Explainable Genetic Programming for Evolving Routing Policies of Uncertain Capacitated Arc Routing Problems

by

Shaolin Wang

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2023

Abstract

The Uncertain Capacitated Arc Routing Problem (UCARP) is a well-known combinatorial optimization problem that has many real-world applications. Genetic programming has successfully evolved routing policies that can make real-time routing decisions for uncertain arc routing problems. Although the evolved routing policies are highly effective, they are typically very complex thus hard for real users to understand and trust. Therefore, it is necessary to improve the interpretability of GP-evolved routing policies.

The overall goal of this thesis is to develop both intrinsic and post-hoc methods to improve the interpretability of GP-evolved routing policies for UCARP.

First, this thesis proposes a novel intrinsic genetic programming approach that simplifies the routing policies during the evolutionary process using a niching technique. The simplified routing policies are stored in an external archive. This thesis also develops new elitism, parent selection, and breeding schemes for generating offspring from the original population and the archive. The experimental results show that the newly proposed approach can achieve significantly better test effectiveness than the current state-of-the-art genetic programming hyper heuristic methods for UCARP. The evolved routing policies are smaller, thus potentially more interpretable.

Second, this thesis develops several multi-objective genetic programming algorithms to evolve a Pareto front of routing policies with different interpretability and effectiveness, so that the end users can choose based on their preferences. This thesis handles two main challenges, i.e., ob-

jective selection bias issue and stochastic fitness evaluation issue, of designing Multi-Objective GP (MOGP) for UCARP. To handle the objective selection bias issue, an α dominance strategy is proposed. To further improve the α dominance strategy, this thesis develops a new α value adjustment scheme. To handle the stochastic fitness evaluation issue, this thesis proposes an archive strategy. The new archive strategy uses an external archive to store the potentially effective individuals that could have been lost during the traditional GP process. The individuals in the archive contribute back to the population in the breeding process. Finally, this thesis combines all the above strategies to develop a new MOGP algorithm. The experimental results show that the new MOGP algorithm outperforms state-of-the-art algorithms for UCARP in terms of HV and IGD. The evolved routing policies are much smaller, thus potentially more interpretable.

Lastly, this thesis proposes a new post-hoc explanation approach to explaining the effective but complex GP-evolved routing policies. This thesis proposes two metrics that can evaluate the quality of a local explanation for GP-evolved routing policies. Then, this thesis investigates the feasibility of using a linear model to construct a local explanation for a complex GP-evolved routing policy in a decision situation. After that, this thesis further improves the local explanation method using Particle Swarm Optimization (PSO) to learn an interpretable linear model that accurately explains the local behavior of the routing policy for each decision situation, and extends the local explanation method to a global explanation method. The experimental results and case studies show that the proposed method can obtain accurate and understandable explanations of GP-evolved routing policies for UCARP.

From the intrinsic aspect, both GP program simplification method and multi-objective GP methods are proposed. The GP program simplification approach addresses the case where only a single interpretable routing policy is required, i.e. where the user's preferences are known before the

routing policy is generated. The multi-target approach is aimed at situations where several different interpretable routing policies are needed, because sometimes we cannot know the user's preferences in advance, and the user can choose their own routing policy according to their preferences. From the post-hoc aspect, this thesis proposes local and global explanation methods to further improve the interpretability of existing complex GP-evolved routing policies. Furthermore, the local and global explanation methods not only produce text explanations but also visual explanations.

List of Publications

* Shaolin Wang, Yi Mei, Mengjie Zhang. "Explaining Genetic Programming Evolved Routing Policies for Uncertain Capacitated Arc Routing Problem". 2023, IEEE Transactions on Evolutionary Computation, DOI:10.1109/TEVC.2023.3238741.

* Shaolin Wang, Yi Mei, Mengjie Zhang. "Genetic Programming With Niching for Uncertain Capacitated Arc Routing Problem". IEEE Transactions on Evolutionary Computation, vol. 26, no. 1, pp. 73–87, 2022.

* Shaolin Wang, Yi Mei, Mengjie Zhang. "A Multi-Objective Genetic Programming Algorithm with α dominance and Archive for Uncertain Capacitated Arc Routing Problem". IEEE Transactions on Evolutionary Computation, 2022, DOI:10.1109/TEVC.2022.3195165.

* Shaolin Wang, Yi Mei, Mengjie Zhang. "Local ranking explanation for genetic programming evolved routing policies for uncertain capacitated Arc routing problems". Genetic and Evolutionary Computation Conference (GECCO), ACM, 2022. pp. 314-322.

* Shaolin Wang, Yi Mei, Mengjie Zhang. "Two-stage multi-objective genetic programming with archive for uncertain capacitated arc routing problem". Genetic and Evolutionary Computation Conference (GECCO), ACM, 2021. pp. 287-295.

* Shaolin Wang, Yi Mei, Mengjie Zhang. "A Multi-Objective Genetic Programming Approach with Self-Adaptive α Dominance to Uncer-

tain Capacitated Arc Routing Problem." IEEE Congress on Evolutionary Computation (CEC). IEEE, 2021. pp. 636-643.

* Shaolin Wang, Yi Mei, Mengjie Zhang. "A Multi-Objective Genetic Programming Hyper-Heuristic Approach to Uncertain Capacitated Arc Routing Problems." IEEE Congress on Evolutionary Computation (CEC). IEEE, 2020. pp. 1-8.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisors A/Prof. Yi Mei and Prof. Mengjie Zhang for their great supports, entrenchments, and guidance. A/Prof Yi Mei spends dedicated time and efforts to help me improve my research skill. He always encourages and guides me through all the hard times during my study. He not only helps me in my research but also assists me in my life. He is always the person I am seeking for help when I have any problem in my life. Prof. Mengjie Zhang is always nice to talk. All the discussions with him always lead me to think deeper and have better ideas. Both professors give me the most help and constructive feedback in writing this thesis and the articles that came before it.

Thank you to my parents (Jisen Wang and Jinping Shao) for their encouragements. From a child to now, it has been difficult but you have always brought me through it with your encouragement, love and support. Thank you to my all family members for their constant moral support!

I wish to thank all my friends in School of Engineering and Computer Science, especially members of Evolutionary Computation Research Group (ECRG). Special thanks to my officemates Ying Bi, Fangfang Zhang, Ruwang Jiao, Qinglan Fan, Peng Wang, Jiabin Lin, Boxiong Tan, Jordan MacLachlan and Tao Shi for their jokes and tasty food. Thank you to the rest of the Evolutionary Computation Research Group, for the many lively and interesting discussions.

Special thanks to my friends, Agudamu, Liying Jin, Yang Guo, Li Li,

Lei Yang, Zhi Zou, Ziyue Teng, Yongbo Yu and Yingliang Shao for their help during my PhD period.

Last but not least, I would like to express my sincere thanks to all those who gave me the assistance and support during my PhD study. It would not be possible to complete this thesis without your help.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter introduces the research problem that this thesis addresses, then describes the motivations, the research goals, and the major contributions of the thesis.

1.1 Problem Statement

The Capacitated Arc Routing Problem (CARP) [72] is a classic combinatorial optimisation problem with a number of real-world applications, such as waste collection [7] and winter gritting [75]. The main goal of CARP is to serve a set of edges in a graph using a fleet of vehicles with the minimum cost. CARP is considered as an arc routing counterpart to the well-known Vehicle Routing Problem (VRP). It has been proven to be NP-hard [204] and received much research interest. Many approaches have been proposed for dealing with CARP [204]. However, most previous studies assume a static environment, where all the parameters (e.g. task demand, travel cost) are fixed and known in advance. This is usually not true in the real world, where the environment is often uncertain. For example, in waste collection, the amount of waste to be collected is not known in advance and varies from one day to another.

Uncertain CARP (UCARP) [121,141,185] was proposed to better reflect

the reality. A variety of uncertainties, such as the stochastic task presence and demand, stochastic edge deadheading cost and stochastic travel time, have been considered [121]. In addition to the NP-hardness inherited from CARP, the main challenge in UCARP is the *route failure* caused by the stochastic task demand. A route failure occurs when the remaining capacity of the vehicle becomes insufficient to serve and edge, because the actual demand of an edge that needs to be served exceeds the vehicle's expected demand. Therefore, the vehicle has to return to the depot to refill and come back to finish serving the edge again. A route failure might lead to a high recourse cost.

UCARP is not just a standalone optimization challenge, it is an integral part of the extensive landscape of logistics and supply chain management [71]. In today's world of commerce and global trade, managing supply chains efficiently is of utmost importance. UCARP, representing a challenge in routing and logistics, plays a pivotal role in tackling the last-mile delivery challenge, optimal vehicle fleet management, and resource allocation within ever-changing and uncertain environments. In sectors like e-commerce, where on-time deliveries are crucial for customer satisfaction, UCARP solutions hold the promise of optimizing routes, reducing delivery costs, and lessening the environmental impact. Moreover, with supply chains constantly evolving in complexity and adaptability, UCARP solutions can aid in streamlining distribution networks, ensuring a seamless flow of products from manufacturers to end consumers. Hence, successfully addressing UCARP goes beyond enhancing routing efficiency. It ripples across the broader logistics and supply chain ecosystem, influencing cost-effectiveness, environmental sustainability, and the overall resilience of operations [86].

Traditional solution optimisation approaches to dynamic routing problems, such as mathematical programming [72], genetic algorithm s [185], particle swarm optimisation [101] and Markov decision processes [170], cannot effectively handle the route failures in UCARP. On the one hand,

the traditional solution optimisation approaches try to optimise a solution (i.e., a solution that performs reasonably well in all possible environments) beforehand. If a route failure occurs, it is repaired by the recourse operator. However, the pre-planned solution is not flexible enough and cannot be sufficiently changed by the recourse operator to cope with the dynamic nature of the route failure situation. Additionally, it may induce a large extra recourse cost in some situations. On the other hand, some solution optimisation methods [41, 170] aim to re-optimise the remaining routes when a route failure occurs. Although this can potentially obtain better solutions, the re-optimisation process by running the solution optimisation algorithm is typically too slow to respond in real-time.

Routing policy-based approaches [166] are promising techniques that can effectively handle the uncertain environment in UCARP. Unlike traditional solution optimization approaches, these approaches do not have to pre-plan any solutions. Instead, they use routing policies to guide a vehicle to determine which task to serve next as soon as it becomes idle [166]. A typical manually designed routing policy is Path Scanning [113]. However, numerous factors, such as the objective(s) and the graph topology, can affect the effectiveness of the routing policy [91].

Recently, Genetic Programming Hyper Heuristic (GPHH) has been considered a promising technique that can automatically evolve effective heuristics for dynamic problems [1], such as scheduling problems [26, 152, 214, 215] and resource allocation problems [180]. GPHH has also been successfully applied to UCARP to evolve effective routing policies automatically [67,122,123,133,143]. Although the GP-evolved routing policies have shown effectiveness, they are often too complex to be understood by real users. With the use of Artificial Intelligence (AI), interpretability has become increasingly important. Interpretability is the degree to which a human can understand the cause of a decision [144]. If people are unable to understand the principles behind the operation of AI models, it is difficult for these models to be trusted by users. As a result, the interpretability

of AI models has become a challenge for practical applications. Therefore, it is essential to improve the interpretability of GP routing policies for UCARPs.

In the world of logistics and supply chain management, where decisions significantly impact operations and finances, the importance of AI model interpretability cannot be emphasized enough [71]. Professionals like logistics managers, supply chain analysts, and decision-makers grapple with the complex orchestration of goods, vehicles, and resources. In environments marked by real-time shifts, unexpected disruptions, and varying customer needs, understanding the reasoning behind AI-generated decisions is crucial. Consider a logistics manager tasked with optimizing delivery routes for a vehicle fleet during peak hours. An interpretable routing policy can offer clear insights into why a particular route was chosen, factoring in elements like traffic conditions, delivery time slots, and cost-effectiveness. This transparency enables decision makers to adapt swiftly, rerouting vehicles to address emerging challenges. In the realm of supply chain planning, interpretable models can illuminate resource allocation, ensuring decisions align with strategic goals and ethical considerations. Additionally, in industries driven by compliance requirements, interpretable AI can furnish verifiable explanations, aiding in audits and regulatory adherence. In essence, interpretability acts as a bridge between the intricate landscape of AI-driven decisions and the human intuition and expertise that drive logistics and supply chain management [102, 139]. It cultivates trust, enhances efficiency, and promotes strategic adaptability.

Methods to improve the interpretability of AI models can be broadly divided into two categories [147]: intrinsic and post-hoc methods. Intrinsic methods are applied *during* the model's training process and are designed to develop specific learning/search mechanisms to generate effective and interpretable models [35]. In contrast, post-hoc methods are applied *after* model training to "explain" trained models. These methods can be applied to both "black-box" and transparent models because

they are typically decoupled from the main model [35]. However, despite the importance of interpretability, few studies have investigated the interpretability aspect of GP-evolved routing policies for UCARP [87,140]. This thesis aims to fill this gap in the literature.

To enhance interpretability of GP-evolved routing policies for UCARP, it is crucial to carefully evaluate the appropriateness of interpretability methods. This domain presents a complex challenge, necessitating a thoughtful and nuanced approach. Intrinsic methods, operating during the model's training or evolution process, hold promise for UCARP by directly shaping the evolved routing policies to enhance interpretability [147]. Through techniques like program simplification and multi-objective optimization, these intrinsic methods can potentially strike a balance between effectiveness and interpretability, catering to various user preferences. On the other hand, post-hoc methods, offering explanations after the model is trained, provide a complementary avenue [147]. Although they might not directly influence the model's structure, they are invaluable for dissecting and visualizing the intricate behavior of evolved routing policies. In UCARP, where routing decisions unfold sequentially, post-hoc methods can shed light on why a particular decision was made in a given context. Therefore, determining the suitability of interpretability methods for UCARP involves a dynamic interplay between intrinsic techniques that streamline the evolution process and post-hoc methods that unravel the intricacies of the final routing policies. Together, they strive to make UCARP solutions both effective and transparent.

The overall goal of this thesis is to develop both intrinsic and post-hoc methods to improve the interpretability of GP-evolved routing policies for UCARP.

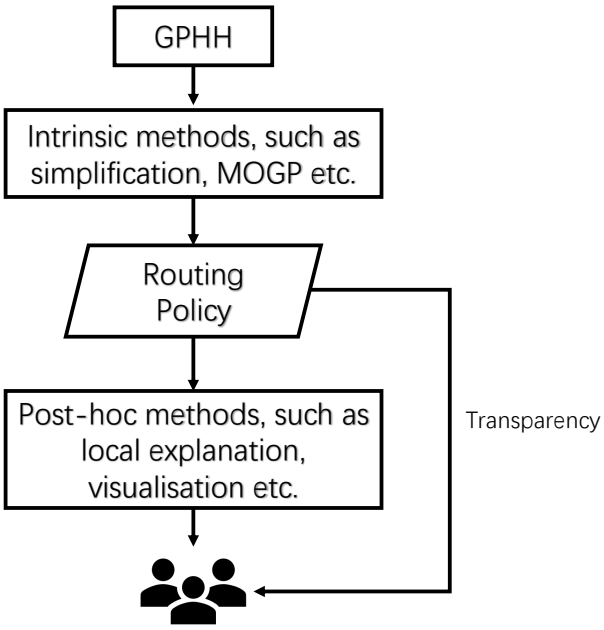


Figure 1.1: GPHH interpretability methods

1.2 Motivations

1.2.1 Why Interpretability

Recently, Artificial Intelligence (AI) has become increasingly popular. AI models can outperform humans on many tasks [14, 174] and often guide human decision-making processes [26, 34, 178]. With the development of AI, interpretability[1] has become a hot topic in the field. It is crucial for human users to understand and trust the AI model [13, 92, 145]. Usually, humans are reluctant to adopt techniques that are not directly interpretable, tractable, and trustworthy [223], especially when they are concerned about ethical AI [73]. An explanation that supports the output of an AI model is essential. For example, when experts make diagnoses in precision medicine, they require much more information from the model than a simple binary prediction to support their diagnosis [183]. Other examples include autonomous vehicles in transportation, security, and finance,

among others [13]. Here are some benefits of improving interpretability of the GP-evolved routing policies of UCARP [13, 139]:

1. Interpretability can help to ensure impartiality in decision-making, i.e., to detect and, consequently, correct bias in the training dataset.

2. Interpretability is able to facilitate the robustness of the model by highlighting potential adversarial perturbations that could change the prediction.

3. Interpretability can ensure that only meaningful variables infer the output. In other words, it can guarantee that a truthful causality exists in the model reasoning.

4. The real-world rule dispatchers can understand the rules and correct wrong decisions made by the rules in time.

5. The real-world dispatchers can understand the rules and easier to adopt them to new scenarios.

GPHH can evolve routing policies that produce solutions that outperform those of other methods as well as manually designed routing policies [122, 203]. However, previous studies of GPHH in UCARP have primarily focused on performance (effectiveness), which has been shown to be impressive. If we only focus on performance, the system will become increasingly opaque because there is a trade-off between a model's performance and its transparency [56]. In contrast to Deep Neural Networks (DNNs), which are considered uninterpretable black-box models [13], some models are considered more transparent or interpretable, such as linear/logistic regression, decision trees, k-nearest neighbours, rule-based learning, general additive models, genetic programming, and Bayesian models [13].

[1]In this thesis, interpretability, explainability and understandability are interchangeable. They can be defined as the ability of a model to explain or to provide the meaning in understandable terms to human operators.

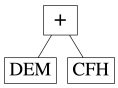


Figure 1.2: A simple and interpretable routing policy

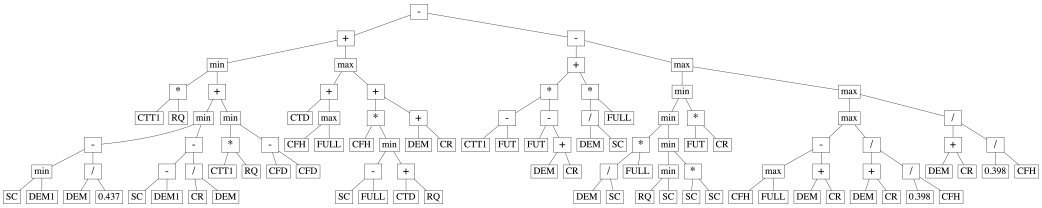


Figure 1.3: A complex and uninterpretable routing policy

However, when these models, such as fuzzy systems based on rule-based learning [5, 6], become too complex, they may not be as easy to interpret as expected. Although the program evolved by GP is considered interpretable because of its tree-based representation [60], if the evolved routing policy becomes too complex (either in program size or structure), it can still be very difficult to interpret. Therefore, interpretability should be considered alongside effectiveness. Most of the existing related works have only focused on the effectiveness of routing policies, ignoring interpretability. As a result, GP-evolved routing policies may be too complicated to interpret and understand. Practitioners may lack confidence in using the routing policies because they lack an understanding of the inner mechanism, despite their effectiveness in training instances. Improving the interpretability of evolved routing policies would make it much easier to adopt them in practice. Besides, it would be possible to fix the evolved routing policies when they make incorrect decisions.

Figs. 1.2 and 1.3 show two different routing policies. The routing policy in Fig. 1.2 is easily interpretable when we understand that CFH indicates the cost from the candidate task to the current location and DEM indicates the expected demand of the candidate task. This policy favors tasks that are near the current location and require less demand. On the other hand, even if we understand all the terminals in the routing policy in Fig. 1.3,

it is difficult to interpret and hard to read. While the routing policy in Fig. 1.2 is more interpretable, its effectiveness is weak. Therefore, it is necessary to develop intrinsic methods that evolve routing policies that are both interpretable and effective, or post-hoc methods to explain the complex routing policies.

In the broader scope of UCARP, the significance of explainability reverberates across various applications and industries. UCARP represents a class of optimization challenges that extend beyond its academic roots and directly impact real-world decision-making processes. From urban waste collection to emergency response planning, the decisions made through UCARP solutions can influence operational efficiency, resource allocation, and even public safety. In these contexts, the ability to understand the reasoning behind AI-driven routing policies is indispensable. Failure to provide transparent and interpretable solutions may lead to suboptimal outcomes, increased operational costs, and ethical concerns [54]. Thus, our research embraces the principles advocated by explainable AI (XAI) [54] and responsible AI (RAI) [96] frameworks. Our proposed methods prioritise transparency, accountability, and fairness, ensuring that the UCARP solutions are not just effective but also ethically sound. By aligning with XAI and RAI principles [189,209], our research empowers decision-makers across diverse domains to trust, understand, and make informed adjustments to AI-driven routing policies, thereby enhancing the broader applicability and ethical integrity of our solutions in the dynamic landscape of UCARP problem-solving.

1.2.2 Challenges of Evolving Interpretable Routing Policies

Even though GP is supposed to be an interpretable approach, the GP-evolved routing policies in UCARP are not easy to interpret [88,196]. Evolving interpretable routing policies is challenging because:

- GP tends to evolve trees containing a number of redundant components that have no contribution to performance. As a result, the GP-evolved routing policies become unnecessarily huge (also known as bloat) and hard to interpret. Firstly, it is difficult to detect those redundant components during the evolutionary process. Secondly, removing them during the evolutionary process can have a negative impact on the search and lead to worsened effectiveness.

- As with any machine learning algorithm, there is a trade-off between interpretability and effectiveness. GP trees that are designed to be more interpretable may sacrifice effectiveness, while more effective routing policies may be less interpretable. Finding the right balance between the two can be challenging.

- What is considered "interpretable" may vary depending on the intended audience and domain. Evolving an interpretable GP tree that meets the needs and preferences of a specific user group can be challenging.

- GP is considered an interpretable approach because of its tree-based representation. However, GP trees can apply complex nonlinear transformations to input variables, which can make it difficult to understand the relationship between the inputs and the output.

Interpretability is a persistent challenge, not only within UCARP but also across various computing and machine learning tasks. The need for models that can be easily interpreted is a common thread in different domains, but UCARP presents unique challenges due to its nature of sequential decision-making and the complexity of evolved routing policies. While the trade-off between model interpretability and effectiveness is a common challenge in many AI and ML applications, UCARP introduces its own complexities. One key difference lies in the fact that traditional interpretation techniques used in tasks like classification and regression may

not seamlessly apply to UCARP's tree-based routing policies. Moreover, while smaller models are typically more interpretable, in UCARP, ensuring that simplification does not compromise the effectiveness of evolved routing policies adds an additional layer of complexity. Addressing interpretability in UCARP calls for a tailored approach that carefully considers its specific intricacies, setting it apart from interpretation challenges in other computing and ML tasks.

Many strategies can be utilised to improve the interpretability of the GP-evolved routing policies, such as intrinsic methods and post-hoc methods.

- GP-evolved routing policies contain many redundant materials, as GP tends to generate larger and larger trees without any improvement in effectiveness. GP simplification is potential way. Simplification method can simplify a GP tree to its simpler version without losing effectiveness.

- We can also optimise the interpretability and the effectiveness simultaneously. Intuitively, in addition to the effectiveness, we can take all the factors that may affect the interpretability, such as the number of nodes, number of distinguished features, tree depth, and model complexity, of the evolved routing policies, into account using multi-objective GP. There is a tradeoff between the interpretability and the effectiveness. In this case, we can use multi-objective GP algorithm to evolve a Pareto front of routing policies in which users can choose routing policies based on their preference.

- Post-hoc methods can be also applied to improve the GP-evolved routing policies. Comparing with intrinsic methods which improve interpretability by evolving simpler routing policies. However, in some very complex scenarios, the GP-evolved routing policies need to maintain a certain level of complexity in order to be effective [25,

95]. The post-hoc methods can potentially improve interpretability without sacrificing effectiveness since they are applied to trained models without modifying them.

1.2.3 Limitations of Current Approaches for Evolving Interpretable Routing Policies for UCARP

To the best of our knowledge, there is not much existing work on improving the interpretability of GP-evolved routing policies for UCARP. Existing studies mainly focus on improving interpretability by limiting the GP tree size, which is the most straightforward approach. However, this may also decrease the effectiveness of the GP-evolved routing policies. Therefore, there is a need to develop new algorithms that can improve interpretability without compromising effectiveness.

GP tree simplification is an effective way to improve the interpretability of the GP-evolved model. It is widely acknowledged that simpler models are easier to interpret [13, 39, 147]. GP typically produces trees that contain a large number of redundant components, and it is desirable to remove as many of these redundancies as possible without sacrificing the exploration ability of GP or simplifying the model by searching for a simpler model with the same phenotypic behavior. In the past, manual simplification of the final GP tree has been attempted, but automatic simplification is preferred [105]. Wong and Zhang et al. [205, 218–220] have explored the use of algebraic tree simplification with hashing techniques and applied it to regression and classification problems. Kinzett et al. [107] and Song et al. [177] have proposed numerical tree simplification methods. However, these simplification methods have several limitations. First, they detect redundant subtrees that need to be removed based on genotypic information (e.g., tree structure) rather than phenotypic information (e.g., behavior in decision-making). Thus, they may fail to detect some implicit redundancies. Second, they require predefined parameters, such as predefined sim-

plification rules or a predefined threshold. The final test performance is highly sensitive to the predefined simplification rules or parameters, making it challenging to set them appropriately. Another limitation is that previous studies only consider removing redundant components from the GP tree. However, the primary goal of simplification is to convert a GP tree into its simpler version with the same phenotypic behavior. Therefore, a search technique can be applied to find a simpler version of the original GP tree instead of simply removing redundant components.

Multi-objective approaches can also be applied to improve the interpretability of GP-evolved routing policies for UCARP. These approaches can evolve both the effectiveness and interpretability of the model, and provide a set of routing policies with varying degrees of interpretability to end-users. This way, users can choose a routing policy based on their preference. However, there are two significant challenges when developing multi-objective GP (MOGP) to evolve effective and small routing policies for UCARP. First, GP is more likely to generate small but ineffective individuals than effective but typically large ones. Under the traditional dominance relation, small and ineffective individuals are more likely to be selected as parents and survive into the next generation, leading to the objective selection bias issue. Second, fitness evaluation for routing policies is stochastic due to the training sample rotation. In other words, it is calculated on a different small subset of training instances in each generation. This stochastic fitness evaluation can cause potentially good routing policies to be discarded if they perform poorly on a small subset of training samples. This phenomenon becomes worse in MOGP for UCARP since most potentially good routing policies have large program size.

Post-hoc interpretability methods, such as local explanation and visualization techniques, are considered promising techniques that can help users to understand black-box models, such as DNNs. They can also be utilised to improve the interpretability of the evolved model in GP. However, they cannot be directly applied to our problem since our problem is

a sequential decision-making problem. For example, visualization techniques for GP, which have been utilised in previous studies, are usually used to visualise the evolutionary process of GP so that users can understand why GP evolves the final model [47,138,151,157], or design new GP tree visualization representation, such as top view cone tree, Robertson cone tree, and cushion tree map, to make GP trees easier to observe [65]. It is also important for users to understand the obtained model. Therefore, it is necessary to apply visualization techniques to interpret the evolved model. Visualization explanation techniques in Explainable Artificial Intelligence (XAI) aim at visualizing the model's behavior. It can be used to visualise the high-dimensional distributed representations by rendering 2D visualizations in which nearby data points are likely to appear close together (t-SNE) [118, 132]. It can also be used to identify the relationship between the input and the output for image classification problems [21]. However, most existing visualization explanation techniques are for classification and regression problems [169, 211]. As existing approaches are mainly for classification and regression problems, they cannot be directly adapted to UCARP, which is a sequential decision-making problem. In UCARP, it is important for us to understand the behavior generated by a GP routing policy, for example, why task A is chosen over a set of other tasks in a decision situation. Thus, it is necessary to develop post-hoc approaches, such as local explanations and visualization methods, for GP-evolved routing policies for UCARP to explain the behavior of the GP-evolved routing policies.

1.3 Research Goals

Overall, the goal of this thesis is to enhance the interpretability of GP-evolved routing policies for UCARP through both intrinsic and post-hoc approaches. For intrinsic interpretability, the thesis will introduce new GP simplification methods that simplify routing policies during the evo-

lutionary process. It will also propose multi-objective GP techniques that optimise both the effectiveness and complexity of routing policies. For post-hoc interpretability, the thesis will present novel local and global explanation methods to better elucidate the GP-evolved routing policies.

The specific research objectives of this work are shown as follows.

1. Develop new GP simplification approaches that can improve interpretability by simplifying GP trees.

   GP trees simplified by previous approaches still contain some implicit redundant components. Besides, they only consider removing redundant components from the GP tree. Simplification can be also the process of converting a GP tree to its simple version which has the same phenotypic behavior. In this objective, we aim to propose new GP simplification approaches for UCARP. To achieve this objective, this research objective leads to sub-objectives as follows.

   (a) To develop a new niching simplification scheme based on phenotypic behaviour to automatically detect and remove redundant materials from GP tree during the evolutionary process.

   (b) To develop new archive strategy to compromise the loss of population diversity due to the niching simplification.

   (c) To design new breeding process and new breeding mechanisms to including archive to the breeding process.

   (d) To develop a new GPHH algorithm with Niching (GPHH-N) that incorporates all the above-designed components.

   (e) To verify the effectiveness of GPHH-N, and analyse the routing policies evolved by GPHH-N.

2. Developing new multi-objective Genetic Programming approaches for UCARP.

   The goal is to find out the Pareto Front that can compromise the effectiveness and the interpretability of the evolved routing policies so

that the GP-evolved routing policies meet the needs and preferences of a different user groups. In this case, we use the number of nodes in routing policy (tree size) and effectiveness as two main objectives. Two main challenges are objective selection bias issue and stochastic evaluation issue. Thus, it is necessary to propose new multi-objective GP approaches that can handle this issue. To achieve this goal, this research objective leads to sub-objectives as follows.

(a) To develop a new MOGP algorithm which uses α dominance (αMOGP) to handle the objective selection bias issue so that the multi-objective GP process will not bias to any objective during the evolutionary process.

(b) To develop a new MOGP algorithm that uses archive strategy (TSMOGP-a) to handle the stochastic evaluation issue so that the MOGP algorithm can compensate the loss of potentially good individuals and evolve more complete Pareto front.

(c) To develop a new MOGP algorithm that uses both the α dominance and the archive strategy (αMOGP-a) to handle both issues simultaneously so that the new MOGP algorithm not bias to any objective and evolve a relatively complete Pareto front.

(d) To verify the effectiveness of αMOGP-a by comparing with the current state-of-the-art on a wide range of UCARP instances, and verify the effectiveness of the newly proposed α adaptation scheme and archive strategy.

(e) To interpret the routing policies evolved by αMOGP-a, and understand the behaviors of the GP-evolved routing policies.

3. Develop a new local/global post-hoc explanation method to further improve the interpretability of complex GP-evolved routing policy.

   The post-hoc explainability technique is considered a promising technique that can be applied to model to explain its decision in ma-

chine learning. It aims at communicating understandable information about how an already developed model produces its predictions for any given input [13]. Under a UCARP decision situation, e.g., when a vehicle completes the current task and becomes idle, it uses a routing policy to prioritise/rank the candidate tasks (those that can be served by the vehicle without violating the capacity constraint) and selects the top-ranked task to serve next. Thus, the behaviour of a routing policy can be characterised from two aspects: (1) its task ranking and (2) the selected task. To explain the behaviour of a routing policy in a decision situation, we need to explain why the routing policy ranks the candidate tasks in a particular way, and why it selects a certain task over others in this decision situation. In addition to explaining the above *local* behaviour of routing policy in each decision situation, it is also important to explain its *global* behaviour in all the different decision situations that occurred during the entire service process. However, the global explanation is much more difficult to obtain than the local explanation for each single decision situation [120]. To achieve this objective, we will propose a new Local-Global Ranking Explanation (LGRE) method to tackle the above challenges. This research objective leads to sub-objectives as follows.

(a) To propose two measures, namely *consistency* and *correlation*, to characterise the local behaviour of routing policy in each decision situation so that we can qualify the local explanation of GP-evolved routing policies.

(b) To investigate the feasibility of using linear model as local explanation of GP-evolved routing policies.

(c) To develop new local explanation method for GP-evolved routing policies for UCARP. The new local explanation method uses Particle Swarm Optimisation (PSO) algorithm to optimise the

coefficients of linear models that is applied as local explanations for the GP-evolved routing policy in a decision situation.

(d) To develop new global explanation method based on local explanations for GP-evolved routing policies for UCARP. The global explanation method utilises clustering mechanisms to summarise the local explanations of all the decision situations occurred in the UCARP instance into a global explanation, in the form of a production rule, for example, "If the decision situation shows pattern X, then the local explanation shows pattern Y".

(e) To design new post-hoc method that can produce both local and global explanations for GP-evolved routing policies for UCARP (LGRE). LGRE uses PSO to evolve linear models for the local explanation, and define a new fitness function that incorporates the consistency, correlation and number of used attributes in the linear model. LGRE uses K-means clustering method to generate global explanations.

(f) To verify the accuracy of the proposed LGRE method, in terms of consistency and correlation in different decision situations.

(g) To show the interpretability of the LGRE method by giving case studies of the local and global explanations.

1.4 Major Contributions

This thesis makes the following major contributions, each of which is discussed in each of the contribution Chapters 3 to 5 as appropriate.

1. This thesis proposes a novel GPHH method with a simplification approach using a niching technique (GPHH-N). This is an intrinsic interpretability method which aims to reduce the complexity of GP-evolved routing policies so that they are much easier to be understood by end-users. In the proposed method, a new niching sim-

plification scheme based on phenotypic behavior is proposed to automatically simplify GP-evolved routing policies during the evolutionary process. An external archive is used to compromise the loss of population diversity due to the niching simplification. A multi-source breeding mechanism is proposed to generate offspring from the original population and the representative archive, respectively. Niching-based elitism and parent selection schemes are designed for breeding from the representative archive. GPHH-N was examined and compared with the basic GPHH approach without simplification (GPHH), the basic GPHH approach with algebraic simplification (GPHH-A), and three representative bloat control methods on 57 UCARP instances. The results suggest that GPHH-N can outperform all the compared approaches in terms of test performance. GPHH-N can also outperform GPHH and GPHH-A in terms of tree size and training time. We also analysed the proposed components' effect through a set of controlled experiments. The results showed that all three new components could contribute to evolving smaller and better routing policies. The niching tournament selection and multi-source breeding components are more effective than the niching elitism component. Overall, GPHH-N can obtain better test performance and smaller and potentially more interpretable routing policies than the current state-of-the-art GPHH approach.

Part of this contribution has been published in:

Shaolin Wang, Yi Mei, Mengjie Zhang. "Genetic Programming With Niching for Uncertain Capacitated Arc Routing Problem". IEEE Transactions on Evolutionary Computation, vol. 26, no. 1, pp. 73–87, 2022.

2. This thesis introduces novel MOGP algorithms that can evolve a set of non-dominated routing policies with different tradeoffs between the effectiveness and the size so that end users can select a routing policy based on their preference. This thesis proposes different meth-

ods for issues in MOGP designing for UCARP. There are two main challenges, i.e., the objective selection bias issue and the stochastic evaluation issue, in MOGP designing for UCARP. Firstly, we have designed new α dominance schemes to handle to objective selection bias issue. In this case, the searching process will not bias to either objective. Then, we have also designed new archive strategy to handle the stochastic evaluation issue. The archive can maintain the potentially good individuals evolved during the evolutionary process. This can compensate the loss of good individuals caused by the stochastic evaluation issue. Finally, an intrinsic MOGP interpretability method is designed to improve the interpretability of GP-evolved routing policies. Additionally, it can provide a relatively complete Pareto front of routing policies to end-users to select policies based on their preferences. The final algorithm (αMOGP-a) contains α-dominance and archive strategy. It uses a new α adaptation scheme that adaptively balances effectiveness and size during the evolutionary process. The new archive strategy is used to maintain potentially useful routing policies. Experimental results demonstrate that αMOGP-a outperforms state-of-the-art algorithms for UCARP. The α adaptation scheme and archive are both effective, with both multi-objective indicators, HV and IGD, worsening when either component is removed. The effectiveness of αMOGP-a is proven through routing policy analysis. As αMOGP-a can provide a set of non-dominated solutions, it is possible to gain domain knowledge about which building blocks are more useful for routing policy by observing these solutions in the same set. Overall, αMOGP-a obtains a better Pareto front and potentially more interpretable routing policies than state-of-the-art algorithms

Part of this contribution has been published in:

Shaolin Wang, Yi Mei, Mengjie Zhang. "A Multi-Objective Genetic Programming Algorithm with α dominance and Archive for Uncer-

tain Capacitated Arc Routing Problem". IEEE Transactions on Evolutionary Computation, 2022, DOI:10.1109/TEVC.2022.3195165.

Shaolin Wang, Yi Mei, Mengjie Zhang. "Two-stage multi-objective genetic programming with archive for uncertain capacitated arc routing problem". Genetic and Evolutionary Computation Conference (GECCO), ACM, 2021. pp. 287-295.

Shaolin Wang, Yi Mei, Mengjie Zhang. "A Multi-Objective Genetic Programming Approach with Self-Adaptive α Dominance to Uncertain Capacitated Arc Routing Problem." IEEE Congress on Evolutionary Computation (CEC). IEEE, 2021. pp. 636-643.

Shaolin Wang, Yi Mei, Mengjie Zhang. "A Multi-Objective Genetic Programming Hyper Heuristic Approach to Uncertain Capacitated Arc Routing Problems." IEEE Congress on Evolutionary Computation (CEC). IEEE, 2020. pp. 1-8.

3. This thesis proposes a new Local-Global Ranking Explanation (LGRE) method to enhance the interpretability of GP-evolved routing policies. Existing post-hoc methods are mainly designed for classification and regression problems. They cannot be adapted to sequential decision making problems, such as UCARP. In addition, it is difficult to explain the behaviors of GP-evolved routing policies, for example why the routing policy ranks the candidate tasks in a particular way, why it selects a certain task over others in this decision situation. LGRE is a post-hoc method that can be applied to existing GP-evolved routing policies. The proposed approach consists of a local ranking explanation module and a global explanation module. The local ranking explanation module employs particle swarm optimization to learn an interpretable linear model that accurately explains the local behavior of the routing policy for each decision situation. The global explanation module then uses a clustering technique to summarize the local explanations into a global explanation. Experi-

mental results and case studies on benchmark datasets demonstrate that the proposed method can provide accurate and understandable explanations of the routing policies evolved for UCARP.

Part of this contribution has been published in:

Shaolin Wang, Yi Mei, Mengjie Zhang. "Explaining Genetic Programming Evolved Routing Policies for Uncertain Capacitated Arc Routing Problem". 2023, IEEE Transactions on Evolutionary Computation, DOI:10.1109/TEVC.2023.3238741.

Shaolin Wang, Yi Mei, Mengjie Zhang. "Local ranking explanation for genetic programming evolved routing policies for uncertain capacitated Arc routing problems". Genetic and Evolutionary Computation Conference (GECCO), ACM, 2022. pp. 314-322.

1.5 Organization of the Thesis

The remainder of this thesis is organised as follows. Chapter 2 introduces the essential background and related work. The major contributions of the thesis are presented in Chapters 3 to 5. Chapter 6 concludes the thesis.

Chapter 2 presents the essential background of Genetic Programming (GP), Hyper Heuristic (HH) and Uncertain Capacitated Arc Routing Problems (UCARPs), an overview of the interpretability of GP. It also reviews the related work in UCARP using GPHH as well as interpretability methods.

Chapter 3 proposes a novel GPHH method with a simplification approach using a niching technique (GPHH-N). A new niching simplification scheme, an archive strategy and a multi-source breeding mechanism are designed and illustrated. It then describes the details of the conducted experiments to compare the proposed methods against the state-of-the-art methods on 57 UCARP instances. The results suggest that GPHH-N can outperform all the compared approaches in terms of test performance and

obtain better interpretability.

