A Parametric Framework for Genetic Programming with Transfer Learning for Uncertain Capacitated Arc Routing Problem

Mazhar Ansari Ardeh, Yi Mei, and Mengjie Zhang

School of Engineering and Computer Science
Victoria University of Wellington

Abstract. The Uncertain Capacited Arc Routing Problem (UCARP) is an important variant of arc routing problems that is capable of modelling uncertainties of real-world scenarios. Genetic Programming is utilised to evolve routing policies for vehicles to enable them to make real-time decisions and handle environment uncertainties. However, when the properties of a solved problem change, the trained routing policy becomes ineffective and a new routing policy is needed to be trained which is a time-consuming process. Nevertheless, there are similarities between the old and the new problem and by extraction and transfer of some knowledge learned from the old problem, the retraining process can be improved. Transfer learning is an intricate task that entails many aspects to decide about, which can influence the degree by which knowledge transfer can be effective. Consequently, in this paper we propose a parametric framework to formalise these details so that it can facilitate studying different aspects of using transfer learning for handling scenario changes of UCARP. Conducting a large number of experiments, we utilise this framework to analyse different transfer learning mechanisms and demonstrate how it can help with understanding dynamics of knowledge transfer for UCARP.

Keywords: Transfer Learning · Genetic Programming · Uncertain Capacitated Arc Routing Problem

1 Introduction

The Uncertain Capacitated Arc Routing Problem (UCARP), first proposed by Mei et al. [13], is an important optimisation problem which simulates a set of vehicles that serve a set of tasks in an uncertain environment. The goal of solving a UCARP is to find a collection of routes that serve all tasks with minimal total cost while respecting some predefined constraints [13]. The approach of using routing policies is a flexible method that can handle the uncertainties of this problem effectively [1, 13]. A routing policy is a real-valued function that assigns a priority value to unserved tasks based on state of environment. Vehicles consider the priority and choose the task to serve next. Liu et. al [13] utilised Genetic Programming as a Hyper Heuristic (GPHH) for evolving UCARP routing policies and showed its superiority to existing methods.

The GPHH method is effective for solving UCARP but it is a time-consuming process. Additionally, if a change, such as number of vehicles, etc., occurs the trained routing policies will not perform optimally for the new problem [13] and, it is required to train new routing policies. However, when the change in problem is not drastic, it is reasonable to believe that transfer learning methods can extract reusable knowledge from the old solution(s) of the old problem to help the training process for the new problem. Previous studies have shown that applying transfer learning methods for handling scenario changes of UCARP is a challenging task [1, 3]. As a result, it is needed to have a formal framework that facilitates the analysis of the knowledge transfer process for UCARP.

Accordingly, in this paper, we propose a generic parametric knowledge transfer framework to formalise the steps involved in performing knowledge transfer for handling scenario changes of UCARP. In our framework, different parameters formulate the influence of different aspects of a knowledge transfer scenario. Consequently, we seek to fulfil the following research goals: 1) Propose a general parametric transfer learning framework for handling scenario changes of UCARP; 2) analyse the effect of each component of the framework, governed by different parameters, on the success of knowledge transfer; 3) speculate on considerations for effective extraction and update of the knowledge. We study this framework for handling scenario changes of UCARP (Section 2.1). Our preliminary work showed that the Probabilistic Prototype Tree (PPT) structure has a good potential for representing transferable knowledge [4]. Therefore, we select this structure as an explicit representation for knowledge. We note that knowledge representation is an abstract term in our generic framework and we intend to use this structure to help analyse our framework experimentally and illuminate the potentials and shortcomings of a transfer learning scenario.

This paper is organised as follows. In Section 2, a background review is given. In Section 3 we describe the considered transfer learning framework. Section 4 presents the experimental setup. This section contains detailed discussion about the different aspects of the framework framework. We conclude the paper in Section 5.

2 Backgrounds

2.1 UCARP

UCARP is an uncertain combinatorial problem defined on a connected undirected graph $G(V, E)$ with V as its set of nodes and E as its set of edges [13]. Each node $v \in V$ represents a location and each edge $e \in E$ is a route between two locations and has a stochastic non-negative demand $d(e)$. Edges with positive demand values are called tasks. In UCARP, a set of vehicles, with limited capacity Q, are considered to serve the tasks that are spread over the edges. In the beginning, all vehicles are stationed at a depot node $v_0 \in V$. Serving a task $e \in E$ incurs two costs: 1) the stochastic deadheading cost, which represents the cost of traversing e with or without serving it and, 2) the positive deterministic cost of serving the task. In UCARP, the goal is to find a set of routes for all vehicles that serves all the tasks while minimising the sum of all costs. There are three constraints that should be considered: 1) a vehicle cannot serve more than its capacity without going back to the depot; 2) all vehicles are required to start and end their routes with the depot; 3) each task can be served only once [13].

