Genetic Programming Hyper-Heuristics with Probabilistic Prototype Tree Knowledge Transfer for Uncertain Capacitated Arc Routing Problems

Mazhar Ansari Ardeh*, Yi Mei† and Mengjie Zhang‡
School of Engineering and Computer Science, Victoria University of Wellington
PO Box 600, Wellington, New zealand
Email: *mazhar.ansariardeh@ecs.vuw.ac.nz, †yi.mei@ecs.vuw.ac.nz, ‡mengjie.zhang@ecs.vuw.ac.nz

***Abstract*—The Uncertain Capacitated Arc Routing Problem (UCARP) is an important combinatorial optimisation problem with extensive real-world applications. Genetic Programming (GP) has shown effectiveness in automatically evolving routing policies to handle the uncertain environment in UCARP. However, whenever a UCARP scenario changes, e.g. when a new vehicle is bought, the previously trained routing policy may no longer work effectively, and one has to retrain a new policy. Retraining a new policy from scratch can be time-consuming but the transfer of knowledge gained from solving the previous similar scenarios may help improve the efficiency of the retraining process. In this paper, we propose a novel transfer learning method by learning the probability distribution of good solutions from source domains and modelling it as a probabilistic prototype tree. We demonstrate that this approach is capable of capturing more information about the source domain compared to transfer learning based on (sub-)tree transfers and even create good trees that are not seen in source domains. Our experimental results showed that our method made the retraining process more efficient and one can obtain an initial state for solving difficult problems that is significantly better than existing methods. The final performance of all algorithms, were comparable, implying that there was no negative transfer.**

***Index Terms*—Genetic Programming, Uncertain Capacitated Arc Routing Problems, Transfer Learning**

I. Introduction

Capacitated Arc Routing Problem (CARP) is a well-known combinatorial optimisation problem. CARP is based on a connected and undirected graph $G(V, E)$, where each node $v \in V$ is an intersection, and each edge $e \in E$ is a street segment. Each edge has a non-negative demand to be served by a set of vehicles with limited capacity. The vehicles are located at the depot (a special node in the graph). Serving and traversing through an edge incurs a serving or deadheading cost. The goal of CARP is to find a set of minimum-cost routes for each vehicle to serve the demand of edges subject to a set of predefined constraints [24]. Street watering, removal of snow [5] and waste collection [1] are real-world applications of CARP.

The Uncertain Capacited Arc Routing Problem (UCARP) is a realistic extension of CARP that addresses the shortcoming of CARP related to its static nature[15]. UCARP is hard to solve with the traditional methods that solve CARP, and Genetic Programming Hyper-Heuristic (GPHH) has achieved considerable success in real-time adjustment of the routes in uncertain environments [13], [15]. However, the existing GPHH approaches evolve routing policies for different scenarios separately without considering the relation and similarity between them. In the real world, different problem scenarios may be correlated with each other. For example, when a company buys a new vehicle, or an existing vehicle breaks down, the problem scenario will be slightly changed (i.e. the number of vehicles increases or decreases by 1). In this case, the current routing policy may no longer work well and a new routing policy will be needed. However, retraining a new routing policy can be very time consuming. On the other hand, due to the similarity between the scenarios, it is reasonable to expect that the knowledge gained from evolving the current routing policy can help improve the efficiency and effectiveness of the retraining process. Therefore, in this paper, we will investigate GPHH with knowledge transfer for evolving routing policies of UCARP more efficiently.

In the context of knowledge transfer in GPHH, a typical strategy is to transfer trees and subtrees evolved in a source domain. While the methods that are based on this strategy have shown good performance in some domains, this general approach has a fundamental shortcoming. Subtrees that are extracted from a source domain might not have the comprehensive information about the problem search space of the source domain problem but just capture partial manifestation of that information. Despite the fact that full trees have more genetic materials than subtrees, it can still be argued that good full trees are just good points in the problem search space of the source domain and therefore, direct transfer of them might not transfer enough helpful knowledge to target domains. Ansari Ardeh et al. [4] recently started addressing this issue by defining GP feature importance as the transferable knowledge but their method was not as effective as subtree-based transfer learning.

In this paper, we intend to capture the underlying distribution of good individuals of the source domain as transferable knowledge in a way that individuals that are sampled from it inherit their good properties. Furthermore, by sampling new individuals from the learned distribution, we aim to create new individuals that are not seen in source domains but are able to perform well in target domains. Ultimately, we aim to enhance GP to solve new but related problems

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

more efficiently. Consequently, this paper fulfils the following research objectives:

- Develop a method to understand the hidden knowledge in the population of an existing knowledge source that reaches beyond the simple transfer of (sub-)trees and utilise it for the task of transfer learning.
- For the knowledge transfer method, design proper mechanisms regarding (1) knowledge representation; (2) knowledge extraction; and (3) knowledge usage in the target domain.
- Examine the effectiveness of the proposed knowledge transfer method and analyse their behaviour.

This paper is organised as follows. A detailed review of the concepts discussed in this paper is given in Section II. Our transfer learning method based on probabilistic prototype tree is presented in Section III. Section IV presents the experiments we conducted and the results we obtained are discussed in Section V. We conclude the paper in Section VI.