Chapter 4 proposes novel MOGP algorithms that evolve a set of non-dominated routing policies with different tradeoffs between the effectiveness and the interpretability so that end users can select a routing policy based on their preference.

Chapter 5 proposes a new Local-Global Ranking Explanation (LGRE) method to improve the interpretability of GP-evolved routing policies. The new approach includes a local ranking explanation and a global explanation module. The experimental results and case studies on the benchmark datasets show that the proposed method can obtain accurate and understandable explanations of the routing policies evolved for UCARPs.

Chapter 6 summarises the work in the thesis and draws overall conclusions. Some possible future research directions are also shown in this chapter.

Chapter 2

Literature Review

This chapter introduces the fundamental concepts of Uncertain Capacitated Arc Routing Problem (UCARP), Genetic Programming (GP), Hyper Heuristic (HH), Interpretability, and related works. Section 2.1 introduces the UCARP. Section 2.2 introduces the routing policies and how it generates a solution to UCARP. Section 2.3 introduces the Hyper Heuristic. Section 2.4 introduces the GP. Section 2.5 introduces the existing studies for GPHH. Section 2.6 introduces the Multi-Objective Evolutionary algorithms. Section 2.7 introduces the Particle Swarm Optimisation algorithm. Section 2.8 introduces the interpretability in Machine Learning (ML). Section 2.9 introduces related work.

2.1 Uncertain Capacitated Arc Routing Problems

A UCARP [141] instance can be represented by a connected undirected graph $G(V, E)$, where V and E indicate the sets of vertices and edges, respectively. The vertex $v_0 \in V$ denotes the depot. Each edge $e \in E$ has a positive random deadheading cost $\bar{\varsigma}(e)$, which indicates the cost to traverse the edge. A fleet of vehicles with a same given positive capacity Q are allocated to serve all the *tasks* $T \subseteq E$. Each task $t \in T$ has a positive random demand $\bar{d}(t)$, and a positive serving cost $sc(t)$. The goal is to min-

imise the total cost of serving all the tasks in T. Several constraints have to be satisfied. First, all the routes must start and end at $v_0 \in V$. Second, each task $t \in T$ must be served exactly once, but it can be traversed multiple times. Third, the total demand served by each route (an edge sequence starts and end at the depot) cannot exceed the capacity of the vehicle.

In UCARP, there are a number of random variables, each of which can be sampling differently, which means that each variable can have infinite values. Accordingly, a UCARP instance may contain different samples. In a UCARP instance *sample*, each random variable, i.e., the task demand $\bar{d}(t)$ and deadheading cost $\bar{\varsigma}(e)$, takes a random realised value. However, the realised demand of a task is unknown until it has been served, and the realised deadheading cost of an edge is unknown in advance and is revealed during the traversal over the edge.

The distribution of a task's demand characteristics, such as its mean and standard deviation, is predetermined, with the actual demand value becoming known only after the task is completed, as seen in activities like waste collection, where the precise amount of waste on the street remains uncertain until the street is fully serviced. Similarly, each edge in the graph has a relatively low likelihood of being absent, often due to temporary roadwork or traffic accidents, and the actual presence of an edge remains unknown until the vehicle reaches its initial point, analogous to how a "closed road" sign is visible only when next to the closed road. Additionally, the distribution of the deadheading cost for an edge is anticipated in advance, relying on historical data, with the actual deadheading cost realised only after traversing the edge, influenced by factors such as real-time traffic conditions, road conditions, and driving skills, introducing variability into the cost estimation.

The nature of the uncertainties of the environment will lead to two kinds of failures during the serving process.

- *Route failure*: the actual demand of a task exceeds the vehicle's remaining capacity.

- *Edge failure*: an edge in the route becomes inaccessible (its deadheading cost becomes infinity).

When a route failure occurs, the vehicle will go back to the depot to refill and return to the failed task to complete the remaining service. Fig. 2.1 shows an example where the vehicle is serving the red edge with expected demand of 3 but actual demand of 5. The vehicle's remaining capacity of 4, thus is expected to be able to serve the task. However, the actual demand is larger than the remaining capacity, and a route failure occurs. In this case, the vehicle has to return to the depot to replenish its capacity in the middle of the service. An edge failure occurs when an edge in the route becomes inaccessible (its deadheading cost becomes infinity). The edge failure can be addressed by finding the shortest detour under the current situation using real-time path-finding algorithms (e.g., Dijkstra's algorithm [51]). The goal of solving UCARP is to minimise the expected total cost of the solution over all the possible values of the random variables after addressing the route and edge failures..

A solution to a UCARP instance is represented as $S = (S.M, S.N)$. $S.M = \{S.M^{(1)}, \ldots, S.M^{(j)}\}$ is a set of vertex sequences, where j is the number of vertex sequences (routes). $S.M^{(k)} = (S.m_1^{(k)}, \ldots, S.m_{L_k}^{(k)})$ stands for the kth route, where L_k represents the number of vertices in the kth route. $S.N = \{S.N^{(1)}, \ldots, S.N^{(j)}\}$ is a set of continuous vectors, and $S.N^{(k)} = (S.n_1^{(k)}, \ldots, S.n_{L_k-1}^{(k)})$ ($S.n_i^{(k)} \in [0, 1]$) is the fraction of demand served along the route $S.M^{(k)}$. The problem can be formulated as follows.

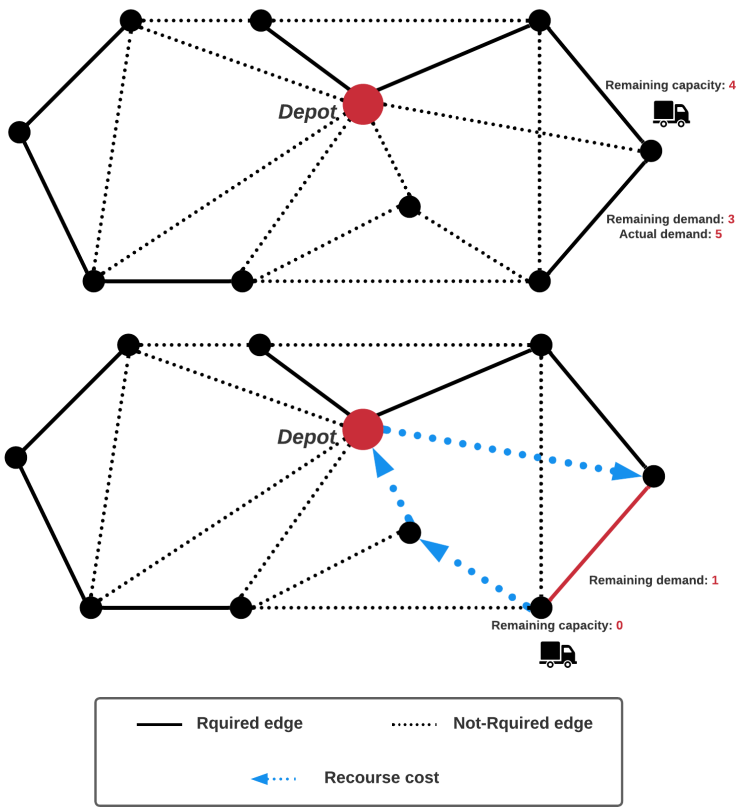


Figure 2.1: An example of a route failure.

$$\min \ \mathrm{E}_{\xi \in \Xi}[C(S_\xi)], \tag{2.1}$$

$$s.t. \ \sum_{i=1}^{L_k-1} d_\xi \left(S_\xi.m_i^{(k)}, S_\xi.m_{i+1}^{(k)}\right) \times S_\xi.n_i^{(k)} \leq Q, \forall k = 1, \ldots, j, \tag{2.2}$$

$$\left(S_\xi.m_i^{(k)}, S_\xi.m_{i+1}^{(k)}\right) \in E, \tag{2.3}$$

$$\left(S_\xi.n_i^{(k)}\right) \in [0, 1], \tag{2.4}$$

$$S_\xi.m_1^{(k)} = S_\xi.m_{L_k}^{(k)} = v_0, \ \ \forall k = 1, 2, \ldots, j, \tag{2.5}$$

$$\sum_{k=1}^{j} \sum_{i=1}^{L_k-1} S_\xi.n_i^{(k)} \times S_\xi.z_i^{(k)}(e) = 1, \ \forall e : d_\xi(e) > 0, \tag{2.6}$$

$$\sum_{k=1}^{j} \sum_{i=1}^{L_k-1} S_\xi.n_i^{(k)} \times (1 - S_\xi.z_i^{(k)}(e)) = 0, \ \forall e : d_\xi(e) = 0, \tag{2.7}$$

where Eq. (2.1) is the objective function. It is used to minimise the E, which is the expected total cost $C(S_\xi)$ of the solution S_ξ over all samples $\xi \in \Xi$. The total cost of a solution on one sample ξ is calculated by Eq. (2.8), where $\varsigma_\xi(u, v)$ is the realised deadheading cost between the node u to v in ξ. The $sc(t)$ refers to the serving cost of a task and $\varsigma_\xi(t))$ refers to deadheading cost of the same task. For any sample ξ, a feasible solution S_ξ is generated by a routing policy which gradually builds the solution on-the-fly. Eq. (2.2) is the capacity constraint, and d_ξ refers to the realised task demand in ξ. Eq. (2.3) and Eq. (2.4) are the domain constraints of $S_\xi.M$ and $S_\xi.N$. Eq. (2.5) indicates that all the routes start and end at the depot. Eq. (2.6) means that each task is served exactly once (the total demand fraction served by all vehicles is 1). $S_\xi.z_i^{(k)}(e)$ equals 1 if $\left(S_\xi.m_i^{(k)}, S_\xi.m_{i+1}^{(k)}\right) = e$, and 0 otherwise. $S_\xi.z_i^{(k)}(e)$ indicates whether the edge is a task or not. Eq. (2.7) ensures that any non-required edge is not served at all, i.e., its service fraction is zero everywhere in the solution.

$$C(S_\xi) = \sum_{k=1}^{j} \sum_{i=1}^{L_k-1} \left(\varsigma_\xi(S_\xi.m_i^{(k)}, S_\xi.m_{i+1}^{(k)})\right) + \sum_{t \in T} (sc(t) - \varsigma_\xi(t)) \tag{2.8}$$

The static CARP is a special case of UCARP, where all the variables are known in advance. Route failure is the main extra challenge of UCARP over the static CARP, and it can lead to large extra recourse cost.

2.1.1 UCARP Datasets

2.1.2 Dataset

In this thesis, we use the *Ugdb* and *Uval* datasets, which are commonly used in UCARP literature [122, 123, 133, 143]. They are extended from *gdb* and *val* which are well known static CARP datasets. The instances in the *gdb* dataset are mostly small. They contain at most 55 tasks that need to be served. The *val* dataset contains instances with the number of tasks ranging from 34 to 97. Table 2.1 shows details of two datasets.

Table 2.1: Details of Ugdb and Uval datasets.

Ugdb				Uval			
index	task	non-task	vehicle	index	task	non-task	vehicle
Ugdb1	22	0	5	Uval1A	39	0	2
Ugdb2	26	0	6	Uval1B	39	0	3
Ugdb3	22	0	5	Uval1C	39	0	8
Ugdb4	19	0	4	Uval2A	34	0	2
Ugdb5	26	0	6	Uval2B	34	0	3
Ugdb6	22	0	5	Uval2C	34	0	8
Ugdb7	22	0	5	Uval3A	35	0	2
Ugdb8	46	0	10	Uval3B	35	0	3
Ugdb9	51	0	10	Uval3C	35	0	7
Ugdb10	25	0	4	Uval4A	69	0	3
Ugdb11	45	0	5	Uval4B	69	0	4
Ugdb12	23	0	7	Uval4C	69	0	5
Ugdb13	28	0	6	Uval4D	69	0	9
Ugdb14	21	0	5	Uval5A	65	0	3
Ugdb15	21	0	4	Uval5B	65	0	4
Ugdb16	28	0	5	Uval5C	65	0	5
Ugdb17	28	0	5	Uval5D	65	0	9
Ugdb18	36	0	5	Uval6A	50	0	3
Ugdb19	11	0	3	Uval6B	50	0	4
Ugdb20	22	0	4	Uval6C	50	0	10
Ugdb21	33	0	6	Uval7A	66	0	3
Ugdb22	44	0	8	Uval7B	66	0	4
Ugdb23	55	0	10	Uval7C	66	0	9
				Uval8A	63	0	3
				Uval8B	63	0	4
				Uval8C	63	0	9
				Uval9A	92	0	3
				Uval9B	92	0	4
				Uval9C	92	0	5
				Uval9D	92	0	10
				Uval10A	97	0	3
				Uval10B	97	0	4
				Uval10C	97	0	5
				Uval10D	97	0	10

For each UCARP instance, the task demand $\bar{d}(t)$ and traversal cost $\bar{\varsigma}(e)$ are random variables. They are transformed from the original task demand $d(t)$ and traversal cost $\varsigma(e)$ from the static CARP instance. The random variables are assumed to follow the truncated normal distribution [122, 133, 143].

$$\bar{d}(t) \sim \mathcal{N}(d(t), \frac{d(t)}{5}), \bar{\varsigma}(e) \sim \mathcal{N}(\varsigma(e), \frac{\varsigma}{5}). \tag{2.9}$$

Any negative sampled task demand is set to 0, and any negative sampled traversal cost is set to ∞, which means that the arc becomes inaccessible.

2.2 Routing Policy

Routing policy is a kind of constructive heuristic designing for routing problems. In general, a heuristic is a mental shortcut that allows people to solve problems and make judgments quickly and efficiently. In other words, any technique or approach, which can be used for solving a problem by employing practical methods that may not be able to guarantee a perfect or optimal solution but instead, can achieve solutions that are sufficiently good in context of the problem at hand, is a heuristic [59, 202]. The concept of heuristics has its origin from the outside of Computer Science (CS) that human judgment is subject to cognitive limitations when they strive to make rational choices [163]. Tversky and Kahneman [188] presented their study of heuristics in human decision-making. Heuristics can be useful intelligent search strategies for computer problem [155]. A Routing policy for UCARP constructs a solution step by step. Liu et al. list five routing policies for UCARP [122]. They are described as follow:

- *H1*: selects the task with the *maximal* distance to the depot;
- *H2*: selects the task with the *minimal* distance to the depot;
- *H3*: selects the task with the *maximal* demand to the depot;

- *H4*: selects the task with the *minimal* demand to the depot;
- *H5*: uses *H1* if capacity ≥ 0.5, and *H2* otherwise.

Suppose there three tasks A,B and C that need to be served and their distances from the depot are 2, 3 and 1, respectively. If we use *H1* as our routing policy to construct the solution. Then *H1* will choose to serve task B as the first step because it is the furthest away from depot, then *H1* will choose to serve task A, and finally task C will be served. Finally, the solution will be B $\rightarrow$ A $\rightarrow$ C.

2.3 Hyper Heuristic

Heuristics have achieved great success in solving real-world computational search problems. However, heuristics are usually problem specific, and are not easy to apply to a new problem or new instances of the same problem. Besides, heuristics are usually expensive to develop because they require substantial expertise and knowledge in problem domain. To overcome these challenges, hyper heuristic is devised. The main goal of hyper heuristic is to automate the development of heuristic methods to solve hard computational search problems [27].

Hyper heuristic is becoming a popular method to design heuristics automatically. The ideas behind hyper heuristics are not new. They have been used in many fields, such as Operational Research, Computer Science and Artificial Intelligence, since the 1960s. However, the term hyper heuristics was first used in a peer-reviewed conference paper in [44]. In [43, 45, 46, 74], the authors further developed the ideas of [44] and applied it to combinatorial optimisation problems. Hyper heuristic is a high-level heuristic that automates the process of selecting or generating low-level heuristics that can solve hard computational search problems [31]. Hyper heuristic can produce a generally applicable algorithm that can solve problems in some given scenarios rather than find out solutions directly.

The primary goal of hyper heuristic is to develop generic methods, which can get good enough solutions, based on a set of easy-to-implement low-level heuristics. So the process of hyper heuristic is that when a particular problem instance or class of instances is given, we apply some low-level heuristics to a learning algorithm, such as genetic programming. Then the learning algorithm will automatically select or generate heuristics based on the low-level heuristics. Finally, a suitable heuristic that can solve the particular problem will be selected or generated.

2.3.1 Taxonomy of Hyper Heuristic

Hyper heuristic approaches can be classified into two categories [28]: heuristic selection and heuristic generation.

- Heuristic selection: the idea of heuristic selection is to select the most suitable heuristic for a given particular problem from pre-existing heuristics that are specified to the given problem. The process of building solution (sequence of heuristic) is incremental. The solution pool starts from empty. The most suitable heuristic for the current problem state is selected and use it to make decision for the current state. The selected heuristic will be added to the solution pool. The process will not stop until a complete solution is built up [27].

- Heuristic generation: the idea of heuristic generation is to construct new heuristics for a given particular problem. Comparing with the heuristic selection which use pre-exist heuristics, heuristic generation constructs new heuristics by combining various small components such as general statistics or operators used in existing heuristics.

2.3.2 Heuristic, Metaheuristic and Hyper Heuristic

A heuristic is a technique that finds a solution to hard computational search problems at reasonable computational cost based on limited knowledge or incomplete information but does not guarantee an optimal solution. Metaheuristics are techniques that can be applied to a broad range of hard computational search problems. Metaheuristics find optimal or near-optimal solutions and its searching space is solutions [28]. The main difference between heuristics and meta-heuristics is that heuristics construct a solution from scratch and usually stops when a complete solution is constructed. But the quality of solution is not guaranteed. There are some typical heuristics, such as greedy heuristic, A* search algorithm. GA and simulated annealing are two popular meta-heuristics. We can also use the Genetic Algorithm (GA) to find a solution. After we set up some parameters initially, GA will automatically generate many solutions randomly and evaluate each solution, and return a good enough solution. However, it is important to consider that in dynamically changing or stochastic environments, the computational cost of regenerating a new solution using Genetic Algorithms (GA) can become significant. GA involves a trial-and-error approach where multiple solutions are generated, evaluated, and evolved over generations to converge towards an optimal or near-optimal solution. This iterative process, although effective in many scenarios, can be time-consuming, especially when the problem space is constantly changing or inherently uncertain due to stochasticity. In such dynamic and unpredictable settings, the efficiency of heuristic algorithms, known for their ability to swiftly find optimal solutions, becomes particularly advantageous. Hyper heuristics [27] is a high-level heuristic that automates the process of selecting or generating low-level heuristics that can solve hard computational search problems [31]. For example, if we want to find a path with the smallest cost from City A to City B. We can utilise A* search algorithm as a heuristic to construct a route from A to B. However, A* search might not be the most suitable algorithm for this scenario to

find a satisfied solution. Hyper heuristic will select or construct heuristic that are suitable for this specific scenario. Then, the generated or selected heuristic can be applied to the problem and get a solution. Note that, both heuristic and metaheuristic search the solution space directly, while hyper heuristic searches the heuristic space to select or generate heuristic to construct solution.

2.4 Genetic Programming

Genetic Programming (GP) is evolutionary computing (EC) technology that automatically solves problems without requiring domain knowledge of solutions. The main aim of GP is to make computers solve problems automatically. In GP, a population is a set of random designed computer programs. We get the final solution by evolving population, generation by generation. For each new generation, GP stochastically transforms programs to hopefully better programs so that they can solve the target problem. GP is very successful in developing new and unexpected ways to solve problems [161]. Fig. 2.2 shows a simple diagram for GP process. Usually, GP can be broken down into the following steps [161]:

- Generate a population of random programs (individuals) firstly;
- Run each program and evaluate its fitness;
- If the quality is good enough, the best individual in the population is returned as the solution; otherwise, select good programs to produce new generation based on the previous generation using genetic operators, such as crossover, mutation and reproduction;
- Repeat above steps until getting good enough solution.

GP has some advantages over other machine learning algorithms in producing heuristics. First, heuristics usually behave in the form of programs

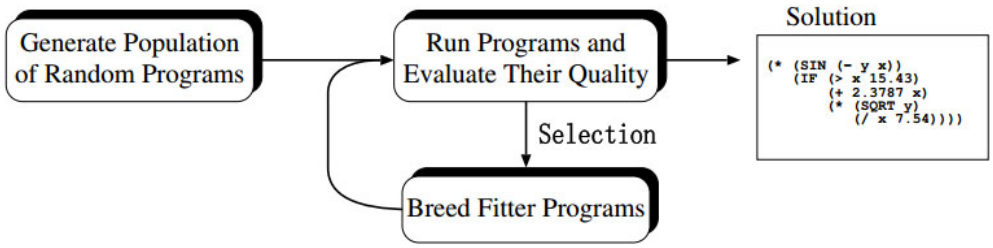


Figure 2.2: A Simple Diagram of GP process [161].

or algorithms. GP can produce programs or algorithms automatically, so GP is an excellent way to produce heuristics. The terminal set can choose from good features of the problem domain. Features from manually designed heuristics can be easily used in GP. A function set, relevant to the problem domain can be determined without too much difficulty [31]. Second, the length of programs is flexible as we do not have the domain knowledge to predefine an optimal length of programs in most cases. In this case, GP can produce heuristics for different scenarios.

2.4.1 Representation

GP is usually expressed as a syntax tree which is different from genetic algorithms which are usually expressed as a one-dimension string. The syntax tree is built up by combining leaf nodes and internal nodes. Leaf nodes, which can be called terminals, are features in the terminal set or constant number. Internal nodes, which can be called functions, are arithmetic operations, such as addition, subtraction, multiplication and protected division, in function set. Typically, the terminal set and function set are determined based on the domain knowledge about a particular problem. There are two constraints for GP trees: closure and sufficiency. Closure means that every function can accept any value or data type that can be returned by any function in the function set or terminal in the terminal set [161]. Sufficiency means that the combination of functions and

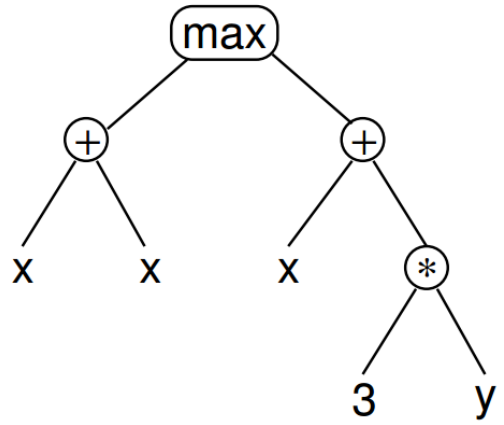


Figure 2.3: An example of tree-based GP program [161].

terminals can produce a solution [161]. Fig. 2.3 is an example of tree-based GP program, which is calculating the formula $max(2x, x + 3 * y)$.

2.4.2 Population Initialisation

Usually, population initialisation is the first step of a GP process. A population is a set of programs or individuals (GP trees). There are three common ways to initialise the population. They are *full*, *grow* and *ramped-half-and-half* [161]. The full method grows all trees to the max-depth limit and then assigns terminals to leaf nodes. The grow method grows all trees by randomly assign function or terminal to each node, if a function is assigned to a node, the node will grow deeper; otherwise, it will stop growing. The ramped-half-and-half method grows a half of the population by the full method and the other half by grow method. There are two limits in tree generation. They are min-depth and max-depth. The min-depth refers to the minimum depth of a tree. The max-depth refers to the maximum depth of a tree. The ramped-half-and-half method is the most popular way to generate population.

2.4.3 Evaluation

In an evolutionary algorithm, it is important to measure how fit or capable each individual is for solving the problem. The fitness function is used to evaluate each individual in a population and determine the quality of the individual and the generation. The definition of the fitness value for an individual is dependent on the problem, it should be a good indicator of the performance of the program for solving the problems. In other words, different fitness measurements can be utilised for different problem. For example, for solving classification/regression problems with GP, each individual is a classifier/regression model and the fitness can be defined as the accuracy of the predictions/loss between the model output and the true output over the training dataset. For solving UCARP, the fitness is the total-cost (time or route length) of the solution generated by GP-evolved routing policy from simulation. Fitness can help to decide which individual can survive to next generation. When reaching the stopping criteria, the individual with the best fitness will be returned as the solution [161].

2.4.4 Selection

During the GP evolutionary process, individuals are selected from the population to generate new individuals. Typically, individuals with better fitness are more likely to survive to the next generation and more likely to contribute to the gene pool in future generations. In the roulette wheel method, a probability distribution is constructed based on fitness value of population member so that individuals with better fitness have a better and higher chance of being selected and then, individuals are selected based on that distribution. Tournament selection is most often used to select individuals in GP [161]. In tournament selection, k (the tournament size) individuals are randomly sampled from the population, and the individual with the best fitness among the sampled individuals is selected to be the parent to genrate offspring.

2.4.5 Genetic Operators

The main goal of genetic operators is to generating a population with better quality by mixing and inheriting genetic materials from old population. To this end, four main genetic operators, i.e. mutation, crossover, reproduction and elitism [161] are applied to GP population with a user-defined probability.

Typical crossover operator is commonly used to generate new individuals based on two randomly selected individuals as parents. Subtree crossover is one of the earliest and most frequently used crossover methods methods for GP [109]. In Subtree crossover, firstly, one node (crossover point) from each parent is randomly selected, and then the subtrees located at the selected nodes are swapped between two parents, finally, two new offsprings are generated. Note that, the majority of nodes in a GP tree are terminals, it is recommended by Koza et al. [109] to discriminate between the terminals and functions in selecting crossover points by selecting functions and terminals with a probability of 90% and 10% respectively.

Unlike crossover, typical mutation operator generates only one new individual based on a single individual. Subtree mutation is commonly used. In the Subtree mutation, one individual is randomly selected, and then a random node (mutation point) in the individual is selected, finally, a new individual is generated by replacing the subtree at the mutation point with a randomly generated new subtree.

Reproduction is a mechanism that allows GP to maintain good candidate solutions found so far and preserve them in the population. A typical reproduction operator [109] simply use tournament selection operator to select good individual from the population and copies it into the population of the next generation without modifying it.

Elitism is quite similar with reproduction. In contrast to reproduction which maintains good individuals, elitism maintains only the best individuals, for example, the best 10 individuals in the population, and copies

them into the population of the next generation without modifying them.

2.5 Genetic Programming Hyper Heuristic

Hyper heuristics aim to automatically select or generate heuristics to solve hard computational search problems [28]. Typically, the search space of hyper heuristics is a space of heuristics rather than solutions, the aim is to increase the robustness of search methods [31], especially for problem instance with large solution space, it might be hard to find a satisfied solution in a huge solution space, but it is likely to find a good heuristic in the fixed heuristic space. It has been widely applied to many problem domains, such as examination timetabling [17], bin packing [167], sports scheduling [38], online container terminal truck dispatching problem [40], job shop scheduling [210] and vehicle routing [192]. GP is considered as a popular method for automatic design of hyper heuristics [150]. Burke et al. [31] summarise the techniques of applying GP as a hyper heuristic to generate new heuristics. They also survey previous work attempting to generate heuristics using genetic programming. There are a number of advantages using GP as hyper heuristic. One of the advantages is that the variable-length encoding is flexible in GP. This can be useful if the best length of the encoding for heuristic representation is not known for a given problem domain. GP can evolve executable programs which can be easily adopted as heuristics. Domain knowledge can be easily incorporated into the fundamental components of the system. For example, humans can easily identify the good features of the problem domain and utilise them as the terminal and function set of a GP approach.

GP has been successfully applied to generate new constructive heuristics in many problem domains and achieve comparable or even better performance than human designed heuristics. Bader-El-Den and Poli [16] utilised GP to quickly generate 'disposable' heuristics to solve the satisfiability problem and the result showed that the generated heuristics per-

formed comparable with the heuristics designed by human. Note that, only a limited search space of heuristics was covered in this work. Burke et al. [30] applied GP evolved heuristics to one dimensional bin packing problems and showed that GP evolved heuristics can outperform the human designed "bestfit" heuristic. In addition, Allen at al. [4] extended the work [30] to three dimensional bin packing problem and achieved competitive results with human designed heuristics. Besides, Burke et al. [29] also applied GP based hyper heuristic on two dimensional strip packing problems. Furthermore, GP based hyper heuristic approaches can also handle travelling salesman problems [99], satisfiability testing problems [66],web service allocation problems [179] and job shop problems [68, 85, 207, 208, 217].

GP is the most commonly used approach that is utilised as a hyper heuristic for UCARP [122, 133, 143]. Simulation is a key element, which is required for evaluating the heuristic functions (routing policies) evolved by GP, in GPHH approach. A simulation for UCARP takes a routing policy and a UCARP instance as input and returns a feasible solution for the instance. The simulation employs a constructive heuristic-like approach. The algorithm starts with an empty route and build up a route step by step, it adds tasks to the end of the route in a sequential approach. As a result, this simulation will not open any new route as long as the current route is open. The algorithm may return back to depot to refill its capacity when the capacity is empty.

The simulation firstly initialises the vehicle's route which starts from the depot. Typically, all tasks are unserved and the capacity of the vehicle is full. Then, the routing policy is utilised to select the next task that needs to be served. To select the next task, all unassigned tasks are considered first and then a filter method is applied to select a subset of all unassigned tasks. If the subset is empty, the vehicle has to return to the depot and refill, close the current route, and open a new route. Otherwise, the routing policy is applied to the unassigned tasks and the task with the best pri-

ority value is selected to be served next. Failures caused by the dynamic nature of the problem are also considered in a simulation. A route failure occurs when the actual demand exceed available capacity of the vehicle. In case of a route failure, the vehicle has to return to the depot to refill. A edge failure occurs when an edge suddenly becomes inaccessible. A detour method is applied to handle edge failures. If no failure occurs, the task is served and removed from the list of unserved tasks. This process will not finish until all tasks are served. When the simulation is finished, a collection of routes are created that serve all the tasks in the UCARP instance.

We can briefly describe a GPHH approach to solving a UCARP instance as:

- Step 1: Initialisation: Create an initial GP population (a population of routing policies) randomly.
- Step 2: Evaluation: Each routing policy is evaluated by applying the simulation to the samples from the input UCARP instance in the training dataset and the fitness value is the average over the total cost of serving tasks.
- Step 3: If the stopping criteria are not met, go to Step 4. Otherwise, return the routing policy that has the best fitness value.
- Step 4: Evolution: Select good routing policies to evolve a new population using the genetic operators of crossover, mutation, and reproduction. Then, go to Step 2.

Comparing with manually designed heuristic and metaheuristic approaches, which can be time consuming and require high-level domain knowledge, GPHH does not require extensive domain knowledge and need to prepare solutions beforehand. It has been shown to be the state-of-the-art for dealing with this problem [8, 9, 122, 124, 133, 134, 143, 203]. However, a drawback of this method is that the evolved routing policies

are hard to be understood by human users. The users need to be able to understand the evolved routing policies to feel confident to use them. In addition, if we can interpret the evolved routing policies, we can determine what contribution each feature makes in the decision-making process, and understand the meaning of each routing policy so that we can reuse the knowledge to other cases. Therefore, it is important to evolve both effective and interpretable routing policies.

When applying to UCARP, each individual of GPHH is a routing policy, which is essentially a priority function. Given a routing policy and UCARP instance, the solution is constructed as follows.

- Step 1: all the vehicles are at the depot, with idle time 0 and their routes are empty. All the tasks are unserved.
- Step 2: find the *vehicle with the earliest idle time* (ties are broken by selecting task with smaller index), and its *candidate tasks*, i.e., the unserved tasks with expected demand less than its remaining capacity.
- Step 3: if no candidate task is found, send the vehicle back to the depot to replenish. Otherwise, use the routing policy to calculate the *priority* of each candidate task, and select the best-priority task to serve next. Set the idle time of the vehicle to the time it completes the service. Fig. 2.4 shows an example of using the CFH+CTD routing policy to select from 3 candidate tasks.
- Step 4: if all the tasks are served, send all the vehicles back to the depot and stop. Otherwise, go to Step 2.

As the adoption of machine learning for solving combinatorial optimization problems gains momentum, it is essential to acknowledge the broader landscape of ML techniques in this domain. Deep Learning and Deep Reinforcement Learning (DRL) have indeed been applied with remarkable success to evolve heuristics for various combinatorial optimization problems. However, it is pertinent to address why these specific tech-

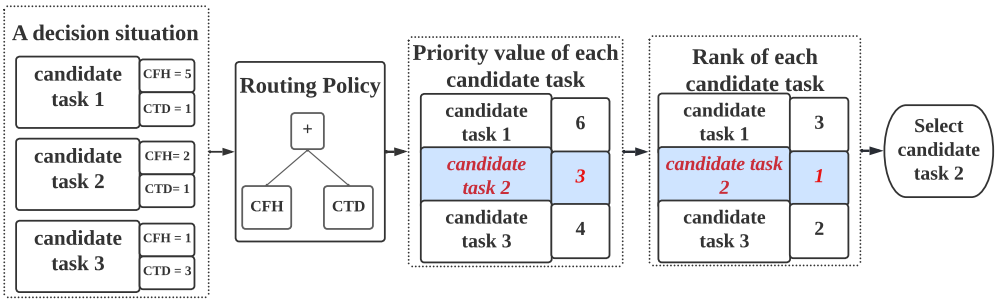


Figure 2.4: An example of decision making process using a simple routing policy in a decision situation with three candidate tasks.

niques were not applied to the investigated Uncertain Capacitated Arc Routing Problem (UCARP). UCARP represents a unique challenge characterised by sequential decision-making and inherent uncertainty. While DRL has shown promise in evolving heuristics for certain combinatorial optimization problems, its application to UCARP is nuanced. The complexity of UCARP's routing policies, with numerous factors influencing decisions, poses challenges in formulating an effective reward structure for DRL. Additionally, the dynamic nature of UCARP, where uncertainties evolve over time, adds an extra layer of complexity not readily amenable to DRL's traditional reinforcement learning paradigms. As such, the choice of genetic programming as the primary optimization technique for UCARP reflects a careful consideration of the problem's intricacies and the suitability of the selected approach to address its specific challenges.

2.5.1 Common Parameter Settings of GPHH

Some common parameter settings of GPHH are shown in Table 2.2. The individuals are initialised with the ramped-half-and-half method with a minimum depth of 2 and a maximum depth of 6. When generating a GP individual, the rates to the terminal and non-terminal to use are 10% and 90%, respectively. The maximal of the depth of a GP individual is 8. To

Table 2.2: The common parameter settings of GPHH.

Parameter	Value
Population initialisation method	ramped-half-and-half
Initial min/max tree depth	2/6
Terminal/function selection rate	10%/90%
Max tree depth	8
Population size	1000
Number of elites	10
Selection method	Tournament selection
Tournament size	7
Crossover/Mutation ratio	85%/15%
Number of generations	101

maintain the quality of the obtained individuals in the evolutionary process of GP, the best 10 individuals (i.e., elites) from the last generation are moved to the next generation directly. The rest individuals for the next generation are produced by crossover and mutation (i.e., genetic operators) with a rate of 85% and 15%, respectively. Tournament selection with size 7 is used to select the parent(s) for the genetic operators. The algorithm is stopped after 101 generations.

The function set is set to $\{+, -, \times, /, \min, \max\}$, where the "/" (protected division) operator returns 1 if divided by 0. The terminals of GP serve as features of the problem to capture sufficient information about the problem. The terminal set of GP in this thesis consists of a number of basic features in UCARP [122, 124, 133, 197], which is shown in Table 2.3.

The Evolutionary Computation Java (ECJ) package [131] is utilised to implement all the algorithms.

Table 2.3: The terminal set in the experiments.

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

2.5.2 Individual Representation

In this thesis, each routing policy is represented as a priority function. The priority function is a combination of the state features, such as the cost from the current location to the candidate task (CFH) and the cost from the candidate task to the depot (CTD). For example, if the priority function is "CFH + CTD", the priority function tends to select tasks that are closer to the current location and also closer to the depot. The routing policy works as a decision-maker during the solution construction process. Typically, each vehicle can only serve one task at a time. The routing policy will be applied to each unserved task to determine each unserved task's priority once a vehicle completes its current task and is ready to serve the next task. The task with the best priority value will be served next. The decision process finishes when all the tasks have been served and the routes returned as the solution constructed by the routing policy.

Algorithm 1: The fitness evaluation process in GPHH.

```
Input: Training set S_train, An unevaluated population pop
Output: An evaluated population pop
Randomly sample a training subset S' ⊆ S_train;
// Evaluate pop using S'
for each routing policy rp ∈ pop do
    // Evaluate rp
    for each training sample s ∈ S' do
        while tasks in queue is not empty do
            if vehicle is ready then
                Select which task to serve using rp;
            end
        end
        Get the total cost of the generated solution, i.e. tc(rp, s);
    end
    Calculate fit(rp) using Eq. (3.1);
end
return evaluated population pop;
```

2.5.3 Fitness Evaluation

Given a routing policy rp, the fitness is defined based on the average quality (i.e. total cost) of the routes that it generates. Specifically,

$$\text{fit}(rp) = \frac{1}{|\mathcal{S}'|} \sum_{s \in \mathcal{S}'} tc(rp, s), \tag{2.10}$$

where $\mathcal{S}'$ is a set of instance samples, $|\mathcal{S}'|$ is the number of instance samples. $tc(rp, s)$ stands for the total cost of the solution obtained by rp on instance sample s. The solution is constructed based on a simulation process that is commonly used in UCARP literature [123, 197]. Algorithm 1 shows the fitness evaluation in GPHH.

One can see that each routing policy is applied as a priority (heuristic) function. When a vehicle is ready (at a decision-making point), a routing policy is applied to determine which candidate task to serve next. The quality of each routing policy can be evaluated by applying it to a set of

training samples $\mathcal{S}'$. A solution can be constructed using the routing policy on a training sample $s \in \mathcal{S}'$.

We randomly re-sample a subset of training samples for the fitness evaluation. Such sample rotation has been commonly used in other studies [80, 122] to improve the generalisation of the evolved heuristics.

2.6 Multi-Objective Evolutionary Algorithms

Multi-Objective Evolutionary Algorithm (MOEA) is a powerful technique used to solve complex optimization problems with multiple conflicting objectives. Unlike traditional single-objective optimization, where there is only one objective to optimise, MOEA considers multiple objectives simultaneously, making it suitable for solving real-world problems where there are often multiple conflicting goals that must be balanced.

MOEA is a type of evolutionary algorithm that uses the principles of natural selection and genetic variation to search for optimal solutions in a multi-dimensional search space. The basic idea behind MOEA is to generate a population of candidate solutions, called individuals, and evaluate their fitness based on multiple objectives. The individuals with the best fitness are then selected for reproduction, where their genetic information is combined and mutated to generate new offspring. The process continues iteratively, with the hope that the population will converge to a set of optimal solutions, called the Pareto front.

The Pareto front is a set of solutions where no solution can be improved in one objective without worsening at least one other objective. In other words, the Pareto front represents the trade-off between different objectives and provides a set of optimal solutions that are not dominated by any other solution in the search space.

Non-dominated sorting is a powerful technique used in multi-objective optimization to identify solutions that are Pareto-optimal, meaning that they are not dominated by any other solution in the search space. The

technique was introduced by Deb et al. [50] and has since become a fundamental component of many state-of-the-art multi-objective optimization algorithms, such as NSGA-II [50], SPEA2 [224] and MOEA/D [221].

The basic idea behind non-dominated sorting is to group solutions into levels or fronts based on their Pareto-optimal status. Specifically, a solution is said to dominate another solution if it is better in at least one objective and not worse in any other objectives. A solution is non-dominated if there is no other solution that dominates it. The first level of non-dominated solutions contains all the solutions that are non-dominated, while the second level contains solutions that are dominated only by the solutions in the first level, and so on.