The best approach for solving UCARP is to search for routing policies that generate solutions in real-time [13]. A routing policy is a real-valued function that assigns a real number as a priority value to unserved tasks, based on the state of environment. Vehicles consider the priority and choose the task with the best priority to serve next. Uncertainty is an important aspect of real-world scenarios [13] and routing policies help vehicles to assess available tasks in real-time and select their next task based on the up-to-date state of their environment. Routing policies can be trained with GP.

GPHH for UCARP Genetic Programming is an Evolutionary Computational (EC) algorithm that evolves a population of computer programs [5]. Like GA, GP applies the genetic operators to its population iteratively to search for an optimal program. When the search space of GP is the space of heuristics, the algorithm is usually referred to as Genetic Programming Hyper-Heuristic (GPHH). Since routing policies are heuristics in essence, GPHH can be utilised to evolve routing policies for UCARP. GPHH initialises a population of routing policies randomly, evaluates the fitness of each population member on a set of training instances, creates a new population by applying the genetic operators (crossover, mutation, reproduction) to the current population, and repeats these steps until a stopping condition is met. Routing policies are represented with GP trees [20].

Routing policies generate the final set of vehicle routes that serve all the tasks. Generating the routes starts with all the vehicles stationed at the depot. Whenever idle, vehicles consider all accessible tasks and utilise the routing policy to assign a priority to each task, select the task with the best priority and, start serving it. This process of selecting and serving tasks is repeated until all tasks are served. The fitness of a routing policy is the mean total cost of the solutions it generates over training instances [13]. Training routing policies is a time-consuming process and this fact is exacerbated by the finding that if a change such as the number of vehicles, occurs it is required to train new routing policies [3]. However, when the change in problem is not drastic, transfer learning methods can help the retraining process.

2.2 Transfer Learning for Genetic Programming

Torrey et al. define transfer learning (TL) as "the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned" [19]. Therefore, instead of learning a new task from scratch, transfer learning allows salvaging the knowledge learned from similar solved tasks to increase the accuracy or decrease the expense of the learning process. In this context, a *negative transfer* happens when extracted knowledge decreases the performance. When the learning technique is Genetic Programming (GP), the same criteria can be considered and one can expect a GP-based TL to 1) create a good initial population; and 2) speed up the evolutionary process; 3) achieve a better

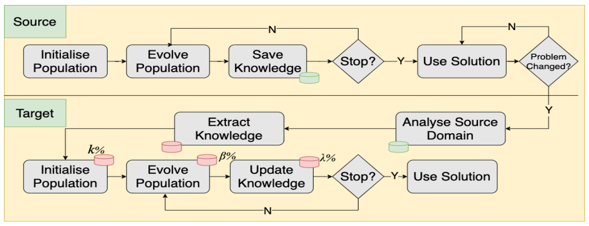


Fig. 1. A general transfer learning framework

Table 1. Framework Parameters

Par.	Description
k	Initialisation rate
β	Search contribution rate
x	Crossover rate
m	Mutation rate
r	Reproduction rate
λ	Learning rate

final performance. This section reviews some of TL methods for evolutionary algorithms and interested readers are referred to [9, 19] for more comprehensive reviews.

Taylor et al. [18] proposed knowledge transfer for evolutionary algorithms by transferring the whole final population from a solved source domain to initialise a Genetic Algorithm (GA) population on a target domain.

Dinh et al. [6] proposed three methods for initialising GP population. In the *FullTree-k* algorithm, k portion of the best individuals in the final population of a source domain are selected as initial GP individuals on a target domain. Their *SubTree-k* method focused again on the final GP population of the source domain but extracted a randomly-selected subtree from each individual and used them as new individuals in the first GP population of the target domain. Their final method, called *BestGen-k* selected k portion of the best individuals of each generation to initialise a new GP.

