70
Fulltree25	461.71	488.36(25.28)	0.50	457.52	477.22(10.45)	0.79
Fulltree50	461.22	481.72(12.81)	0.90	457.79	478.85(10.62)	0.67
Fulltree75	466.17	483.72(11.71)	0.46	458.69	476.94(10.90)	0.87
Fulltree100	462.93	485.17(15.32)	0.45	457.69	477.66(10.16)	0.82
Subtree25	464.02	484.00(14.62)	0.73	465.94	480.31(9.93)	0.33
Subtree50	466.92	483.14(11.59)	0.74	466.56	478.15(8.26)	0.76
Subtree75	465.35	485.55(14.28)	0.47	463.35	481.12(12.25)	0.39
Subtree100	460.52	480.40(12.74)	0.42	462.92	480.50(12.07)	0.32

for GP and consequently, helped GP reach the comparable results in the target domain much faster. Furthermore, the experiments allowed us to identify pitfalls and shortcomings of GP (sub)tree transfer methods that impact their effectiveness. To be specific, we noticed that abundance of redundancy and introns, code bloat and convergence to local optima in source domain are likely to transfer to target domain and lead to negative transfer.

This paper provides a useful benchmark for investigating scalability and transferability of GP transfer methods for solving UCARP that reveals interesting gaps that needs to be addressed in future works. In future works, we will address the shortcomings raised in this work to address the dynamics of UCARP and provide more effective transfer learning methods such as feature weighting and contribution.

REFERENCES

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

[2] S. Wøhlk, *A Decade of Capacitated Arc Routing*. Boston, MA: Springer US, 2008, pp. 29–48.

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

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

[5] L. Muyldermans and G. Pang, *Chapter 10: Variants of the Capacitated Arc Routing Problem*, pp. 223–253.

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

[7] Y. Liu, Y. Mei, M. Zhang, and Z. Zhang, "Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '17. New York, NY, USA: ACM, 2017, pp. 290–297.

[8] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct 2010.

[9] 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, Oct 2015.

[10] R. J. Marshall, "Adapting a hyper-heuristic to respond to scalability issues in combinatorial optimisation," 2015.

[11] J. Wang, K. Tang, and X. Yao, "A memetic algorithm for uncertain capacitated arc routing problems," in *2013 IEEE Workshop on Memetic Computing (MC)*, April 2013, pp. 72–79.

[12] 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, Feb 2016.

[13] Y. Mei and M. Zhang, "Genetic programming hyper-heuristic for multi-vehicle uncertain capacitated arc routing problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '18. New York, NY, USA: ACM, 2018, pp. 141–142.

[14] J. MacLachlan, Y. Mei, J. Branke, and M. Zhang, "An improved genetic programming hyper-heuristic for the uncertain capacitated arc routing problem," in *AI 2018: Advances in Artificial Intelligence*, T. Mitrovic, B. Xue, and X. Li, Eds. Cham: Springer International Publishing, 2018, pp. 432–444.

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

[16] F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with multi-tree representation for dynamic flexible job shop scheduling," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 472–484.

[17] B. Al-Helali, Q. Chen, B. Xue, and M. Zhang, "A hybrid gp-knn imputation for symbolic regression with missing values," in *AI 2018: Advances in Artificial Intelligence*, T. Mitrovic, B. Xue, and X. Li, Eds. Cham: Springer International Publishing, 2018, pp. 345–357.

[18] S. Azari, M. Zhang, B. Xue, and L. Peng, "Genetic programming for preprocessing tandem mass spectra to improve the reliability of peptide identification," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–8.

[19] J. Lu, V. Behbood, P. Hao, H. Zuo, S. Xue, and G. Zhang, "Transfer learning using computational intelligence: A survey," *Knowledge-Based Systems*, vol. 80, pp. 14 – 23, 2015, 25th anniversary of Knowledge-Based Systems.

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

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

[22] 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 *2017 IEEE Congress on Evolutionary Computation (CEC)*, June 2017, pp. 1287–1294.

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

[24] M. Iqbal, M. Zhang, and B. Xue, "Improving classification on images by extracting and transferring knowledge in genetic programming," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, July 2016, pp. 3582–3589.

[25] W. Fu, B. Xue, M. Zhang, and X. Gao, "Transductive transfer learning in genetic programming for document classification," in *Simulated Evolution and Learning*, Y. Shi, K. C. Tan, M. Zhang, K. Tang, X. Li, Q. Zhang, Y. Tan, M. Middendorf, and Y. Jin, Eds. Cham: Springer International Publishing, 2017, pp. 556–568.