To perform non-dominated sorting, a set of candidate solutions is first evaluated on the multiple objectives of the optimization problem. The solutions are then ranked into levels based on their non-dominated status, with non-dominated solutions being assigned to the first level. Once the non-dominated solutions are identified, the sorting process is repeated on the remaining solutions, but the non-dominated solutions from the previous level are removed from consideration. This process is repeated until all solutions have been sorted into levels.

Non-dominated sorting is a useful technique in multi-objective optimization because it provides a way to identify solutions that are not dominated by any other solution in the search space. These solutions are considered to be Pareto-optimal and represent the trade-offs between different objectives. By identifying the Pareto-optimal solutions, a decision-maker can make informed decisions on which solution to choose based on their preferences and constraints.

There are several variations of MOEA, each with its own strengths and weaknesses. One popular approach is the Non-dominated Sorting Genetic Algorithm (NSGA-II) [50], which uses a fast and efficient sorting algorithm to maintain a diverse set of solutions along the Pareto front. Another popular approach is the Strength Pareto Evolutionary Algorithm 2

(SPEA2) [224], which uses a fitness sharing scheme to encourage diversity and prevent premature convergence.

MOEA has been successfully applied to a wide range of real-world problems, including engineering design, finance, environmental management, and many others. However, MOEA is not a silver bullet and requires careful tuning of its parameters and selection of appropriate objective functions and performance metrics to achieve good results. Additionally, MOEA may be computationally expensive, especially when dealing with high-dimensional search spaces, and may require significant computational resources to achieve good results.

2.7 Particle Swarm Optimisation

Particle Swarm Optimization (PSO) is a population-based metaheuristic algorithm that has gained significant attention in the field of optimization in recent years. The algorithm is inspired by the social behavior of birds flocking, where individuals interact with each other to achieve a common goal [100]. PSO operates by maintaining a population of particles, each representing a candidate solution in the search space.

During the optimization process, each particle evaluates its fitness based on an objective function, and updates its position p and velocity V based on its own best known position and the best known position of its neighbors in the search space. The position and velocity updates are guided by a set of user-defined parameters. These parameters control the exploration and exploitation capabilities of the algorithm and must be carefully tuned to achieve good performance. Inertia weight w refers to the parameter that controls the balance between exploration and exploitation during the search process. A high inertia weight value allows for more exploration, while a low value enables more exploitation. Cognitive factor c refers to the parameter that controls the influence of the particle's best known position on its movement. A high cognitive factor value favors exploitation,

while a low value favors exploration.

PSO has several advantages over traditional optimization techniques, including its simplicity and scalability. The algorithm can efficiently search for solutions in high-dimensional search spaces and has been successfully applied in various domains, including engineering, finance, and machine learning.

The implementation of PSO is shown as below:

- Step 1: Initialise a population of particles with random positions and velocities in the search space.
- Step 2: For each particle
  - Evaluate the particle x_i using a predefined fitness function.
  - Compare the particle's fitness with its best fitness. If current value is better than its best fitness then set new fitness as best fitness, and $pbest$ equal to the current location.
  - Identify the $gbest$ which is the best position that explored by all the particles in the swarm.
  - Update the velocity and position of the particle according to the following equation:

$$V_i(t+1) = wV_t + c_1 r_1 (p_{best} - p_{x_i}(t)) + c_2 r_2 (g_{best} - p_{x_i}(t)) \quad (2.11)$$

$$p_{x_i}(t+1) = p_{x_i}(t) + V_i(t+1) \quad (2.12)$$

    where r_1 and r_2 are random numbers between 0 and 1. $V_i(t+1)$ is the updated velocity. w is the inertia weight. r_1 and r_2 are cognitive factors. $p_{x_i}(t+1)$ is the updated position of the particle x_i.

- Step 3: If a stopping criterion is met, then stop, otherwise go back to Step 2.

2.8 Machine Learning and Interpretability

Machine Learning (ML) is increasingly popular in the Artificial Intelligence (AI) field. The aim of ML is to build computer programs that can learn to perform complex tasks and can be improved automatically through experience [53]. It can be utilised to many real-world problems for which creating a detailed design can be quite hard despite a clear specification [164]. Typically, ML algorithms solve a problem by training a model based on training data, then the trained model is applied to the unseen test data. ML systems can outperform humans on many tasks and decisions since they are able to consider all the instances in a dataset without any human bias due to the prior knowledge [164]. ML is able to observe the structures and patterns within large datasets. A learning problem refers to the problem of improving some performance indicators through some type of training experience during execution. ML can handle a variety of many learning problems, such as classification, clustering and regression [146]. The main practical objective of ML is to design efficient and robust algorithms to produce accurate predictions for unseen data.

In machine learning, it is common to split the available data into two sets: the training set and the test set. The purpose of this split is to evaluate the performance of the machine learning model on unseen data, which is a crucial step in assessing the generalization capability of the model. The training set is used to train the model, which involves adjusting the model parameters or weights to minimise the error or loss function. The model is trained using an optimization algorithm such as gradient descent, which updates the weights in the direction that reduces the loss function. The test set is used to evaluate the performance of the trained model on unseen data. The test set is not used during the training process, and its purpose is to provide an estimate of how well the model is likely to perform on new, unseen data. It is important to use separate training and test sets to avoid overfitting, which occurs when the model becomes too complex and

fits the training data too closely, resulting in poor generalization to new data. The test set provides a way to estimate the generalization error of the model, which is the difference between its performance on the training set and the test set. The performance of the model on the test set can be measured using various metrics, depending on the nature of the problem. Common metrics include accuracy, precision, recall, F1 score, and AUC-ROC curve.

Generalisation and overfitting are two important concepts in machine learning that relate to the ability of a model to perform well on new, unseen data. Generalisation refers to the ability of a machine learning model to perform well on new, unseen data that is drawn from the same distribution as the training data. In other words, a model that generalises well is able to capture the underlying patterns and relationships in the training data and apply them to new, unseen data. Overfitting, on the other hand, occurs when a model becomes too complex and fits the training data too closely. This can result in poor generalisation to new, unseen data because the model has memorised the training data instead of learning the underlying patterns and relationships. Overfitting can occur when the model is too complex, the training data is noisy or insufficient, or the model is trained for too long. One way to detect overfitting is to compare the performance of the model on the training set and the test set. If the performance on the training set is much better than the performance on the test set, it is likely that the model has overfit the training data.

The expected task performance is not the only concern for the deployment of ML systems in complex applications. The *right to explanation* [73] becomes more and more important. Especially, when decisions derived from ML systems can ultimately affect humans' lives (as in e.g. medicine, law or defense), there is an emerging need for understanding how such decisions are furnished by AI methods [73]. Interpretability is becoming one of the main barriers lying between the AI and its practical implementation [13]. If we want to integrate machines and algorithms into our daily

lives, interpretability is required to increase the social acceptance [147]. Humans are reticent to adopt techniques that are not directly interpretable, tractable and trustworthy [223], especially when humans care more and more on ethical AI [73]. In addition, ML models usually pick up biases from the training data. This can lead the ML contains unexpected discrimination against protected groups. Interpretability can be a useful debugging tool for detecting bias in machine learning models [147]. Doshi-Velez and Kim [54] also claim that if we can ensure that the ML models can explain their decisions, it will be much easier to check the fairness, privacy, robustness, causality and trust of the models.

2.8.1 Definition of Interpretability

It is difficult to provide a formal definition of interpretability since there is no common understanding in the literature on what interpretability means [13]. To understand the concept of interpretability, let us begin by examining the meaning of the word "interpret" from dictionary, " *explain, tell or to present in understandable terms*" (Merriam-Webster dictionary accessed on 2023-3-23). Many studies claim to achieve interpretable models and techniques that enable explainability [13]. Therefore, different definitions may be given in the context of machine learning (ML) systems. For instance, Kim et al. [103] define interpretability as *"the degree to which a human can consistently predict the model's result"*, while Miller [144] defines it as " *the degree to which a human can understand the cause of a decision."* Therefore, it can be concluded that interpretability refers to a passive characteristic of a model that refers to the level at which a given model makes sense to a human observer [13]. The higher the interpretability of a machine learning model, the easier it is for someone to understand why certain decisions or predictions have been made. In other words, a model is more interpretable than another model if its decisions are easier for a human to comprehend than those of the other model [147]. In conclusion, inter-

pretability can be defined as the ability to explain or provide meaning of a ML model in understandable terms to a human users [13, 69, 111].

2.8.2 Techniques for Interpretability

There are two categories of approaches to interpretability based on the techniques and the model properties [120]. The first one relates to transparency which indicates the mechanism by which the model works, and the other one consists of post hoc explanations. Usually, these two categories can be distinguished by identifying whether interpretability is achieved through constraints imposed on the complexity of the ML model (intrinsic) or by applying methods that analyse the model after training (post-hoc) [147].

2.8.2.1 Intrinsic Interpretability

Intrinsic mainly focus on the transparency of the model. Transparency refers to models that are interpretable by themselves [35]. Intrinsic mainly focus on the transparency of the model. Transparency can be achieved through imposition of constraints on the model, such as sparsity, monotonicity, causality, or physical constraints that come from the domain knowledge [168]. Transparency can answers the question of how the model works [120]. Lipton [120] considers the transparency at three different levels: the entire model (simulatability), the individual components such as parameters (decomposability) and the training algorithm (algorithmic transparency).

- Simulatability can be denoted as the ability of a model being simulated by human users [13]. If an entire model can be understood and simulated by a person at once, then the model can be called transparent. In this case, an interpretable model should be a simple model. Thus, model complexity takes a dominant place in transparency [14]. For example, a simple but extensive (e.g., contains too

large number of rules) rule-based system is not transparent, however, a single perceptron neural network is transparent. This claim is consistent with argument that sparse linear models are more interpretable than dense linear models learned on the same inputs [182]. Ribeiro et al. [165] also agree with this claim and suggest that an interpretable model is one that can be easily presented to a human by means of text and visualizations. Lipton [120] suggests that simulatability may admit two subtypes: the first one based on the size of the model and the other one based on the computation required to make the prediction.

- Decomposability is the second notion of the transparency. It denotes that ability to explain each of the components (input, parameter, and calculation) of a model. This is consistent with the property of intelligibility in [127]. For example, each node in a decision tree can be easily translated to a plain text description. By contrast, the weights in a deep neural network are hard to explain to users. Note that, Decomposability requires every inputs themselves be individually interpretable. Therefore, an algorithmically transparent model can become decomposable when each part of the model can be understandable by a human without the need for additional tools [35].

- Algorithmic transparency is the final notion of transparency which at the level of the learning algorithm itself. It deals with the ability of the user to understand the process followed by the model to produce any given output from its input data. For example, a linear model is considered transparent because users can understand the shape of the error surface, this allows users to understand how the model will work in every unseen data it may handle [93]. On the other hand, it is not possible to understand it in deep architectures of deep neural network [97]. Lipton [120] also argues that the heuristic optimization procedures for neural networks are demonstrably powerful, but

we cannot understand how they work and guarantee that they will work on new problems. [35]. Besides, Carvalho et al. [35] argue that the main constrain for algorithmically transparent models is that the model has to be fully explorable by means of mathematical analysis and methods.

2.8.2.2 Post-Hoc Interpretability

Post-hoc (post-model) interpretability refers to apply explanation methods to the learned model as some models that are not readily interpretable by design. It can extract information from the learned model. Note that, post-hoc interpretations often do not elucidate precisely how a model works, they may nonetheless confer useful information for practitioners and end users of machine learning [120]. It can also answer the question what else can the model tell us. Post-hoc methods can be applied to not only "black-box" models, but also transparent models, since post-hoc methods are usually decoupled from the main model [35]. Some common approaches to post hoc interpretations include text explanations, visualizations of learned representations or models, local explanations and example-based explanations [13, 120, 147].

- Text explanations handle the interpretability problem by providing explainability for a model by means of learning to generate text explanations that can help explaining the results from the model to end users [69]. It is much easier for humans to understand text comparing with a ML model as humans often justify decisions verbally [120]. Krening et al. [110] propose a system in which one model chooses actions to optimise cumulative discounted return, and the other model to provide verbal explanations of strategy from the first model. In the field of recommendation systems, McAuley and Leskovec [137] provide text explanation to explain the decisions of a latent factor model. They train a latent factor model for rating prediction and

a topic model for product reviews simultaneously. They aim to minimise the squared error on rating prediction and maximise the likelihood of review text. Rather than train a separate model to provide text explanation, the top words presented in the topic models can be also a explanation technique [37].

- Visualisation explanations aim at visualizing the model's behavior. Many existing visualisation techniques in the literature come along with dimension reduction techniques that can help human to understand simpler visualisation. Visualisation explanations can be applied with other techniques to improve their understanding, and are considered as the most suitable way to introduce complex interactions within the variables involved in the model to users not acquainted to ML modeling [13]. One popular method is to visualize high-dimensional distributed representations with t-distributed stochastic neighbor embedding (t-SNE) [132] that can renders 2D visualizations in which nearby data points are likely to appear close together. Mordvintsev et al. [148] try to explain a learned neural network for image classification by altering the input through gradient descent to enhance the activations of certain nodes selected from the hidden layers. The relationship between the input image and the output label can be investigated by inspected the the perturbed inputs of the nodes. They observed that enhancing some nodes caused certain dog faces to appear throughout the input image.

- Local explanations handle the interpretability by explaining learned model locally. It is quite difficult to describe the full mapping learned by a model, but it is much easier to give explanations to less complex solution subspaces that are relevant for the whole model [120]. Saliency map [175] is popular local explanation method for DNN. It can accurately show the relationship between the output and a given input vector by highlighting regions of the input that, if changed,

would most influence the output. However, these local explanations might be misleading. Because users may get a totally different saliency map when they move a single pixel in the input image. Local interpretable model-agnostic explanations (LIME) is another popular local explanation method which is proposed in [165]. It is actually a local surrogate model that trained to approximate the predictions of the underlying black box model. Instead of training a global surrogate model, LIME focuses on training local surrogate models to explain individual predictions.

- Example-based explanations explain the behavior of ML models or explain the underlying data distribution by selecting particular instances of the dataset [147]. It is similar to the process of how humans behave when attempting to explain a given operation, example-based explanations can extract representative examples that represent the inner relationships and correlations found by the model being analysed [13]. The most similar example could be k-nearest neighbors method, we can identify a prediction based on other instances that have been already labeled. Another example is that doctors often refer to case studies to support a planned treatment protocol [120]. Doshi-Velez et al. [55] and Kim et al. [104] have done related work for Bayesian methods, they try to interpret generative models by example-based explanation approaches. Le et al. [116, 117] utilise counterfactual to improve the model understanding and trust.

2.9 Related Work

2.9.1 Approaches to CARP

A variety of approaches have been proposed to solve CARP. The *Exact* approaches [18, 72] were firstly applied. The advantage of the exact approaches is that they can get the optimal solutions. However, they can

only be applied to small-sized instances due to their high computational complexity. Constructive heuristics, such as path scanning heuristic [70], augment-merge heuristic [72], and Ulusoy's split heuristic [190], were proposed to obtain reasonably good solutions in a short time. Constructive heuristics are generally much faster than the exact approaches because the one-pass solution construction process typically has a linear complexity. In contrast, to guarantee optimality, the exact approaches have exponential complexity for NP-hard problems such as UCARP. Meta-heuristic approaches, such as tabu search [24,78], genetic algorithms [112], memetic algorithms [113,142,181] and ant colony optimisation [52,94,206], were also applied to CARP. Meta-heuristic approaches usually iteratively improve one or more initial solutions. They can also provide promising solutions within a limited time budget. Since meta-heuristic algorithms can take the solution from constructive heuristics as their initial solution, they are not worse than constructive heuristics. Transfer learning techniques, such as meme learning and selection [62], memetic computing paradigm [61], are also utilised to enhance the problem solving capability for CARP. These approaches can capture useful structures or latent patterns from previous experiences of problem-solving, and transfer the structures and patterns to future evolutionary search. As a result, the effectiveness and search efficiency can be improved.

2.9.2 Approaches to UCARP

The CARP in reality can contain a variety of different uncertainties. Liu et al. [121] gave a comprehensive review on the uncertain factors in CARP, and the corresponding approaches to address them. Liu et al. [121] summarised four important uncertain factors in CARP, i.e., stochastic task demand, stochastic edge deadheading cost, stochastic task presence and stochastic edge existence, all of which are important to be considered. Thus, in this paper, we aim to solve the UCARP [141] model that considers all the

above four uncertain factors.

There are two types of approaches to the UCARP model with the four uncertain factors. The first one aims to find solutions that are expected to handle all the possible realisations of the random variables, which can be namely "*proactive*" approaches. The second one uses Genetic Programming Hyper Heuristics (GPHH) to evolve routing policies that construct solutions gradually in real time, which can be namely "*reactive*" approaches.

Typically, proactive approaches output a pre-planned solution and a recourse operator. They use an optimisation algorithm to generate the pre-planned solution, and then a recourse operator to repair the solution when failure, such as route failure, occurs. The performance of these approaches is highly dependent upon the accuracy of the prediction of the stochastic environment as well as the effectiveness of the recourse operators.

Fleury et al. [64] adapted the best method published for CARP, i.e.,hybrid genetic algorithm (HGA), to CARP with stochastic task demand (SCARP). HGA not only considers the solution cost but also the robustness of solutions. Wang et al. [194] proposed a new MA for UCARP. This new MA contains an integrated fitness function and a large step-size local search operator. It makes use of the integrated fitness function to direct the search process while the large step-size local search operator prevents MA from trapping in the local optima. Babaee et al. [15] proposed hybrid meta-heuristic approach integrated with a mix integer programming model to deal with the demand uncertainty. The hybrid meta-heuristic approach utilises simulated annealing as the optimiser and a constructive heuristic to provide initial solution. Wang et al. [193] developed an Estimation of Distribution Algorithm with Stochastic Local Search (EDASLS) for UCARP. The EDASLS aims to get solutions that can perform well over a set of UCARP instances. It is developed based on the edge histogram based sampling algorithm [187]. A novel stochastic local search is proposed and integrated. The stochastic local search can effectively handle the uncertainties in UCARP. Tong et al. [186] proposed a generalised meta-heuristic

framework for UCARP. In addition, Tong et al. [184] proposed a hybrid local search framework which can handle the small dynamic changes in UCARP.

In contrast, reactive approaches produce a routing policy without needing any pre-planned solution. GP is typically used to evolve a population of routing policies, and the best evolved routing policy can be applied to unseen/future scenarios to make routing decisions in real time.

Weise et al. [203] firstly adapted a GPHH to a single-vehicle UCARP and showed that routing policies evolved by GPHH could outperform manually designed routing policies. After that, to improve the effectiveness of GPHH, Liu et al. [122] proposed a novel pool filer to omit irrelevant candidate tasks during the decision-making process. In real world, tasks in UCARP are usually served by a fleet of vehicles. Thus, the model was extended from a single-vehicle to multi-vehicle version by Mei et al. [143]. MacLachlan et al. [133] improved the GPHH by proposing a novel pool filer which can prevent generating too small routes and an effective look-ahead terminal which can prevent route failure. Then, MacLachlan et al. [134] examined the advantage of vehicle collaboration in handling the uncertain environment. To speed up the training efficiency, Ardeh et al. [10, 11] applied transfer learning techniques to GPHH. Liu et al. [124] proposed a new paradigm for GPHH. The new paradigm utilises GP and cooperative coevolution to evolve a task sequence and a recourse policy simultaneously. A new recourse strategy called OneFAll was proposed to handle route failure by using one vehicle take charge of all the failed tasks [126]. In addition, Liu et al. [125] proposed a GPHH method which consider the stability of the routes generated by routing policies. Recently, Wang et al. [195, 197] took the interpretability of the GP-evolved routing policies into consideration by evolving smaller routing policies. Table 2.4 shows the categorisation of the approaches to CARP and UCARP.

Table 2.4: Classification of approaches to (U)CARP.

Problem	Category	Literature
CARP	Exact approaches	[18,72]
	Constructive heuristics	[70,72,190]
	Metaheuristics	[24,52,61,62,78,94,112,113,142,181,206]
UCARP	Proactive approaches	[15,64,184,186,193,194]
	Reactive approaches	[10,11,122,124,133,134,143,195,197,203]

2.9.3 Interpretable GP

2.9.3.1 Interpretability in GP

Genetic Programming (GP) is considered an interpretable approach since its flexible tree-based representation [33, 83]. Therefore, it is often utilised to explain other "black-box" models, such as Deep Neural Networks (DNNs) [60, 63], Support Vector Machines (SVMs) [63], Random Forest (RF) [63] and Reinforcement Learning (RL) [77]. Evans et al. [60] proposed a global model extraction method which uses multi-objective genetic programming to construct accurate, simplistic and model-agnostic representations of complex black-box estimators. The result showed that the newly proposed method can offer drastically simpler models, with statistically equivalent test accuracy over DNNs and RF. Rather than reproducing a global model, Ferreira et al. [63] utilised GP to evolve a local explanation model. The Genetic Programming Explainer (GPX) was proposed to the problem of explaining decisions computed by AI systems. GPX generates a noise set located in the neighbourhood of the point of interest and fits a local explanation model for the analysed sample. The result showed that GPX can reflect the local behaviour of the complex models, such as Deep Neural Networks, Support Vector Machines and Random Forest.

In addition to helping to explain "black-box" models, GP can be also directly utilised to evolve interpretable programs for classification, regression and clustering problems. Berlanga et al. [19] utilised GP to evolve compact and accurate GP trees for high-dimensional classification prob-

lems. The evolved GP classification system showed a good performance in terms of accuracy and interpretability. Hein et al. [77] utilised GP to discover interpretable policies for RL. The authors proposed a Genetic Programming for Reinforcement Learning (GPRL) approach based on model-based batch reinforcement learning and genetic programming, which autonomously learns policy equations from pre-existing default state–action trajectory samples. GPRL can produce well-performing interpretable reinforcement learning policies from the pre-existing default state–action trajectory data. Hu et al. [82] designed a linear GP (LGP) algorithm and feature importance evaluation methods for the bioinformatics application problem of predicting disease risk using metabolite abundance levels in blood samples. LGP can evolve highly accurate classification models and the feature selection method based on feature importance evaluation can help further improve the prediction accuracy and discover the causality between the input and output. Zhao [222] utilised multi-objective GP to evolve a Pareto Front of interpretable decision tree-style rules for classification problem. Although GP can be utilised to evolve interpretable models, one issue is that the built-in interpretability is insufficient to handle complex problems [139]. Mei et al. [139] also claim that the evolved GP models become difficult to interpret when they become too large size. Consequently, it becomes important to consider interpretability as a key aspect of the design of GP algorithms. This will enable GP models not only effectively perform critical tasks such as recognition, prediction, and decision-making, but also provide insights into the reasoning behind their decision-making processes. By incorporating interpretability, users can comprehend the internal workings of the model, including its capabilities and limitations, its potential for making errors in particular circumstances, and how to enhance its performance.

There have been various studies to reduce tree size during the GP evolutionary process. The most straightforward way is to limit the number of nodes of the tree. Wang et al. [195–197] intended to handle this issue

by directly reducing the size of the evolved GP programs. In addition, the number of Splitting points was considered by Evans et al. [60]. However, this strategy has two main limitations. First, it limits the search space and may weaken the exploration ability of GP, and thus lead to worse effectiveness. Second, it is nontrivial to predefine a reasonable limit for the number of nodes [23]. To deal with these issues, Silva and Costa [171–173] proposed a method that dynamically adjust the limit during the evolutionary process. The method in [171] initial set a small limit and then increase the limit once it finds a new best individual exceed the limit. The limit can be also decreased if a new best individual has a smaller size [172]. Another commonly used strategy is to penalise large trees during the selection, known as the *parsimony pressure* [109]. The parsimony pressure can be utilised in either fitness evaluation stage [36, 89, 153, 212] or parent selection stage [20, 23, 48, 58, 129]. The penalty term can be a fixed value based on domain knowledge [36, 89, 153] or dynamically changed based on the accuracy and size of the current best individual [212]. One way to use lexicographic parsimony pressure methods [129] is to prioritise accuracy when comparing solutions, and then compare their size only if their accuracy is identical. Kinzett et al. [108] point out that it is difficult to obtain a small and effective solution in some scenarios. The solution could be biased by either tree size or fitness in GP. Luke et al. [130] review a set of bloat control methods in GP and compare them with the depth limiting method on different problems. The authors argue that linear parsimony pressure [36] performed the best in the cross-problem comparison, and the double tournament selection [128] performed the second best. However, the conclusions obtained in [130] are purely empirical.

General genetic operators can be modified to evolve simpler GP trees. Rather than resorting to the arbitrary generation and subsequent rejection or penalization of large offspring in genetic algorithms, it is possible to design genetic operators specifically aimed at generating small offspring. A frequently employed strategy involves creating offspring that bear re-

semblance to their parents in terms of structure. In tree-based genetic programming, one can adopt crossover techniques that select only sub-trees possessing the same structural characteristics as the parents' upper segments, such as one-point crossover [159], or sub-trees with similar sizes, such as size-fair crossover [114]. Homologous crossover [154] represents a hybrid approach that takes into account both sub-tree size and inherited structure. When it comes to mutation, one may modify only a single node's value (point mutation [160]), instead of replacing an entire sub-tree. Alternatively, the size of newly generated sub-trees can be controlled to ensure that it does not surpass the replaced sub-tree, using techniques like size-fair mutation [115] or the prune and plant operator [3].

Keijzer and Babovic [98] and Mei et al. [140] handle the interpretability issue by applying dimensionally aware GP which considers the physical dimensions of the problem features, and favours the combinations between the features with the same physical dimension. Alba et al. [2] and Hein et al. [76] utilised strongly-typed GP to constrain the combinations of different feature types to evolve interpretable GP programs. GP tree simplification methods are an alternative for tree size reduction. Hooper et al. [81] utilised the expression simplification to simplify the trees. Their simplification method employed over 200 simplification rules to simplify an expression. To speed up the simplification process, Zhang and Wong et al. [205,218–220] proposed the use of algebraic tree simplification with the hashing techniques. However, these algebraic methods cannot detect all kind of redundancies [106]. It can easily simplify the expression $A - A$ to 0. However, $A + 10^{-100}$ cannot be simplified to A directly, although 10^{-100} does not play any significant role in the final output. In addition to the algebraic approach, Kinzett [107] and Song et al. [177] proposed numerical simplification methods. These methods are based on the local effect of a subtree. Kinzett et al. [107] removed a node or a subtree if its numerical contribution is smaller than a predefined threshold. Song et al. [177] replaced the parent node with its child node if the difference between their

outputs is below a predefined threshold. Most GP simplification methods can only detect redundancies based on genotypic information (contribution of the genotypic subtrees) rather than phenotypic information (output of the whole GP tree). Thus, many redundancies will still exist. Also, they require manually selected simplification rules or predefined threshold parameter. The final test performance is highly sensitive to these parameters. It is hard to set these parameters properly.

2.9.3.2 Interpretability in GPHH for UCARP

There are two categories of approaches to interpretability based on the techniques and the model properties [120]. The first one is mainly about the intrinsic which aims to the complexity of the model directly, and the other one consists of post-hoc explanations. Usually, these two categories can be distinguished by identifying whether interpretability is achieved through constraints imposed on the complexity of the ML model (intrinsic) or by applying methods that analyse the model after training (post-hoc) [147].

Intrinsic approaches mainly focus on the transparency of the model. Transparency refers to the degree that a model can be understood by users directly [35]. If an entire model can be understood and simulated by a person at once, then the model can be called transparent. In this case, an interpretable model should be a simple model. Thus, model complexity takes a dominant place in transparency [14]. For example, a rule-based system that contains a large number of simple rules is not transparent, however, a single perceptron neural network is transparent. This claim is consistent with the argument that simple linear models (having few coefficients) are more interpretable than complex linear models (having many coefficients) learned on the same inputs [182]. Intrinsic interpretability is typically achieved by imposing constraints on the model, such as monotonicity, causality, sparsity, or physical constraints that come from the domain knowledge [168].

Branke et al. [25] investigated the use of a simple linear model as an alternative to the GP-evolved rules, however, the linear model gave significantly worse results than the GP-evolved model. Jia et al. [95] also explored different representations of routing policies, and eventually found that a routing policy represented as a GP tree was the best choice. To evolve more interpretable routing policies for UCARP, Wang et al. [195, 197] take the intrinsic interpretability of the GP-evolved routing policies into consideration by evolving simpler routing policies. Wang et al. [195, 197] firstly add size limitation for the GP tree to evolve smaller routing policies. Even though users can understand relatively simple routing policies, in some cases GP needs to evolve very complex routing policies to ensure effectiveness. It is necessary to find ways to explain these complex routing policies.

Post-hoc (post-model) interpretability refers to applying explanation methods to the learned model as some models are not readily interpretable by design. It can extract information from the learned model. Note that, post-hoc interpretations often do not elucidate precisely how a model works, they may nonetheless confer useful information for practitioners and end users of machine learning [120]. It can also answer the question what else the model can tell us. Post-hoc methods can be applied to not only "black-box" models, but also transparent models (the model itself is easy to interpret), since post-hoc methods are usually decoupled from the main model [35]. Some common approaches to post-hoc interpretations include text explanations, visualisations of learned representations or models, local explanations and example-based explanations [13, 120, 147].

To our best knowledge, there is not any existing post-hoc interpretability work for GPHH for UCARP. we can see that previous research has focused on intrinsic methods, while post-hoc has yet to be investigated. It is necessary to develop novel post-hoc methods that help end-users understand GP-evolved routing policies.

2.10 Chapter Summary

This chapter reviewed the main concepts of UCARP. The essential backgrounds of GP, HH and GPHH. Also, this chapter reviewed the related work of approaches for CARP and UCARP. We discuss the limitations of existing work of current GPHH works for UCARP. The current studies of GPHH for UCARP is lack of interpretability. GP-evolved routing policies are typically very complex thus hard to interpret by human users. Interpretability is very important if we want to apply the GP-evolved routing policies in practice as users will refuse to use GP-evolved routing policies if they are not trustworthy. In this chapter, we have also reviewed related works of interpretability of GP from two aspects, i.e. using GP to interpret other machine learning models and interpreting GP-evolved models. This chapter showed that improving GP-evolved routing policies have many challenges and difficulties. The limitations of the existing work that form the motivations of this research were also discussed, which can be summarised as follows.

1. Existing studies have shown that GPHH methods can evolve effective routing policies that well handle UCARP. However, these methods still have limitations of the interpretability of the evolved routing policies. Therefore, improving the interpretability of GP-evolved routing policies is still an open issue. Research needs to be conducted to propose new methods to improve the interpretability in both intrinsic and post-hoc ways.

2. Existing studies to evolve simpler routing policies mostly limit the tree size directly. One key reason of complex routing policies is that GP tree contains many redundant build blocks. However, to our best knowledge, there is no study trying to removing the redundant build blocks from GP-evolved routing policies for UCARP.

3. There is a tradeoff between the interpretability of effectiveness of

GP-evolved routing policies. Thus, we can treat our problem as a multi-objective problem. In addition, evolving interpretable routing policies that meets the needs and preferences of a specific user group can be challenging. Existing method have not consider using multi-objective methods to evolve a diverse non-dominated routing policies to different users.

4. To our best knowledge, there is no study trying to propose post-hoc methods to improve the interpretability of GP-evolved routing policies for UCARP as GP is considered an interpretable approach because of its tree-based representation. However, GP trees can apply complex nonlinear transformations to input variables, which can make it difficult to interpret the relationship between the inputs and the output. Therefore, it is necessary to propose post-hoc methods that can explain the relationship between the inputs and the final decision made by GP-evolved routing policies.

By proposing both intrinsic and post-hoc methods for GP-evolved routing policies for UCARP, the above-mentioned issues will be addressed in the following three chapters. Chapter 3 will develop intrinsic method that can automatically remove the redundant build blocks from GP-evolved routing policies during the evolutionary process. Chapter 4 will develop new multi-objective GP methods to include both interpretability and effectiveness as main objectives and evolve a Pareto front of routing policies meet the needs and preferences of different user groups. Chapter 5 will introduce new post-hoc methods that give explanations of GP-evolved routing policies to end users.

Chapter 3

Genetic Programming with Niching for Uncertain Capacitated Arc Routing Problem

3.1 Introduction

GP evolved routing policies are usually too large and complex, which usually leads to poor interpretability. Evolving small and simple routing policies (without losing effectiveness) can help end users understand the evolved routing policies better so that they can use them confidently and modify them when necessary. Intuitively, the size (number of nodes) of the routing policy is an essential factor that can affect the interpretability, and a larger (smaller) routing policy tends to be harder (easier) to interpret. Usually, the tree size of the routing policy evolved by GPHH is too large. The main reason is that GP tends to continuously increase the size of its individuals, which is known as *bloat* [23, 49, 162]. For bloat control, one can simply limit the tree size or depth during the GP evolution (e.g., [161, 197]) or design specific genetic operators to consider both tree size and effectiveness (e.g., [49, 108, 130]). However, these approaches cannot balance the two aspects well. Their performance could be biased by

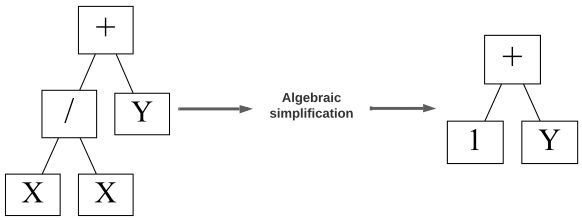


Figure 3.1: An example of GP tree algebraic simplification.

either tree size or effectiveness.

GP program simplification is another way to reduce the tree size by removing redundant materials from GP trees. GP produces trees that typically contain a large number of redundant components (subtrees). It is desirable to remove as much of these redundant materials as possible without sacrificing the exploration ability of GP. Manual simplification on the final GP tree has been tried in the past. However, it is more preferred to simplify GP trees during the search process [105]. Wong and Zhang et al. [205, 218–220] have explored the use of algebraic tree simplification with the hashing technique and applied it to regression and classification problems. Kinzett et al. [107] and Song et al. [177] have proposed numerical tree simplification methods. Both methods are based on the local effect of a subtree.

The above simplification methods have several limitations. First, they detect redundant subtrees based on genotypic information (e.g. tree structure) rather than phenotypic information (e.g. behaviour in decision making). Thus, they may fail to detect some implicit redundancies. Second, they need predefined parameters, i.e. simplification rules and threshold. The final effectiveness is highly sensitive to these parameters, and it is hard to set them properly. To the best of our knowledge, there is no existing study that implements the simplification based on the phenotype of GP trees. Also, there is no existing study that applies GP program simplification to GPHH for UCARP to evolve both effective and simple routing

policies.

The phenotypic behaviour, in this context, captures the actual sequence of routing decisions made by the evolved solutions. To illustrate this concept, consider a simplified UCARP scenario with a pool of tasks (e.g., t1, t2, t3) awaiting service. A routing policy, as the outcome of the evolutionary process, selects a specific order to serve these tasks. For instance, if the routing policy prioritises t3 first, followed by t1, and finally t2, the resulting phenotypic behavior of the routing policy would be represented as 't3, t1, t2.' This sequence embodies the concrete routing actions taken by the vehicles, dictating the order in which tasks are served.

To simplify a GP tree based on phenotypic behaviour, we aim to replace a large tree with another smaller tree with the same phenotypic behaviour. An intuitive approach is to group the individuals in the population based on their phenotypic behaviour (each group contains the individuals with the same behaviour), and simplify all the individuals in each group by replacing it with the smallest individual in that group. We call this approach *niching simplification*, since it is similar to the niching techniques [84,136] that divides the individuals in the population into different groups/niches.

3.1.1 Chapter Goals

The overall goal of this chapter is to propose a novel GPHH that simplifies the individuals during the evolutionary process based on their phenotypic behaviour. In this way, we can obtain both more effective and smaller routing policies, which can be potentially more interpretable and generalisable. To achieve this goal, we have the following research objectives:

1. To develop a new niching simplification scheme based on phenotypic behaviour. Specifically, the individuals with the same phenotypic behaviour (i.e., fitness in this study) are grouped into a niche. In each niche, all the individuals are replaced by the individual with

the smallest size (so-called *representative* of the niche);

2. To compromise the loss of population diversity due to the niching simplification, we still keep the original population, and store the niche representatives in an external archive;

3. To develop a multi-source breeding mechanism to generate offspring from the original population and the representative archive, respectively;

4. To design niching-based elitism and parent selection schemes for the breeding from the representative archive;

5. To develop a new GPHH algorithm with Niching (GPHH-N) that incorporates all the above designed components.

6. To verify the effectiveness of GPHH-N, and analyse the routing policies evolved by GPHH-N.

3.1.2 Chapter Organisation

The rest of this chapter is organised as follows. Section 3.2 describes the proposed approach. Section 3.3 gives the experimental studies. Section 3.4 conducts the results and the further analysis. Finally, Section 3.6 gives the conclusions and possible future directions.

3.2 Proposed Method

3.2.1 Overall Framework

The overall framework of GPHH-N is shown in Fig. 3.2. Firstly, a population of routing policies (each represented as a GP tree) is initialised randomly. Then, the routing policies are evaluated at each generation based on their simulation effectiveness (total cost). If the stopping criteria are not

reached, the population undergoes the evolutionary process. In the evolutionary process, routing policies are first simplified by the niching simplification. Details of the niching simplification will be given in Section 3.2.4. After that, the smallest tree in each niche is considered the niche representative and is stored in a representative archive. The niching elitism and niching tournament selection are then applied to both the original population and representative archive to produce offspring. The niching elitism selects diverse trees for the next generation. It will be described in Section 3.2.5. The newly developed niching tournament selection can ensure that diverse parents are selected from the simplified population. Details will be discussed in Section 3.2.6. The multi-source breeding process selects parents from both the original and simplified populations so that the simplified blocks are not completely lost. We will describe its details in Section 3.2.7. The relationship between the population and representative archive and how the newly proposed operations update them are shown in Fig. 3.3.

3.2.2 Individual Representation

In this thesis, each routing policy is represented as a priority function. The priority function is a combination of the state features, such as the cost from the current location to the candidate task (CFH) and the cost from the candidate task to the depot (CTD). For example, if the priority function is "CFH + CTD", the priority function tends to select tasks that are closer to the current location and also closer to the depot. The routing policy works as a decision-maker during the solution construction process. Typically, each vehicle can only serve one task at a time. The routing policy will be applied to each unserved task to determine each unserved task's priority once a vehicle completes its current task and is ready to serve the next task. The task with the best priority value will be served next. The decision process finishes when all the tasks have been served and the routes

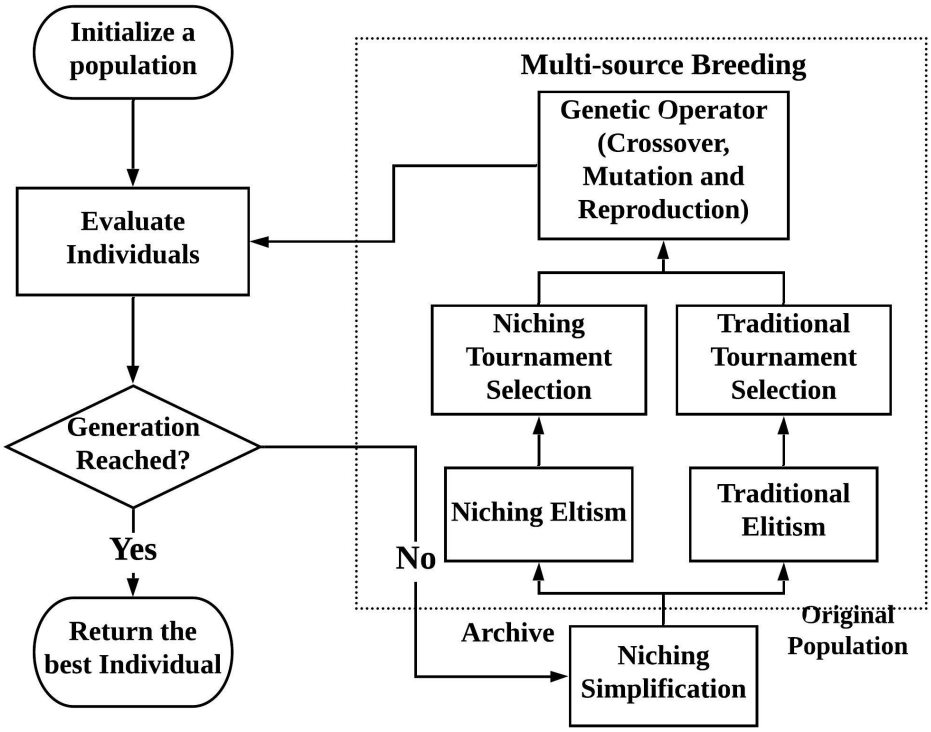


Figure 3.2: The overall Framework of GPHH-N

returned as the solution constructed by the routing policy.