Ardeh et al. [3] believed that the subtrees that appear frequently in a source domain must be important that GP created them repeatedly. Their algorithm, *FreqSub*, considered them as new individuals to initialise GP of a target domain. Later [2], they measured the importance of subtrees based on their contribution to fitness and transferred the most contributing ones as new individuals.

Ardeh et al. also performed transfer learning as transfer of feature/function importance [1,4]. In a more recent work, Ardeh et al. [4] learned the probability distribution of creating good individuals from a source domain. They utilised PPTs for representing the learned probabilities and used the transferred PPTs for initialising GP in target domains.

3 The Proposed Parametric Framework for Transfer Learning

Transfer learning is needed when a change in a solved problem creates a new but related problem. In this context, there are a few key decisions: 1) how to represent the transferable knowledge? 2) how to extract the knowledge? 3) how to use the knowledge? 4) how to update the knowledge?

Knowledge representation can be either *explicit* or *implicit*. In explicit representations, the transferable knowledge is extracted into a reusable model $\mathcal{M}$ that expresses some form of understanding about the source domain. The works of Feng et al. [8] and Ardeh et al. [4] are in this category that respectively, use

Algorithm 1: Pseudocode for PKGPHH

```
    Input  : Training dataset $\wp$
    Output: The best learned heuristic $ind^*$
 1: $ind^* \leftarrow null$, $gen \leftarrow 0$, $k \leftarrow 0$
    // Load final population of source domain, if available.
 2: $\Im \leftarrow$ loadFinalPopulationSource ()
    // Specify portion of target domain population to create from
       extracted knowledge
 3: $k \leftarrow$ getInitPercent ()
 4: $\Im' \leftarrow$ Select top members of $\Im$ to learn from
 5: if $\Im' = \varnothing$ then
 6:     $\mathcal{M} \leftarrow \varnothing$
 7: end
 8: else
 9:     $\mathcal{M} \leftarrow$ Extract knowledge from the selected individuals (e.g extract a
          PPT).
10: end
11: Initialisation: Create $k \times Popsize$ individuals from the $\mathcal{M}$ by selecting
      functions/terminals for each node based on the probability distribution.
12: Randomly initialise $(1 - k) \times Popsize$
13: while $gen < maxGen$ do
14:     Evaluate each individual on $\wp$ and update $ind^*$
15:     Apply clearing
16:     if $\mathcal{M} \neq \varnothing$ then
17:         Update $\mathcal{M}$
18:         Breed $\beta \times Popsize$ of the population from $\mathcal{M}$
19:     end
20:     Apply crossover, mutation and reproduction
21:     $gen \leftarrow gen + 1$
22: end
23: return $ind^*$
```

memes of statistical dependency and probability prototype trees as the explicit representations. In the implicit approach, the model $\mathcal{M}$ is not extracted but some manifestation of the hidden knowledge are considered from the source domain. The work of Dinh et al. [6] is in this category. Knowledge extraction is heavily dependent on the selected representation. It is important to consider that when the two domains are related, the knowledge contributes to the *exploitation* phase of EC methods and relying too much on it may run the risk of early convergence. In the context of EC-based algorithms, the transferred knowledge can be used to 1) initialise $k \in [0, 1]$ portion of the initial population; 2) guide the evolution by contributing to β percent of the search effort. The value of β should be selected in ordinance with crossover rate x and mutation rates m.

The knowledge from a source domain may be good enough for giving EC algorithm a better-than-random state but as search becomes more focused on the characteristics of the target domain, its usefulness degrades. Hence, it is needed to update the knowledge with new information. The important decision here is the learning rate $\lambda \in [0, 1]$ that specifies what portion of the old knowledge should be preserved with $\lambda = 0$ meaning no update at all. The details of the update mechanism depends on the knowledge representation.

Figure 1 presents a flowchart of this general framework and a more pseudo code of the framework is given in Algorithm 1. Table 1 gives a summary of the parameters introduced here. In the remainder of this paper, we investigate the proposed general framework with the help of *FullTree*, *SubTree* [6] and *FreqSub* [3] methods. Since all these methods have an implicit knowledge representation and only focus on initialisation, we also extend the model-based method in [4] with the capability of updating the model during the GP run so that it demonstrates different aspects of the framework. This method will be referred to as Probabilistic Knowledge-enabled GPHH (*PKGPHH*). The goal here is to help the better understanding of the framework and illustrate how it can help with gaining a better insight of a transfer learning pipeline rather than proposing a new and more efficient algorithm.