II. Background

A. UCARP

A UCARP is defined as a graph $G(V, E)$ in which the set of nodes is V and E designates the set of edges. Vehicles in UCARP have a capacity Q and are located at the depot $v_0 \in V$. Each edge $e \in E$ has two cost values. The first one is the deadheading cost $dc(e)$ which is the cost of traversing the edge without serving it which is a positive random variable. The second is the serving cost $sc(e)$ which is positive and deterministic. Demand of an edge, $d(e)$, is a non-negative random variable that quantifies the amount of work that needs to be done for the task. An edge e with $d(e) > 0$ is called a *task* and needs to be served. Finding a solution for UCARP is the act of finding vehicle routes so that the total cost (the sum of all the serving and deadheading costs) is minimised. Feasible UCARP solutions need to comply with the following constraints:

- Each vehicle departs from the depot and returns to the depot after completing its services. The vehicles can go back to the depot to refill their capacity in the middle of a service, and then continue the interrupted service.
- The total demand served by a vehicle between two refills does not exceed its capacity.
- Each task is served exactly once.

In general, proactive and reactive methods are the two main categories of approaches for solving UCARP. In the proactive approach, a robust solution is considered and optimised for UCARP. The reactive method makes use of Genetic Programming Hyper Heuristics (GPHH) to search for routing policies that can generate solutions. The works described in [15], [21], [22] are good examples of techniques in the category of proactive methods.

A shortcoming of the proactive methods is that they do not possess the flexibility that is required for real-world and real-time applications [13]. Liu et. al [15] overcame this issue with a GPHH approach to evolve routing policies to model the decision making procedure of a vehicle when selecting the next task to serve and applied it to a set of benchmark UCARP instances. Liu et. al [16] extended that work for the case of multiple vehicles. Recently, Wang et al. proposed a GPHH approach to training an ensemble of routing policies for UCARP [23].

B. GPHH for UCARP

Popularised by Koza [12], GP is an extension of Genetic Algorithm (GA) that applies genetic operators to evolve computer programs and has found extensive applications in different research areas [2]. A Genetic Programming Hyper-Heuristic method searches the space of heuristic functions with a standard GP.

Considering the fact that a routing policy is just a priority function, it is natural to conclude that routing policies can be evolved with GPHH. GPHH for evolving routing policies has the following four principal states:

1) Initialise a population of GP trees representing routing policies.
2) Evaluate the fitness of each GP tree over a set of training instances.
3) Select a set of individuals for evolution.
4) Apply the genetic operators of evaluation, evolution(crossover, mutation) and reproduction to evolve a new population from the current one.
5) Until a stopping criteria is met, repeat the steps in 2 through 4.

Each GP tree is a routing policy that prioritises the tasks that a vehicle can serve. After a vehicle finishes serving a task, it considers all unserved tasks and a few filtering methods are applied to them to remove tasks that the vehicle cannot serve. Based on the state of the problem, the routing policy assigns a priority to each task and the vehicle serves the task with the best priority. Because of the dynamic nature of UCARP, two failures may occur when a vehicle serves a task. Since task demands are stochastic, actual demand of a task, may be greater than the capacity of the vehicle which leads to *route failure*. Furthermore, it is possible that vehicles arrive at a task e that is inaccessible because $dc(e) = \infty$ which leads to an *edge failure*. This is why traditional optimisation methods, that try to discover exact solutions, fail in case of UCARP. When a route failure happens, the vehicle must go back to the depot and replenish its capacity and then serve the failed task. Edge failures can be overcome with a detour around the failed task. The fitness of a policy is the average of total cost of the routes that it generates on predefined training instances.

The GPHH approach to solving UCARP does not need any preplanned solutions or extensive domain expertise and to the best of our knowledge, it outperforms other methods for solving UCARP [13], [16]. GPHH requires model training which is not trivial. On the other hand, there exists the case in which problem state changes over time in real world applications. As an example, transportation systems frequently change their fleet size with recruiting new vehicles into their system or retiring old ones. When problem state changes,

existing routing policies are applicable but it was shown that their performance degrades greatly [16] and as a result, new policies will be required. A simple remedy is to train new routing policies but, as was pointed out earlier, the application of GPHH for solving UCARP is typically time-consuming and it is better to have methods that can use the previous solutions and speed up the training phase of new routing policies. This is the exact case in which transfer learning is most applicable [14], [19].

C. GP Transfer Learning

Transfer learning can be described as "the ability of a system to recognise and apply knowledge and skills learned in previous tasks to novel tasks" [18] to improve the learning process for the new tasks. When the primary learning method is GP, transfer learning can be used to (1) create better initial population, (2) improve its convergence speed and (3) help it achieve better final solutions [18]. Transfer learning is a rather new concept, especially when compared with the main body of machine learning literature but nevertheless, a plethora of research has been conducted on it [14], [19]. Because the main learning method in this work is GP, we consider the transfer learning methods that were designed exclusively for GP.