3.2.3 Fitness Evaluation

Given a routing policy rp, the fitness is defined based on the average quality (i.e. total cost) of the routes that it generates. Specifically,

$$\text{fit}(rp) = \frac{1}{|\mathcal{S}'|} \sum_{s \in \mathcal{S}'} C(s_\xi), \tag{3.1}$$

where $\mathcal{S}'$ is a set of instance samples, $|\mathcal{S}'|$ is the number of instance samples. $C(s_\xi)$ stands for the total cost of the solution obtained by rp on instance sample s, which can be calculated by Eq. (2.8). The solution is constructed based on a simulation process that is commonly used in UCARP

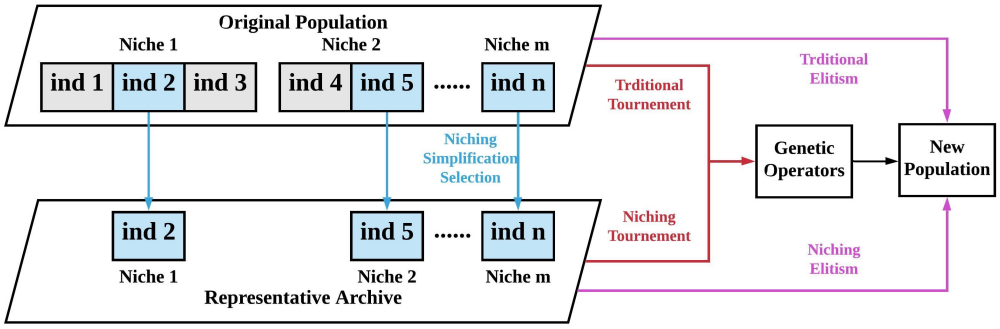


Figure 3.3: The illustration of the offspring generation in GPHH-N.

literature [123,197]. Algorithm 2 shows the fitness evaluation in GPHH-N.

One can see that each routing policy is applied as a priority (heuristic) function. When a vehicle is ready (at a decision-making point), a routing policy is applied to determine which candidate task to serve next. The quality of each routing policy can be evaluated by applying it to a set of training samples $\mathcal{S}'$. A solution can be constructed using the routing policy on a training sample $s \in \mathcal{S}'$.

We randomly re-sample training samples for the fitness evaluation. Such sample rotation has been commonly used in other studies [80, 122] to improve the generalisation of the evolved heuristics.

3.2.4 Niching Simplification

The niching simplification method is shown in Algorithm 3. It firstly places the trees with the same phenotype into the same niche. Then, each niche identifies the tree with the smallest tree size (number of nodes in the tree) in the niche as the niche representative, and replaces all the other trees with this representative. If there are multiple trees with the smallest tree size, the first identified tree is chosen as the representative. In this way, each niche is represented by a tree with the smallest tree size under

Algorithm 2: The fitness evaluation in GPHH-N.

```
Input: Training set S_train, An unevaluated population pop
Output: Fitness values of pop
Randomly re-sample a training samples S' ⊆ S_train;
// Evaluate pop using S'
for each routing policy rp ∈ pop do
    // Evaluate rp
    for each training sample s ∈ S' do
        while tasks in queue is not empty do
            if vehicle is ready then
                Select which task to serve using rp;
            end
        end
        Get the total cost of the generated solution, i.e. C(s_ξ);
    end
    Calculate fit(rp) using Eq. (3.1);
end
return fitness values of pop;
```

the same phenotype. Fig. 3.4 shows two trees with the same phenotype but different tree sizes. Tree A can be hardly simplified to Tree B by the algebraic or the numerical simplification method. However, Tree A can be easily simplified to Tree B by the niching simplification approach.

Note that fitness is used as a high-level phenotypic behaviour of a GP individual. This is a specific design for GPHH for UCARP, since our preliminary study has shown that many routing policies have different tree structures but the same fitness value. Although we can use other more precise phenotypic characterisations such as the structure of the obtained routes for a given instance, or the phenotypic vector [79], the fitness-based characterisation is shown to be effective enough empirically, and is very efficient to compute.

Algorithm 3: The niching simplification.

```
Input: Original Population pop
Output: Simplified Population pop', Representative Archive archive
Initialise archive = ∅ ;
for each routing policy rp ∈ pop do
    inNiche(rp) = false ;
end
for each routing policy rp ∈ pop do
    if inNiche(rp) == true then
        continue ;
    end
    Create a new niche = {rp};
    Set inNiche(rp) = true;
    for each routing policy rp' ∈ pop do
        if fit(rp') equals to fit(rp) then
            Add rp' to niche;
            Set inNiche(rp') = true;
        end
    end
    Find out the individual with smallest tree size ind_s in niche;
    Add ind_s to archive;
    Replace all other individuals in the niche with ind_s and update pop' ;
end
return pop' and archive;
```

3.2.5 Niching Elitism

Elitism is used in GP to avoid losing the previously found best individuals. In the traditional elitism scheme, we directly copy the best individuals (e.g. the top 10 individuals in terms of fitness) from the population to the next generation. However, this will lead to a significant loss of diversity after the niching simplification since all the best individuals might come from the same niche. We propose a niching elitism scheme to address this issue, which selects the elitists only from the representative archive to maintain diversity. Firstly, the archive is sorted based on the fitness

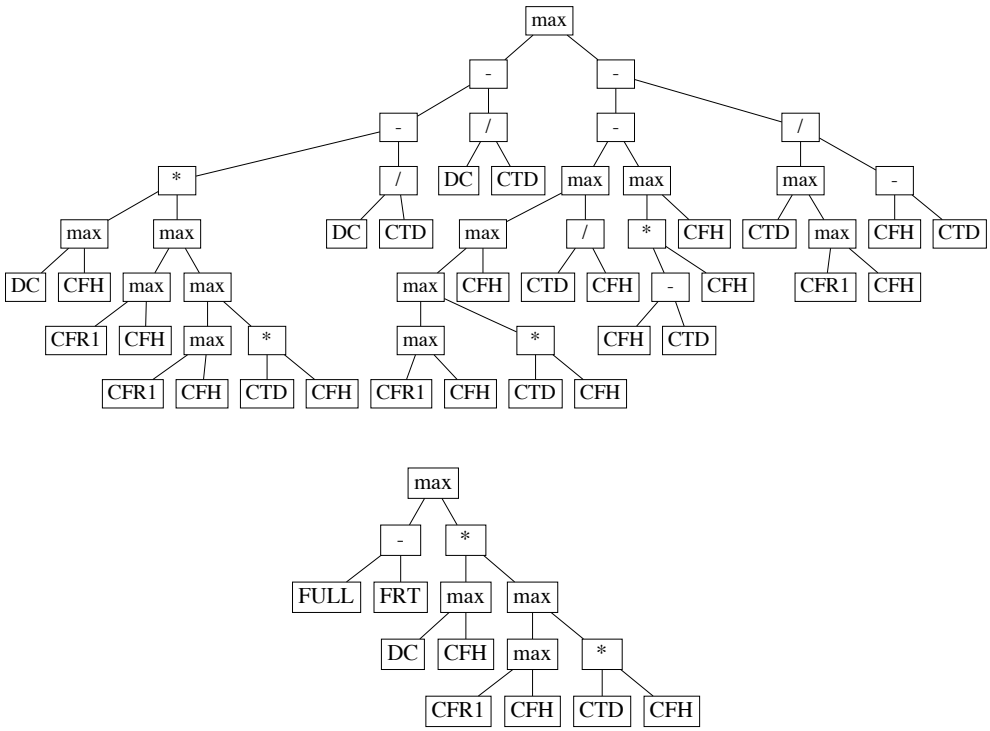


Figure 3.4: Two trees with the same fitness but different tree sizes.

value. Then, top individuals are returned, and they can survive to the next generation.

3.2.6 Niching Tournament Selection

In GP, the parents are typically selected by tournament selection. In GPHH-N, each niche can contain many individuals with identical (good) fitness, and traditional tournament selection tends to select identical parents. To avoid such a loss of diversity, we modified the tournament selection operator. The new niching tournament selection selects only from the representative archive so that the parents for crossover are more likely to be different. We set the probability of selecting a representative proportional

to its niche size using the following equation,

$$P(rep_i) = \frac{H_i^\alpha}{\sum_{i=1}^{\mathcal{N}} H_i^\alpha}, \alpha \in [0, 1] \tag{3.2}$$

where H_i stands for the number of individuals in $niche_i$, $\mathcal{N}$ refers to the total number of niches. α $(0 \leq \alpha \leq 1)$ is a parameter to control the balance between exploration and exploitation. The H_i^α represents the α power of H_i. When α equals 1, the niching tournament selection becomes the traditional tournament selection, which selects parents directly from the simplified population. On the other hand, when α equals 0, each representative will have the same probability of selection regardless of the number of individuals in its niche. A parameter sensitivity analysis on α will be given in Section 3.4.1.

3.2.7 Multi-Source Breeding

The niching simplification process may lead to the loss of the potential useful building blocks in the large trees. To compensate for such loss, we design a multi-source breeding scheme. The multi-source breeding produces offspring from two sources, i.e. the original population and the representative archive. Each source is used to produce half of the offspring population. The pseudocode of the multi-source breeding is shown in Algorithm 4. We breed offspring from two sources so that each source will produce half of the population in the next generation.

3.3 Experiment Design

To verify the effectiveness of the proposed GPHH-N, we compare it with the baseline GPHH (which evolves routing policies using GPHH without any simplification methods) [122], GPHH with the algebraic simplification (GPHH-A) [219]. Bloat control approaches are most commonly used

Algorithm 4: The multi-source breeding

Input: Original population pop, Representative archive $archive$, population size S, elitism number k
Output: New population $newpop$
Add k best individuals from pop to $newpop$;
while $size(newpop) < S/2$ **do**
  Select parents from pop using $traditional\ tournament\ selection$;
  Generate an offspring by applying (crossover/mutation/reproduction operators);
  Add the offspring to $newpop$;
end
Add k best individuals from $archive$ to $newpop$;
while $size(newpop) < S$ **do**
  Select parents from $archive$ using $niching\ tournament\ selection$;
  `// Eq.(3.2) applied`
  Generate an offspring by applying (crossover/mutation/reproduction operators);
  Add the offspring to $newpop$;
end
return $newpop$;

to consider both effectiveness and tree size in GP. In this case, we compare our GPHH-N with three representative bloat control approaches, i.e. Tarpeian, linear parametric parsimony pressure (LPPP), double tournament (DT) in the literature [130]. The comparisons are conducted within the scope of GP approaches. First, the purpose of the experiments is to verify whether our new GP algorithm can obtain both effective and small routing policies. The tree size can only be compared with other GP approaches. Second, the previous study [135] has already shown that GPHH is a competitive approach for UCARP, which is no worse than the state-of-the-art non-GP approaches such as EDASLS [193].

For each UCARP instance, we randomly generate 500 training samples and 500 test samples by randomly sampling the stochastic task demands and deadheading costs. The samples share the same graph topology of the

UCARP instance but different realised values for the stochastic demands and deadheading costs.

A GP run on a UCARP instance contains training and test phases. In the training phase, routing policies are trained on 500 training samples. These samples are split into 100 batches, each containing 5 samples ($\mathcal{S}'$ in Eq. 3.1). A different batch is used for the training in each of the 100 generations. The best routing policy in the final population is returned as the trained policy. In the test phase, the trained routing policy is tested on the 500 test samples. The test effectiveness of a routing policy on the UCARP instance is defined as the average total cost over the 500 test samples, calculated as follows.

$$\text{Effectiveness}(rp) = \frac{1}{|\mathcal{S}_{\text{test}}|} \sum_{s \in \mathcal{S}_{\text{test}}} tc(rp, s), \tag{3.3}$$

where $\mathcal{S}_{\text{test}}$ is the test set, and $|\mathcal{S}_{\text{test}}| = 500$ is the size of the test set. Fig. 3.5 shows the training and test phases of one GP run on one UCARP instance.

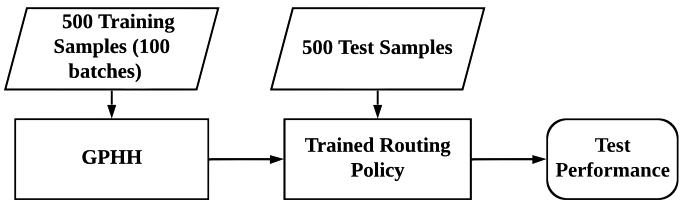


Figure 3.5: The training and test phases of a GP Run on a UCARP Instance.

3.3.1 Dataset

In this work, we use the *Ugdb* and *Uval* datasets, which are commonly used in UCARP literature [122, 123, 133, 143]. They are extended from *gdb* and *val* which are well known static CARP datasets. Details of the datasets have been discussed in Table 2.1.

3.3.2 Specific Parameter Setting

For GPHH approaches, the Parameter Setting has been discussed in Section 2.5.1. For the compared bloat control methods, the parameters are set according to the literature [130]. The simplification rules for GPHH-A are set according to the literature [219]. GPHH refers to the state-of-the-art GPHH method for UCARP [143]. The meanings of each terminal can be found in Table 2.3

We use Evolutionary Computation Java (ECJ) package [131] to implement all the algorithms. For each UCARP instance, each compared algorithm is run 30 times independently (each run trains a routing policy on the 500 training samples, and then test it on the 500 test samples).

3.4 Results and Further Analysis

We compared the algorithms using the Wilcoxon rank sum test with the significance level of 0.05. In the tables, "+", "-" or "=" next to each compared algorithm indicates that the compared algorithm performed statistically significantly better than, worse than, or comparable to GPHH-N.

3.4.1 Parameter Sensitive Analysis On α

The parameter α is important in balancing exploration and exploitation in the niching tournament selection. To set the α value, we compared the GPHH-N with 3 different α values, i.e. 0, 0.5 and 1.

Tables 3.1 and 3.2 show the results of the pairwise comparison among GPHH-N-0.0, GPHH-N-0.5 and GPHH-N-1.0. Each entry is represented in the Win-Draw-Lose format. Win (Lose) indicates the number of instances where the row approach performs significantly better (worse) than the column approach. Draw indicates the number of instances where the two approaches show no significant difference. Table 3.1 shows the results in terms of the effectiveness, and Table 3.2 shows the results in terms of the

tree size. From Table 3.1, we can see that both GPHH-N-0.0 and GPHH-N-0.5 achieved the best test effectiveness among the compared algorithms. From Table 3.2, GPHH-N-0.5 achieved significantly smaller tree size than GPHH-N-0.0. Overall, GPHH-N-0.5 is the best in terms of the effectiveness and the tree size.

Table 3.1: The Win-Draw-Lose table for the pairwise comparisons between different α values in terms of **effectiveness(average total cost)**.

Approach	**GPHH-N-1.0**	**GPHH-N-0.5**	**GPHH-N-0.0**
GPHH-N-1.0		0-29-28	1-20-36
GPHH-N-0.5	28-29-0		2-50-5
GPHH-N-0.0	36-20-1	5-50-2	

Table 3.2: The Win-Draw-Lose table for the pairwise comparisons between different α values in terms of **tree size**.

Approach	**GPHH-N-1.0**	**GPHH-N-0.5**	**GPHH-N-0.0**
GPHH-N-1.0		57-0-0	57-0-0
GPHH-N-0.5	0-0-57		55-0-2
GPHH-N-0.0	0-0-57	2-0-55	

Fig. 3.6 shows the scatter plot of GPHH-N with different α values on a representative instance *Uval2B*. Each shape represents the mean tree size and Effectiveness of the 30 independent runs. We can see that GPHH-N-0.5 and GPHH-N-0.0 achieved much better effectiveness than GPHH-N-1.0. In addition, GPHH-N-0.5 obtained a much smaller tree size than GPHH-N-0.0. Therefore, $\alpha = 0.5$ is used in the subsequent experiments.

3.4.2 Effectiveness

Tables 3.3 and 3.4 show the mean and standard deviation for the effectiveness (average total cost) over the 30 independent runs on *Ugdb* and *Uval* instances.

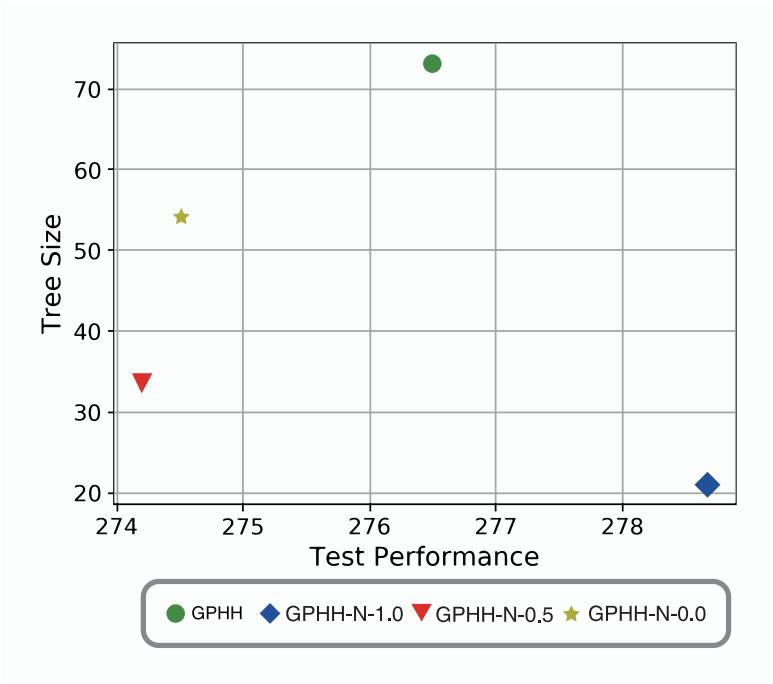


Figure 3.6: Scatter plot of GPHH-N with different α value on *Uval2B*

On the *Ugdb* dataset, we can see that GPHH-N significantly outperformed all the compared algorithms. It performed significantly better than GPHH on 11 out of 23 instances and never performed worse than GPHH. Besides, GPHH-N significantly outperformed GPHH-A on 9 out of 23 instances while was defeated by GPHH-A on only 2 instances. GPHH-N performed significantly better than Tarpeian on 19 out of 23 instances and slightly worse on only 1 instance. GPHH-N was defeated by DT on only 1 instance. However, it outperformed DT on 10 out of 23 instances. GPHH-N outperformed LPPP on all the instances except *Ugdb16*. On average, GPHH-N (283.85) performed the best among all the compared approaches on the *Ugdb* dataset.

The same pattern can be observed for the *Uval* dataset. GPHH-N significantly outperformed GPHH on 22 out of 34 instances while never performed worse. GPHH-N outperformed GPHH-A on 24 instances while

Table 3.3: The mean and standard deviation for **Effectiveness (Average total cost)** of compared approaches on the **Ugdb** instances of 30 runs.

Instance	GPHH	GPHH-A	Tarpeian	DT	LPPP	GPHH-N
Ugdb1	351.25(14.66)(-)	352.1(12.22)(-)	360.34(15.85)(-)	359.03(24.37)(-)	360.89(8.73)(-)	344.12(6.23)
Ugdb2	367.73(4.77)(=)	369.51(12.19)(=)	374.35(9.26)(-)	369.88(6.38)(-)	383.72(10.26)(-)	367.49(13.31)
Ugdb3	307.03(2.87)(=)	306.35(3.1)(=)	308.69(2.55)(=)	307.03(4.3)(=)	312.42(5.27)(-)	307.63(3.64)
Ugdb4	324.9(6.42)(=)	321.95(3.08)(+)	325.91(3.15)(-)	322.87(4.33)(=)	329.18(10.06)(-)	323.43(4.13)
Ugdb5	422.17(6.09)(=)	426.57(15.3)(=)	429.59(8.93)(-)	425.49(9.32)(-)	450.11(14.66)(-)	422.22(13.32)
Ugdb6	344.3(8.37)(-)	344.9(9.11)(-)	359.26(7.17)(-)	342.51(6.46)(-)	362.24(0.0)(-)	337.6(5.27)
Ugdb7	353.24(4.41)(-)	352.58(3.67)(=)	359.59(0.68)(-)	354.95(4.05)(-)	359.85(0.0)(-)	351.68(4.68)
Ugdb8	430.79(8.16)(-)	436.28(37.62)(-)	439.64(11.37)(-)	427.71(7.06)(=)	451.64(13.39)(-)	425.96(5.95)
Ugdb9	389.02(9.86)(-)	387.8(9.17)(-)	397.22(10.41)(-)	392.48(14.55)(-)	408.01(13.95)(-)	380.47(10.02)
Ugdb10	293.0(7.22)(=)	293.75(7.29)(=)	298.22(5.56)(-)	293.79(7.58)(=)	298.7(2.09)(-)	291.47(7.01)
Ugdb11	433.32(8.52)(-)	435.88(6.31)(-)	446.76(7.19)(-)	437.03(6.03)(-)	446.48(4.31)(-)	429.82(11.32)
Ugdb12	604.8(17.36)(=)	606.66(15.48)(=)	618.57(13.27)(-)	604.82(18.0)(=)	622.06(15.8)(-)	604.19(13.94)
Ugdb13	577.11(8.53)(=)	580.12(9.02)(=)	586.62(8.97)(-)	578.01(6.93)(=)	599.42(15.69)(-)	574.93(7.15)
Ugdb14	107.02(1.31)(=)	110.24(12.68)(=)	108.25(1.59)(-)	107.75(2.58)(=)	117.81(2.88)(-)	106.8(1.27)
Ugdb15	58.34(0.81)(=)	58.26(0.19)(-)	58.26(0.24)(-)	58.27(0.41)(=)	62.01(0.0)(-)	58.11(0.09)
Ugdb16	134.52(0.55)(=)	134.64(0.09)(=)	134.51(0.07)(+)	134.61(0.11)(=)	134.47(0.0)(+)	134.91(1.92)
Ugdb17	91.47(1.9)(-)	91.08(0.12)(+)	91.23(0.09)(=)	91.18(0.46)(+)	93.82(0.48)(-)	91.29(0.35)
Ugdb18	167.49(1.77)(-)	167.62(3.06)(-)	168.81(2.19)(-)	168.18(5.61)(-)	180.78(5.52)(-)	166.06(0.92)
Ugdb19	63.29(1.61)(=)	64.16(1.47)(=)	63.95(1.61)(=)	63.66(1.56)(=)	67.81(1.07)(-)	63.63(1.41)
Ugdb20	127.09(1.57)(=)	127.19(1.53)(=)	128.85(2.05)(-)	128.0(4.57)(=)	137.31(0.0)(-)	126.67(2.16)
Ugdb21	164.74(2.27)(-)	167.54(17.58)(-)	166.29(2.96)(-)	165.35(2.82)(-)	181.52(5.49)(-)	163.25(1.58)
Ugdb22	210.11(2.22)(-)	209.26(1.38)(=)	211.01(3.61)(-)	209.28(1.69)(=)	221.46(2.44)(-)	209.09(1.94)
Ugdb23	250.62(2.98)(-)	251.07(4.16)(-)	251.66(2.49)(-)	249.52(1.86)(-)	258.34(6.08)(-)	247.73(2.13)
Average	285.80	286.76	290.76	286.58	297.39	283.85

showed comparable results on the remaining 10 instances. GPHH-N beat Tarpeian on 32 out of 34 instances but never showed significantly worse effectiveness. GPHH-N also significantly outperformed DT on 20 out of 34 instances and showed comparable effectiveness on the other instances. GPHH-N beat LPPP on all the instances. The average effectiveness of GPHH-N (384.27) is better than all the other approaches.

3.4.3 Tree Size

Tables 3.5 and 3.6 show the mean and standard deviation for the tree size of routing policies of the compared algorithms on *Ugdb* and *Uval* instances.

In Table 3.5, we can see that GPHH-N can evolve much smaller routing policies than GPHH on 18 out of 23 instances on the $Ugdb$ dataset. GPHH-

Table 3.4: The mean and standard deviation for **Effectiveness (Average total cost)** of compared approaches on the **Uval** instances of 30 runs.

Instance	GPHH	GPHH-A	Tarpeian	DT	LPPP	GPHH-N
Uval1A	175.43(2.73)(=)	176.47(3.33)(-)	180.75(2.32)(-)	176.51(6.15)(=)	188.73(6.65)(-)	174.37(1.71)
Uval1B	184.2(2.73)(=)	183.87(1.46)(=)	185.65(2.44)(-)	183.74(1.3)(=)	190.04(3.09)(-)	183.69(1.29)
Uval1C	311.93(10.22)(-)	313.69(10.77)(-)	318.29(5.53)(-)	314.01(9.08)(-)	328.39(5.82)(-)	306.5(12.67)
Uval2A	228.68(1.7)(=)	229.37(3.36)(=)	230.97(3.46)(=)	229.06(2.77)(=)	237.65(4.18)(-)	229.25(2.48)
Uval2B	276.49(4.06)(-)	277.95(4.12)(-)	279.74(5.18)(-)	278.63(3.25)(-)	293.33(15.04)(-)	274.2(3.36)
Uval2C	593.34(23.87)(-)	592.11(16.74)(-)	618.37(26.69)(-)	593.6(22.97)(-)	615.61(14.72)(-)	585.09(32.83)
Uval3A	82.18(1.47)(=)	82.44(1.32)(=)	82.36(0.91)(-)	83.63(4.19)(=)	88.37(2.15)(-)	81.97(0.66)
Uval3B	96.04(2.22)(-)	97.13(2.39)(-)	100.0(3.71)(-)	95.91(1.7)(-)	101.36(0.67)(-)	94.09(1.45)
Uval3C	176.45(7.56)(-)	176.18(5.65)(-)	177.49(7.76)(-)	175.66(5.63)(-)	191.54(14.79)(-)	171.31(4.22)
Uval4A	420.29(9.25)(-)	419.32(6.39)(-)	426.99(18.41)(-)	418.89(6.68)(-)	430.38(9.0)(-)	415.05(3.25)
Uval4B	440.85(5.92)(-)	440.07(5.77)(=)	449.25(12.83)(-)	441.81(7.28)(-)	452.09(10.8)(-)	437.64(4.84)
Uval4C	490.73(12.69)(-)	487.45(10.77)(-)	496.52(11.98)(-)	488.95(12.07)(-)	502.05(11.07)(-)	481.42(7.47)
Uval4D	699.52(32.49)(-)	699.17(41.57)(-)	723.26(32.38)(-)	694.53(30.51)(-)	726.66(39.43)(-)	679.62(24.65)
Uval5A	440.36(3.98)(-)	441.58(4.16)(-)	444.51(4.6)(-)	439.69(5.3)(=)	450.26(4.75)(-)	437.46(4.75)
Uval5B	469.81(5.51)(-)	469.55(5.51)(-)	477.66(12.84)(-)	473.99(20.39)(-)	480.97(11.09)(-)	465.81(4.25)
Uval5C	513.04(5.46)(-)	514.82(8.29)(-)	518.32(7.1)(-)	514.65(7.88)(-)	531.0(12.04)(-)	508.33(4.32)
Uval5D	723.04(14.21)(=)	725.36(27.47)(=)	750.36(24.91)(-)	724.44(16.5)(=)	749.13(25.04)(-)	720.0(14.98)
Uval6A	229.0(2.36)(=)	230.67(10.98)(=)	229.39(2.19)(-)	228.69(2.95)(=)	232.31(5.11)(-)	228.36(2.89)
Uval6B	257.32(3.71)(=)	257.53(4.5)(=)	259.23(3.15)(-)	256.1(3.5)(=)	266.56(9.88)(-)	255.71(4.6)
Uval6C	400.99(11.4)(-)	403.55(12.32)(-)	422.21(22.31)(-)	402.19(11.42)(-)	428.03(23.79)(-)	395.1(8.26)
Uval7A	289.74(15.08)(=)	294.17(20.28)(-)	289.73(5.6)(-)	288.03(9.94)(=)	298.57(6.06)(-)	287.14(10.39)
Uval7B	293.51(8.59)(=)	298.33(20.17)(-)	297.86(8.31)(-)	293.96(6.33)(-)	308.95(9.0)(-)	289.62(5.3)
Uval7C	405.06(6.32)(=)	404.95(8.67)(=)	415.7(18.46)(-)	405.42(8.09)(=)	414.11(11.49)(-)	402.77(5.21)
Uval8A	398.75(8.67)(-)	400.7(18.68)(-)	398.24(2.34)(-)	397.08(1.84)(-)	404.15(4.65)(-)	396.17(4.7)
Uval8B	426.22(6.28)(-)	425.1(5.71)(-)	432.02(10.94)(-)	425.21(6.16)(-)	437.98(6.99)(-)	421.27(4.98)
Uval8C	664.62(19.93)(-)	662.56(18.46)(-)	682.25(15.4)(-)	657.38(14.87)(=)	688.07(15.69)(-)	651.12(18.18)
Uval9A	333.75(3.75)(-)	335.02(6.79)(-)	336.69(3.47)(-)	333.52(2.04)(-)	342.35(6.31)(-)	330.88(2.36)
Uval9B	348.89(4.77)(-)	349.08(4.68)(-)	351.79(4.28)(-)	349.35(4.07)(-)	356.58(5.8)(-)	345.14(4.46)
Uval9C	363.89(6.27)(-)	364.33(4.45)(-)	369.69(6.37)(-)	361.15(4.06)(=)	373.94(5.01)(-)	360.36(5.16)
Uval9D	476.86(10.82)(-)	475.31(9.89)(=)	486.36(18.25)(-)	478.3(12.22)(-)	491.21(9.41)(-)	469.8(12.01)
Uval10A	439.33(4.74)(=)	441.52(19.54)(-)	441.37(4.74)(-)	438.55(3.63)(=)	458.51(20.73)(-)	436.58(1.09)
Uval10B	458.65(7.58)(-)	459.46(5.03)(-)	461.27(4.45)(-)	458.54(4.98)(-)	471.01(17.59)(-)	453.76(4.0)
Uval10C	479.37(6.92)(-)	478.5(6.38)(-)	482.29(6.73)(-)	478.85(5.93)(-)	485.69(10.49)(-)	474.86(6.0)
Uval10D	622.17(16.77)(=)	619.15(6.68)(=)	623.39(23.78)(=)	619.56(7.05)(=)	628.2(10.06)(-)	620.74(15.32)
Average	388.54	389.01	395.29	388.22	401.29	384.27

Table 3.5: The mean and standard deviation for **tree size** of routing policies of the compared algorithms on **Ugdb** instances of 30 runs.

Instance	GPHH	GPHH-A	Tarpeian	DT	LPPP	GPHH-N
Ugdb1	83.0(21.64)(-)	74.8(19.29)(-)	21.4(12.68)(+)	83.2(25.28)(-)	7.47(3.14)(+)	50.0(16.13)
Ugdb2	88.2(23.16)(-)	62.87(14.11)(-)	24.13(10.46)(+)	81.67(21.61)(-)	8.67(3.45)(+)	51.33(20.92)
Ugdb3	69.13(23.69)(-)	63.67(19.22)(-)	18.8(10.18)(+)	73.93(25.56)(-)	6.53(1.25)(+)	34.13(17.46)
Ugdb4	66.0(28.63)(-)	65.6(30.39)(-)	16.27(7.6)(+)	64.8(21.08)(-)	8.47(3.86)(+)	45.2(16.53)
Ugdb5	80.6(19.27)(-)	70.13(20.0)(-)	25.13(11.31)(+)	91.87(23.2)(-)	7.0(3.06)(+)	49.47(23.74)
Ugdb6	66.13(16.19)(-)	57.67(18.81)(-)	8.8(19.42)(+)	72.47(24.81)(-)	1.0(0.0)(+)	41.73(17.74)
Ugdb7	43.4(17.63)(-)	35.07(20.06)(=)	2.13(2.81)(+)	44.13(17.97)(-)	1.0(0.0)(+)	32.73(14.35)
Ugdb8	62.33(19.67)(=)	65.33(16.77)(=)	26.27(15.83)(+)	79.8(27.69)(-)	8.27(4.15)(+)	65.0(21.69)
Ugdb9	66.53(21.91)(=)	67.4(20.17)(=)	24.27(16.46)(+)	68.2(18.71)(=)	8.4(3.9)(+)	69.27(20.43)
Ugdb10	64.6(21.4)(-)	60.87(20.74)(-)	14.4(13.52)(+)	63.13(22.69)(-)	5.13(0.51)(+)	29.93(18.77)
Ugdb11	51.27(31.21)(=)	54.6(17.94)(-)	8.73(12.02)(+)	54.53(18.23)(-)	1.73(2.32)(+)	38.67(15.1)
Ugdb12	75.0(20.6)(-)	66.33(17.07)(-)	25.13(11.87)(+)	85.53(28.04)(-)	14.0(4.54)(+)	48.47(17.08)
Ugdb13	69.4(22.05)(-)	66.0(19.76)(-)	17.13(6.39)(+)	64.07(20.08)(=)	8.47(2.97)(+)	54.27(20.55)
Ugdb14	80.4(26.39)(-)	62.13(16.87)(-)	20.33(9.6)(+)	75.53(22.11)(-)	1.53(1.38)(+)	42.53(24.73)
Ugdb15	96.0(30.83)(-)	71.07(32.92)(-)	21.07(12.3)(=)	91.33(39.31)(-)	1.0(0.0)(+)	22.8(16.04)
Ugdb16	19.0(17.62)(=)	10.27(5.52)(=)	2.67(4.07)(+)	28.0(14.86)(-)	1.0(0.0)(+)	20.6(22.37)
Ugdb17	85.73(35.77)(-)	55.67(20.89)(-)	10.87(18.69)(+)	80.47(41.31)(-)	1.07(0.37)(+)	31.53(36.35)
Ugdb18	80.07(29.18)(-)	75.4(25.14)(-)	25.0(13.54)(+)	92.87(36.2)(-)	5.33(2.58)(+)	47.2(23.31)
Ugdb19	71.8(27.87)(-)	71.87(33.84)(-)	16.27(14.05)(+)	74.33(31.86)(-)	1.4(0.81)(+)	26.47(17.58)
Ugdb20	78.93(28.74)(-)	63.53(20.6)(-)	19.8(14.16)(+)	85.53(31.95)(-)	1.0(0.0)(+)	41.27(19.22)
Ugdb21	81.07(32.49)(-)	67.53(19.86)(-)	20.6(9.53)(+)	79.8(31.31)(-)	2.73(2.77)(+)	53.6(25.61)
Ugdb22	80.87(26.72)(-)	67.33(15.09)(-)	23.07(9.24)(+)	78.87(22.34)(-)	1.4(1.22)(+)	55.47(17.49)
Ugdb23	81.33(28.09)(=)	71.13(20.14)(=)	24.73(12.65)(+)	81.2(20.73)(=)	5.4(2.54)(+)	73.2(26.88)
Average	71.33	62.01	18.13	73.70	4.70	44.56

N can also obtain smaller routing policies than GPHH-A on 18 out of 23 instances. In Table 3.6, the results are consistent with that in Table 3.5. GPHH-N can evolve much smaller routing policies than GPHH on 24 out of 34 instances on the $Uval$ dataset. It can evolve much smaller routing policies than GPHH-A on 19 out of 34 instances.

Note that Tarpeian and LPPP achieved much smaller routing policies on both datasets. However, their effectiveness is much worse than GPHH-N in most instances. The tree size of routing policies evolved by DT is comparable with that in GPHH on both datasets. This is because DT firstly selects the parents based on the fitness and then considers the tree size. Thus, if all the good individuals selected from the first tournament selection are

Table 3.6: The mean and standard deviation for **tree size** of routing policies of the compared algorithms on **Uval** instances of 30 runs.