3.1 Probabilistic Knowledge-enabled GPHH

Knowledge Representation and Extraction We consider the probability distributions of good individuals of a source domain as an explicit transferable knowledge and utilise PPTs to store the probability distributions. A PPT is a complete binary tree in which each node holds the probabilities that each GP function/terminal may appear in a tree at the corresponding node. It can be proved with maximum likelihood estimation [7] that the probability of a GP function/terminal g appearing in a particular position in the tree is $p_g = M_g/N$ in which N is the total number of individuals considered for learning; M_g is the number of GP trees in the population that contain g at the given node. Figure 2 demonstrates an example PPT for a simple GP configuration for which the terminal set is $T = \{a, b\}$, the function set is $F = \{+, *\}$ and the maximum tree depth is two.

Presence of code bloat and duplicates can have a negative effect on the quality of transfer learning techniques [3]. To fight off this issue, we utilise the clearing method [15]. The clearing method relies on the phenotypic characterisation of GP trees for measuring similarity and is based on the work in [12].

Knowledge Usage To create new GP individuals in target domains, first a GP function/terminal g is selected for the root node with the probability specified in the PPT root node for g. Then, for each child of the root (if it can have a child), a GP terminal/function is selected based on its PPT probability and this process is repeated until maximum tree depth is reached. The transferred PPT is used to create k portion of the initial GP population and, β portion of new populations in each generation alongside crossover and mutation operators.

Knowledge Update To keep transferred PPT updated to the specifications of the target domain, we update each probability value incrementally with $P_g^{t+1} = \lambda P_g^t + (1 - \lambda)\bar{P}_g$. Here, P_g^t is the probability value of GP function/terminal g at generation t and $\bar{P}_g = M_g^t/N$ is the estimated probability value for g from the

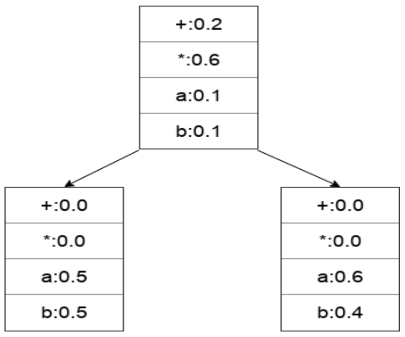


Fig. 2. A simple PPT

Table 2. UCARP Terminal Settings

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

best individuals of the current population (M_g^t is the number of individuals that contain g at a corresponding node).

4 Experimental Studies

In order to investigate the proposed framework, we consider the scenario in which the difference between source and target UCARP domains is based on the number of vehicles (Table 3). We consider *SubTree*, *FullTree* [6] and *FreqSub* [3] from the body of existing works alongside the *PKGPHH* method (research question in Section 1). The GP terminal set is presented in Table 3.1. GP function set is $\{+, -, \times, /, min, max\}$ [3]. A population of 1024 individuals is evolved for 50 generations with a reproduction rate of 0.05. Max tree depth is 8, and 10 elites of a generation are moved to the next generation. For training, 5 samples are considered which are rotated each generation to improve generalisation while 500 samples are used for testing. We obtained the best results with the $\epsilon = 0, c = 1$ for niching radius and capacity settings [15]. All comparisons are performed with the Wilcoxon rank sum test of 30 independent runs with 0.05 significance level. *SubTree*, *FullTree* and *FreqSub* initialise 10% of GP population in the target domain and are referred to as *SubTree-0.1*, *FullTree-0.1* and *FreqSub-0.1* respectively. For PKGPHH, we extracted the transferable PPT from top 10% individuals of the source domain and considered different initialisation rates k, breeding rate β, learning rate λ and crossover x rate and, is referred to as *PKGPHH(β, x, λ, k)*. For these experiments, we rely on domain expertise for the relatedness of the source and target domains.