The work of Kocer et. al [11] is one of the earliest transfer learning solutions for an evolutionary algorithm. In their paper, to transfer knowledge, the individuals with the best and median fitness values are selected in each generation of GA when solving source domains problem and saved into an "individual pool". On target domains, 30% of the initial population are selected randomly from the pool [11]. Building on work of Kocer, Dinh et. al [6] also proposed three transfer learning methods based on (sub)tree transfer for GP. In the *BestGen* method, which bears some similarities to the work of Kocer [11], k best individuals of each generation of GP on a source domain are saved into a pool and on target domain, they are used as initial population. In the second algorithm, called *FullTree* , $k\%$ of the best individuals in the final population of source domain are selected to initialise the GP population on target domain. In their third method, *SubTree*, one of root sub-trees of each individual in the final population of GP on source domain is selected into a pool to create the initial GP population on target domain. Haslam et al [9] improved the work of Dinh et al further with considering performance of transferred items on target domain too.

The idea of knowledge transfer through sub-tree transfer was further investigated by Iqbal et. al [10]. In their work, the authors dubbed the root sub-trees of GP individuals as code fragments and extracted them from good individuals of the final population of GP on a source domain and stored them to be used again on a target domain. In a target domain, during the initialisation phase and with a probability of 50%, root children are selected from the stored code fragments or generated from terminals and functions. This sub-tree creation procedure is utilised during the GP run's mutation too.

Rather than transferring exact (sub-)trees, O'Neill et. al [18] took a more interesting approach. In the *Common SubTree in Related Problems* method, subtrees that are commonly repeated in the population of source domain are identified. These subtrees are converted to functions and added to the function set of GP on target domain. Ansari Ardeh et al. [3] relaxed some of the conditions of this work and only considered the frequent exact subtrees and used them to initialise GP on target domain. In a more recent work, the same authors proposed selection of subtrees based on cumulative contributions that they make to their individuals which also favours more frequent subtrees implicitly [2].

As it can be observed from this review, a number of existing GP transfer learning methods pursued the first goal of transfer learning, i.e. creating a better initial population, that was mentioned early in this section. In this work, we also focus on this goal but intend to take a step further and learn the probability distribution of GP trees that have shown good performance in the source domain and create a probabilistic prototype tree based on it to be used as the transferable knowledge. As it was pointed out earlier, each GP tree represents a single point in the search space of a source problem and hence, it can contain limited knowledge about that search space and GP subtrees are even more limited in this regard. This issue can be alleviated to some degree by transferring more trees but this is also limited to GP population size. However, by learning the probability distribution of good trees, new individuals can be created that have acquired the properties of good individuals of the source domain.

III. GPHH with Probabilistic Prototype Tree Transfer

In the context of GPHH, we make a basic assumption that abundant knowledge about a source domain is contained in the individuals evaluated during the GPHH process in the source domain. Therefore, a *generic* GPHH with knowledge transfer contains the following steps:

- Select a knowledge representation for transferable knowledge.
- Extract knowledge from the individuals evaluated for the source domain.
- Modify the GPHH in the target domain to use the extracted knowledge.
- Run GPHH to evolve individuals for the target domain.

This GPHH with knowledge transfer process consists of three key design decisions: (1) knowledge representation; (2) knowledge extraction; and (3) knowledge usage in the target domain. In this paper, we consider a novel GPHH with knowledge transfer based on the concept of probabilistic prototype tree (PPT) [20] for knowledge representation and its detailed description, including knowledge representation, extraction and usage are given in this section. The high-level pseudocode of modified GPHH, referred to as *GPHH-PPT* throughout this paper, is given in Algorithm 1. In our approach, first a source domain is solved as usual with GPHH and when a problem change is detected, *GPHH-PPT* learns the probability distribution of good individuals of the final population from source domain into a PPT as the means

Algorithm 1: Pseudocode for GPHH-PPT

Input : Training instances $\wp$, Final population of source domain $\Im$, Percent of initial population created from transferred knowledge k ($\in [0, 1]$)

Output: The best evolved function ind^*

set $ind^* \leftarrow null$, $gen \leftarrow 0$

`// Knowledge Extraction`

Use a tournament of size t to select a subset of $\Im$ to learn from

Extract knowledge as PPT from the selected individuals by calculating the probability of each node

`// Knowledge Usage`

while $N_{ind} < k \times Popsize$ **do**

  Initialisation: Sample new individuals from the PPT by sampling GP functions/terminals based on the learned probability distribution nodes

end

`// GPHH`

while $N_{ind} < (1-k) \times Popsize$ **do**

  Initialisation: Randomly initialise an individual

end

while $gen < maxGen$ **do**

  Evaluate each individual on $\wp$ and update ind^*

  Apply selection

  Apply evolution

  $gen \leftarrow gen + 1$

end

return ind^*

for knowledge representation and then, on target domain, the learned PPT is used to create a good initial GP population, after which standard GP operators are applied to the population iteratively until a stopping criteria is met.