Instance	GPHH	GPHH-A	Tarpeian	DT	LPPP	GPHH-N
Uval1A	68.2(24.45)(-)	66.0(26.61)(-)	11.33(6.1)(+)	77.47(27.33)(-)	3.73(2.55)(+)	29.0(19.81)
Uval1B	67.0(20.1)(-)	58.8(15.63)(-)	18.73(9.26)(=)	67.6(22.71)(-)	6.53(2.21)(+)	23.2(10.11)
Uval1C	69.13(20.58)(=)	65.27(25.22)(=)	24.4(12.68)(+)	82.67(29.53)(=)	8.4(3.45)(+)	69.73(14.58)
Uval2A	73.07(29.08)(-)	69.67(21.98)(-)	19.87(8.27)(=)	76.4(29.6)(-)	7.53(1.66)(+)	31.27(27.17)
Uval2B	73.2(23.7)(-)	72.67(22.92)(-)	19.67(9.93)(+)	76.27(30.29)(-)	6.93(4.05)(+)	33.67(23.4)
Uval2C	81.73(31.01)(=)	64.2(24.4)(=)	23.0(11.72)(+)	68.67(20.69)(=)	13.33(6.06)(+)	72.2(19.26)
Uval3A	72.2(25.59)(-)	65.6(22.63)(-)	27.6(12.87)(+)	77.33(24.2)(-)	5.6(1.19)(+)	36.73(15.87)
Uval3B	68.87(32.24)(-)	67.13(25.94)(-)	11.87(8.92)(+)	72.47(19.43)(-)	3.13(0.51)(+)	41.07(19.93)
Uval3C	75.6(19.26)(-)	69.67(17.05)(=)	27.13(12.94)(+)	84.67(36.86)(-)	8.0(3.35)(+)	63.6(18.07)
Uval4A	68.27(22.76)(-)	68.33(22.54)(-)	22.27(11.54)(+)	76.73(25.47)(-)	8.13(2.45)(+)	47.2(16.13)
Uval4B	72.53(20.41)(-)	60.93(19.11)(=)	25.67(13.97)(+)	70.6(20.15)(-)	11.47(3.95)(+)	54.33(19.05)
Uval4C	68.8(19.02)(=)	66.87(21.05)(=)	26.6(10.54)(+)	77.2(23.41)(-)	12.6(3.12)(+)	64.33(19.84)
Uval4D	66.8(22.98)(=)	66.6(19.79)(=)	29.2(12.65)(+)	70.67(23.35)(=)	16.8(8.78)(+)	70.4(18.52)
Uval5A	68.33(20.16)(-)	68.4(20.92)(-)	21.87(10.14)(+)	68.53(19.31)(-)	6.87(2.4)(+)	40.47(16.47)
Uval5B	84.8(33.43)(-)	70.07(23.32)(-)	28.27(14.21)(+)	80.53(29.0)(-)	10.4(3.57)(+)	52.67(17.61)
Uval5C	63.67(18.48)(-)	60.67(24.65)(-)	21.07(10.3)(+)	64.8(21.97)(-)	11.0(4.39)(+)	46.87(17.08)
Uval5D	70.93(20.65)(=)	64.8(19.2)(=)	19.73(9.73)(+)	72.6(23.76)(=)	12.47(5.12)(+)	66.33(15.85)
Uval6A	77.87(25.58)(-)	73.87(28.8)(-)	23.47(11.79)(+)	74.33(24.76)(-)	9.53(3.6)(+)	39.87(14.09)
Uval6B	74.27(23.92)(-)	62.0(18.58)(-)	22.8(10.28)(+)	71.6(17.94)(-)	9.67(3.8)(+)	47.13(17.4)
Uval6C	66.67(21.32)(=)	63.93(19.06)(=)	26.27(12.54)(+)	74.07(20.5)(-)	9.73(6.23)(+)	63.67(16.71)
Uval7A	73.07(25.02)(-)	67.4(22.51)(-)	27.27(16.71)(+)	76.93(17.78)(-)	10.13(2.91)(+)	40.07(22.85)
Uval7B	74.6(22.3)(-)	66.87(22.17)(-)	22.0(8.55)(+)	80.07(25.13)(-)	10.33(4.28)(+)	37.67(15.57)
Uval7C	73.67(21.13)(=)	64.6(15.24)(=)	25.53(10.94)(+)	75.47(21.46)(-)	14.4(5.07)(+)	63.8(23.58)
Uval8A	70.07(18.06)(-)	65.13(18.99)(-)	24.87(21.78)(+)	66.6(18.44)(-)	7.47(1.87)(+)	43.33(17.63)
Uval8B	68.33(22.62)(-)	63.13(20.36)(-)	18.4(12.87)(+)	66.33(20.56)(-)	7.0(3.28)(+)	45.87(15.9)
Uval8C	73.6(18.66)(-)	68.53(19.25)(=)	27.73(14.53)(+)	71.2(21.27)(=)	14.93(4.65)(+)	62.07(16.31)
Uval9A	68.67(19.55)(-)	70.33(21.3)(-)	22.2(10.99)(+)	69.47(22.59)(-)	8.27(3.3)(+)	48.67(20.53)
Uval9B	60.4(17.84)(=)	64.67(20.89)(=)	25.33(12.55)(+)	69.2(21.97)(-)	7.87(2.27)(+)	53.8(21.86)
Uval9C	69.2(30.56)(=)	55.73(15.96)(=)	24.33(14.18)(+)	84.13(29.38)(-)	8.2(3.18)(+)	61.0(17.9)
Uval9D	80.73(30.09)(-)	76.2(22.81)(-)	29.33(12.95)(+)	77.8(24.92)(-)	14.87(3.48)(+)	62.07(15.83)
Uval10A	62.73(18.84)(-)	58.0(19.28)(-)	22.13(12.39)(+)	62.13(23.1)(-)	5.87(3.14)(+)	38.2(19.19)
Uval10B	70.87(21.73)(-)	60.73(21.42)(=)	21.73(11.71)(+)	68.4(22.21)(-)	8.73(4.06)(+)	56.27(20.32)
Uval10C	65.73(18.13)(-)	63.2(17.34)(=)	26.27(12.64)(+)	70.4(18.58)(-)	8.53(2.81)(+)	55.07(15.14)
Uval10D	67.8(20.45)(=)	66.07(22.53)(=)	31.27(14.69)(+)	77.4(21.98)(-)	16.33(4.4)(+)	63.93(21.05)
Average	70.93	65.77	23.51	73.55	9.55	50.76

large, there is no chance to select small individuals in the second tournament selection process. Overall, GPHH-N performed well on the tree size, which is expected, as it can effectively remove redundant components in GP trees.

3.4.3.1 Effectiveness Vs Tree Size

To show the results more clearly, we also plot the results in a scatter map, where the x-axis is mean effectiveness on 30 independent runs and the y-axis is the mean tree size. Fig. 3.7 shows the scatter plot on Ugdb1 and Uval2B. Each compared algorithm is represented as a single dot with a specific shape in each figure.

From Fig. 3.7, we can see that GPHH-N dominates all the other methods except Tarpeian and LPPP, which achieve smaller tree size but much worse effectiveness. Consider that the effectiveness is relatively more important than the tree size, we can see that GPNN-N is better than all the other compared algorithms.

3.4.3.2 Training Time

Table 3.7 shows the mean training time of GPHH, GPHH-A and GPHH-N on *Ugdb* and *Uval* instances. As expected, both GPHH-A and GPHH-N can significantly reduce the training time on most instances. This is mainly because the simplification operation can remove the redundant components. The simplified routing policies need less time to be evaluated. We can see that GPHH-N can further reduce training time comparing with GPHH-A. This is mainly because GPHH-N can remove more redundant components than GPHH-A and makes evolved trees smaller.

Overall, we can see obvious advantage of GPHH-N over GPHH, GPHH-A and the compared bloat control methods. GPHH-N can obtain better and smaller routing policies in a shorter training time.

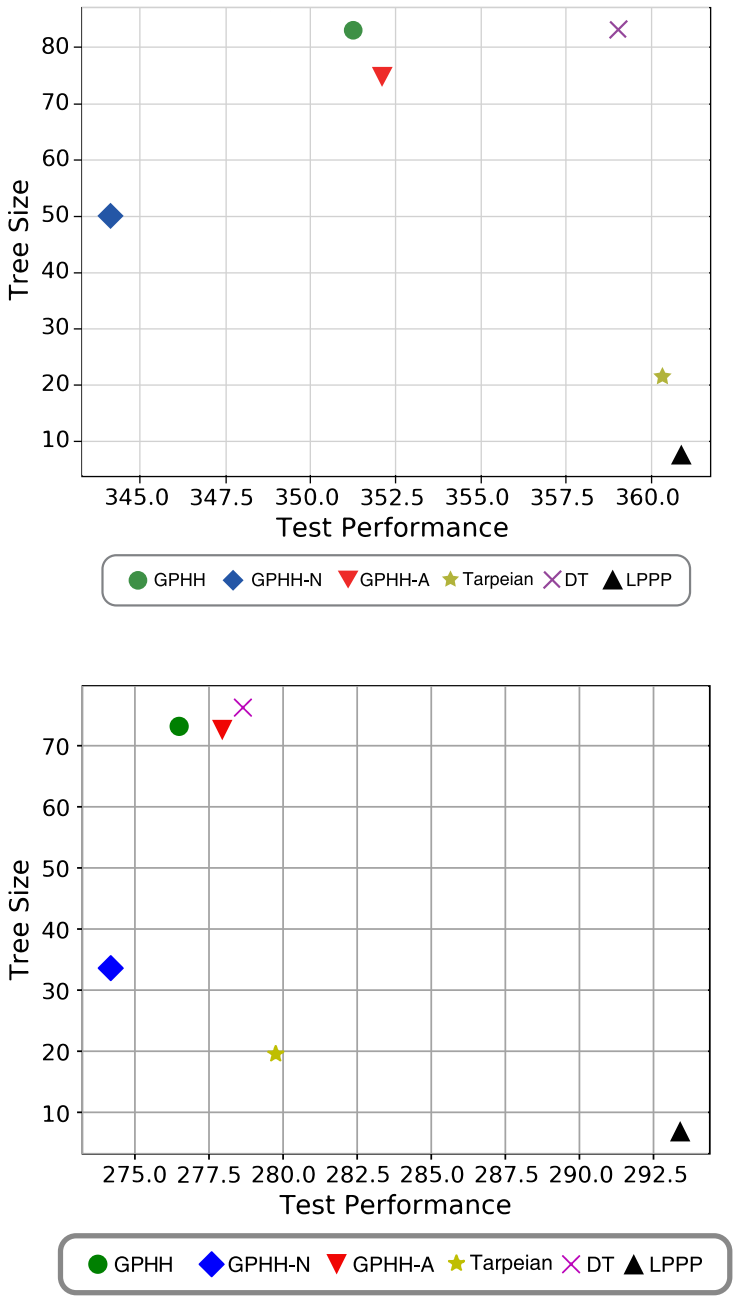


Figure 3.7: Scatter map of compared approaches on representative instances (Top one is on Ugdb1, bottom is on Uval2B).

Table 3.7: The mean **training time** (seconds) of the compared algorithms on **Ugdb** and **Uval** datasets.

Dataset	GPHH	GPHH-A	GPHH-N
Ugdb (Average)	1027.86	901.05	863.07
Uval (Average)	7794.32	7385.92	7178.14

3.5 Further Analysis

3.5.1 Effect of Each Component

It has been shown that GPHH-N can evolve both better and smaller routing policies. There are four main components in GPHH-N. They are the niching simplification (Section 3.2.4), niching elitism scheme (Section 3.2.5), niching tournament selection (Section 3.2.6) and multi-source breeding (Section 3.2.7). In order to verify the effectiveness of each component of GPHH-N, we designed some controlled experiments. Due to the page limit, we will present the experimental results in the Win-Draw-Lose format. Win (Lose) indicates that GPHH-N can significantly perform better (worse) than the compared algorithm. Draw indicates that GPHH-N can achieve comparable results with the compared algorithm. Table 3.8 shows the results in terms of effectiveness and Table 3.9 shows the results in terms of the tree size of the evolved routing policies.

Table 3.8: The Win-Draw-Lose table for the controlled experiments between the compared algorithms and GPHH-N in terms of **effectiveness (average total cost)**.

	v.s. no niching elitism	**v.s. no multi-source breeding**	**v.s. no niching tournament selection**
W-D-L	1-55-1	15-42-0	28-29-0

From Table 3.8, one can see that the niching tournament selection made the most contribution to GPHH-N in terms of effectiveness. The effectiveness decreases on 28 out of 57 instances when niching tournament selec-

Table 3.9: The Win-Draw-Lose table for the controlled experiments between the compared algorithms and GPHH-N in terms of **tree size**.

	v.s. no niching elitism	**v.s. no multi-source breeding**	**v.s no niching tournament selection**
W-D-L	33-0-24	42-0-15	0-0-57

tion is not used (using traditional tournament selection on the simplified population instead). The multi-source breeding method can make some contributions. The niching elitism does not play a major role in improving the effectiveness.

From Table 3.9, one can observe that the niching tournament selection tends to increase the tree size. The niching tournament selection can improve the diversity of the parent selection process and lead to better effectiveness. However, it will also increase the tree size.

Fig. 3.8 shows the scatter plot of the GPHH-N with and without different components. We can see that all the three components are important in terms of effectiveness. We can see that the dot of GPHH-N without niching tournament selection is located at the bottom right area of the plot. This indicates that niching tournament selection can improve the effectiveness at the cost of larger tree size.

3.5.2 Analysis of Evolved Policies

To gain further understanding of the behaviour of the routing policies, a representative routing policy is selected. Eqs. (3.4) – (3.7) show a selected policy evolved by GPHH-N for the Ugdb19 instance. The policy has a promising effectiveness (63.39, while the mean effectiveness of GPHH-N is 63.63). In addition, it has 21 nodes, which is much smaller than the routing policies evolved by other algorithms with the similar effectiveness. The meanings of each terminal can be found in Table 2.3

$$RP = S_1 + \max(S_2, S_3) \tag{3.4}$$

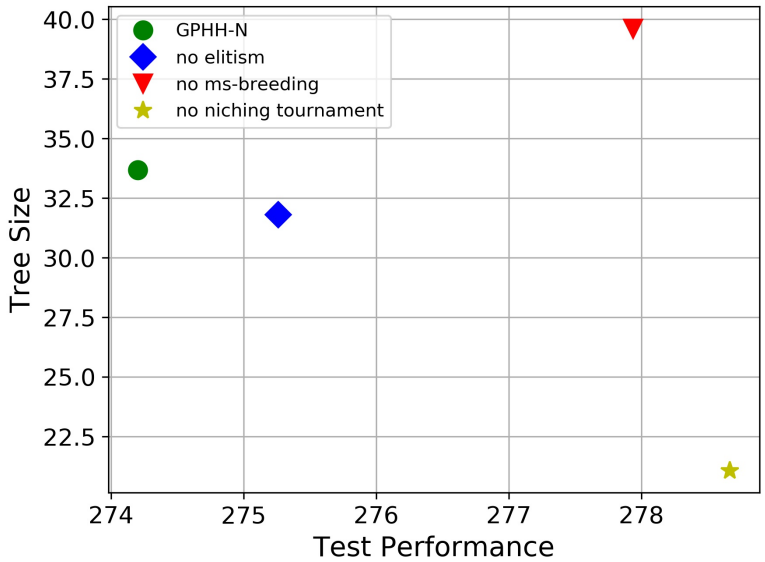


Figure 3.8: Scatter map of controlled experiment on a representative instance *Uval2B*

where

$$S_1 = 2DEM + CFH - CTD \tag{3.5}$$

$$S_2 = DEM + CFH - CTD \tag{3.6}$$

$$S_3 = PUT + RQ - \max(CFR1, CTT1) \tag{3.7}$$

To make it easier to understand, we can also transform RP to the following IF-ELSE format rule set.

if $S_2 \geq S_3$ **then**
  $RP = 3DEM + 2CFH - 2CTD$
else
  if $CFR1 > CTT1$ **then**
    $RP = 2DEM + CFH - CTD + PUT + RQ - CFR1$
  else
    $RP = 2DEM + CFH - CTD + PUT + RQ - CTT1$
  end if
end if

We can identify the following patterns and interpretations from the above rule set.

- When $S_2 \geq S_3$, the RP becomes $RP = 3DEM + 2CFH - 2CTD$. There are two possible cases for $S_2 \geq S_3$ to happen:
  - S_2 is large. This indicates that all the remaining tasks have large demand, far away from the current location of the vehicle and close to the depot;
  - S_3 is small. This indicates that there are not many remaining tasks, the vehicle is almost full, and all the remaining tasks have a large $CFR1$ and $CTT1$ (they are far away from the other vehicles and each other);

  In these two cases, RP prefers the tasks with small demands, close to the current location of the vehicle and far away from the depot.

- Otherwise, $S_2 < S_3$ indicates that the remaining tasks can have small demands, are close to the current location of the vehicle, and far away from the depot, and the vehicle is relatively empty. It also has two possible cases:
  - $CFR1 > CTT1$, this indicates that the task is close to some other tasks, but far away from other vehicles. In this case, in addition to small demands, close to the current location of the vehicle and far away from the depot, RP also prefers the tasks with larger $CFR1$, i.e. farther away from other vehicles;
  - $CFR1 < CTT1$, this indicates the task is far away from other tasks, but can be close to some other vehicles. In this case, in addition to small demands, close to the current location of the vehicle and far away from the depot, RP also prefers the tasks with larger $CTT1$, i.e. far away from other tasks. In other words, RP prefers the isolated tasks towards the beginning of the routes.

3.6 Chapter Summary

The goal of this chapter is to evolve both effective and smaller/simpler routing policies for UCARP. This goal has been successfully achieved by the newly proposed novel GPHH with a simplification approach using a niching technique (GPHH-N). GPHH-N is examined and compared with the basic GPHH approach without simplification (GPHH), the basic GPHH approach with algebraic simplification (GPHH-A) and three representative bloat control methods on 57 UCARP instances. The results suggest that GPHH-N can outperform all the compared approaches in terms of effectiveness. GPHH-N can also outperform GPHH and GPHH-A in terms of tree size and training time. We also analysed the effect of the newly proposed components by a set of controlled experiments. The results showed that all the three new components could contribute to evolve smaller and better routing policies. The niching tournament selection and multi-source breeding components are more effective than the niching elitism component. Overall, GPHH-N can obtain better effectiveness and smaller and potentially more interpretable routing policies than the current state-of-the-art GPHH approach.

Chapter 4

Multi-Objective Genetic Programming Algorithms for Uncertain Capacitated Arc Routing Problem

4.1 Introduction

As discussed in the previous chapter, the GP-evolved routing policies are usually too large and complex thus hard to interpret. A main reason is that GP tends to continuously increase the size of its individuals during the evolutionary process, which is known as *bloat* [49, 162]. A number of strategies have been proposed to deal with bloat [23, 106, 109, 161, 197]. In the previous chapter, we have also proposed a new GP simplification method to reducing the GP tree size to increase the interpretability. In this chapter, we will propose new multi-objective GP (MOGP) method to optimise effectiveness and GP tree size simultaneously. In this case, we can have GP-evolved routing policies with different degrees of interpretability and effectiveness. Different users can have different understanding of

interpretability. Users can pick routing policies based on their preference by using MOGP. In addition, it is possible to get some knowledge from the Pareto front evolved by MOGP method by analysing the difference among GP-evolved routing policies in the Pareto front.

There are two major challenges for developing MOGP to evolve effective and small routing policies for UCARP. First, GP is much more likely to generate small (but ineffective) individuals than effective (and typically large) ones. Under the traditional dominance relation, small ineffective individuals are more likely to be selected as parents and survive into the next generation. This is known as the *objective selection bias* issue [198]. Second, the fitness evaluation for routing policies is stochastic due to the training sample rotation [26, 122], i.e., it is calculated on a different small subset of training instances in each generation. The *stochastic fitness evaluation* issue can cause potentially good routing policies to be discarded if they happen to perform poorly on a small subset of training samples. This phenomenon becomes worse in MOGP, since most potentially good routing policies have large program size.

In this chapter, we have proposed different strategies for addressing the above issues. To handle the objective selection bias issue, firstly, we adapted the α dominance strategy to our MOGP framework as α dominance strategy [90] has the potential to handle the objective selection bias issue. A detail introduction of α dominance strategy will be given in Section 4.1.2. In this case, the newly proposed MOGP algorithm is named αMOGP). The detail of αMOGP is discussed in the Section 4.2. After proposing αMOGP, we found that simply adapted α dominance strategy to our problem only partially handle the objective selection bias issue. Especially, the performance of the algorithm depends on the α value, but it is difficult to find the best α parameter setting in different scenarios. Thus, we proposed a new α value adjustment scheme to automatically adjust the α value based on the bias status during the evolutionary process. This new MOGP algorithm is named self-adaptive αMOGP (αMOGP-sa).

αMOGP-sa can automatically identify whether the Pareto front is bias to size or effectiveness, and then increase/decrease the α value. αMOGP-sa can find the suitable α value for different scenarios. The detail of αMOGP-sa will be discussed in Section 4.3. To handle the stochastic fitness evaluation issue, we proposed an archive strategy. The new archive strategy uses an external archive to store the potentially effective individuals that could have been lost during the traditional GP process. The individuals in the archive contribute back to the population in the breeding process. In this case, we can avoid losing the potentially good individuals in the non-dominant sorting process. This new algorithm is named Two-Stage MOGP with Archive (TSMOGP-a). TSMOGP-a is discussed in Section 4.4.

Finally, we combine all the above strategies together to develop a new MOGP algorithm with the α-dominance and archive strategy which is named αMOGP-a. αMOGP-a aims to address the following limitations of our previous works in MOGP for evolving routing policies. First, the effectiveness of the α-dominance method highly depends on the α value, which is challenging to determine. Intuitively, α should be increased if the population is biased to small ineffective individuals, and decreased if most individuals in the population are too large. However, the α adaptation is non-trivial due to the difficulty of estimating the accurate boundaries of the effectiveness and program size. Second, the existing archive update strategies are not effective enough. It can include duplicate individuals in the archive and reduce the diversity of the archive. To address the issue of sensitivity to the α value, in the αMOGP-a we propose a new scheme to estimate the relative program size of the population, and a new α adaptation strategy guided by the relative program size estimation. To address the archive diversity issue, we propose a new archive update strategy that avoids including phenotypic duplicate individuals in the archive. αMOGP-a will be discussed in Section 4.5.

4.1.1 Chapter Goals

The overall goal of this chapter is to propose novel MOGP algorithms that can evolve a set of non-dominated routing policies that have different tradeoffs between the effectiveness and the size so that end users can select a routing policy based on their preference. To achieve this goal, we have the following specific research objectives:

1. To develop new MOGP algorithms that can evolve effective and interpretable routing policies for UCARP,
2. To develop new strategies that can deal with the objective selection bias issue,
3. To develop new strategies that can deal with the stochastic fitness evaluation issue,
4. To verify the performance of MOGP algorithms by comparing with the current state-of-the-art on a wide range of UCARP instances, and
5. To interpret the routing policies evolved by new MOGP algorithms, and understand the behaviors of the GP-evolved routing policies.

4.1.2 α Dominance Strategy

The α-dominance strategy [90] is a more general form of traditional dominance. The basic idea of α-dominance is to set tradeoff rates between objectives. For example, we have two solutions, A and B, with two objectives, obj_1 and obj_2. $\mathbf{f}(A) = (a_1, a_2)$ and $\mathbf{f}(B) = (b_1, b_2)$. When using traditional Pareto-dominance, if $a_1 < b_1$ and $a_2 > b_2$, A and B are nondominated. However, A might dominate B when using α-dominance strategy. For a problem to minimize all objectives. If a feasible solution x dominates another solution x' using α-dominance strategy. The problem can be formulated as follows.

$$\min(f_1(o),\ f_2(o),\ f_3(o), ...,\ f_m(o)), \tag{4.1}$$

$$s_i(x, x') \leq 0, \ \forall i \in \{1, 2, 3, ..., m\}, \ \wedge \tag{4.2}$$

$$s_i(x, x') < 0, \ \exists i \in \{1, 2, 3, ..., m\} \tag{4.3}$$

where

$$s_i(x, x') = f_i(x) - f_i(x') + \sum_{j \neq i}^{1...m} \alpha_{ij}(f_j(x) - f_j(x')) \tag{4.4}$$

There is one thing that needs to be noticed that traditional dominance is a special form of α dominance while all α_{ij} equal to 0.

In our problem, we expect that a solution with medium size and good performance will not be dominated by a solution with extremely small size but poor performance. Thus, we modified Eq. 4.3 as follows.

$$s_{size}(x, x') = size(x) - size(x') + \alpha_s(\mathit{eff}(x) - \mathit{eff}(x')) \tag{4.5}$$

$$s_{\mathit{eff}}(x, x') = \mathit{eff}(x) - \mathit{eff}(x') + \alpha_p(size(x) - size(x')) \tag{4.6}$$

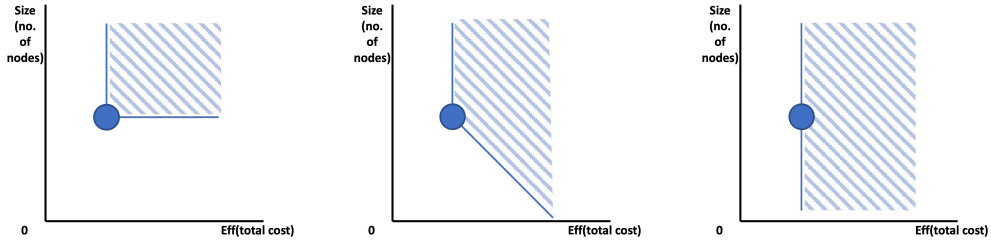
where $size(x)$ refers to the size (i.e., number of nodes) of the individual x. $\mathit{eff}(x)$ refers to the effectiveness (i.e., test performance) of the individual x. α_s equals to α (all the α value mentioned later in the paper) and α_p equals to 0.

Specifically, given two solutions x and x', we say that x *α-dominates* x' if $s_{size}(x, x') \leq 0, s_{\mathit{eff}}(x, x') \leq 0$.

Fig. 4.1 shows the how the dominance area changes when α value changes. It can be seen that when α equals 0, the dominance area is same as traditional dominance criteria in which two objectives are treated equally. When the α is increased, the effectiveness is subsequently given more importance. When α equals positive ∞, it only considers the single objective effectiveness and ignore the size.

4.1.3 Chapter Organisation

The rest of this chapter is organised as follows. Section 4.2 describes the newly proposed αMOGP. Section 4.3 introduces the αMOGP-sa. Section



(a) $\alpha = 0$ (b) $\alpha = 1$ (c) $\alpha = \infty$

Figure 4.1: The dominance area when α changes.

4.4 introduces the TSMOGP-a. Section 4.5 describes the αMOGP-a. Finally, Section 4.6 gives the summary of this chapter.

4.2 MOGP Approach with Adaptive α Dominance Strategy

4.2.1 Overall Framework

The basic idea of αMOGP is using α-dominance strategy to replace the traditional Pareto-dominance. The α-dominance strategy is combined with tournament selection as α tournament selection. After evaluating each routing policy, α tournament selection will be applied to select parent individuals from the old population. The parent individuals will be used to breed new population. The, $crossover$, $mutation$ and $reproduction$, operators will be applied in the breeding process. The pseudocode of the αMOGP approach is shown in Algorithm 5. Given the number of generations G. Each routing policy is evolved using a different training subset randomly sampled from the training set S_{train}.

The α tournament selection uses the α dominance to select the parents. Specifically, it first randomly samples T individuals, and then select the best one among them in terms of α dominance. If there are multiple non-dominated individuals, the first one identified during the comparison is

Algorithm 5: The overall framework of αMOGP.

```
Input: Training set S_train, number of generations G
Output: A set of non-dominated routing policies RP
initialise the population pop;
g = 0;
while g < G do
    randomly sample a training subset S' ⊆ S_train;
    evaluate pop using S';
    while |pop'| < popsize do
        Breed pop' using parent selection (Algorithm 6) and genetic
          operators;
        g = g + 1;
    end
    pop = pop';
    update the α value based on different update schemes;
end
return the non-dominated routing policies in pop;
```

selected. The pseudocode of the α Tournament Selection method is shown in Algorithm 6.

The α-dominance strategy presents a challenge in finding the right balance between performance and size throughout the evolutionary process. In our exploration, we have assessed the behaviors of three distinct adaptation schemes for the α value: linear, sigmoid, and cosine.

The linear scheme implements a gradual reduction in the emphasis on performance, resulting in a shift of pressure from performance to size as evolution progresses. This scheme seeks to strike a balance by gradually tilting the objectives towards favoring smaller, more interpretable routing policies.

In contrast, the sigmoid scheme introduces a nuanced approach. It initiates a slow reduction in the emphasis on performance during the early

Algorithm 6: The α tournament selection approach

```
Input: size of Tournament T, Population pop
Output: best routing policy
t= 1;
best = random selected rp from pop;
for t < T do
    rp = random selected rp' from pop;
    if rp' α-dominates best (Eq. (4.5,4.6)) then
        best = rp';
    end
    t = t + 1;
end
return best;
```

stages of evolution, followed by a more rapid reduction in the middle stages. This rapid decrease sharpens the shift of pressure from performance to size. The sigmoid scheme aims to adapt to the evolving needs of the GP process, responding dynamically to the tradeoff between performance and size.

The cosine scheme takes a cyclical approach, consistently oscillating between increasing and decreasing emphasis on performance. This repetitive pattern enables a continuous shifting of pressure between the objectives of performance and size. The cosine scheme provides an intriguing perspective on how periodic changes in emphasis can influence the evolutionary trajectory of GP.

Each of these adaptation schemes offers a unique perspective on the dynamics of the α value during the GP process. By examining their behaviors, we gain valuable insights into how variations in the α value impact the tradeoff between performance and size, ultimately aiding in the pursuit of more interpretable and effective routing policies.

The three adaptation scheme curves are shown in Fig. 4.2. The formu-

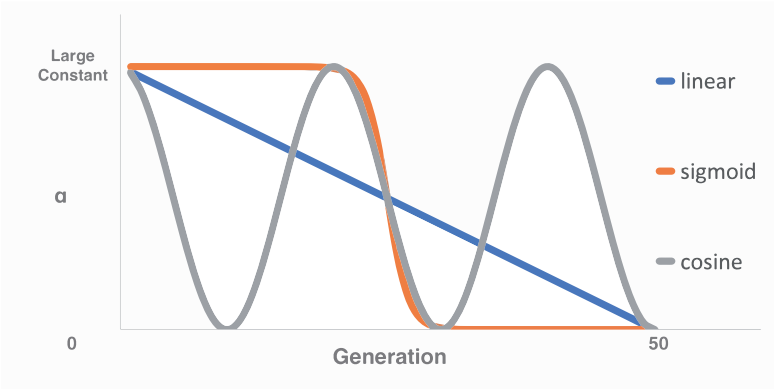


Figure 4.2: Three adaptation scheme curves.

las for all three schemes are shown as follow.

$$f_{linear}(generation) = \mathrm{C} + \frac{-\mathrm{C} * generation}{50} \tag{4.7}$$

$$f_{sigmoid}(generation) = \mathrm{C} * \frac{1}{1 + e^{generation-25}} \tag{4.8}$$

$$f_{cosine}(generation) = \frac{\mathrm{C}}{2} * (\cos(\frac{\pi * generation}{10}) + 1) \tag{4.9}$$

where C is a sufficiently large constant which is 99999999 in this paper.

4.2.2 Experimental Study

To evaluate the proposed approach, we test them on a number of UCARP instances which are commonly used in UCARP literature [122, 143]. For the sake of convenience, we denote the αMOGP with different adaptation schemes as follows: αMOGP-linear (αMOGP-l), αMOGP-sigmoid (αMOGP-s) and αMOGP-cosine (αMOGP-c) . αMOGP-l adjust the α value based on a linear adaptation scheme. αMOGP-s adjusts the α value based on a sigmoid adaptation scheme. αMOGP-c adjusts the α value based on a cosine adaptation scheme. We compare these three algorithms with the SimpleGP [143], which evolves a single routing policy using GPHH,

Two-stage GPHH [196], which apply single-objective GPHH in the first stage and multi-objective GPHH in the second stage, and some other traditional MOGP algorithms, such as NSGA-II [50] and SPEA2 [22]. These two widely-used and easily implementable methods, NSGA-II and SPEA2, lend themselves well to adaptation within the context of GPHH for UCARPs. They rely on traditional dominance criteria and serve as useful benchmarks to evaluate the effectiveness of our proposed method. Both NSGA-II and SPEA2 in this work utilise the original idea of NSGA-II [50] and SPEA2 [22] but with GP adaptation to make it compatible in GPHH. While originally designed for static problems, the core principles of selection, reproduction, and elitism can be seamlessly applied to UCARPs. Our approach utilizes GP crossover and mutation operators and employs simulation-based fitness assignment.

We select 8 commonly used UCARP instances to evaluate the performance of the proposed approach. The problem size varies from 22 tasks and 5 vehicles (small) to 97 tasks and 10 vehicles (large). Thus, we can examine the effectiveness of the proposed methods in different problem scenarios. In the training phase, routing policies are trained based on the training set $\mathcal{S}_{train}$. There are 5 training samples during the evaluation and they are re-sampled each generation. In the test phase, the routing policy will be tested on an unseen test set $\mathcal{S}_{test}$, which contains 500 test samples that can avoid testing bias.

The population size for all the compared algorithms is 1000. Besides that, the total number of generations for all the compared algorithms set to 50. For the initialisation, ramp-half-and-half initialisation is used. The ratio of crossover to mutation to reproduction is $0.8 : 0.15 : 0.05$. For all αMOGP algorithms, the α-tournament selection size is 7. For NSGA-II, the tournament selection size is 2 which is a common setting for NSGA-II. For SPEA2, the tournament selection size is 7 which is a common setting for SPEA2. For SimpleGP, the tournament selection size is 7 which is commonly used in UCARP experiments. The maximal depth is 8, which

has been commonly used in previous studies (e.g., [122]). Elitism size for all compared approaches is 10 except NSGA-II and SPEA2. There is no elitism parameter for NSGA-II and SPEA2 approaches. The meanings of each terminal can be found in Table 2.3

We use Evolutionary Computation Java (ECJ) package [131] to implement all the algorithms. The result is collected based on 30 independent runs for each algorithm on each UCARP instance.

4.2.3 Results

We use two commonly adopted measures for multi-objective optimisation, i.e. hyper-volume (HV, the larger the better) and Inverted Generational Distance (IGD, the smaller the better). The Wilcoxon rank sum test with a significance level of 0.05 is used to verify the performance of the proposed approaches. Each $+$, $-$ and $=$ in parentheses refer the Wilcoxon rank sum test significance. The first parentheses refer to the Wilcoxon rank sum test significance between compared algorithms and NSGA-II. The second parentheses refer to the Wilcoxon rank sum test significance between compared algorithms and SPEA2.

Table 4.1 shows the mean and standard deviation of HV of the compared algorithms. It can be seen that all αMOGP algorithms significantly outperforms NSGA-II and SPEA2 on all instances. This indicates the effectiveness of the proposed αMOGP. To make further comparison, we also make a pairwise comparison between the algorithms. Table 4.2 shows the pairwise comparison results between the algorithms. In the table, each entry represents the comparison result between the column algorithm and the row algorithm. The entry is formatted in W-D-L format. W (L) indicates the number of instances where the column approach performs significantly better (worse) than the row approach. D indicates the number of instances where the two approaches showed no significant difference.

From the table, we can see that all αMOGP algorithms can generate

Table 4.1: The mean and standard deviation for HV of the compared algorithms in test process. For each method, (+), (-) and (=) indicates it is significantly higher (better) than, lower (worse) than, and comparable with NSGA-II (the first parentheses) and SPEA2 (the second parentheses).

Instance	NSGA-II	SPEA2	TS-GPHH	αMOGP-l	αMOGP-s	αMOGP-c
Ugdb1	0.9071(0.0252)	0.8645(0.0408)	0.9378(0.0258)(+)(+)	0.9389(0.0356)(+)(+)	0.9427(0.0263)(+)(+)	0.9423(0.0292)(+)(+)
Ugdb2	0.9153(0.0154)	0.8894(0.0355)	0.9399(0.0199)(+)(+)	0.9395(0.0281)(+)(+)	0.9572(0.0121)(+)(+)	0.9423(0.0175)(+)(+)
Ugdb8	0.9142(0.0228)	0.8625(0.0466)	0.9395(0.0333)(+)(+)	0.9404(0.0302)(+)(+)	0.9505(0.0195)(+)(+)	0.9427(0.0251)(+)(+)
Ugdb23	0.8889(0.0206)	0.8738(0.0259)	0.9303(0.0198)(+)(+)	0.9295(0.0376)(+)(+)	0.9416(0.0186)(+)(+)	0.9341(0.0247)(+)(+)
Uval9A	0.9756(0.0052)	0.9577(0.0178)	0.9838(0.0037)(+)(+)	0.9781(0.0159)(+)(+)	0.9853(0.0115)(+)(+)	0.9811(0.0084)(+)(+)
Uval9D	0.919(0.0174)	0.8528(0.053)	0.9508(0.012)(+)(+)	0.9393(0.0324)(+)(+)	0.9581(0.0121)(+)(+)	0.948(0.0215)(+)(+)
Uval10A	0.9736(0.0067)	0.9534(0.0161)	0.9861(0.0097)(+)(+)	0.9832(0.0112)(+)(+)	0.9905(0.0047)(+)(+)	0.9859(0.0117)(+)(+)
Uval10D	0.9302(0.0238)	0.8986(0.0325)	0.9643(0.0123)(+)(+)	0.9518(0.0411)(+)(+)	0.9724(0.0106)(+)(+)	0.963(0.0138)(+)(+)

Table 4.2: The WDL table for the pairwise comparisons between the algorithms in terms of HV.

Approach	NSGA-II	SPEA2	TS-GPHH	αMOGP-l	αMOGP-s	αMOGP-c
NSGA-II	0-8-0	0-0-8	8-0-0	8-0-0	8-0-0	8-0-0
SPEA2	8-0-0	0-8-0	8-0-0	8-0-0	8-0-0	8-0-0
TS-GPHH	0-0-8	0-0-8	0-8-0	0-8-0	6-2-0	0-8-0
αMOGP-l	0-0-8	0-0-8	0-8-0	0-8-0	5-3-0	0-8-0
αMOGP-s	0-0-8	0-0-8	0-2-6	0-3-5	0-8-0	0-5-3
αMOGP-c	0-0-8	0-0-8	0-8-0	0-8-0	3-5-0	0-8-0

better Pareto fronts than NSGA-II and SPEA2. Also, αMOGP-s performs best among all αMOGP algorithms and TS-GPHH. It is significantly better than αMOGP-l on 5 instances and significantly better than αMOGP-c on 3 instances. It can also generate better Pareto fronts than TS-GPHH. We can see that it perform significantly better than TS-GPHH on 6 out of 8 instances. Besides that, it does not perform significantly worse on any instance than other algorithms.

Table 4.3 shows mean and standard deviation of IGD of compared algorithms. It can be seen that αMOGP-l outperforms NSGA-II on 7 out of total 8 instances. Besides that, all other αMOGP algorithms and TS-GPHH outperform NSGA-II on all instances. It is clear to see that all compared algorithms can outperform SPEA2 on all instances. This is consistent with

Table 4.3: The mean and standard deviation for IGD of the compared algorithms in test process. For each method, (+), (-) and (=) indicates it is significantly lower (better) than, higher (worse) than, and comparable with NSGA-II (the first parentheses) and SPEA2 (the second parentheses).

Instance	NSGA-II	SPEA2	TS-GPHH	αMOGP-l	αMOGP-s	αMOGP-c
Ugdb1	0.0972(0.0205)	0.1316(0.0373)	0.0835(0.028)(+)(+)	0.0786(0.0375)(+)(+)	0.0733(0.021)(+)(+)	0.0729(0.0284)(+)(+)
Ugdb2	0.1619(0.0078)	0.1806(0.0277)	0.112(0.0252)(+)(+)	0.1197(0.023)(+)(+)	0.105(0.0218)(+)(+)	0.1212(0.0214)(+)(+)
Ugdb8	0.0748(0.0198)	0.1205(0.0406)	0.0672(0.0306)(+)(+)	0.0778(0.0448)(=)(+)	0.0466(0.0169)(+)(+)	0.0676(0.041)(+)(+)
Ugdb23	0.1376(0.0171)	0.1487(0.0225)	0.0909(0.0203)(+)(+)	0.083(0.0421)(+)(+)	0.0717(0.0177)(+)(+)	0.0791(0.0366)(+)(+)
Uval9A	0.1018(0.0215)	0.1194(0.0237)	0.0548(0.0154)(+)(+)	0.0608(0.0195)(+)(+)	0.0568(0.025)(+)(+)	0.0499(0.0194)(+)(+)
Uval9D	0.1403(0.02)	0.195(0.0507)	0.0767(0.0194)(+)(+)	0.0742(0.0387)(+)(+)	0.0713(0.0191)(+)(+)	0.0696(0.0319)(+)(+)
Uval10A	0.1029(0.0196)	0.1332(0.0282)	0.0635(0.016)(+)(+)	0.0619(0.0167)(+)(+)	0.0647(0.0189)(+)(+)	0.0593(0.0232)(+)(+)
Uval10D	0.0751(0.0198)	0.098(0.0275)	0.0507(0.0226)(+)(+)	0.0645(0.0509)(+)(+)	0.0331(0.0131)(+)(+)	0.0543(0.0251)(+)(+)

Table 4.4: The WDL table for IGD on all 8 instances in W-D-L format for each compared approach.

Approach	NSGA-II	SPEA2	TS-GPHH	αMOGP-l	αMOGP-s	αMOGP-c
NSGA-II	0-8-0	0-0-8	8-0-0	7-1-0	8-0-0	8-0-0
SPEA2	8-0-0	0-8-0	8-0-0	8-0-0	8-0-0	8-0-0
TS-GPHH	0-0-8	0-0-8	0-8-0	1-7-0	3-5-0	2-6-0
αMOGP-l	0-1-7	0-0-8	0-7-1	0-8-0	3-5-0	1-7-0
αMOGP-s	0-0-8	0-0-8	0-5-3	0-5-3	0-8-0	0-6-2
αMOGP-c	0-0-8	0-0-8	0-6-2	0-7-1	2-6-0	0-8-0

HV. We also make a pairwise comparison on each algorithm on IGD, and the results are shown in Table 4.4. In the table, each entry represents the comparison result between the column algorithm and the row algorithm. The entry is formatted in W-D-L format. W (L) indicates the number of instances where the column approach performs significantly better (worse) than the row approach. D indicates the number of instances where the two approaches showed no significant difference. The result is consistent with HV, all compared algorithms can obtain better front than NSGA-II and SPEA2, and αMOGP-s performs best among all αMOGP algorithms.