4.1 Experiment Results

A major goal of transfer learning is to reduce the time for retraining the routing policies. Accordingly, we considered the earliest generation in which test

Dataset	gdb1	gdb1	gdb2	gdb2	gdb3	gdb3	gdb4	gdb4	gdb5	gdb5	gdb6
Source	5	5	6	6	5	5	4	4	6	6	5
Target	4	6	5	7	4	6	3	5	5	7	4
Dataset	**gdb6**	**gdb7**	**gdb7**	**gdb12**	**gdb12**	**gdb14**	**gdb14**	**gdb15**	**gdb15**	**gdb16**	**gdb16**
Source	5	5	5	7	7	5	5	4	4	5	5
Target	6	4	6	6	8	4	6	3	5	4	6
Dataset	**gdb17**		**gdb17**	**gdb13**	**gdb13**	**gdb19**	**gdb19**	**gdb20**	**gdb20**	**gdb21**	**gdb21**
Source	5		5	6	6	3	3	4	4	6	6
Target	4		6	5	7	2	4	3	5	5	7

Table 3. Number of vehicles on source and target problems

performance of a transfer learning method is statistically similar to the final population of vanilla GP without knowledge transfer for at least three consecutive generations. Figure 4 depicts the percentage of generations on target domain that could be saved by applying each transfer algorithm, averaged over all experiments. According to Table 4, PKGPHH achieved the final performance of vanilla GP at least 57.2% earlier which indicates its potentials for reducing the training time. Interestingly, based on Table 4, the best improvement is achieved by the *Full-Tree* method. Table 4 presents a summary of how the final performance of each algorithm compares to that of vanilla GP without transfer in the form of ('wins', 'draws', 'losses') triples in which the number of 'wins', 'draws' and 'losses' of an algorithm indicate the number of times its final performance was statistically better than, similar to or worst than the baseline algorithm. According to Table 4, for the majority of the experiments, *PKGPHH* is on par with the existing methods and Table 4 indicate that *PKGPHH* has the potential for transfer learning that is similar to existing methods. In the following subsections, we will consider our research questions and use the results to give an extensive analysis of the framework and gain a better understanding about its dynamics.

4.2 Exploration vs Exploitation

The effect of transferred knowledge on the GP evolution is governed by different parameters. Large values of β place a high focus on the exploitation of old

Algorithm	SubTree-0.1	FullTree-0.1	PKGPHH(0.1, 0.7, 0.2, 0.1)
Performance	(1, 30, 1)	(1, 28, 3)	(0, 32, 0)
Improvement	63%	63.3%	57.2%
Algorithm	FreqSub-0.1		PKGPHH(0.1, 0.7, 0.2, 0.5)
Performance	(4, 20, 8)		(3, 25, 4)
Improvement	43.6%		51%

Table 4. Performance of algorithms in terms of number of "wins", "draws" and "losses" of algorithms against vanilla GP and, average improvements in training effort achieved

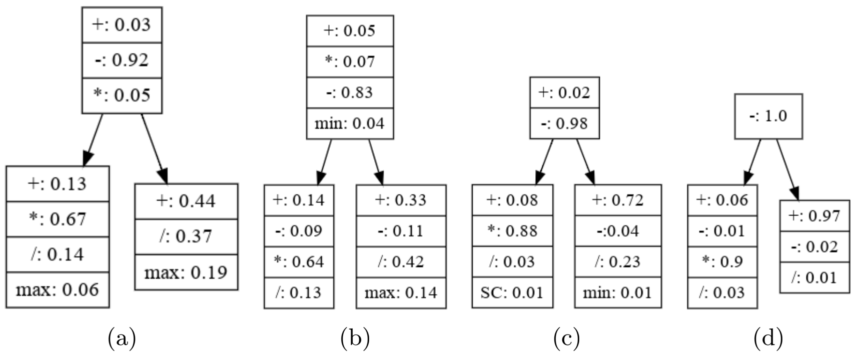


Fig. 3. Evolution of an extracted PPT (a) after generation 2 (b), 25 (c) and 50 (d) in target domain.

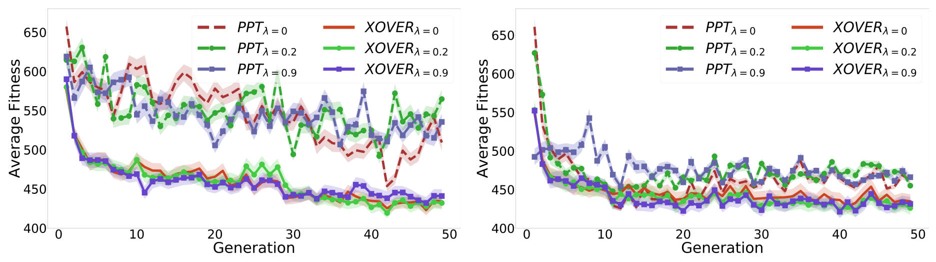
knowledge and can deter GP's search mechanism. We observed that when β is large, GP converges to local optima and even more exploration through large values of mutation rate could not overcome it. For example, for the case of PKG-PHH(0.325, 0.325, 0.3, 0.5), the improvement was 47.13%. We noticed that GP's performance for UCARP is sensitive to the initial state (controlled with k) and GP will converge to local optima if large portion of the initial population is created from the transferred knowledge. This effect was observed in all algorithms. For instance, in Table 4, increasing k from 0.1 to 0.5 reduced improvement from 57.2% to 51.0% and added 4 more negative transfer cases. However, more cases of significantly better final performance were observed with larger initialisation rates too. This usually happens when important regions of the target domain's search space are seen in the source domain and the initialisation can bias GP towards these regions.