A. GPHH with Probabilistic Prototype Tree Transfer

1) Knowledge Representation: The core idea of this paper is that, having a source domain, if we can learn the probability distribution of good individuals of the source domain, we can use this probability distribution to sample individuals that can have the properties of good individuals of the source domain. In our definition, the probability distribution of the good individuals should be able to know the probability that a GP terminal or function may occur at a given node of GP programs.

Therefore, if we define an arbitrary indexing I on the nodes of GP trees, we can define the random variable X_i to be the value that node $i \in I$ can have from the combined set $T \cup F$, where F and T are the GP function and terminal sets respectively. We denote the probability of X_i having the value $r \in F \cup T$ as $P(X_i = r) = p_{i,r}$. It is interesting to note that this definition of the random variable X_i has the categorical probability distribution, considering the fact that it can have $\rho = |T \cup F|$ exclusive outcomes and the probability of each outcome does not change over time in a standard GP [17]. Furthermore, the random variable $Y_{i,r}$, defined to be the number of times that an $r \in T \cup F$ has appeared in node $i \in I$ in a population of GP individuals, has the multinomial probability distribution [8].

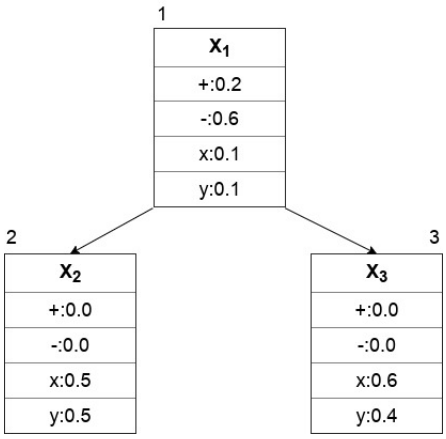

Fig. 1 *An example of a PPT*

With these considerations, we define the probabilistic prototype tree to be a complete binary (or q-ary with q being the maximum arity of GP functions) tree in which each node $i \in I$ holds the probability vector $P_i = (p_{i,r})_{r \in T \cup F}$, that is, P_i is probability vector in which each of its components denotes the probability of selecting an item r from either the terminal or the function set and represents the underlying categorical distribution of the item r appearing at i. Figure 1 shows a simple PPT of depth 2 for $T = \{x, y\}$ and $F = \{+, -\}$ as an example.

2) Knowledge Extraction: With the definition of PPT given earlier, it is fairly straightforward to learn the PPT as the probability distribution of good GP individuals from an existing population. As it was mentioned earlier, the random variable $Y_{i,r}$ which is the number of times that the item r has appeared in location i has the multinomial distribution. Considering the relationship between X_i and Y_i, we can find the approximation of the parameter $p_{i,r}$ by performing maximum likelihood estimation over the existing population [7]. Consequently it is fairly easy to prove that $p_{i,r} \approx \overline{p_{i,r}}$ where $\overline{p_{i,r}}$ is the average number of times that the item r has appeared in location i in the population.

It is possible to consider the whole population set for learning the PPT but it has been shown that code bloat and the presence of duplicates is an important factor that can deter effective transfer learning [2], [3]. In order to address this, we propose a tournament selection to be performed on the population so that the algorithm can have a control on the quality of the individuals that are selected for learning the PPT. In this regard, the selection mechanism samples s number of individuals from the population by performing tournament selection with a tournament of size t.

It should be noted that the structure of the prototype tree has some similarities with the PIPE tree that is presented in [20].

TABLE I: The source and target domains in the experiments.

Small Dataset	gdb1		gdb1	gdb2	gdb2	gdb21	gdb21	
#Tasks	22		22	26	26	33	33	
Source	5		5	6	6	6	6	
Target	4		6	5	7	5	7	
Medium Dataset	gdb8		gdb8	gdb9	gdb9	gdb23	gdb23	
#Tasks	46		46	51	51	55	55	
Source #Vehicles	10		10	10	10	10	10	
Target #Vehicles	9		11	9	11	9	11	
Large Dataset	val9C	val9C	val9D	val9D	val10C	val10C	val10D	val10D
#Tasks	92	92	92	92	97	97	97	97
Source #Vehicles	5	5	10	10	5	5	10	10
Target #Vehicles	4	6	9	11	4	6	9	11

However, our work does not consider the constants $R_{d,w}$ in the PPT that this work uses and also defines a more rigorous modelling of the underlying probability distribution of the GP individuals that is absent in that work. Furthermore, that paper takes a completely different learning approach from the one that we propose. Finally, that work is not in the context of GP transfer learning.

3) Knowledge Usage: When a PPT is learned from the source domain, it can be used to sample new GP individuals to help GP on the target domain. To create a new individual from a transferred PPT, a GP item $r \in T \cup F$ is selected for node $i \in I$ with the probability $p_{i,r}$. In this paper, a roulette wheel mechanism is used to make the selection. For example, consider the PPT in Figure 1 for creating new individuals. Instead of randomly selecting a GP function/terminal for node 1 (root), $+$, $-$, x or y will be selected with respective probabilities of 0.2, 0.6, 0.1, 0.1 and nodes number 2 and 3 will be created similarly. The nodes are created in a prefix order, starting from the root node. If the selected item is of type ephemeral random constant (ERC), then a value in the range $[0, 1]$ is generated randomly and assigned to it. In this paper, the PPT is used to create k percent of the initial population of GP on the target domain to give it an initial performance boost.