Performance is still the primary objective of this study. Thus, we take the routing policies with the best performance from every front to compare the mean performance and size for all compared algorithms. The

result is shown in Table 4.5 and Table 4.6, respectively. Table 4.5 shows the mean and standard deviation of test performance of the compared algorithms. Wilcoxon rank sum test with a significance level of 0.05 is used to compare each approach with SimpleGP. From Table 4.5, one can see that both NSGA-II and SPEA2 perform worse than SimpleGP since they cannot handle the challenge of premature convergence on smaller individuals. All αMOGP algorithms can obtain a comparable result with SimpleGP. Especially, both αMOGP-l and αMOGP-c perform significantly better than SimpleGP on 2 out of 8 instances.

Table 4.6 shows the mean and standard deviation of size of the compared algorithms. All the compared approaches can evolve much smaller routing policies than SimpleGP. This is as expected since we use size as an objective. The result indicates that using size as an objective can reduce routing policy size effectively. There is one thing that needs to be noticed that both NSGA-II and SPEA2 achieves much smaller size on all instances than other algorithms. This is because either NSGA-II or SPEA2 has the problem of premature convergence to small individuals which will lead to poor performance (shown in Table 4.5).

Overall, the proposed αMOGP can generate better Pareto fronts than NSGA-II, SPEA2 and TS-GPHH. Besides that, it can achieve much smaller rule size than SimpleGP with comparable performance. Primarily, with proper adaptation scheme, it can obtain better performance.

4.2.4 Further Analysis

To analyse the routing policies evolved by αMOGP, we picked one of the best performing routing policies from αMOGP-s for Ugdb1 as an example. The routing policy evolved by αMOGP-s, denoted as RP1, is shown in Eq. 4.10.

$$RP1 = \max(S_1, S_2) \tag{4.10}$$

Table 4.5: The mean and standard deviation for test performance (total cost) of the compared algorithms. For each method, (+), (-) and (=) indicates it is significantly lower (better) than, higher (worse) than, and comparable with SimpleGP.

Instance	SimpleGP	NSGA-II	SPEA2	TS-GPHH	αMOGP-l	αMOGP-s	αMOGP-c
Ugdb1	355.47(14.83)	373.2(9.8)(-)	389.15(15.56)(-)	356.4(10.7)(=)	354.8(12.0)(=)	358.4(10.6)(=)	354.6(12.5)(=)
Ugdb2	371.72(7.67)	392.8(6.5)(-)	404.08(15.62)(-)	372.0(9.1)(=)	370.2(7.9)(=)	370.8(6.4)(=)	370.5(7.0)(=)
Ugdb8	463.34(54.3)	476.3(15.1)(-)	509.82(29.91)(-)	452.0(23.0)(=)	441.1(10.8)(+)	448.9(12.7)(=)	443.7(11.2)(+)
Ugdb23	252.47(3.11)	260.2(2.9)(-)	262.35(3.66)(-)	253.0(3.3)(=)	250.5(2.2)(+)	252.0(3.1)(=)	251.1(2.7)(=)
Uval9A	335.13(3.8)	351.3(6.0)(-)	371.24(19.77)(-)	336.7(4.6)(=)	336.0(3.7)(=)	336.4(3.4)(=)	336.3(3.8)(=)
Uval9D	478.14(16.68)	522.7(17.4)(-)	586.58(51.35)(-)	480.7(10.1)(=)	474.2(11.9)(=)	479.3(13.4)(=)	474.8(12.2)(=)
Uval10A	439.41(5.97)	460.0(7.0)(-)	481.59(16.89)(-)	442.0(10.9)(=)	440.5(3.9)(=)	440.9(4.4)(-)	439.7(4.0)(=)
Uval10D	620.91(7.97)	668.9(24.0)(-)	699.8(32.83)(-)	624.2(8.6)(=)	619.6(8.4)(=)	622.2(10.0)(=)	617.5(10.4)(+)

Table 4.6: The mean and standard deviation for size of routing policies of the compared algorithms.

Instance	SimpleGP	NSGA-II	SPEA2	TS-GPHH	αMOGP-l	αMOGP-s	αMOGP-c
Ugdb1	74.6(23.84)	10.0(3.99)	8.4(5.18)	30.53(18.02)	27.67(15.97)	17.33(11.98)	27.87(15.27)
Ugdb2	71.93(23.79)	6.93(3.13)	5.73(3.22)	38.33(19.8)	28.07(14.12)	26.53(13.27)	29.0(14.08)
Ugdb8	65.47(24.33)	7.07(3.08)	5.6(4.49)	42.27(36.62)	51.67(19.03)	30.87(13.97)	41.6(22.49)
Ugdb23	71.8(25.22)	8.27(3.66)	8.6(4.53)	31.13(19.5)	47.53(26.57)	33.07(24.67)	43.27(24.64)
Uval9A	56.93(18.27)	9.73(4.65)	8.53(4.83)	30.6(13.72)	28.4(14.79)	26.07(14.04)	30.6(12.68)
Uval9D	69.27(29.46)	10.33(4.85)	7.53(6.15)	42.67(19.31)	58.0(29.24)	37.0(22.18)	49.87(20.02)
Uval10A	60.47(18.59)	8.07(3.47)	4.53(4.54)	24.07(10.86)	19.27(9.74)	15.73(6.0)	23.6(12.99)
Uval10D	65.33(14.35)	8.93(3.46)	9.2(5.18)	35.0(21.22)	47.67(23.7)	34.13(17.92)	45.73(21.52)

where

$$S_1 = DC * CFH + CTT1 \tag{4.11}$$

$$S_2 = \frac{DEM1}{DC - CFR1} \tag{4.12}$$

From S_1, we can observe that the policy tends to select the tasks with smaller CFH (close to the current place) and smaller CTT1. From S_2, one can see that the policy tends to select the tasks with smaller DEM1. Thus RP1 tends to select the tasks close to current place or the tasks that their closest task has a small demand.

We have also analysed the final Pareto front. One interesting observation is that a large amount of small and good routing policies contains the same building block "DC * CFH". The final Pareto front contains a large

number of individuals with one single terminal. Another interesting observation is that most of the single-terminal individuals are "CFH". This indicates that "CFH" is critical for build small and good individuals.

4.2.5 Summary

In summary, this work proposes some different manual settings of α adaptation schemes to control the balance between the effectiveness and size of routing policies in MOGP. The experimental results show that with a proper adaptation scheme, the proposed approach could evolve much smaller routing policies without losing the test performance. The results also show that the proposed approach can effectively reduce the objective selection bias issue. In addition, different manual settings of α adaptation schemes can all outperform the standard GPHH.

4.3 MOGP Approach with Self Adaptive α Dominance Strategy

Although the manually designed α adaptation schemes outperform the standard MOGP, we still see that the best alpha depends on the scenarios, and we should set α differently in different scenarios. In this case, we will design new self α adaptation scheme to adjust α value during the evolutionary process.

4.3.1 Overall Framework

Fig. 4.3 shows the diagram of the αMOGP-sa. The basic idea of αMOGP-sa is using α-dominance criteria to replace the traditional dominance criteria. After evaluating each routing policy, the population will be non-dominated sorted based on α dominance. Rank and sparsity will be assigned to each individual in the population. Then, we can detect the bias

of the Pareto front of the current generation and adjust the α value. After that, a new population will be generated. The tournament selection will be applied to select parent individuals based on each candidate individual's rank and sparsity. The $crossover$ and $mutation$ operators will be applied to breed a new population. When the stopping criteria are reached, the final Pareto front based on traditional dominance criteria will be returned.

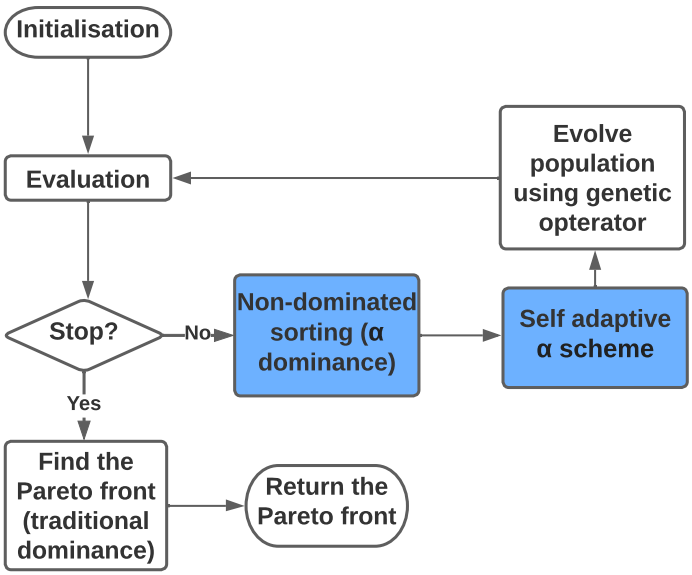


Figure 4.3: The diagram of αMOGP-sa.

4.3.2 Self-Adaptive α Scheme

One of the main challenges of the α dominance is to find a proper α value [198]. For different instances, we need different α value as the search space can be quite different, and the range of the objectives can also be quite different for different instances. In this paper, we propose a self-adaptive α scheme. The scheme will first identify the upper and lower boundary of the effectiveness and size to know the range of the objective space, and if the new boundary is greater than the original, then the original bound-

ary is replaced by the new boundary. We assume that a good Pareto front should cover the boundary as much as possible. Then, the current Pareto front based on traditional dominance criteria is utilised to identify the bias. If the current front is biased to effectiveness, the α should be decreased. If the current front is biased to size, the α should be increased. The pseudocode of the self-adaptive α scheme is shown in Algorithm 7.

Algorithm 7: The self-adaptive α scheme.

Input: The population *pop*, The α value, Upper boundary and lower boundary of effectiveness u_{eff} and l_{eff}, Upper boundary and lower boundary of size u_{size} and l_{size}, learning rate lr

Output: The α value, Updated upper and lower boundary of effectiveness u_{eff} and l_{eff}, Updated upper and lower boundary of size u_{size} and l_{size}

find the current upper and lower boundary of effectiveness and size u'_{eff}, l'_{eff}, u'_{size}, l'_{size} from *pop*;

if $u'_{eff} > u_{eff}$ **then**
  $u_{eff} = u'_{eff}$;
end

if $l'_{eff} < l_{eff}$ **then**
  $l_{eff} = l'_{eff}$;
end

if $u'_{size} > u_{size}$ **then**
  $u_{size} = u'_{size}$;
end

if $l'_{size} < l_{size}$ **then**
  $l_{size} = l'_{size}$;
end

Find the current Pareto front pf using traditional dominance criteria from *pop*;

Calculate the average mean effectiveness avg_{eff} of the pf;

Calculate the average mean size avg_{size} of the pf;

if $\frac{u_{eff}-avg_{eff}}{avg_{eff}-l_{eff}} > 1.0$ *and* $\frac{u_{size}-avg_{size}}{avg_{size}-l_{size}} < 1.0$ **then**
  `// bias to effectiveness, then decrease the` α
  $\alpha = \alpha - lr$;
end

if $\frac{u_{eff}-avg_{eff}}{avg_{eff}-l_{eff}} < 1.0$ *and* $\frac{u_{size}-avg_{size}}{avg_{size}-l_{size}} > 1.0$ **then**
  `// bias to size, then increase the` α
  $\alpha = \alpha + lr$;
end

return α,u_{eff}, l_{eff},u_{size},l_{size};

4.3.3 Experimental Study

To evaluate the performance of the αMOGP-sa, we test all compared algorithms on a set of UCARP instances. We compare the αMOGP-sa with

SPEA2, TSMOGP [196] and the αMOGP [198] which is the current state-of-the-art MOGP approach for UCARP in terms of effectiveness and size.

We select eight commonly used UCARP instances to evaluate the performance of the proposed approach. The problem size varies from small (22 tasks and 5 vehicles) to large (97 tasks and 10 vehicles). The experiment consists of two phases, i.e. training and test. In the training phase, routing policies are trained based on the training set. There are 5 training samples during the evaluation, and they are re-sampled each generation. In the test phase, the routing policy will be tested on 500 unseen test samples, which can avoid testing bias. The meanings of each terminal can be found in Table 2.3

For all compared algorithms, the generation is 50, and the population size is 1000. Ramp-half-and-half initialisation is used. The ratio of crossover to mutation is 0.85 : 0.15. The tournament selection size is 7 for all compared algorithms. The maximal depth is 8, which has been commonly used in previous studies (e.g., [122]). Elitism size for all compared approaches is 10. There is no elitism parameter for SPEA2 as it has an archive to store elites. When α value equals to 0, the α dominance criteria is the traditional dominance criteria. Previous studies [196, 198] have demonstrated the existence of objective selection bias issue when using traditional dominance criteria. To test our hypothesis that the self-adaptive α scheme can detect the bias and find a suitable α value to address the bias, the α value is set to 0 at beginning for αMOGP-sa. The learning rate for the self-adaptive α scheme is 0.2. The Evolutionary Computation Java (ECJ) package [131] is utilised to implement all the algorithms. The results are collected based on 30 independent runs for each algorithm on each UCARP instance.

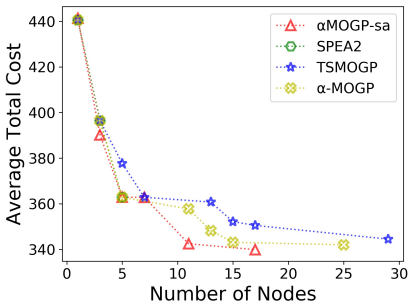
Table 4.7: The mean and standard deviation for **HV** of the compared algorithms in test process.

Instance	**SPEA2**	**TSMOGP**	**αMOGP**	**αMOGP-sa**
Ugdb1	0.8625(0.04)	0.937(0.03)	0.9385(0.03)	0.9578(0.02)(+)(+)(+)
Ugdb2	0.8902(0.04)	0.9334(0.02)	0.9316(0.03)	0.954(0.01)(+)(+)(+)
Ugdb8	0.8392(0.05)	0.9274(0.04)	0.9284(0.03)	0.9705(0.01)(+)(+)(+)
Ugdb23	0.866(0.03)	0.9274(0.02)	0.9286(0.04)	0.9478(0.01)(+)(+)(+)
Uval9A	0.9577(0.02)	0.9815(0.0)	0.9735(0.02)	0.9887(0.0)(+)(+)(+)
Uval9D	0.8524(0.05)	0.9477(0.01)	0.9326(0.04)	0.9592(0.01)(+)(+)(+)
Uval10A	0.9542(0.02)	0.9863(0.01)	0.9747(0.02)	0.9931(0.0)(+)(+)(+)
Uval10D	0.9019(0.03)	0.9684(0.01)	0.9566(0.04)	0.9775(0.01)(+)(+)(+)

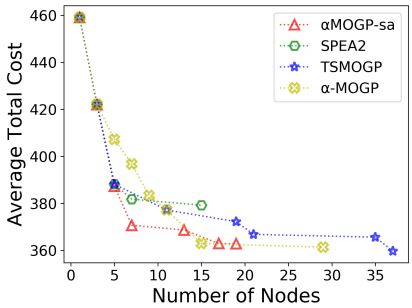
4.3.4 Results

The Wilcoxon rank sum test with a significance level of 0.05 is used to compare all algorithms. For each compared algorithm, if the proposed algorithm is statistically significantly better than, worse than, and comparable with that of the compared algorithm, the corresponding entry is marked with (+), (-), or (=), respectively.

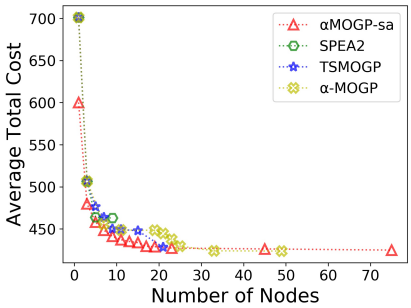
We adopt two commonly used indicators for multi-objective optimisation, i.e. Hyper-Volume (HV) [225] and Inverted Generational Distance (IGD) [42]. The results are shown in Tables 4.7 and 4.8. It can be seen that the αMOGP-sa perform best among all compared algorithms on almost all instances. This is as expected as the self-adaptive α scheme can automatically find a suitable α value for different scenarios. With the suitable α value, αMOGP-sa can generate the better Pareto front. To make the results more obvious, the final Pareto front over the 30 independent runs on each instance is shown in Fig. 4.4. We can see that the Pareto fronts of the αMOGP-sa are better distributed in the objective space. The Pareto fronts of αMOGP is close to that of αMOGP-sa. This is expected as both algorithms utilised α dominance instead of traditional dominance. The Pareto fronts of αMOGP-sa are better than that of αMOGP. This can also indicate the effectiveness of the self-adaptive α scheme.



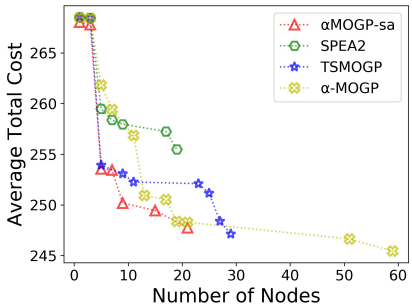
(a) Ugdb1



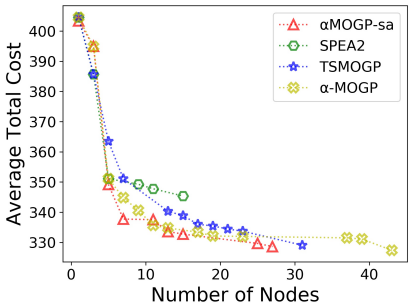
(b) Ugdb2



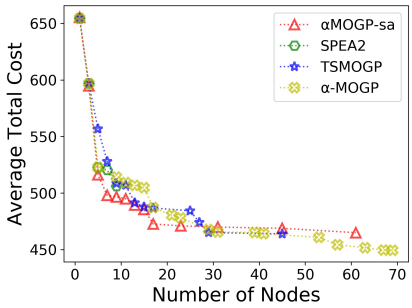
(c) Ugdb8



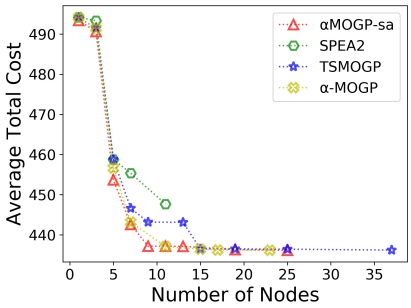
(d) Ugdb23



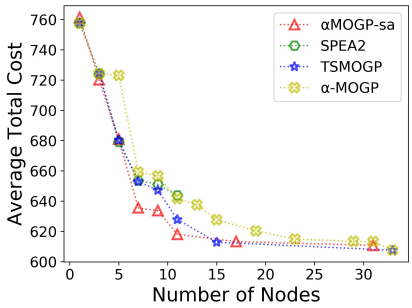
(e) Uval9A



(f) Uval9D



(g) Uval10A



(h) Uval10D

Figure 4.4: The final Pareto front over 30 independent run of compared algorithms.

Table 4.8: The mean and standard deviation for **IGD** of the compared algorithms in test process.

Instance	**SPEA2**	**TSMOGP**	**αMOGP**	**αMOGP-sa**
Ugdb1	0.0921(0.03)	0.0678(0.03)	0.066(0.04)	0.0345(0.01)(+)(+)(+)
Ugdb2	0.1329(0.03)	0.0892(0.03)	0.0865(0.03)	0.048(0.01)(+)(+)(+)
Ugdb8	0.1359(0.04)	0.0973(0.03)	0.1113(0.05)	0.0244(0.01)(+)(+)(+)
Ugdb23	0.1207(0.02)	0.0788(0.02)	0.0851(0.05)	0.047(0.01)(+)(+)(+)
Uval9A	0.1218(0.03)	0.0522(0.01)	0.0598(0.02)	0.0486(0.02)(+)(=)(+)
Uval9D	0.202(0.05)	0.0842(0.02)	0.0807(0.04)	0.0545(0.02)(+)(+)(+)
Uval10A	0.097(0.03)	0.0444(0.02)	0.0466(0.02)	0.0271(0.01)(+)(+)(+)
Uval10D	0.1033(0.03)	0.0522(0.03)	0.0674(0.05)	0.0316(0.01)(+)(+)(+)

In this paper, we are still concerned about effectiveness because we would like some effective and compact routing policies. In this case, we extract the individual with the best average total cost in the final Pareto front from each run of each MOGP algorithm. We also compare our algorithm with the state-of-art single-objective GPHH approach. Table 4.9 shows the mean and standard deviation of the average total cost (effectiveness) of the compared algorithms. It can be seen that αMOGP-sa can perform comparable with GPHH on all instances. Besides, it can outperform SPEA2 on all instances because SPEA2 has the objective selection bias issue on size. This is consistent with the pattern of the final Pareto front of the SPEA2 in Fig. 4.4. αMOGP-sa can outperform TSMOGP on 3 out 8 instances and achieve comparable effectiveness on other instances. αMOGP-sa performs slightly worse than αMOGP on 1 instance, but it performs significantly better than αMOGP on 3 out of 8 instances and comparable on other instances. Table 4.10 shows the mean and standard deviation of the size of compared algorithms. αMOGP-sa can evolve significantly smaller routing policies than GPHH. This is because αMOGP-sa considers both size and effectiveness during the evolutionary process. We can see that SPEA2 obtains much smaller routing policies than αMOGP-sa. This is because of the objective selection bias issue. All the routing policies obtained

Table 4.9: The mean and standard deviation of the **average total cost** of the best individual from compared algorithms.

Instance	GPHH	SPEA2	TSMOGP	αMOGP	αMOGP-sa
Ugdb1	355.4678(14.83)	389.147(15.56)	356.4119(10.74)	354.7738(12.05)	351.8234(7.41)(=)(+)(=)(=)
Ugdb2	371.7201(7.67)	404.084(15.62)	372.0391(9.14)	370.1717(7.91)	372.9289(6.03)(=)(+)(=)(-)
Ugdb8	430.3441(8.16)	509.8187(29.91)	451.9689(22.96)	441.0539(10.75)	433.7355(5.58)(=)(+)(+)(+)
Ugdb23	252.4679(3.11)	262.3478(3.66)	253.0202(3.27)	250.4665(2.16)	251.2397(1.95)(=)(+)(+)(=)
Uval9A	335.1342(3.8)	371.2449(19.77)	336.7092(4.57)	336.0367(3.65)	333.8784(2.22)(=)(+)(+)(+)
Uval9D	478.1405(16.68)	586.5754(51.35)	480.6811(10.14)	474.2416(11.94)	477.6783(7.09)(=)(+)(=)(=)
Uval10A	439.4139(5.97)	481.591(16.89)	442.041(10.9)	440.5193(3.92)	438.1817(2.47)(=)(+)(=)(+)
Uval10D	620.9134(7.97)	699.7957(32.83)	624.1609(8.61)	619.5877(8.45)	620.2146(4.96)(=)(+)(=)(=)

Table 4.10: The mean and standard deviation of the **average number of nodes** of the best individual from compared algorithms.

Instance	GPHH	SPEA2	TSMOGP	αMOGP	αMOGP-sa
Ugdb1	74.6(23.84)	8.4(5.18)	30.6(18.01)	27.67(15.97)	34.47(31.68)(+)(-)(=)(=)
Ugdb2	71.93(23.79)	5.73(3.22)	38.33(19.8)	28.8(13.88)	28.07(13.26)(+)(-)(+)(=)
Ugdb8	65.47(24.33)	5.6(4.49)	42.27(36.62)	51.67(19.03)	51.2(33.63)(+)(-)(=)(=)
Ugdb23	71.8(25.22)	8.6(4.53)	31.33(19.58)	47.53(26.57)	37.07(36.73)(+)(-)(=)(+)
Uval9A	56.93(18.27)	8.53(4.83)	30.6(13.72)	28.4(14.79)	32.27(19.78)(+)(-)(=)(=)
Uval9D	69.27(29.46)	7.53(6.15)	42.67(19.31)	58.0(29.24)	40.7(16.45)(+)(-)(=)(+)
Uval10A	60.47(18.59)	4.53(4.54)	24.07(10.86)	19.27(9.74)	25.8(13.99)(+)(-)(=)(-)
Uval10D	65.33(14.35)	9.2(5.18)	35.0(21.22)	47.67(23.7)	42.4(18.06)(+)(-)(-)(=)

by SPEA2 are small but poor. This is consistent with the results in Table 4.9. The effectiveness of SPEA2 is significantly worse than αMOGP-sa. αMOGP-sa can achieve comparable size with TSMOGP and αMOGP on 6 out of 8 instances. Besides, αMOGP-sa outperforms TSMOGP on Ugdb2, and it also outperforms αMOGP on Ugdb23.

4.3.5 Further Analysis

To further verify the self-adaptive α scheme's effectiveness, we also compare the NSGP-II with the self-adaptive α scheme (NSGP-II-sa) with the traditional NSGP-II and a Two-Stage NSGP-II (TSNSGP-II). From Table 4.11, we can see that the NSGP-II-sa can outperform the other two algo-

Table 4.11: The experiment result (HV) to show the effectiveness of the self-adaptive α scheme.

Instance	NSGP-II	TS-NSGP-II	NSGP-II-sa
Ugdb1	0.7512(0.06)	0.7929(0.06)	0.9082(0.04)(+)(+)
Ugdb2	0.8342(0.03)	0.8629(0.04)	0.9261(0.02)(+)(+)
Ugdb8	0.8707(0.03)	0.8798(0.04)	0.9508(0.03)(+)(+)
Ugdb23	0.5643(0.08)	0.6(0.09)	0.8213(0.04)(+)(+)
Uval9A	0.728(0.06)	0.8074(0.06)	0.8939(0.02)(+)(+)
Uval9D	0.7125(0.07)	0.794(0.06)	0.8851(0.02)(+)(+)
Uval10A	0.5867(0.1)	0.7001(0.09)	0.9001(0.03)(+)(+)
Uval10D	0.6329(0.14)	0.7697(0.11)	0.9228(0.02)(+)(+)

rithms without self-adaptive α scheme. This is expected as the scheme can automatically adjust the dominance criteria to reduce the objective selection bias to generate better Pareto front. The NSGP-II is a simple adaption of popular NSGA-II algorithm to GP. TSNSGP-II is the combination of NSGP-II with the two-stage framework which is introduced in [196]. The NSGP-II-sa is the combination of NSGP-II with the self-adaptive α scheme, the self-adaptive α scheme is introduced in Section 4.3.2.

We also plot the convergence curve of α value on all instances on 30 independent runs. The convergence curve is shown in Fig. 4.5. We can see that the curves grow rapidly at first, and as generation increases, the curves stabilise and eventually converge. Besides, the curve eventually converges at different values for different instances. For Ugdb23 and Uval10A, the α value converge to large values. For Uval9A, α value converges to a medium value. For the rest of the instances, α value converge to small values. This is consistent with the assumption that α value is scenario specified. The self-adaptive α scheme is capable of identifying suitable α values for different scenarios. This is also why the αMOGP-sa can outperform αMOGP on HV and IGD.

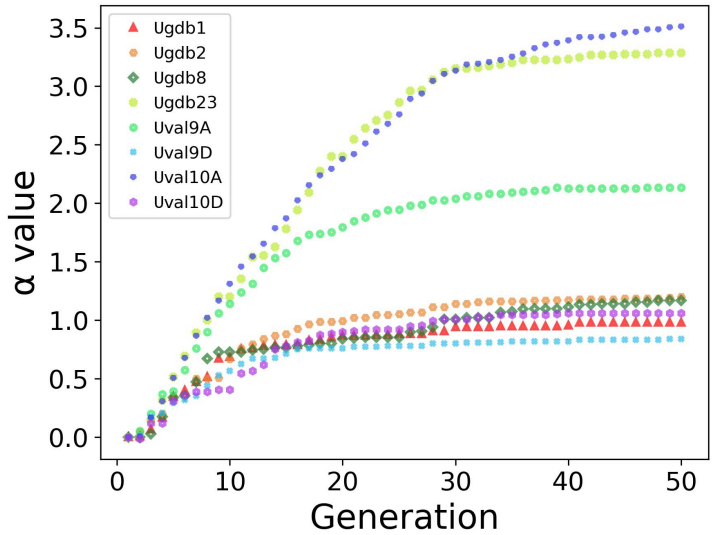


Figure 4.5: The convergence curve of α value.

4.3.6 Summary

In summary, this work proposes a simple yet effective α dominance criterion based Multi-Objective GP with a self-adaptive α scheme (αMOGP-sa). The experimental results showed that the newly proposed (αMOGP-sa) better handle the objective selection bias issue than the state-of-art GPHH approach. The αMOGP-sa generates much better Pareto fronts in terms of HV and IGD than other MOGP approaches from the literature. We also employ the newly proposed self-adaptive α scheme on NSGP-II. The experiment results also indicate that the scheme can effectively address the objective selection bias issue. The convergence curve of α value indicates that the self-adaptive α scheme can find proper α values for different instances. In the following section, we will design algorithm to handle the stochastic evaluation issue.

4.4 Two-Stage MOGP with Archive Strategy

GPHH is essentially a learning approach rather than direct optimisation, and the ultimate goal is to apply the evolved routing policies to future unseen UCARP instances. Generally, obtaining accurate fitness values requires a large number of independent samples, but it is time consuming, and the use of a small number of samples may mislead GP searches. To improve the generalisation of the evolved routing policies, a common strategy is to use a small subset of training samples and change the samples at each generation (so-called *training sample rotation*). Such strategy has been widely used in other problems such as dynamic scheduling (e.g. [80, 213, 214, 216]) and is conceptually similar to the mini-batch strategy in machine learning. As a result, the effectiveness evaluation of a routing policies becomes stochastic and changes generation by generation. In this way, effective routing policies may be lost simply because its effectiveness happens to be not so promising in the small subset of training samples in a particular generation. This issue becomes more serious in MOGP in UCARP, since it is already difficult to retain the potentially effective routing policies. This is stochastic evaluation issue in MOGP algorithm design in UCARP. In this section, we design a new archive strategy to store the potentially effective individuals that could have been lost during the traditional GP process.

4.4.1 Overall Framework

The flowchart of the proposed TSMOGP-a is given in Fig. 4.6. It consists of two stages. Both stages utilise an external archive to store potentially good individuals. The first stage is from generation 0 to generation $G/2$ (G is the maximal number of generations), i.e. half of the entire search process. During the first stage, we only consider minimising the total cost, and ignore the size temporarily. In other words, the first stage is a single-objective optimisation stage. Initially, the population is randomly gener-

ated and an empty external archive is created. Then, each individual is evaluated purely based on the effectiveness. At each generation, the offspring population is first generated from the current population and the external archive by the evolutionary operators (e.g. crossover and mutation). After that, the current and offspring populations are combined and evaluated. Finally, the individuals in the combined population are sorted based on the total cost, and the top N (N is the population size) individuals are selected to be in the next generation.

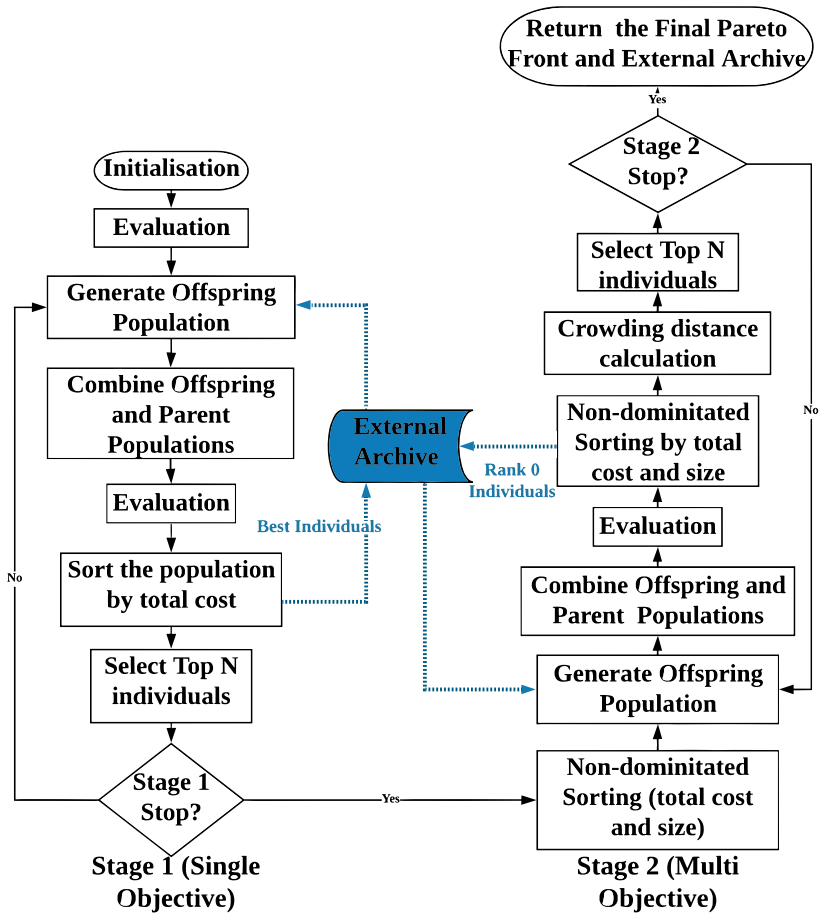


Figure 4.6: Overview of TSMOGP-a.

Note that the first stage of TSMOGP-a is different from the standard single-objective GP, which directly passes all the offspring to the next generation. Therefore, the first stage of TSMOGP-a is greedier than the standard GP. In addition, due to the training sample rotation, the parents are re-evaluated on the new training samples to be compared with the offspring in the combined population.

The second stage of TSMOGP-a is from generation $G/2 + 1$ to generation G. This stage considers both the objectives of total cost and size, and treats them as equally important. It follows the NSGA-II [50] framework, except that at each generation the parents are selected from both the current population and the external archive. At each generation, the offspring population is first generated from the current population and the archive. Then, the current offspring populations are combined with the parent population, and non-dominated sorting and crowding distance calculation are applied. Finally, the top N individuals are selected into the next generation. At each generation, the external archive is updated with the non-dominated individuals in the population, and contributes itself to generate offspring during the breeding process. The details can be seen in Section 4.4.2.

4.4.2 External Archive

The external archive is used to store potentially good individuals generated during the evolutionary process. As shown in Fig. 4.6, potentially good individuals are put into the external archive at the end of each generation. The external archive contains genotypically unique individuals. Whenever a new individual is to be added into the archive, its *genotype* (i.e. tree structure) is compared with that of the individuals in the archive. If there is a duplicate already in the archive, the new individual will be discarded. The maximal size of external archive is set the same as the population size. When the size of the archive is exceeded, individuals will be

re-evaluated, then non-dominated sorting is applied to the individuals in the archive, and only retain population size non-dominated individuals. The pseudocode of the updating process of the external archive is shown in Algorithm 8.

The external archive contributes itself during the breeding process. Specifically, when selecting the parents for crossover and mutation, the current population is combined with the external archive to form the mating pool. Then, the parents are selected from the combined mating pool by the tournament selection (of size 7). To avoid re-applying the non-dominated sorting and crowding distance calculation on the combined mating pool, we define the following archive-aided comparison. The pseudocode of the archive-aided breeding process is shown in Algorithm 9.

- Set all the individuals in the archive as rank 0.5;
- Individual A is better than individual B if $rank(A) < rank(B)$;
- If $rank(A) = rank(B)$, then
  - if A and B are from the current population, then we use the pre-calculated crowding distance, and select the one with a larger crowding distance.
  - if A and B are from the archive, then we select them randomly.

4.4.3 Experimental Study

To verify the performance of the proposed TSMOGP-a, we compare it with several state-of-the-art MOGPs for UCARP. Specifically, we compare with the SPEA2 [23] that weakens the bias towards size, the αMOGP [198] which is the current state-of-the-art MOGP approach for UCARP in terms of effectiveness and size. We also compare with the baseline single-objective GPHH [122] in terms of effectiveness.

Algorithm 8: The updating process of the external archive.

```
Input: The current number of generation g, The total number of generation G,
       the current population pop, the external archive, the population size
       popsize
Output: The updated archive
if g ≤ G/2 then
    // Stage 1
    Obtain the best individual in the population in terms of effectiveness: ind ;
    Set duplicate = false;
    for ind' ∈ archive do
        if genotype(ind) equals to genotype(ind') then
            // two inds have same genotype (tree structure)
            duplicate = true;
            break;
        end
    end
    if duplicate = false then
        Re-assign the rank of ind to 0.5;
        Put ind into archive;
    end
else
    // Stage 2
    Obtain the non-dominated (rank-0) solutions pop_nd in pop;
    for ind ∈ pop_nd do
        Set duplicate = false;
        for ind' ∈ archive do
            if genotype(ind) equals to genotype(ind') then
                tcptwo inds have same genotype (tree structure)
                  duplicate = true;
                break;
            end
        end
        if duplicate = false then
            Re-assign the rank of ind to 0.5;
            Put ind into archive;
        end
    end
end
if |archive| ≥ popsize then
    Non-dominated sort the archive;
    Remove duplicates;
    Retain popsize individuals archive_nd;
    Set archive = archive_nd;
end
return archive;
```

Algorithm 9: The archive-aided breeding process.

Input: The current population pop, the external $archive$
Output: The offspring population pop'
Create an empty population $pop' = \emptyset$;
Set the mating pool as $pool = archive \cup pop$;
while $|pop'| < popsize$ **do**
  Select the $parents$ by the tournament selection **with the archive-aided comparison**;
  Generate a $child$ using $parents$ by crossover/mutation;
  Add $child$ to pop';
end
return pop';

We select 6 representative UCARP instances from the commonly used Ugdb and Uval datasets [122,143] for experiments which have been introduced in Table 2.1. The experiment is divided into the training and test phases. In the training phase, 5 training samples are used for evaluation, which is changed at each generation. In the test phase, the best trained routing policy is tested on an unseen test set containing 500 test samples.

The number of generations is 50 for all compared algorithms. For the single-objective GPHH, the population size is 1000. Since the NSGP-II double the number of evaluations per generation due to the re-evaluations, for fair comparison, we set the population size to 500 for all the NSGP-II algorithms. For other MOGPs, the population size is 1000. Ramp-half-and-half initialisation is used. The tournament selection size is 7. The maximal depth is 8. The crossover and mutation rates are 0.85 and 0.15. All the algorithms are implemented on Evolutionary Computation Java (ECJ) package [131]. Each algorithm was run 30 times independently for each UCARP instance. The meanings of each terminal can be found in Table 2.3

4.4.4 Results

4.4.4.1 Comparison with State-Of-The-Art Approach

Tables 4.12 and 4.13 shows the results of the Hyper-Volume (HV) and Inverted Generational Distance (IGD) of the SPEA2, αMOGP and TSNSGP-II-a on the 6 test instances. For HV, we normalised the objective values first, and then use (1,1) as reference point. For IGD, we exact all the solutions from all algorithms for each instance and find a reference Pareto front from these solutions. For each instance, the Wilcoxon rank sum test with a significance level of 0.05 is conducted between the 30 independent runs of TSNSGP-II-a and each compared algorithm. If TSNSGP-II-a is statistically significantly better than, worse than, and comparable with that of the compared algorithm, the corresponding entry is marked with (+), (-), or (=), respectively.

From the tables, we can see that TSNSGP-II-a perform best among the compared algorithms. It can outperform SPEA2 and αMOGP on all the instances in both HV and IGD. Fig. 4.7 plots the non-dominated set of single run that has the median HV value of the 30 runs of each compared algorithm on each instance. It can be seen that the solutions obtained by TSNSGP-II-a are better distributed in the objective space and can mostly dominate that obtained by SPEA2 and αMOGP. It can be seen that the solutions of SPEA2 are biased to size, and cannot achieve good effectiveness. Typically, SPEA2 can maintain the diversity of the population as it has an internal archive to store good individuals. However, in our problem, we do training sample rotation every generation. In this case, the internal archive of SPEA2 is not able to maintain the diversity of the population. We can see that αMOGP is sensitive to instance. It can obtain good solutions on 4 out 6 instances. However, it is still slightly biased to size on Ugdb2 and Ugdb23. Overall, the results indicate that TSNSGP-II-a can generate better non-dominated solutions than the state-of-the-art MOGP approaches, and it is much easier for users to choose a routing policy from

Table 4.12: The mean and standard deviation for **HV** of the compared algorithms in test process.