4.3 Quality of Learned Knowledge

Figure 3 shows evolution of a PPT (Figure 3a) extracted from a source domain (only top two levels are shown to save space) and tracks its structure after generation 2 (Figure 3b), generation 25 (Figure 3c) and generation 50 (Figure 3d). The first thing to notice is that one GP function at each shown node has a probability that is very higher than others. We observed the same property in lower levels of the PPT. This indicates a lack of diversity in the source domain. Considering that niching is performed in the source domain, this figure shows that even if the niching algorithm increases phenotypic diversity, the population still can lose its genotypic diversity in the subset of top-performing individuals. Increasing the number of individuals to learn the PPT does not help since either their fitness is not good or they have low probabilities and will not be selected.

4.4 Sensitivity to Learning Rate

Another issue that is evident from Figures 3 is that the probability values do not change over time, even when the learning rate λ is zero. To understand



(a) (b)

Fig. 4. Average fitness of individuals bred from PPT (PPT) and crossover (XOVER) for different learning rates λ (gdb2, 6 to 7 vehicles) (a) clearing is enabled versus (b) clearing is disabled.

the effect of learning rates, we recorded the fitness of all individuals that were created from the PPT breeding and crossover operators at each generation and Figure 4a plots them for different learning rates for one of the experiments with $\beta = 0.1, k = 0.1$. According to this figure, the fitness of individuals created from PPT (labelled as *PPT* in the plot) are usually worse than the fitness of the individuals created from crossover (labelled as *XOVER* in the plot). Also, it is noticeable that different learning rates do not have any effect and, for all learning rates, standard deviations of PPT-bred individuals is more than crossover-bred individuals. Consequently, we can conclude that the learning mechanism is not very efficient and the population suffers from some sort of convergence to local optima from which GP cannot escape.

4.5 Effectiveness of Clearing

Previous studies showed that diversity in source domains affects quality of knowledge transfer [3]. To verify this, we disabled the clearing method in both source and target domains and we identified a few more cases of significant final improvements and less cases of negative transfer than the results reported in Table 4. Figure 4b shows that again, different values of λ do not have important effect but disabling the clearing method has an observable effect and it actually improves the quality of the individuals that are created from the PPT. Looking at the structure of the PPT for this experiment showed that the lack of diversity has lead to a PPT that has single probability values in many nodes. This leads to creation of trees that are very similar to the best individuals the PPT was learned from. Conversely, the PPTs that represent more diversity due to the clearing procedure create trees that are more diverse but do not have good qualities because the current structure of PPT does not consider inter-dependence of probability distributions (more investigations in Subsection 4.6). Overall, we suspect that genotypic [17] niching may alleviate this problem.

4.6 Effectiveness of Knowledge Representation

It is possible that the selection of the PPT structure for knowledge representation, $\mathcal{M}$, is limited. A basic assumption of our model is the independence of probability distributions governing each PPT node. While this assumption was effective in the area of probabilistic optimisation [16], other works have also noticed its flaws and have proposed multivariate modelling of the search space with Bayesian networks [10, 11, 14]. In our investigations, we noticed that good individuals that, for example, have / as the root's right child node, usually have $*$ as the root's left child node. However, the PPT structure does not represent this relationship.

5 Conclusions and Future Work

In this paper, first we proposed a parametric framework for transfer learning and investigated it with a simple implementation that offers straightforward mechanisms for updating and using the transferred knowledge during GP run. Utilising this method, we probed the framework's features with a case study of tackling the UCARP scenario changes. Our experiments demonstrated that the framework can model delicacies of transfer learning.