IV. Experiment Design

To investigate the effectiveness of the proposed GPHH with knowledge transfer for UCARP, we design a number of source and target domain settings. Specifically, we define the source and target domains based on the same UCARP instance with the same graph topology and distribution of the random variables but they differ from each other in terms of the number of vehicles, reflecting the situations where a new vehicle is bought or an existing vehicle breaks down. For the UCARP instances, we select several small-sized, medium-sized and large-sized instances (Table I). This way, we can have a comprehensive study on the performance of knowledge transfer under different scenarios.

In both source and target domains, 500 unseen test samples are randomly generated. For GP training, 5 training samples are used for fitness evaluation, which are rotated at each generation to reduce overfitting. Specifically, we design three stages for experiments: source domain is considered as n vehicles in a UCARP instance and GP is utilised to train routing policies for solving the problem and is evaluated on test dataset. We consider a target domain by changing number of vehicles to $n-1$ or $n+1$. The final stage of our experiment is to extract knowledge from the source domain and try to alleviate the training of routing policies for target domain.

We examine the *Fulltree*, *Subtree* and *FreqSub-root* configurations. These methods are used to select 50% of the final population of source domain to be transferred to initialise 50% of the population on the target domain. The reason that *FullTree* and *FreqSub* are selected for comparison is that they both transfer whole GP trees to target domain. Since our novel method also creates whole trees on the target domain, we would like to compare their performance with the performance of the trees that are created from the learned PPT. *SubTree* method is also selected because this algorithm transfers a subtree of a GP tree and therefore, it transfer less amount of genetic material and we would like to see how our method can perform against this limited form of transfer learning.

For *GPHH-PPT*, we experiment on learning the PPT from the final population of the source domain with different configurations and report the best configuration. To have the same learning pool as existing methods, we also experimented with disabling tournament selection of our algorithm. To keep it consistent with the existing transfer learning literature, $k = 50\%$ of the population of target domains are initialised with the algorithm. Table II presents the GP settings and terminals used in experiments which is based on the work in [16]. GP function set is $\{+, -, \times, /, \min, \max\}$ in which the division operator $/$ is protected and returns 1 when divided by 0. A summary of these methods is given in Table III. All algorithms are run 30 times independently. To evaluate different aspects of our algorithm, we conduct a set of experiments with different settings and present the settings that produce the best results.

TABLE II: GP settings.

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

V. Results and Discussions

For our experiment settings, we report that in terms of the final performance after 50 generations, there was no

TABLE III: The examined algorithms

Algorithm	Subtree selection mechanism
FreqSub-Root	Select most frequent subtrees in the final population [3]
Fulltree-k	Select k of best of individuals of the final population [6]
Subtree-k	Select a random subtree of each individual of final population [6]
GPHH-PPT-(s,t)	Learns the underlying distribution of the population with a sample size s and tournament size t, from the final population ($k = 50\%$ in all experiments)

significant difference between all the algorithms and the case of GPHH without transfer learning, verified by the Wilcoxon test of 30 independent runs. There are some cases where some knowledge transfer methods are significantly better. However, we cannot find a general pattern where a GPHH with knowledge transfer consistently outperformed the one without transfer. Due to space limit, we choose not to give the results on the final performance here, but focus more on the initial performance on the target domain. Because the focus of all algorithms, including ours, was on improving the quality of the initial state, it can be inferred that any improvement to the initial population of GPHH did not persist until the final stage of the algorithm. We attribute this outcome to the dynamic nature of UCARP for which good individuals for one state of environment could perform poorly for some other state. Nevertheless, this result has an important implication in the context of transfer learning which is the indication that the changes did not incur any negative transfer which is an important concern of transfer learning algorithms.

We experimented with different values for the s and t parameters of *GPHH-PPT*. Also, we experimented on the case for which the tournament selection part of the algorithm was disabled so that *GPHH-PPT* had the exact same learning pool of the methods *FullTree-50* and *SubTree-50*. In our experiments, the best initial performance was obtained for

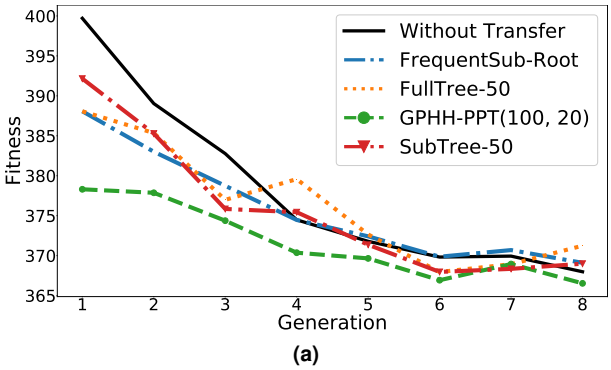

Fig. 2 *Convergence curves of the compared algorithms on small-sized dataset **gdb1** from 5 to 4 vehicles (8 generations out of 50 shown).*