Instance	**SPEA2**	**αMOGP**	**TSNSGP-II-a**
Ugdb1	0.8676(0.04)	0.9446(0.03)	0.9643(0.01)(+)(+)
Ugdb2	0.8953(0.04)	0.9524(0.03)	0.9674(0.01)(+)(+)
Ugdb8	0.8378(0.05)	0.8999(0.06)	0.9673(0.01)(+)(+)
Ugdb23	0.8664(0.03)	0.9366(0.03)	0.9509(0.02)(+)(+)
Uval9D	0.8524(0.05)	0.9326(0.04)	0.963(0.01)(+)(+)
Uval10D	0.9019(0.03)	0.9553(0.04)	0.9779(0.01)(+)(+)

Table 4.13: The mean and standard deviation for **IGD** of the compared algorithms in test process.

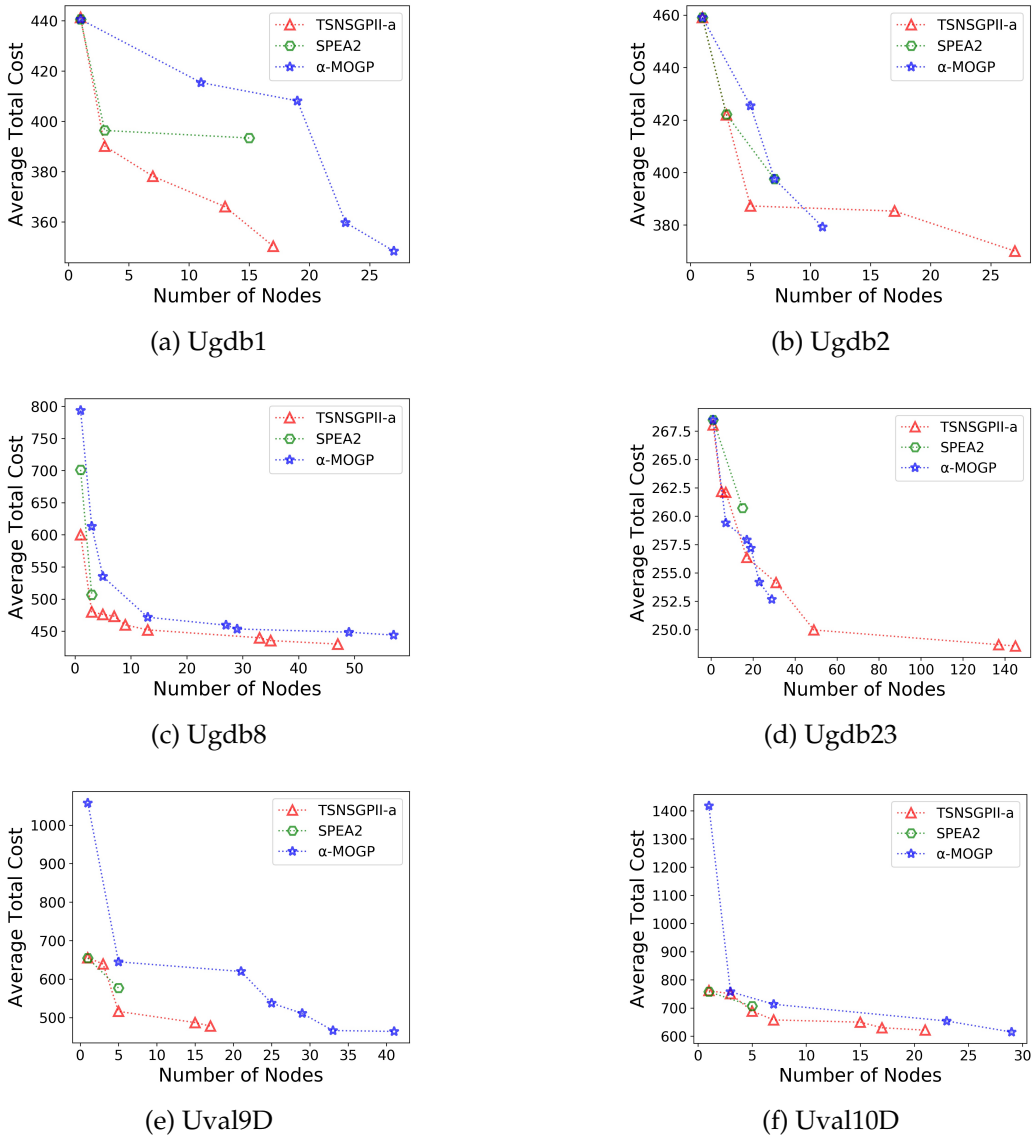
Instance	**SPEA2**	**αMOGP**	**TSNSGP-II-a**
Ugdb1	0.1022(0.04)	0.058(0.02)	0.0381(0.01)(+)(+)
Ugdb2	0.0814(0.03)	0.0544(0.01)	0.0325(0.01)(+)(+)
Ugdb8	0.2637(0.04)	0.2002(0.03)	0.0799(0.03)(+)(+)
Ugdb23	0.1124(0.03)	0.0873(0.02)	0.0412(0.01)(+)(+)
Uval9D	0.2104(0.05)	0.1262(0.03)	0.0604(0.03)(+)(+)
Uval10D	0.089(0.03)	0.0496(0.01)	0.0268(0.01)(+)(+)

the Pareto front from TSNSGP-II-a based on their preference.

4.4.4.2 Effectiveness of the External Archive

The use of the external archive is a major contribution in TSNSGP-II-a. To verify the effectiveness of the external archive, we compare TSNSGP-II-a with TSNSGP-II, which is the version without the archive. In addition, we also compare NSGP-II-a with NSGP-II. Both the algorithms have a single stage, which is the second stage of TSNSGP-II-a. NSGP-II-a has the archive, while NSGP-II has not.

Tables 4.14 and 4.15 show the results of the Hyper-Volume (HV) and Inverted Generational Distance (IGD) of NSGP-II, NSGP-II-a, TSNSGP-II and TSNSGP-II-a. We can see that NSGP-II-a can outperform NSGP-II in HV for 5 out of 6 instances, and in IGD for all instances. In addition, it can



(a) Ugdb1 (b) Ugdb2

(c) Ugdb8 (d) Ugdb23

(e) Uval9D (f) Uval10D

Figure 4.7: The non-dominated solution of the run with the median HV among the 30 runs of each compared algorithm.

be seen that TSNSGP-II-a can outperform TSNSGP-II in both HV and IGD for all the 6 instances. Overall, we can see that the external archive can significantly improve the performance of the MOGP, no matter whether

the two-stage framework is used or not.

Table 4.14: The HV of NSGP-II, NSGP-II-a, TSNSGP-II and TSNSGP-II-a.

Instance	NSGP-II	NSGP-II-a	TSNSGP-II	TSNSGP-II-a
Ugdb1	0.9104(0.03)	0.9333(0.02)(+)	0.9272(0.02)	0.9643(0.01)(+)
Ugdb2	0.9217(0.02)	0.9355(0.02)(+)	0.9383(0.02)	0.9674(0.01)(+)
Ugdb8	0.897(0.03)	0.9544(0.01)(+)	0.9046(0.03)	0.9673(0.01)(+)
Ugdb23	0.8826(0.02)	0.9135(0.02)(+)	0.8964(0.03)	0.9509(0.02)(+)
Uval9D	0.9184(0.02)	0.9227(0.02)(=)	0.9389(0.01)	0.963(0.01)(+)
Uval10D	0.9336(0.02)	0.9452(0.02)(+)	0.9572(0.02)	0.9779(0.01)(+)

Table 4.15: The IGD of NSGP-II, NSGP-II-a, TSNSGP-II and TSNSGP-II-a.

Instance	NSGP-II	NSGP-II-a	TSNSGP-II	TSNSGP-II-a
Ugdb1	0.0669(0.02)	0.056(0.01)(+)	0.058(0.02)	0.0381(0.01)(+)
Ugdb2	0.0591(0.01)	0.0531(0.01)(+)	0.0544(0.01)	0.0325(0.01)(+)
Ugdb8	0.2144(0.03)	0.1243(0.03)(+)	0.2002(0.03)	0.0799(0.03)(+)
Ugdb23	0.0981(0.02)	0.0689(0.02)(+)	0.0873(0.02)	0.0412(0.01)(+)
Uval9D	0.1563(0.02)	0.116(0.03)(+)	0.1262(0.03)	0.0604(0.03)(+)
Uval10D	0.0648(0.02)	0.0535(0.02)(+)	0.0496(0.01)	0.0268(0.01)(+)

4.4.5 Further Analysis

The ultimate goal of GPHH for UCARP is to obtain an effective and small routing policy. Here, we assume that the effectiveness is preferred, and the routing policy from the final set with the lowest total cost is selected. Then, we compare the selected routing policy from those obtained by TSNSGP-II-a with the routing policy obtained by the baseline single-objective GPHH [122].

Table 4.16 shows the mean and standard deviation of the average total cost (effectiveness) of TSNSGP-II-a and GPHH, along with that of SPEA2 and αMOGP. We can see that TSNSGP-II-a outperformed GPHH on 2 out of 6 instances, and achieved comparable effectiveness on the other instances. Besides, TSNSGP-II-a can outperform SPEA2 on all instances.

Table 4.16: The mean and standard deviation of the **average total cost** of best individual from each compared algorithms.

Instance	GPHH	SPEA2	αMOGP	TSNSGP-II-a
Ugdb1	355.47(14.83)	366.54(9.04)	354.77(12.05)	350.55(5.34)(=)(+)(=)
Ugdb2	371.72(7.67)	384.02(11.53)	370.17(7.91)	370.18(5.89)(=)(+)(=)
Ugdb8	443.34(54.3)	468.61(19.12)	441.05(10.75)	433.17(8.48)(+)(+)(+)
Ugdb23	252.47(3.11)	258.23(4.11)	250.47(2.16)	250.91(2.3)(+)(+)(=)
Uval9D	478.14(16.68)	500.72(14.71)	474.24(11.94)	475.01(7.9)(=)(+)(=)
Uval10D	620.91(7.97)	644.7(18.95)	619.59(8.45)	621.63(7.54)(=)(+)(=)

This is consistent with the patterns shown in Fig. 4.7. The median non-dominated set of SPEA2 is biased to size. TSNSGP-II-a also achieved comparable results with αMOGP on 5 out 6 instances and outperform αMOGP on 1 instance.

Table 4.17 shows the mean and standard deviation of the average number of nodes (size) of compared algorithms. We can see that SPEA2 achieves the smallest size. This is because its search process is still biased to the size, and the effectiveness is largely ignored due to the objective selection bias issue. TSNSGP-II-a obtains significantly smaller routing policies than αMOGP on 3 out of 6 instances. This is consistent with Fig. 4.7. TSNSGP-II-a can obtain comparable effectiveness with αMOGP but with smaller size. Therefore, the median non-dominated set of TSNSGP-II-a can dominate that of αMOGP. Overall, TSNSGP-II-a can evolve significantly smaller routing policies than GPHH without sacrificing effectiveness on all instances. This verifies the effectiveness of the two-stage framework and external archive used in TSNSGP-II-a.

4.4.5.1 Analysis of Routing Policies

We randomly select two best routing policies from GPHH and TSNSGP-II-a for Ugdb23, one for each algorithm. These two routing policies are shown in Fig. 4.8. The effectiveness of the routing policy obtained by GPHH is 249.89, and the effectiveness of the routing policy obtained by

Table 4.17: The mean and standard deviation of the **size** of the best individual from each compared algorithms.

Instance	GPHH	SPEA2	αMOGP	TSNSGP-II-a
Ugdb1	74.6(23.84)	12.13(9.41)	27.67(15.97)	46.33(34.29)(+)(-)(-)
Ugdb2	71.93(23.79)	16.27(12.83)	28.8(13.88)	36.67(26.14)(+)(-)(=)
Ugdb8	65.47(24.33)	16.07(20.08)	51.67(19.03)	45.47(24.12)(+)(-)(=)
Ugdb23	71.8(25.22)	14.4(13.93)	47.53(26.57)	35.53(28.19)(+)(-)(+)
Uval9D	69.27(29.46)	20.3(15.16)	58.0(29.24)	40.47(17.24)(+)(-)(+)
Uval10D	65.33(14.35)	11.4(5.39)	47.67(23.7)	34.67(17.95)(+)(-)(+)

TSNSGP-II-a is 249.27. It can be seen that the routing policy from GPHH is large and complex. We cannot gain useful knowledge from it easily. In contrast, it is easier to understand the routing policy from TSNSGP-II-a as it is smaller and simpler. The routing policy obtained by TSNSGP-II-a can be represented as follows:

$$RP = \max(S_1, S_2) \tag{4.13}$$

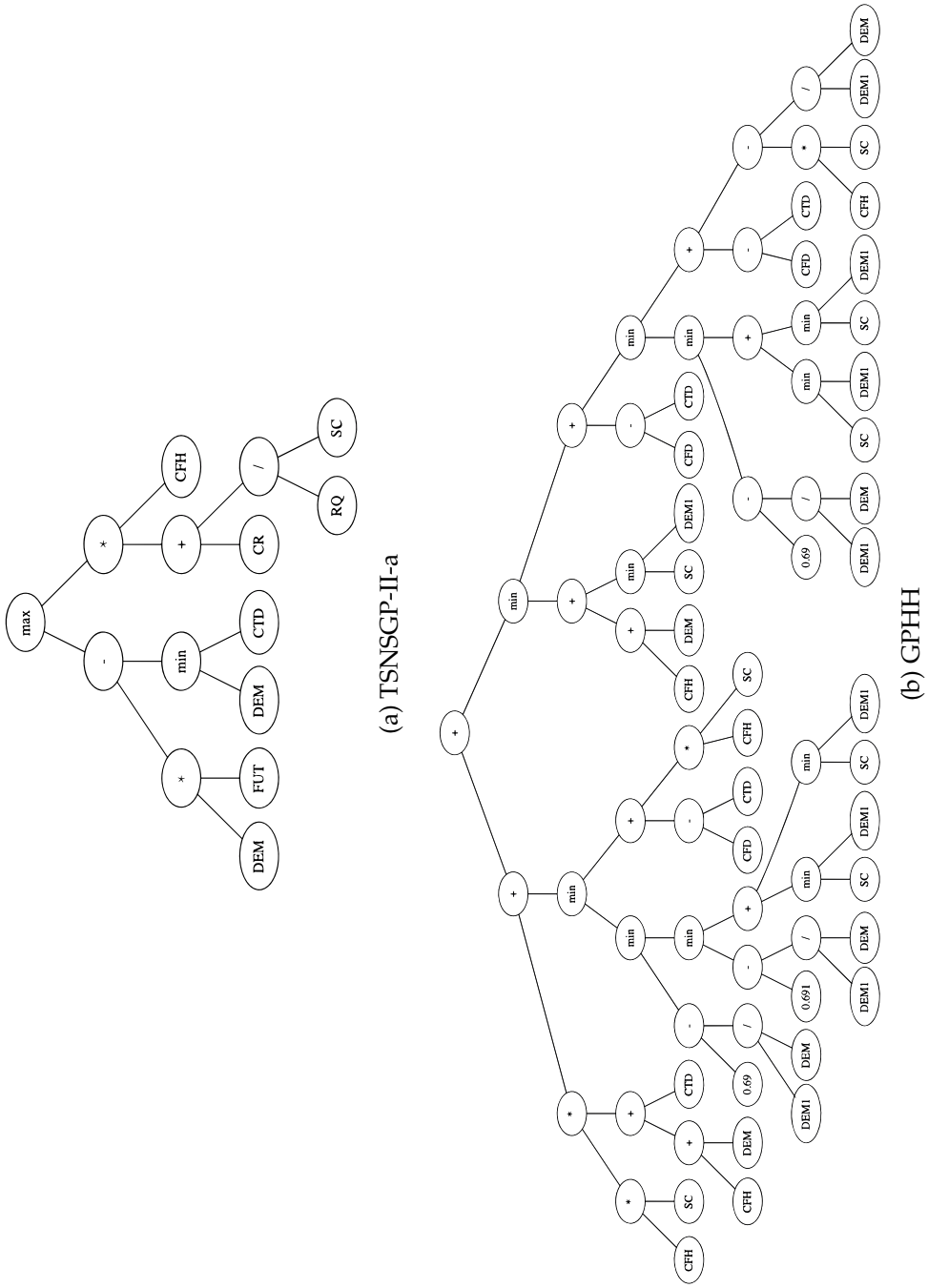
where

$$S_1 = FUT * DEM - \min(DEM, CTD), \tag{4.14}$$

$$S_2 = (CR + \frac{RQ}{SC}) * CFH \tag{4.15}$$

To gain detailed understandings of the behaviours from the routing policies evolved by TSNSGP-II-a and make it easier to understand, we transform RP to the following IF-ELSE format rule set.

if $S_1 \geq S_2$ **then**
  if $DEM \leq CTD$ **then**
    $RP = FUT * DEM - DEM$
  else
    $RP = FUT * DEM - CTD$
  end if
else
  $RP = (CR + \frac{RQ}{SC}) * CFH$
end if



(a) TSNSGP-II-a

(b) GPHH

Figure 4.8: Example routing policies obtained by TSNSGP-II-a and GPHH.

We can extract the following patterns and understandings from the rule set:

- When $S_1 \geq S_2$, RP becomes S_1. There are two possible cases for $S_1 \geq S_2$ to happen:
  - S_1 is large. This indicates that there are many remaining tasks, and all the remaining tasks have large demand and are close to the depot.
  - S_2 is small. This indicates that vehicle is relatively empty and is close to the depot. All the remaining tasks are close to the current location.

  There are two possible cases of S_1:
  - When $DEM \leq CTD$, S_1 becomes $FUT * DEM - DEM$ or $(FUT - 1) * DEM$. Note that, FUT refers to the fraction of the unassigned tasks. The value of FUT is between 0 and 1. It is hard for $(FUT - 1)$ to greater than 0. Thus, the output of $(FUT - 1) * DEM$ will be a negative value. This leads to a negative output of S_1. However, this contradicts the assumption that $S_1 \geq S_2$ because the output of S_2 is usually a positive value. Therefore, this case would never happen.
  - When $DEM > CTD$, which means that all the remaining tasks are very close to the depot and require large demand. RP becomes $FUT * DEM - CTD$.

  In this case, RP prefers the tasks with small demands and far away from the depot.

- Otherwise, $S_1 < S_2$ indicates that the vehicle is far away from the depot, and there are only a few remaining tasks with small demands, all the remaining tasks are close to the current location of the vehicle and far away from the depot. RP becomes $(CR + \frac{RQ}{SC}) * CFH$, and

RP prefers the tasks with large serving cost and close to the current location.

4.4.6 Summary

In summary, the external archive strategy stores a wide range of potentially effective individuals, in case they are lost from the population due to the training sample rotation. The experimental results show that the proposed method managed to obtain both effective and smaller routing policies than the state-of-the-art MOGPs for UCARP. It also provides a better distributed set of routing policies, making it easier for the end-users to select better routing policy based on their preferences. We have also verified the efficacy of the external archive. The external archive can significantly improve the performance of the MOGP in terms of HV and IGD. The evolved routing policies are much easier to understand than that in standard GPHH.

4.5 MOGP with Self Adaptive α Dominance and Archive Strategies

Previous sections proposed different strategies to handle either the objective selection bias issue or the stochastic evaluation issue. This section aims to propose an approach that handle both issues simultaneously. The proposed approach contains both α dominance strategy and archive strategy.

This work aims to address the following limitations of the existing α-dominance and archive strategies in MOGP for evolving routing policies. First, the effectiveness of the α-dominance method highly depends on the α value, which is challenging to determine. Intuitively, α should be increased if the population is biased to small ineffective individuals, and decreased if most individuals in the population are too large. However,

the α adaptation is non-trivial due to the difficulty of estimating the accurate boundaries of the effectiveness and program size. Second, the existing archive update strategies are not effective enough. It can include duplicate individuals in the archive and reduce the diversity of the archive. To address the issue of sensitivity to the α value, in the new algorithm we propose a new scheme to estimate the relative program size of the population, and a new α adaptation strategy guided by the relative program size estimation. To address the archive diversity issue, we propose a new archive update strategy that avoids including phenotypic duplicate individuals in the archive.

4.5.1 Overall Framework

Fig. 4.9 shows the overall framework of the newly proposed αMOGP-a. Firstly, a population is randomly initialised and an empty archive is created. Each individual, which is represented as a GP tree, is evaluated based on a set of training instances (simulations). Specifically, the size (number of nodes) and the effectiveness (the mean total cost of the solutions generated for the training instances) are calculated for each individual. After that, an α non-dominated sorting is carried out to sort all the individuals in the population. The tournament selection is utilised to select parents to generate offspring. The offspring population is combined with the parent population. This follows the process of NSGA-II [50]. As we use the training sample rotation in UCARP, the parents in the combined population are re-evaluated on the instances used to evaluate the offspring. The boundary of individual size will be identified and updated. The boundary is utilised to identify the bias. The combined population is then sorted using the α non-dominated sorting and the non-dominated individuals are stored in the archive. The individuals in the archive will be involved in the breeding process. The utilisation of the archive will be discussed in Section 4.5.3. The α parameter is then updated based on

the distribution of the population. The new α adaptation scheme will be introduced in Section 4.5.2. Then, the top N (N is the population size) individuals will be selected to next generation. When the stopping criteria are met, the archive individuals will be re-evaluated using a validation set of samples which is different from the training and test samples. A traditional non-dominated sorting is applied to the archive and the final non-dominated solutions are returned.

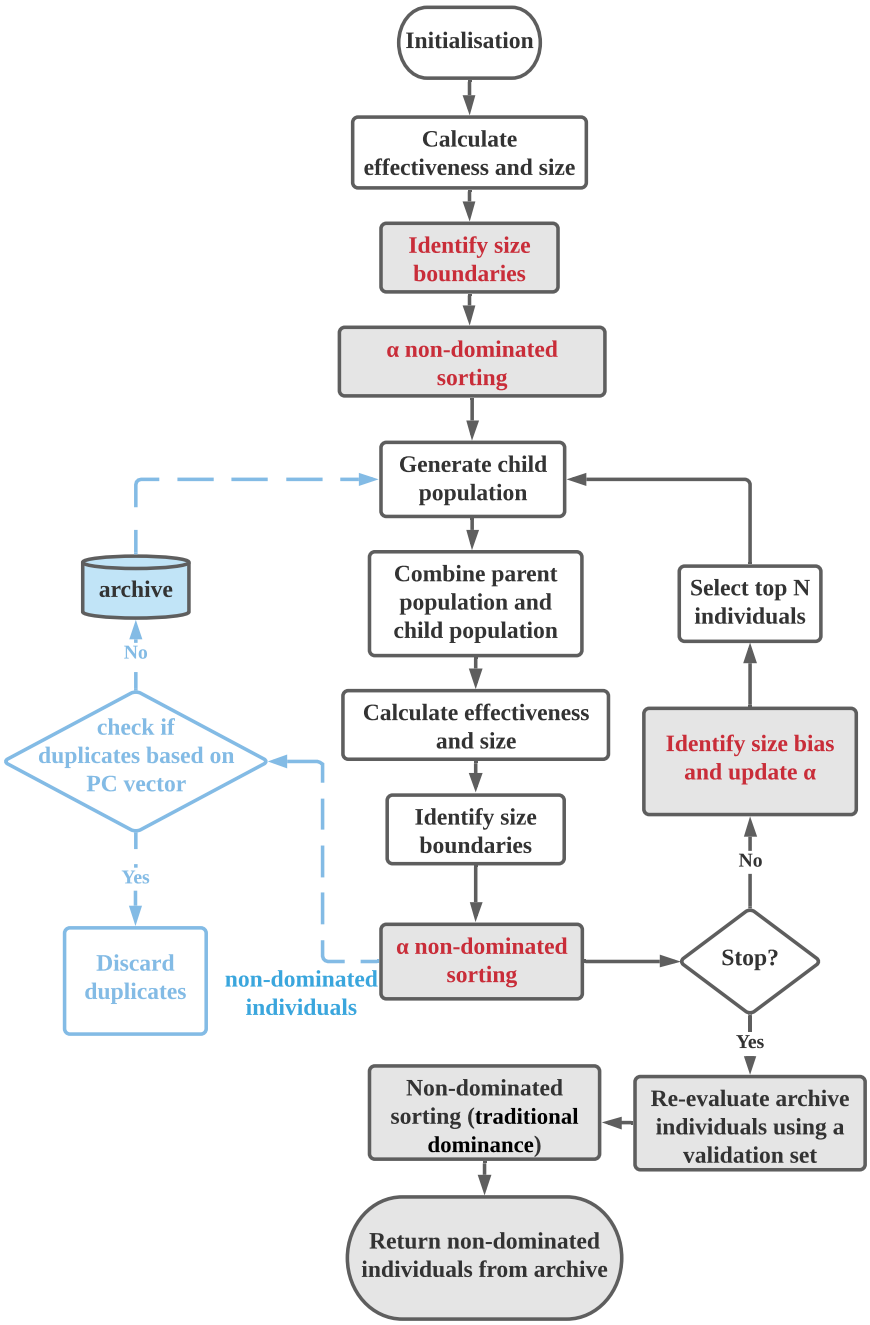


Figure 4.9: The overall Framework of αMOGP-a.

4.5.2 New α Adaptation Scheme

The α parameter controls the balance between the effectiveness and size during the search. To control the balance better, it is necessary to adaptively adjust the α value during the evolutionary process. In this chapter, we propose a novel α adaptation scheme that can automatically adjust the α value based on the distribution of current α non-dominated solutions generated by α dominance. Here, we use size alone to estimate the bias, since it is difficult to estimate the range of effectiveness of routing policies during the search process (e.g., a random routing policy can be very poor). If the current α non-dominated solutions is biased away from size, we will decrease the α value. Otherwise, if the current α non-dominated solutions is biased towards size, we will increase the α value.

The procedure of estimating the current bias to size is as follows. First, we calculate the mean size of the α non-dominated individuals in the current population, denoted as S_{pop}. Then, we compare S_{pop} with the historical range of the size [S_{min}, S_{max}]. If S_{pop} leans towards S_{min}, then we consider the current population to bias towards size. If S_{pop} is closer to S_{max}, then the current population tends to bias away from size. In addition, the amount of increase/decrease in α should depend on the degree of the bias. For example, we should increase α more if S_{pop} is closer to S_{min}.

Taking the above into account, we design the α adaptation scheme as follows.

$$\theta_{old} = \arctan(\alpha_{old}) \tag{4.16}$$

$$\theta_{new} = \theta_{old} + \frac{S_{max} + S_{min} - 2S_{pop}}{S_{max} - S_{min}} * \phi \tag{4.17}$$

$$\alpha_{new} = \tan(\theta_{new}) \tag{4.18}$$

where α_{old} and α_{new} are the α values before and after the update. θ_{old} and θ_{new} are angles corresponding to α_{old} and α_{new} (the angle between the horizontal line and the boundary of the dominance region induced by α in Fig. 3). ϕ is a step size which indicates how many degrees we need increase or

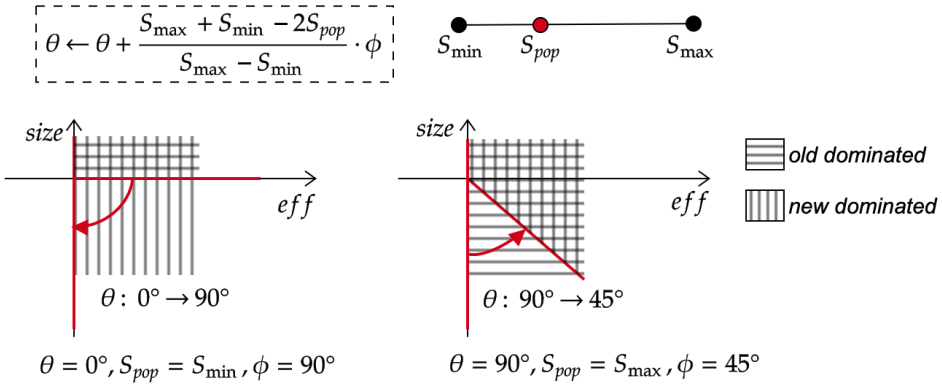


Figure 4.10: The relationship between changes in the dominance area and the location the current Pareto front.

decrease the angle to adjust the dominance area. In other words, the angle θ changes linearly from $-\phi$ to ϕ when S_{pop} changes from S_{min} to S_{max}. θ is unchanged if $S_{pop} = (S_{max} + S_{min})/2$, i.e., in the middle of the historical sizes.

Fig. 4.10 shows two examples of the α adaptation scheme. In the first example, the current α is 0, $S_{pop} = S_{min}$, and $\phi = 90°$. In this case, the current population is strongly biased towards size. Thus, we increase the angle by $90°$ (the maximum possible change) so that the subsequent search will ignore size and fully focus on effectiveness. In the second example, the current α is ∞ (angle is $90°$), $S_{pop} = S_{max}$, and $\phi = 45°$. In other words, the current population is biased away from size (the individuals are very large), and we need to decrease α. In this case, we decrease the angle by $45°$ (due to the step size ϕ), and α is decreased from ∞ to 1.

4.5.3 New Archive Strategy

In this paper, the external archive is used to store the potentially good individuals occurred in the evolutionary process. At the end of each generation, the α non-dominated individuals are added to the archive if it is

not a duplicate of any individual in the archive. The duplication checking process is based on the comparison of the Phenotypic Characterisation vector (PC vector) [215]. The phenotypic characterisation of an individual is an vector of decisions it makes in a list of representative decision situations. For each decision situation, the decision is the index of its selected task for the idle vehicle. If two individuals have similar PC vectors, they tend to have similar behaviour and effectiveness. Since the phenotypic characterisation is a vector of task indices, we use the following equation to calculate the distance between two PC vectors $\mathbf{p}_1$ and $\mathbf{p}_2$:

$$\Delta(\mathbf{p}_1, \mathbf{p}_2) = \sum_{i=1}^{L} \mathbb{I}_{p_{1i}=p_{2i}} \tag{4.19}$$

where $\mathbb{I}_{p_{1i}=p_{2i}} = 1$ if $p_{1i} = p_{2i}$, and $\mathbb{I}_{p_{1i}=p_{2i}} = 0$, otherwise. If the distance of two individuals is zero, we consider the two individuals to be duplicates. If a new individual and an existing individual in the archive are duplicates, then the new individual will replace the existing individual only if it has a smaller size. The archive updating process is shown in Algorithm 10.

To fully utilise the archive, we use it to provide parents for breeding. When the breeding process starts, the archive is merged with the current population to create a mating pool. Then, we select parents from the mating pool using tournament selection. Note that the individuals in the archive are evaluated on different instance samples from the current population, due to the instance rotation. To compare between the individuals in the archive and the current population, we assume that the individuals in the archive are slightly worse than the non-dominated individuals in the current population, but better than other dominated individuals. If two selected individuals are both from current population, the better individual is the one that is dominated by fewer individuals or dominated by same number of individuals but has smaller crowding distance. If two individuals are both selected from the archive, we can randomly pick one

Algorithm 10: The updating process of the archive.

```
Input: The current population pop, the external archive, the population size
       popsize
Output: The updated archive
Obtain the α non-dominated individuals pop_nd in pop;
for ind ∈ pop_nd do
    Set duplicate = false;
    for ind' ∈ archive do
        calculate distance between ind and ind' using Eq.4.19;
        if distance = 0 then
            duplicate = true;
            if size(ind) < size(ind') then
                replace ind' with ind
            end
            break;
        end
    end
end
return archive;
```

as a parent. If one is from the archive and the other one is from the current population, as long as the individual from the current population is not a non-dominated individual, the archive individual will be chosen as a parent.

Typically, the final non-dominated solutions are chosen based on the performance of the solution on the training set at the last generation. However, this may lead to the final non-dominated solution being biased towards the training set of the last generation, which in turn results in the non-dominated solution performing poorly on the test set. In addition, to select non-dominated solutions from the archive, solutions in the archive should be re-evaluated. Therefore, at the end of the GP process, the archive will be passed to a validation set of samples to select the non-dominated solutions.

4.5.4 Experiment Design

To verify the effectiveness of the newly proposed αMOGP-a, we design two sets of experiments:

1. Compare with the current state-of-the-art MOGP algorithm for UCARP (TSMOGP-a [199]), to verify the quality of the obtained Pareto front, using Hyper Volume (HV) and Inverted Generational Distance (IGD).

2. Compare with the state-of-the-art algorithms for reducing program size (i.e., bloat control methods, such as Tarpeian, linear parametric parsimony pressure (LPPP), double tournament (DT) in the literature [130] and GPHH-N [201]), to verify the effectiveness and size of the obtained most effective routing policy.

A GP run contains two main phases, i.e., training and test. In the training phase, routing policies are trained on 350 training samples (250 samples for training and 100 sample for validating). The 250 samples are divided into 50 subset. Each subset contains 5 training samples. For each generation, one subset is used for fitness evaluation. At the end of training process, 100 validating samples are used to select final Pareto front from the archive. In the test phase, the final Pareto front is tested on 500 test samples. All the samples share the same graph topology of a UCARP instance but are generated by randomly sampling the stochastic task demands and deadheading costs.

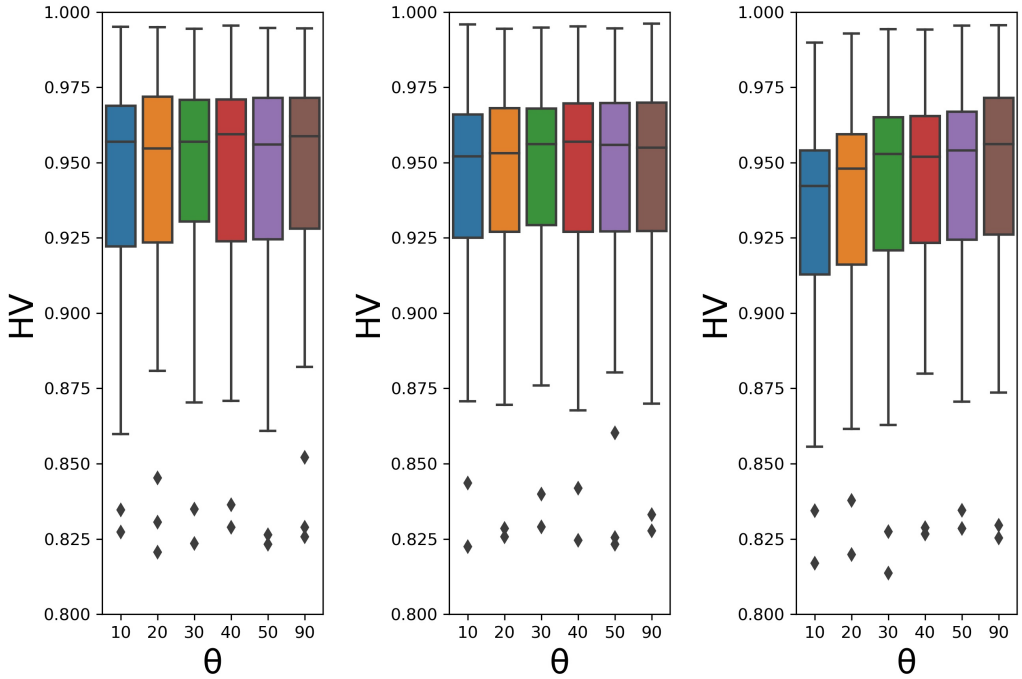
In this work, αMOGP-a is verified on two commonly used large UCARP datasets, i.e., the *Ugdb* and *Uval* which have been introduced in Table 2.1.

The initialisation method of the population for all compared algorithms is Ramped Half-and-Half. All these parameter settings are commonly used in the GPHH literature [29, 32, 143]. The population size for MOGP algorithms are set to 500. For single-objective GP algorithm, it is set to 1000. This is because MOGP combines parent and offspring population to generate a new combined population which will double the original

population size. For fair comparison, the population size of the MOGP algorithms is set as the half of the single-objective algorithm. The maximal number of generations is set to 50. The tournament selection size is 2. The ratio of crossover and mutation operator is set as 0.85 : 0.15. The maximal tree depth of all GP algorithm is set to 8. The elitism size of the single-objective GP is set to 10. All the compared MOGP algorithms are non-dominated sorting based, thus there is no need to use elitism. The Evolutionary Computation Java (ECJ) package [131] is utilised to implement all the algorithms. The results are collected based on 30 independent runs for each algorithm on each UCARP instance. The meanings of each terminal can be found in Table 2.3

4.5.5 Parameter Sensitive Analysis

For the α dominance, the initial value of α and the step size ϕ are very important. A good initial α value can help the algorithm find good individuals faster. A good step size can help α faster converge to the proper value. Thus, we design a set of experiments to identify a proper initial α value and step size. All the parameters are tested in the newly proposed αMOGP-a algorithm. We set 3 different initial α values, i.e., 0, 1 and 100. When initial α value is set to 0, the algorithm starts from the traditional dominance that treats the two objectives equally. When initial α value is set to 1, the algorithm focuses more on the effectiveness. When initial α value is set to 100, the algorithm almost becomes a single-objective algorithm that only optimise the effectiveness. We also have 5 different step size, 10, 20, 30, 50 and 90. Due to the page limit, we summary the experimental results in the boxplot format, each box in the boxplot represents the distribution of the 30 runs of the proposed algorithm with the corresponding parameter setting over the 57 instances. We examined the performance on both HV and IGD. As IGD has the same pattern as the HV, we omit the IGD results here due to the page limit. The results are shown in Fig. 4.11.



(a) $\alpha = 0$ (b) $\alpha = 1$ (c) $\alpha = 100$

Figure 4.11: The experimental results for parameter sensitivity analysis in terms of HV.

From Fig. 4.11, we can observe that the algorithms with initial α value of 0 and 1 perform better than algorithms with initial α value of 100 as they can obtain more stable results and which is independent of the θ value. This indicates that a smaller initial value is a better choice. By looking into the results, when α equals 100, the algorithm will focus too much on the effectiveness, and almost becomes a single-objective algorithm. The population is occupied by only large individuals. This makes it hard for the algorithm to find small effective individuals. As a result, it is difficult for the algorithm to find a complete Pareto front. Intuitively, 0 is a better initial value than 1 since α dominance becomes traditional dominance when α equals 0. We can use the traditional dominance as a start point. For the step size, it can be seen that step size is not very sensitive when initial

value equals 0 and 1, but 90 seems to be a slightly better choice. Therefore, the initial α value of 0 and step size of 90 are utilised in the subsequent experiments.

4.5.6 Results

We use the Wilcoxon rank sum test with the significance level of 0.05 for statistical analysis. In each table, each "+", "-" or "=" next to αMOGP-a indicates that αMOGP-a performed statistically significantly better than, worse than, or comparable to the compared algorithm.

4.5.6.1 Hyper-Volume and IGD Test Performance

Tables 4.18 and 4.19 show the results of the mean and standard deviation of HV and IGD obtained by the compared MOGP methods on the test samples.