Formulating a transfer learning task based on our framework, helped us note that although transfer learning can have strong potentials, devising a powerful structure for knowledge representation and updating it effectively are key factors for producing an effective and efficient performance in target domains. Striking a balance between exploitation and exploration is another important challenge. In our framework, the transferred knowledge can be used during both GP initialisation and evolution and our experiments showed that both of them should be considered carefully to reach the balance. The transferred knowledge can add to the exploitation phase of GP and relying on it excessively can lead to convergence to local optima. Relatedness of a source domain to a corresponding target domains is also an important considerations that can affect the performance. To the best of our knowledge, there exists no methods for measuring the similarity between UCARP domains. In our experiments, we relied on domain expertise for defining related source and target domains. However, this assumed relatedness is not guaranteed. We leave the investigation of this issue and its integration into the framework for future works.

Although we utilised an experimental transfer learning algorithm implementing the framework, we believe the potentials of the framework are general in nature and our experiments with GP and UCARP are examples that demonstrate how different aspects of the framework can impact the success of knowledge transfer which we aim to demonstrate in our future works.

References

1. M. A. Ardeh, Y. Mei, and M. Zhang. Genetic Programming Hyper-heuristic with Knowledge Transfer for Uncertain Capacitated Arc Routing Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 334–335. ACM, 2019.

2. M. A. Ardeh, Y. Mei, and M. Zhang. A novel genetic programming algorithm with knowledge transfer for uncertain capacitated arc routing problem. In *PRICAI 2019: Trends in Artificial Intelligence*, pages 196–200. Springer, 2019.
3. M. A. Ardeh, Y. Mei, and M. Zhang. Transfer learning in genetic programming hyper-heuristic for solving uncertain capacitated arc routing problem. In *2019 IEEE Congress on Evolutionary Computation*, pages 49–56, 2019.
4. M. A. Ardeh, Y. Mei, and M. Zhang. Genetic programming hyper-heuristics with probabilistic prototype tree knowledge transfer for uncertain capacitated arc routing problems (accepted to appear). In *2020 IEEE Congress on Evolutionary Computation (CEC)*, 2020.
5. A. De Lorenzo, A. Bartoli, M. Castelli, E. Medvet, and B. Xue. Genetic programming in the twenty-first century: a bibliometric and content-based analysis from both sides of the fence. *Genetic Programming and Evolvable Machines*, 2019.
6. T T H Dinh, T H Chu, and Q U Nguyen. Transfer learning in genetic programming. In *IEEE Congress on Evolutionary Computation*, pages 1145–1151, 2015.
7. Scott R. Eliason. *Maximum likelihood estimation : logic and practice*. Sage, 1993.
8. 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*, 19(5):644–658, 2015.
9. A. Gupta, Y. Ong, and L. Feng. Insights on transfer optimization: Because experience is the best teacher. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):51–64, 2018.
10. Y. Hasegawa and H. Iba. Optimizing programs with estimation of Bayesian network. In *IEEE Congress on Evolutionary Computation*, pages 1378–1385, 2006.
11. Y. Hasegawa and H. Iba. A Bayesian Network Approach to Program Generation. *IEEE Transactions on Evolutionary Computation*, 12(6):750–764, dec 2008.
12. Torsten Hildebrandt and Jürgen Branke. On using surrogates with genetic programming. *Evolutionary Computation*, 23(3):343–367, 2015.
13. Y. Liu and Y. Mei. Automated Heuristic Design Using Genetic Programming Hyper-Heuristic for Uncertain Capacitated Arc Routing Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 290–297, 2017.
14. M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian Optimization Algorithm. In *IlliGAL Report*, GECCO'99, pages 525–532, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
15. A. Petrowski. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of IEEE Conference on Evolutionary Computation*, 1996.
16. R. Sałustowicz and J. Schmidhuber. Probabilistic Incremental Program Evolution: Stochastic search through program space. In *Lecture Notes in Computer Science*, pages 213–220. Springer, 1997.
17. Ofer M Shir. *Niching in Evolutionary Algorithms*, pages 1035–1069. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
18. M. Taylor, S. Whiteson, and P. Stone. Transfer learning for policy search methods. In *Proceedings of the International Conference on Machine Learning*, 2006.
19. Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global, 2010.
20. F. Zhang, Y. Mei, and M. Zhang. A two-stage genetic programming hyper-heuristic approach with feature selection for dynamic flexible job shop scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 347–355, 2019.