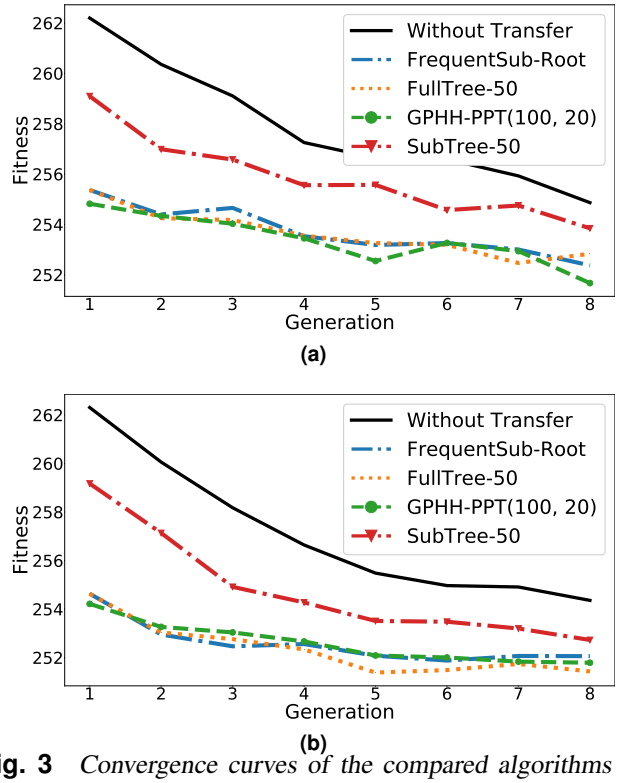

Fig. 3 *Convergence curves of the compared algorithms on the mediums-sized dataset **gdb23** from 10 to (a) 9 and (b) 11 vehicles (8 generations out of 50 shown).*

$s = 100, t = 20$. It should be noted this setting gives the algorithm a smaller learning pool than *FullTree-50* and *SubTree-50* but to be thorough, we also tested *FullTree-10* and *SubTree-10* with the same pool and observed the performance of these algorithms was significantly worse that *FullTree-50* and *SubTree-50*. Therefore, we omitted *FullTree-10* and *SubTree-10* from our discussions due to space limit.

In terms of the quality of the initial performance, the results are more interesting. Figures 2 and 3 present early performance of the compared algorithms on 2 different datasets as the representative of all the experiments performed. The examination of the convergence curve of the algorithms suggested that in almost all experiments, GPHH-PPT was able to create initial populations that were better than vanilla GPHH. Furthermore, the curves suggest that GPHH-PPT has been able to surpass the *SubTree* method too in almost all experiments. Overall, we can see that GPHH-PPT is able to create initial populations than that are either better the *FullTree* and *FrequentSub* methods or are as good as the ones created by them.

To confirm these hypotheses, we conducted the Wilcoxon test of the 30 runs to compare the quality of individuals in first generation of each algorithm. For the sake of convenience and brevity, the summary of these tests are presented in Table IV by summing up the number of times GPHH-PPT was significantly better than ("wins"), same as ("draws") and worse

TABLE IV: Summary of the quality of initial GP population of 30 runs, grouped by dataset sizes (small, medium, large). The number of "wins", draws" and "losses" of GPHH-PPT against the compared algorithms.

Baseline	No Transfer	SubTree-50	FullTree-50	FrequentSub
Small	6-0-0	6-0-0	0-6-0	0-6-0
Medium	6-0-0	6-0-0	4-2-0	4-2-0
Large	8-0-0	8-0-0	8-0-0	8-0-0

than ("losses") the baseline algorithm.

Inspecting Table IV, our first conjecture is confirmed that *GPHH-PPT* is always better than vanilla GP, i.e., it is better than the random creation of initial populations. This fact is the first achievement of our method. The table also confirms our second hypothesis that *GPHH-PPT* can outperform the *SubTree* method. This result was highly expected. The *SubTree* method just transfers a random part of GP trees and therefore, it just transfers random and partial manifestations of the knowledge from source domains and cannot be expected to have a complete view of the knowledge. This indicates that *GPHH-PPT* has achieved more comprehensive understanding of the hidden knowledge.

Comparison between *GPHH-PPT* and *FullTree* reveals even more interesting insights. For small and medium-sized datasets, the performance of *GPHH-PPT* is comparable to or better than the performance of *FullTree*. However, the most intriguing result comes for the large tasks for which *GPP-PPT* is the clear and unanimous winner over all algorithms, including *FullTree*. This is an important observation that, we believe, stems from the fundamental difference between *GPHH-PPT* and *FullTree* (or any method based on transfer of genetic materials). Earlier in this paper, we hypothesised that we intend to learn the underlying knowledge hidden in a GP population in contrast to simple transfer of genetic materials with the hope of capturing some of the knowledge. The superior performance of *GPHH-PPT* on large and difficult datasets indicates that when the problem becomes difficult, having a deeper knowledge is more effective and confirms the benefit of this approach. Knowing that *FullTree* transfers the best individuals, the result that the performance of *GPHH-PPT* is comparable to that of *FullTree* for small and medium-sized tasks implies that *GPHH-PPT* has achieved the level of understanding of the underlying knowledge about the source domain that it can mimic the performance of the best individuals.

As it was mentioned in Section II-C, the first goal of transfer learning is to create a better initial state than randomness so that the better initial state can reduce the training effort. Consequently, we calculated the reduction in training time of algorithms in the target domain by finding the first generation in which the test performance of the algorithms is statistically similar to the last generation of vanilla GP for at least 3 consecutive generations and averaged it over all experiments. The improvements are presented in Table V and as is evident, *GPHH-PPT* can reduce the training effort of GP in target domains significantly and it is even better than other

TABLE V: Improvements in GP training time averaged over all experiment

Algorithm	SubTree-50	FullTree-50	FrequentSub	GPHH-PPT
Improvement	50.22%	53.33%	50.23%	59.44%

algorithms. It should be noted that the improvement are seen in all experiments regardless of their size.

1) PPT Analysis: The main idea in PPT-based transfer learning is to learn the probability distribution of GP functions and features in a source domain to be used in target domain. Figure 4 presents a PPT that is learned from a source domain (*gdb1, 5 vehicles*) in the smaller frame and the magnified view of its top 2 level of nodes. The numbers inside nodes indicate the probability of selecting corresponding GP function/terminal at that location and the labels beside each node show the indexing order defined over this PPT. The probability of missing items is zero In this work we define the underlying knowledge in the source domain as being the probability distribution of good GP individuals, granulated at the node level and Figure 4 basically depicts this. For example, the figure shows that in the source domain, the node at $i = 3$ has a categorical distribution with parameters $p_{3,max} = 0.06, p_{3,*} = 0.54, p_{3,-} = 0.36, p_{3,min} = 0.04$.

It is interesting to note that PPTs can capture the relation between GP items with respect to their position in trees. For example, based on Figure 4, we can see that $\{max, *\}$ are important at node 2 and $\{max, *, -min\}$ are important at node 4 and therefore, any combination of these sets can be considered important, specifically between $\{max\}$, from node 2 and $\{*, -\}$ from node 4. It is possible to capture this type of relation with methods like *FullTree* but when tree size and the number of such relations is large, the number of possible combinations of GP items increase exponentially and the number of needed transferable trees to capture all these relations become intractably large. Consequently, our approach allows for the creation of trees that may not have been seen before. For example, the tree *(((((0.87 max CFH) max (SC - CTD)) max (FUT * 0.13)) min (((SC - CTD) min (FUT * (FUT * SC))) - CTD)) max ((((0.66 max CFH) max (SC - CTD)) max (((RQ - CTD) max CR) min (SC * SC))) * (((SC - CTD) min (FUT * (FUT * SC))) max CFH))) max ((CFD max (((SC + CFH) - (SC - CTD)) max ((SC min (FRT * DC)) max (CTT1 * CTD)))) * (RQ - DC)))* is one of the trees that is sampled from this PPT. With a fitness value of 372.74, this was the best individual in the first population in target domain (*gdb1, 6 vehicles*). This tree, or similar trees, was not present in the source domain and therefore, methods like *FullTree* could not transfer it. Interestingly, the best individual that *FullTree* transferred in this experiment had a fitness value of 380.94. Another interesting feature to notice is that the PPT can prune a lot of possible values as many nodes contain a single value.

VI. Conclusions and Future Work

In this paper, we proposed a novel GPHH approach with knowledge transfer to improve the efficiency of the retraining

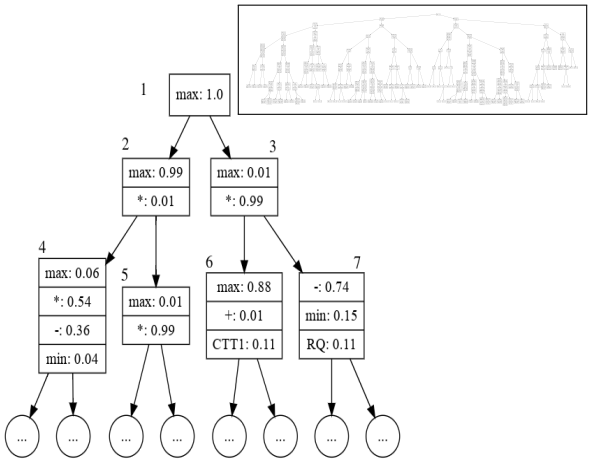

Fig. 4 *A learned PPT*

process for routing policies of UCARP. We proposed a novel transfer learning algorithm based on modelling the transferable knowledge as a probabilistic prototype tree and tested our knowledge transfer algorithm on a range of UCARP source and target domains with differences in the number of vehicles. The experimental results showed that the knowledge transfer can greatly improve the quality of initial states in the target domain, especially for large and difficult problems, and hence, improve the efficiency of GPHH retraining on the target domains by reducing the training time significantly. Also, the experiments showed that the proposed method was better than vanilla GP as well as existing methods. Since the final performance is statistically comparable with vanilla GPHH and other transfer learning methods, we conclude that no negative transfer occurred which demonstrates the potential of learning the probability distribution of good individuals of the source domain and transferring it as probabilistic prototype trees.

In the future, we will explore more effective knowledge transfer methods which will include better knowledge extraction and knowledge usage. For example, we will consider simplifying and transforming the trees in the source domain, to improve the compactness of the transferred probabilistic prototype trees and reduce the noise in them. We will also develop other learning methods to estimate the probability tree and see if they can perform more accurately. Furthermore, we will consider the possibility of using the transferred method during GPHH run to achieve better final performance too.

VII. Acknowledgement

This work was supported in part by the Marsden Fund of New Zealand Government under Contracts VUW1509 and VUW1614.

References

[1] Amponsah, S.K., Salhi, S.: The investigation of a class of capacitated arc routing problems: the collection of garbage in developing countries. Waste Management **24**(7), 711–721 (2004)

[2] Ansari Ardeh, M., Mei, Y., Zhang, M.: A Novel Genetic Programming Algorithm with Knowledge Transfer for Uncertain Capacitated Arc Routing Problem. In: PRICAI 2019: Trends in Artificial Intelligence, pp. 196–200. Springer International Publishing (2019)

[3] Ansari Ardeh, M., Mei, Y., Zhang, M.: Transfer Learning in Genetic Programming Hyper-heuristic for Solving Uncertain Capacitated Arc Routing Problem. In: IEEE Congress on Evolutionary Computation. pp. 49–56 (2019)

[4] Ardeh, M.A., Mei, Y., Zhang, M.: Genetic programming hyper-heuristic with knowledge transfer for uncertain capacitated arc routing problem. In: The Genetic and Evolutionary Computation Conference Companion. pp. 334–335. ACM (2019)

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

[6] Dinh, T.T.H., Chu, T.H., Nguyen, Q.U.: Transfer learning in genetic programming. In: 2015 IEEE Congress on Evolutionary Computation. pp. 1145–1151 (2015)

[7] Eliason, S.R.: Maximum likelihood estimation : logic and practice. Sage (1993)

[8] Forbes, C., Evans, M., Hastings, N., Peacock, B.: Statiscitcal Distributions. John Wiley & Sons (2011)

[9] Haslam, E., Xue, B., Zhang, M.: Further investigation on genetic programming with transfer learning for symbolic regression. Proceeding of IEEE Congress on Evolutionary Computation pp. 3598–3605 (2016)

[10] Iqbal, M., Xue, B., Al-Sahaf, H., Zhang, M.: Cross-Domain Reuse of Extracted Knowledge in Genetic Programming for Image Classification. IEEE Transactions on Evolutionary Computation **21**(4), 569–587 (2017)

[11] Kocer, B., Arslan, A.: Genetic transfer learning. Expert Systems with Applications **37**(10), 6997 – 7002 (2010)

[12] Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)

[13] Liu, Y., Mei, Y., Zhang, M., Zhang, Z.: Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem. In: The Genetic and Evolutionary Computation Conference. pp. 290–297. ACM (2017)

[14] Lu, J., Behbood, V., Hao, P., Zuo, H., Xue, S., Zhang, G.: Transfer learning using computational intelligence: A survey. Knowledge-Based Systems **80**, 14–23 (2015)

[15] Mei, Y., Tang, K., Yao, X.: Capacitated arc routing problem in uncertain environments. In: IEEE Congress on Evolutionary Computation. pp. 1–8 (2010)

[16] Mei, Y., Zhang, M.: Genetic Programming Hyper-heuristic for Multi-vehicle Uncertain Capacitated Arc Routing Problem. In: The Genetic and Evolutionary Computation Conference Companion. pp. 141–142. ACM (2018)

[17] Murphy, K.P.: Machine Learning: A Probabilistic Perspective. The MIT Press (2012)

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

[19] Pan, S.J., Yang, Q.: A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering **22**(10), 1345–1359 (2010)

[20] Sałustowicz, R., Schmidhuber, J.: Probabilistic Incremental Program Evolution: Stochastic search through program space. In: Lecture Notes in Computer Science, pp. 213–220. Springer, Berlin, Heidelberg (1997)

[21] Wang, J., Tang, K., Lozano, J.A., Yao, X.: Estimation of the Distribution Algorithm With a Stochastic Local Search for Uncertain Capacitated Arc Routing Problems. IEEE Transactions on Evolutionary Computation **20**(1), 96–109 (2016)

[22] Wang, J., Tang, K., Yao, X.: A memetic algorithm for uncertain Capacitated Arc Routing Problems. In: 2013 IEEE Workshop on Memetic Computing (MC). pp. 72–79 (2013)

[23] Wang, S., Mei, Y., Zhang, M.: Novel ensemble genetic programming hyper-heuristics for uncertain capacitated arc routing problem. In: The Genetic and Evolutionary Computation Conference. pp. 1093–1101. ACM Press (2019)

[24] Wøhlk, S.: A Decade of Capacitated Arc Routing, pp. 29–48. Springer US (2008)