From Tables 4.18 and 4.19, we can see that αMOGP-a outperforms TSMOGP-a on 20 out of 57 instances and achieves comparable results on remaining instances in terms of HV. The αMOGP-a also performs significantly better than TSMOGP-a on 16 out of 57 instances and achieves comparable results on 40 instances in terms of IGD. This is as expected since αMOGP-a can better handle the objective selection bias issue and the issue of potentially good individuals. By looking further into the results, we can also see that even though αMOGP-a and TSMOGP-a obtain competitive results on many instances, the mean value of each instance of αMOGP-a is slightly better than that achieved by TSMOGP-a in terms of HV and IGD. All these observations can indicate that αMOGP-a can generate better Pareto front than the TSMOGP-a on some instances and generate at least competitive Pareto front on other instances.

The Pareto front with the median HV over the 30 independent runs on some representative instances is shown in Fig. 4.12. It can be seen that the non-dominated solutions generated by αMOGP-a is better than TSMOGP-

Table 4.18: The mean and standard deviation of the **HV** and **IGD** of the compared MOGP methods on the on **Ugdb** Dataset.

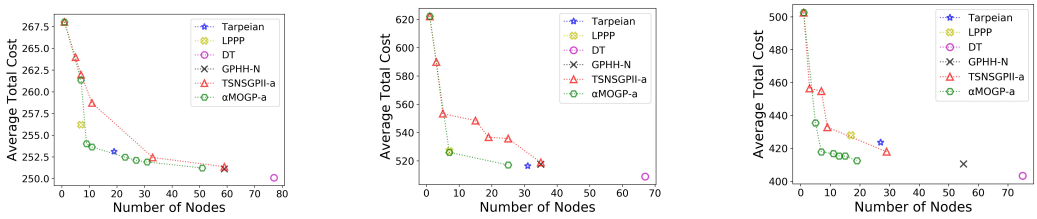
	HV		IGD	
Instance	TSMOGP-a	αMOGP-a	TSMOGP-a	αMOGP-a
Ugdb1	0.9327(0.015)	0.9459(0.017)(+)	0.0571(0.015)	0.0428(0.012)(+)
Ugdb2	0.9331(0.026)	0.9501(0.015)(+)	0.0618(0.024)	0.0488(0.012)(+)
Ugdb3	0.9648(0.005)	0.9662(0.005)(=)	0.1111(0.024)	0.1350(0.023)(-)
Ugdb4	0.9749(0.018)	0.9755(0.011)(=)	0.0672(0.016)	0.0710(0.016)(=)
Ugdb5	0.9416(0.020)	0.9512(0.016)(+)	0.0569(0.018)	0.0467(0.012)(+)
Ugdb6	0.9146(0.027)	0.9137(0.024)(=)	0.0610(0.017)	0.0597(0.015)(=)
Ugdb7	0.9447(0.006)	0.9531(0.012)(+)	0.0361(0.003)	0.0334(0.007)(+)
Ugdb8	0.9260(0.011)	0.9358(0.012)(+)	0.066(0.0150)	0.0616(0.016)(=)
Ugdb9	0.8962(0.022)	0.9068(0.022)(=)	0.1039(0.022)	0.0990(0.022)(=)
Ugdb10	0.9553(0.014)	0.9600(0.013)(=)	0.0698(0.015)	0.0674(0.013)(=)
Ugdb11	0.8233(0.028)	0.8257(0.021)(=)	0.1321(0.026)	0.1293(0.019)(=)
Ugdb12	0.8835(0.036)	0.9025(0.033)(+)	0.0819(0.030)	0.0636(0.025)(+)
Ugdb13	0.9215(0.019)	0.9307(0.018)(=)	0.0498(0.013)	0.0453(0.014)(=)
Ugdb14	0.9438(0.021)	0.9405(0.014)(=)	0.0467(0.014)	0.0429(0.008)(=)
Ugdb15	0.9827(0.008)	0.9842(0.013)(=)	0.0250(0.009)	0.0216(0.010)(=)
Ugdb16	0.9401(0.004)	0.9392(0.002)(=)	0.0946(0.017)	0.0774(0.016)(+)
Ugdb17	0.9858(0.001)	0.9945(0.004)(+)	0.0218(0.001)	0.0145(0.004)(+)
Ugdb18	0.9606(0.014)	0.9647(0.012)(=)	0.0408(0.013)	0.0366(0.010)(=)
Ugdb19	0.9697(0.036)	0.9811(0.024)(+)	0.0705(0.024)	0.0630(0.016)(=)
Ugdb20	0.9359(0.024)	0.9346(0.026)(=)	0.0493(0.017)	0.0471(0.015)(=)
Ugdb21	0.9305(0.017)	0.9337(0.016)(=)	0.0626(0.015)	0.0581(0.013)(=)
Ugdb22	0.9079(0.030)	0.9281(0.016)(+)	0.0658(0.024)	0.0476(0.014)(+)
Ugdb23	0.9151(0.013)	0.9232(0.012)(+)	0.0639(0.011)	0.0549(0.007)(+)

a. Specifically, αMOGP-a can find solutions with better effectiveness than that obtained by TSMOGP-a. This is because αMOGP-a use the α dominance to adaptively give more pressure to the effectiveness during the evolutionary process when necessary. This can help αMOGP-a generating more effective and compact individuals during the evolutionary process. As other algorithms are single-objective algorithms, their Pareto fronts have only one point. We can see that the solutions of Tarpeian and LPPP

Table 4.19: The mean and standard deviation of the **HV** and **IGD** of the compared MOGP methods on the on **Uval** Dataset.

	HV		IGD	
Instance	TSMOGP-a	αMOGP-a	TSMOGP-a	αMOGP-a
Uval1A	0.9772(0.009)	0.9777(0.013)(=)	0.0239(0.006)	0.0218(0.007)(=)
Uval1B	0.9747(0.015)	0.9753(0.009)(=)	0.0215(0.010)	0.0209(0.010)(=)
Uval1C	0.8787(0.031)	0.8910(0.026)(=)	0.0784(0.028)	0.0701(0.021)(=)
Uval2A	0.9723(0.009)	0.9735(0.012)(=)	0.0323(0.006)	0.0281(0.007)(+)
Uval2B	0.9587(0.012)	0.9649(0.012)(+)	0.0372(0.009)	0.0330(0.007)(=)
Uval2C	0.8048(0.049)	0.8289(0.034)(+)	0.1389(0.039)	0.1163(0.024)(+)
Uval3A	0.9673(0.010)	0.9751(0.009)(+)	0.0415(0.006)	0.0357(0.007)(+)
Uval3B	0.8827(0.024)	0.8959(0.020)(+)	0.0878(0.018)	0.0766(0.016)(+)
Uval3C	0.8513(0.047)	0.8521(0.051)(=)	0.1219(0.040)	0.1224(0.044)(=)
Uval4A	0.9673(0.011)	0.9712(0.008)(=)	0.0433(0.012)	0.0466(0.010)(=)
Uval4B	0.9672(0.011)	0.9689(0.013)(=)	0.0382(0.009)	0.0386(0.011)(=)
Uval4C	0.9538(0.013)	0.9628(0.011)(+)	0.0510(0.010)	0.0454(0.010)(+)
Uval4D	0.8814(0.036)	0.8979(0.036)(=)	0.0771(0.023)	0.0685(0.022)(=)
Uval5A	0.9757(0.006)	0.9772(0.002)(=)	0.0253(0.008)	0.0226(0.005)(=)
Uval5B	0.9662(0.009)	0.9728(0.006)(+)	0.0296(0.008)	0.0238(0.007)(+)
Uval5C	0.9643(0.011)	0.9674(0.008)(=)	0.0329(0.010)	0.0285(0.007)(+)
Uval5D	0.9265(0.012)	0.9428(0.015)(+)	0.0778(0.013)	0.0713(0.014)(=)
Uval6A	0.9785(0.006)	0.9782(0.008)(=)	0.0297(0.010)	0.0307(0.012)(=)
Uval6B	0.9644(0.008)	0.9626(0.016)(=)	0.0437(0.010)	0.0424(0.016)(=)
Uval6C	0.9071(0.033)	0.9114(0.040)(=)	0.0726(0.024)	0.0683(0.029)(=)
Uval7A	0.8902(0.035)	0.8821(0.037)(=)	0.0712(0.022)	0.0737(0.024)(=)
Uval7B	0.9399(0.017)	0.9476(0.019)(=)	0.0558(0.009)	0.0493(0.012)(=)
Uval7C	0.9328(0.025)	0.9444(0.025)(+)	0.0556(0.019)	0.0474(0.018)(=)
Uval8A	0.9752(0.006)	0.9789(0.006)(+)	0.0386(0.013)	0.0350(0.013)(=)
Uval8B	0.9673(0.010)	0.9697(0.006)(=)	0.0342(0.009)	0.0331(0.007)(=)
Uval8C	0.9126(0.020)	0.9197(0.015)(=)	0.0776(0.016)	0.0724(0.015)(=)
Uval9A	0.9692(0.007)	0.9713(0.005)(=)	0.0282(0.006)	0.0258(0.006)(=)
Uval9B	0.9646(0.012)	0.9656(0.010)(=)	0.0332(0.009)	0.0365(0.008)(=)
Uval9C	0.9577(0.007)	0.9588(0.011)(=)	0.0301(0.006)	0.0295(0.009)(=)
Uval9D	0.9176(0.012)	0.9211(0.008)(=)	0.0617(0.011)	0.0584(0.010)(=)
Uval10A	0.9765(0.010)	0.9764(0.012)(=)	0.0254(0.008)	0.0254(0.009)(=)
Uval10B	0.9772(0.004)	0.9781(0.005)(=)	0.0481(0.015)	0.0530(0.018)(=)
Uval10C	0.9689(0.008)	0.9707(0.007)(=)	0.0388(0.011)	0.0384(0.011)(=)
Uval10D	0.9643(0.012)	0.9701(0.010)(+)	0.0408(0.010)	0.0377(0.010)(=)

are allocated on the left side of each figure. This indicates that Tarpeian AND LPPP evolve small but less effective solutions. The solutions of DT



(a) Ugdb23 (b) Uval5C (c) Uval6C

Figure 4.12: The non-dominated solution of the run with the median HV over 30 runs on compared algorithms.

and GPHH-N are allocated on the right side of each figure. This indicates that DT and GPHH-N evolve effective but large solutions. The αMOGP-a can evolve Pareto front which contains solutions varies from small (ineffective) to large (effective). As a result, users can choose a solution based on their preference.

4.5.6.2 Most Effective Individual

Although a set of non-dominated solutions are obtained, only a single individual needs to be selected as the routing policy in practical use. Here, we consider the effectiveness as the primary objective, and select the individual with the best effectiveness from the non-dominated set. Then, we compare the effectiveness and size of the most effective individuals obtained by each algorithm. Each "+", "-" or "=" next to the compared algorithms indicates that αMOGP-a performs statistically significantly better than, worse than, or comparable to the compared algorithm.

Tables 4.20 and 4.21 show the mean and standard deviation of the average total cost (effectiveness) of the compared algorithms. It can be seen that αMOGP-a performs significantly better than Tarpeian on 32 out of 57 instances, and never performs significantly worse than Tarpeian. The αMOGP-a also outperforms LPPP on almost all instances. There is an interesting observation that both DT and GPHH-N can outperform αMOGP-

Table 4.20: The Experimental Results of (**Effectiveness/Average total cost**) for Compared Algorithms on **Ugdb** Dataset.

Instance	Tarpeian	LPPP	DT	GPHH-N	TSMOGP-a	αMOGP-a
Ugdb1	359.741(9.67)(+)	361.5418(8.43)(+)	353.9708(17.169)(=)	347.0554(5.705)(-)	354.4534(6.151)(+)	349.8263(6.501)
Ugdb2	376.6364(11.098)(+)	385.6952(10.432)(+)	371.489(7.852)(=)	369.7039(7.791)(=)	375.5819(9.299)(+)	369.4091(5.863)
Ugdb3	307.3752(1.659)(=)	311.9899(4.892)(+)	306.3066(2.038)(=)	307.6944(2.444)(+)	307.1549(1.537)(=)	306.8357(1.532)
Ugdb4	324.5011(4.963)(+)	331.275(16.411)(+)	322.1242(1.958)(=)	322.646(1.942)(=)	322.7723(4.693)(=)	322.0611(2.534)
Ugdb5	432.4777(9.581)(=)	451.353(14.359)(+)	425.016(8.723)(=)	422.739(6.281)(-)	433.6248(11.01)(=)	428.6176(8.313)
Ugdb6	362.5787(16.472)(+)	362.7825(1.89)(+)	343.9608(9.035)(-)	338.8735(3.449)(-)	349.2738(8.652)(=)	349.8394(7.416)
Ugdb7	359.4155(0.96)(+)	359.8538(0.0)(+)	357.3026(3.399)(+)	355.1464(4.582)(=)	357.2867(1.504)(+)	354.9296(3.213)
Ugdb8	441.1566(12.024)(+)	456.8996(13.306)(+)	432.0626(10.601)(=)	430.0516(6.924)(=)	437.2732(4.744)(+)	432.5656(5.593)
Ugdb9	399.2624(8.284)(+)	406.644(11.598)(+)	392.9335(10.982)(=)	383.0641(8.999)(-)	396.7845(8.619)(+)	392.4596(8.449)
Ugdb10	296.5628(4.87)(+)	300.0145(2.93)(+)	291.9685(5.712)(=)	289.9431(4.95)(-)	294.8434(3.747)(=)	293.4669(3.696)
Ugdb11	443.5919(5.715)(+)	446.8559(4.755)(+)	438.2663(5.222)(=)	432.2868(8.233)(-)	439.2069(4.559)(=)	438.8171(3.598)
Ugdb12	618.1571(14.697)(+)	624.9459(18.742)(+)	606.285(16.707)(=)	605.8586(9.306)(=)	610.9021(15.611)(+)	604.2255(13.99)
Ugdb13	585.3549(7.584)(=)	593.7929(10.522)(+)	581.0675(7.594)(=)	576.8475(7.093)(-)	585.7524(6.656)(=)	582.4751(6.353)
Ugdb14	108.6919(2.039)(=)	118.0813(2.556)(+)	108.072(2.439)(-)	107.0618(1.437)(-)	108.0788(1.482)(=)	108.4235(1.018)
Ugdb15	58.3416(0.316)(=)	62.0083(0.0)(+)	58.5233(0.974)(=)	58.4673(1.037)(=)	58.2869(0.187)(=)	58.2604(0.356)
Ugdb16	134.5298(0.06)(+)	134.4749(0.0)(=)	134.5552(0.052)(+)	134.4879(0.411)(=)	134.393(0.222)(=)	134.4488(0.14)
Ugdb17	91.2706(0.0)(+)	93.8657(0.0)(+)	91.1897(0.251)(+)	91.2842(0.107)(+)	91.2649(0.03)(+)	91.076(0.091)
Ugdb18	169.0591(2.795)(=)	182.0322(6.18)(+)	168.7085(4.76)(=)	166.973(1.606)(-)	169.0211(2.14)(=)	168.5346(1.795)
Ugdb19	61.7611(1.723)(=)	68.0709(0.802)(+)	61.5426(1.902)(=)	60.8448(0.547)(=)	61.7781(1.601)(=)	61.3057(1.071)
Ugdb20	128.8132(2.175)(=)	137.3079(0.0)(+)	127.914(4.238)(=)	126.8208(2.337)(-)	127.794(1.565)(=)	127.9286(1.693)
Ugdb21	166.2336(3.428)(=)	180.8426(5.53)(+)	165.9424(4.636)(=)	164.6705(3.54)(-)	166.5348(1.982)(=)	166.0947(1.814)
Ugdb22	211.7671(3.45)(+)	221.2032(2.724)(+)	210.1016(2.105)(=)	210.0523(2.478)(=)	210.8277(2.0)(+)	209.5899(1.034)
Ugdb23	251.9495(2.512)(=)	257.2459(4.187)(+)	251.1037(2.753)(=)	248.748(1.801)(-)	251.8207(1.736)(=)	251.0229(1.492)

a on a number of instances. DT will first use tournament selection to select multiple effective individuals and then select the smallest one from the selected ones as parent. So DT still prefers to select the effective individuals as parent. For GPHH-N, it utilises the niching simplification method to reduce the size but its main goal is still on the effectiveness. Both algorithms focus more on the effectiveness. Thus, they can achieve better effectiveness than αMOGP-a on some instances. However, if we go a step further and look into the experimental results, we see that the effectiveness of αMOGP-a is just slightly worse than that in DT and GPHH-N. Furthermore, as we will see later, αMOGP-a can obtain much smaller size than DT and GPHH-N. In addition, αMOGP-a can outperform the TSMOGP-a on 18 instances and never perform significantly worse than TSMOGP-a on remaining instances.

Tables 4.22 and 4.23 show the mean and standard deviation of the average size (number of nodes) of the compared algorithms. We can see that

Table 4.21: The Experimental Results of (**Effectiveness/Average total cost**) for Compared Algorithms on **Uval** Dataset.

Instance	Tarpeian	LPPP	DT	GPHH-N	TSMOGP-a	αMOGP-a
Uval1A	181.9436(4.437)(+)	188.8427(6.633)(+)	177.7145(5.283)(=)	174.5455(1.57)(-)	178.3407(2.691)(=)	178.2425(3.673)
Uval1B	186.0304(2.533)(=)	190.4595(3.237)(+)	183.8868(1.058)(-)	183.7689(0.869)(-)	186.0899(3.287)(=)	185.6556(1.88)
Uval1C	320.5283(10.524)(+)	329.6197(8.438)(+)	315.8533(8.309)(=)	309.8156(12.435)(-)	317.9679(8.34)(=)	314.4565(7.278)
Uval2A	233.3371(6.258)(=)	239.207(5.083)(+)	231.5573(4.049)(=)	230.5274(3.451)(-)	232.9362(2.52)(=)	232.7639(3.327)
Uval2B	283.4977(12.472)(+)	295.0745(15.143)(+)	279.484(3.985)(=)	278.0283(3.801)(=)	280.5628(4.185)(=)	278.6683(4.082)
Uval2C	617.0144(18.002)(+)	622.1776(13.246)(+)	598.0613(22.492)(=)	591.1879(14.467)(-)	615.4274(21.033)(+)	606.9756(14.595)
Uval3A	82.8063(1.183)(=)	88.3126(2.176)(+)	82.4896(1.249)(=)	81.7668(0.698)(-)	83.3861(0.956)(+)	82.5919(0.906)
Uval3B	100.1533(3.669)(+)	101.8713(2.175)(+)	96.5034(1.66)(=)	95.3129(1.293)(-)	97.4254(1.338)(=)	96.8018(1.08)
Uval3C	179.7858(9.375)(=)	193.8458(14.893)(+)	177.4681(5.522)(-)	173.7863(4.013)(-)	182.4238(6.7)(=)	182.3861(7.16)
Uval4A	430.4099(27.157)(+)	429.4005(7.934)(+)	420.1009(5.788)(=)	417.5135(4.281)(-)	424.1895(6.985)(=)	421.1683(4.948)
Uval4B	448.8171(10.208)(=)	457.2661(12.784)(+)	444.0003(8.243)(=)	439.2253(6.023)(-)	446.7894(7.41)(=)	445.7183(9.372)
Uval4C	499.4249(12.378)(+)	503.3278(13.547)(+)	493.0886(17.139)(=)	486.7424(11.729)(-)	497.6095(10.661)(+)	490.4748(8.96)
Uval4D	726.6945(34.562)(=)	728.593(39.509)(=)	701.2216(31.662)(=)	689.6775(22.469)(-)	722.9498(28.646)(=)	709.5947(28.792)
Uval5A	444.9818(4.721)(+)	451.2758(3.761)(+)	440.4567(3.934)(-)	440.6779(8.742)(-)	443.9832(4.71)(=)	442.8144(2.026)
Uval5B	477.6904(8.99)(+)	479.6288(7.16)(+)	475.9497(27.624)(=)	467.0353(4.985)(-)	476.082(6.158)(+)	471.7457(3.856)
Uval5C	522.0263(6.645)(+)	530.2507(12.634)(+)	518.6073(8.35)(=)	513.4305(6.425)(=)	518.4106(7.573)(=)	516.0506(6.337)
Uval5D	747.1994(20.341)(+)	751.3212(20.016)(+)	728.8122(13.066)(=)	725.5403(15.047)(=)	735.3973(10.431)(+)	722.9408(11.808)
Uval6A	229.8105(2.513)(=)	232.7176(4.722)(+)	229.3166(2.951)(=)	228.1973(1.736)(=)	229.7946(2.232)(=)	229.4904(2.935)
Uval6B	260.987(4.221)(+)	268.2549(10.851)(+)	257.3596(4.19)(=)	256.8229(4.516)(=)	258.3724(3.261)(=)	259.3178(6.059)
Uval6C	423.3921(21.584)(=)	431.228(22.874)(+)	405.4078(8.073)(-)	401.385(6.803)(-)	416.9153(14.703)(=)	415.2576(17.743)
Uval7A	291.7678(6.221)(=)	300.7732(7.065)(+)	289.6651(10.374)(-)	285.0704(4.362)(-)	291.7044(5.248)(=)	292.8277(5.272)
Uval7B	299.9158(8.327)(=)	310.1024(9.85)(+)	294.6197(6.177)(-)	291.9621(6.751)(-)	304.7538(7.55)(=)	301.0962(7.967)
Uval7C	417.0744(17.023)(+)	413.6364(10.863)(=)	407.9503(8.14)(=)	406.1523(4.853)(=)	416.4156(12.029)(+)	411.1995(12.162)
Uval8A	398.8698(3.124)(+)	404.9157(5.031)(+)	398.6413(3.961)(=)	400.2917(12.315)(=)	400.0483(3.017)(+)	397.5535(2.686)
Uval8B	435.4858(14.72)(+)	438.6551(8.929)(+)	428.9821(9.303)(=)	424.48(5.044)(-)	429.9672(6.382)(=)	428.3036(4.022)
Uval8C	689.1992(14.948)(+)	690.6813(14.953)(+)	666.4134(18.001)(-)	661.1327(14.932)(-)	677.2028(13.883)(=)	672.4655(10.534)
Uval9A	336.7522(3.242)(=)	343.2224(7.196)(+)	334.4052(2.242)(-)	332.6537(2.86)(-)	337.6159(3.129)(+)	336.4662(2.551)
Uval9B	351.4968(3.71)(=)	356.7118(5.774)(+)	350.7538(3.445)(=)	347.8707(3.993)(=)	350.8984(6.068)(=)	350.211(5.154)
Uval9C	369.8666(5.713)(+)	375.803(4.828)(+)	363.2856(5.335)(=)	362.1799(4.098)(-)	367.4177(4.386)(=)	366.1998(6.142)
Uval9D	490.5418(13.782)(+)	492.4926(10.905)(+)	480.2601(10.236)(=)	474.6573(12.982)(-)	483.0787(8.324)(=)	480.3587(6.0)
Uval10A	444.3772(10.897)(=)	461.3147(22.035)(+)	439.147(4.092)(-)	437.5081(2.54)(-)	444.0889(5.229)(=)	444.2145(6.384)
Uval10B	462.9993(6.095)(=)	471.7224(17.878)(+)	460.3632(5.944)(=)	456.189(4.304)(-)	462.8599(3.279)(=)	461.9855(4.145)
Uval10C	483.9717(5.967)(=)	489.8595(11.404)(+)	480.028(6.693)(=)	476.3406(6.219)(-)	482.8358(6.115)(=)	481.3485(5.005)
Uval10D	622.7074(7.153)(=)	624.3295(7.891)(+)	619.4135(4.145)(=)	618.6131(5.903)(=)	625.817(10.896)(+)	620.4912(8.373)

αMOGP-a performs significantly worse than Tarpeian and LPPP. Tarpeian and LPPP can evolve much smaller routing policies than αMOGP-a. However, this size reduction is compromised by sacrificing effectiveness. Tarpeian and LPPP can only evolve small but ineffective routing policies. In contrast, DT and GPHH-N can evolve very effective routing policies, but the size of the evolved routing policies is much larger than αMOGP-a. The αMOGP-a can outperform DT and GPHH-N on almost all instances. It is worth noting that although αMOGP-a and TSMOGP-a obtain competitive results in terms of size, αMOGP-a outperforms TSMOGP-a in terms of the effectiveness according to Tables 4.20 and 4.21.

Table 4.22: The Experimental Results of (**Size**) for Compared Algorithms on **Ugdb** Dataset.

Instance	Tarpeian	LPPP	DT	GPHH-N	TSMOGP-a	αMOGP-a
Ugdb1	21.7333(13.24)(-)	7.2667(3.095)(-)	84.9333(22.416)(+)	60.6(24.732)(+)	38.8(32.65)(=)	37.0(32.415)
Ugdb2	26.5333(11.407)(=)	9.1333(4.783)(-)	80.8667(29.172)(+)	52.1333(15.037)(+)	32.8(19.975)(=)	35.4667(19.929)
Ugdb3	19.4(8.589)(=)	7.5333(2.097)(-)	68.8(22.562)(+)	43.2667(23.511)(+)	26.8(17.47)(+)	18.2667(10.339)
Ugdb4	22.4(15.078)(=)	7.9333(3.352)(-)	62.0(22.935)(+)	55.0667(32.477)(+)	23.1333(18.827)(=)	24.5333(20.981)
Ugdb5	24.0667(10.837)(-)	8.4(5.096)(-)	77.1333(20.764)(+)	49.6(22.469)(+)	29.8(25.296)(-)	36.6667(26.57)
Ugdb6	12.0667(17.599)(-)	4.1333(5.138)(-)	62.6667(23.265)(+)	40.6667(17.905)(+)	20.2(11.454)(=)	18.8667(12.193)
Ugdb7	4.9333(10.069)(-)	1.0(0.0)(-)	41.2667(19.998)(+)	25.5333(12.616)(+)	10.3333(6.546)(-)	20.3333(21.22)
Ugdb8	31.2(16.949)(=)	8.4667(4.2)(-)	73.7333(29.231)(+)	60.1333(17.322)(+)	42.7333(35.544)(=)	43.7333(33.398)
Ugdb9	29.0(20.477)(=)	8.6667(3.155)(-)	62.8(20.317)(+)	61.4667(15.498)(+)	26.9333(18.274)(=)	26.3333(10.807)
Ugdb10	15.2667(8.737)(-)	5.0667(0.365)(-)	55.6667(21.575)(+)	29.3333(16.66)(=)	16.4667(12.789)(-)	25.5333(19.75)
Ugdb11	18.4(19.098)(=)	1.8667(2.609)(-)	55.4667(22.759)(+)	41.2(11.854)(+)	16.9333(15.461)(=)	17.4667(12.292)
Ugdb12	26.4(11.77)(=)	14.0667(4.571)(-)	80.2(26.343)(+)	53.0(16.526)(+)	36.0(21.047)(=)	28.6667(11.698)
Ugdb13	23.2667(10.657)(=)	7.2667(2.333)(-)	62.8(20.774)(+)	54.0(24.148)(+)	26.4(16.569)(=)	26.6667(15.296)
Ugdb14	23.5333(9.38)(=)	1.4(1.221)(-)	70.7333(21.053)(+)	46.9333(23.368)(+)	30.0(14.218)(=)	23.6667(8.949)
Ugdb15	25.8667(15.71)(=)	1.0(0.0)(-)	89.8667(37.522)(+)	32.8(25.877)(=)	18.2(9.974)(-)	33.2(24.57)
Ugdb16	2.5333(1.634)(+)	1.0(0.0)(=)	19.4667(18.35)(+)	9.0667(15.478)(+)	3.3333(7.485)(=)	1.8(3.221)
Ugdb17	4.0667(1.363)(-)	1.0(0.0)(-)	94.6667(49.248)(+)	6.8(13.166)(-)	5.2(9.253)(-)	21.3333(21.381)
Ugdb18	32.4667(16.358)(=)	5.2(2.987)(-)	83.5333(37.12)(+)	57.4667(31.191)(+)	36.8667(22.966)(=)	29.8667(15.22)
Ugdb19	16.1333(5.625)(+)	1.2(0.61)(-)	81.2(32.464)(+)	16.2667(14.888)(=)	15.5333(12.897)(=)	13.7333(13.781)
Ugdb20	25.1333(16.559)(=)	1.0(0.0)(-)	79.1333(41.556)(+)	43.4(22.963)(+)	24.6667(13.986)(=)	21.2667(9.318)
Ugdb21	25.3333(13.586)(=)	3.2(2.797)(-)	76.2667(33.307)(+)	58.4667(27.827)(+)	28.6(13.87)(=)	30.7333(14.713)
Ugdb22	21.9333(10.356)(-)	1.5333(1.383)(-)	65.9333(25.438)(+)	53.2(23.835)(+)	32.6(20.807)(=)	35.3333(22.561)
Ugdb23	28.4(12.286)(=)	6.1333(1.634)(-)	78.0(23.895)(+)	76.5333(34.894)(+)	38.2(18.108)(=)	29.9333(11.552)

To sum up, from the perspective of both benchmark datasets and real-world dataset, αMOGP-a can evolve better Pareto front than TSMOGP-a. For the benchmark datasets, αMOGP-a evolves much smaller routing policies by slightly losing effectiveness comparing with DT and GPHH-N. Tarpeian and LPPP can evolve much smaller routing policies than αMOGP-a, but their effectiveness is much worse. For the real-world instance, αMOGP-a evolves much smaller routing policies without losing effectiveness comparing with DT and GPHH-N. It evolves significantly better routing policies in terms of both effectiveness of size than Tarpeian and LPPP. Thus, all the results indicate that αMOGP-a can evolve compact and effective routing policies.

Table 4.23: The Experimental Results of (**Size**) for Compared Algorithms on **Uval** Dataset.

Instance	Tarpeian	LPPP	DT	GPHH-N	TSMOGP-a	αMOGP-a
Uval1A	15.6(11.193)(-)	3.8667(2.713)(-)	73.1333(25.249)(+)	30.5333(20.238)(=)	22.6(14.464)(=)	23.6(14.764)
Uval1B	19.9333(10.113)(=)	6.6(2.43)(-)	61.1333(22.505)(+)	26.4667(16.358)(=)	19.8(13.01)(=)	22.8667(15.429)
Uval1C	25.1333(11.708)(=)	9.6(4.207)(-)	75.7333(34.519)(+)	64.9333(20.854)(+)	33.1333(28.657)(=)	31.2667(16.789)
Uval2A	25.2667(12.426)(+)	7.1333(1.655)(-)	72.3333(29.405)(+)	26.5333(15.471)(+)	22.7333(16.161)(=)	17.8667(9.138)
Uval2B	24.6667(13.184)(=)	7.4667(4.289)(-)	74.2667(31.573)(+)	48.2667(27.884)(+)	33.4(16.917)(=)	28.2667(17.265)
Uval2C	26.7333(11.741)(=)	13.8(5.573)(-)	67.1333(24.166)(+)	71.9333(22.245)(+)	41.6(30.477)(=)	33.2(16.055)
Uval3A	28.8667(13.024)(=)	5.6(1.192)(-)	73.1333(28.796)(+)	36.0(15.972)(+)	26.7333(14.881)(=)	26.4(10.906)
Uval3B	14.6667(18.263)(-)	3.0(0.743)(-)	67.5333(24.624)(+)	50.4667(19.355)(+)	23.1333(10.345)(=)	21.2(9.618)
Uval3C	26.2(12.041)(=)	7.9333(3.14)(-)	76.8(37.3)(+)	65.6667(21.867)(+)	30.8(17.697)(=)	32.0(15.148)
Uval4A	25.2(9.357)(=)	8.7333(2.664)(-)	66.4(32.594)(+)	47.3333(19.526)(+)	31.2(14.644)(=)	27.8667(14.352)
Uval4B	28.2(12.743)(=)	12.4667(5.894)(-)	66.2667(23.297)(+)	47.4667(16.698)(+)	30.5333(15.247)(=)	25.0(10.544)
Uval4C	28.2(17.893)(=)	13.4667(3.665)(-)	73.5333(31.015)(+)	56.8(28.218)(+)	35.4667(20.623)(=)	30.9333(12.86)
Uval4D	28.0667(14.477)(-)	18.6667(10.131)(-)	67.6667(18.867)(+)	68.2667(23.893)(+)	29.1333(13.846)(=)	37.0(15.438)
Uval5A	26.8667(17.834)(=)	6.2667(1.437)(-)	59.6667(15.856)(+)	44.9333(16.613)(+)	28.3333(15.654)(=)	24.4(12.353)
Uval5B	30.1333(16.54)(=)	10.8667(3.235)(-)	75.0667(33.336)(+)	46.8667(19.333)(+)	30.5333(19.839)(=)	30.7333(14.732)
Uval5C	19.5333(7.947)(-)	9.3333(4.816)(-)	58.6667(20.186)(+)	41.4667(13.418)(+)	26.2667(18.175)(=)	25.0667(11.441)
Uval5D	20.9333(12.323)(-)	13.2(6.288)(-)	64.6667(22.61)(+)	63.6(19.842)(+)	31.4667(13.602)(=)	27.2(13.435)
Uval6A	23.0(12.171)(=)	9.5333(3.441)(-)	61.5333(29.389)(+)	39.0(21.953)(+)	36.6667(18.952)(+)	25.2(11.559)
Uval6B	23.6(11.205)(=)	9.8(3.773)(-)	68.8(21.88)(+)	48.6(18.977)(+)	34.2(35.929)(=)	30.2(17.377)
Uval6C	29.4(15.725)(=)	9.6667(5.809)(-)	70.0(20.408)(+)	62.6(16.336)(+)	28.4(11.425)(=)	28.5333(14.294)
Uval7A	27.3333(14.726)(=)	9.5333(2.569)(-)	65.0667(27.505)(+)	36.1333(15.046)(+)	27.3333(16.138)(=)	25.7333(10.392)
Uval7B	29.2(13.291)(=)	10.1333(3.739)(-)	81.6(31.488)(+)	38.0(18.411)(+)	33.1333(17.035)(=)	27.9333(11.552)
Uval7C	26.8667(11.59)(-)	14.0667(4.571)(-)	70.7333(25.006)(+)	58.4(19.849)(+)	42.6667(29.758)(=)	38.4(23.764)
Uval8A	24.9333(11.283)(=)	7.4(2.253)(-)	63.4667(16.739)(+)	39.0667(19.415)(+)	24.2(13.783)(=)	27.6667(14.126)
Uval8B	22.4(15.714)(=)	6.5333(2.662)(-)	57.6667(18.257)(+)	41.2667(21.027)(+)	30.5333(17.963)(=)	24.1333(10.887)
Uval8C	27.3333(13.392)(-)	15.0(5.872)(-)	62.6667(22.792)(+)	59.4667(23.286)(+)	36.0(16.54)(=)	36.6(20.358)
Uval9A	21.8667(8.593)(=)	8.1333(2.956)(-)	64.0(27.361)(+)	46.0(21.112)(+)	26.4667(14.659)(=)	21.4667(11.646)
Uval9B	26.0(11.951)(=)	8.0(2.716)(-)	62.4(22.653)(+)	49.4(18.491)(+)	32.8(14.339)(+)	29.0(19.476)
Uval9C	22.4667(10.814)(-)	8.0667(3.778)(-)	75.6(38.129)(+)	53.6667(17.736)(+)	28.4(16.147)(=)	31.7333(13.98)
Uval9D	31.4(11.196)(=)	16.2(4.055)(-)	66.4(20.077)(+)	59.8667(16.1)(+)	47.9333(38.445)(=)	38.0667(15.911)
Uval10A	24.9333(12.935)(=)	5.6(3.529)(-)	62.4(20.545)(+)	31.4(19.035)(+)	23.9333(12.722)(=)	22.2(13.01)
Uval10B	23.2667(12.089)(=)	8.4(3.682)(-)	61.4667(18.463)(+)	50.6(16.141)(+)	21.6667(10.984)(=)	24.0(18.471)
Uval10C	25.4(15.035)(=)	8.3333(3.032)(-)	62.6(16.587)(+)	52.0(13.884)(+)	23.9333(10.329)(=)	26.8(14.681)
Uval10D	27.9333(12.202)(-)	16.6667(5.122)(-)	77.9333(33.362)(+)	66.8(18.029)(+)	36.1333(20.928)(=)	34.9333(12.12)

4.5.7 Further Analysis

4.5.7.1 Component Analysis

To verify the effectiveness of each component of the newly proposed αMOGP-a, we designed a set of control experiments on some representative instances. We compare αMOGP-a which utilises the new α adaptation scheme with an existing α adaptation scheme proposed in [198]. The existing α adaptation scheme utilises both the effectiveness and size to identify bias and adjust α value by simply +/- 0.1 each time. We also compare αMOGP-

Table 4.24: The **HV** of the compared algorithms in the control experiment of α adaptation scheme and archive.

Instance	αMOGP-a	αMOGP-a-+/- 0.1	αMOGP
Ugdb1	0.94(0.017)	0.93(0.014)(+)	0.88(0.024)(+)
Ugdb2	0.95(0.015)	0.92(0.018)(+)	0.88(0.029)(+)
Ugdb8	0.93(0.012)	0.92(0.012)(+)	0.84(0.071)(+)
Ugdb23	0.92(0.012)	0.89(0.020)(+)	0.84(0.052)(+)
Uval9A	0.97(0.005)	0.96(0.007)(+)	0.92(0.025)(+)
Uval9D	0.92(0.008)	0.91(0.011)(+)	0.84(0.043)(+)
Uval10A	0.97(0.012)	0.96(0.009)(+)	0.93(0.021)(+)
Uval10D	0.97(0.010)	0.96(0.008)(+)	0.90(0.046)(+)

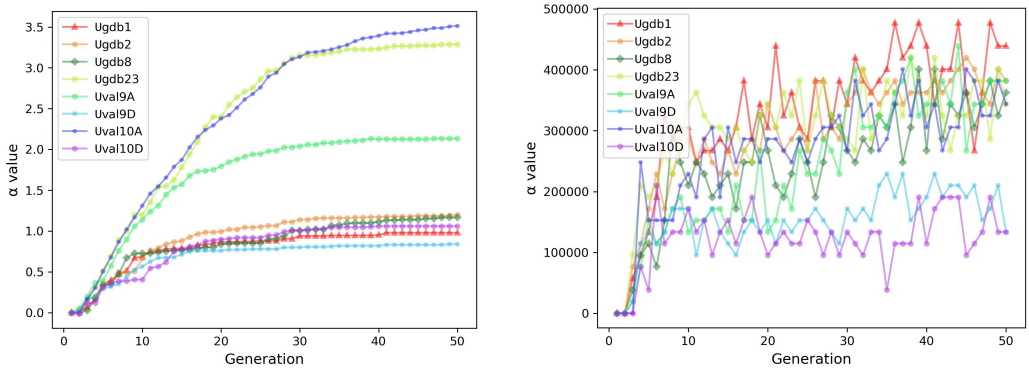
a with the counterpart without using the archive (αMOGP). The experimental results are shown in Table 4.24. As IGD has the same pattern as the HV, we omit the IGD results here due to the page limit. Each "+", "-" or "=" next to the compared algorithms indicates that αMOGP-a performs statistically significantly better than, worse than, or comparable to the compared algorithm.

From Table 4.24, we can see that αMOGP-a significantly outperforms the one using the existing α adaptation scheme. This indicates the effectiveness of the new α adaptation scheme. In addition, we can see that without using the archive, the performance of αMOGP-a significantly deteriorates. This demonstrates the effectiveness of using the archive.

4.5.7.2 α Adaptation

To further analyse how the new α adaptation scheme outperforms the existing one, we plot the convergence curve of α value on 8 representative instances on 30 independent runs. The convergence curve is shown in Fig. 4.13. Each dot in the figure is the mean α value over 30 independent runs at the generation.

From Fig. 4.13a, it can be seen that with the existing α adaptation scheme,



(a) Old α adaptation scheme

(b) New α adaptation scheme

Figure 4.13: The Convergence Curve of α value over old adaptation scheme and new adaptation scheme on Representative Instances.

α value keeps increasing until the end. This shows that 0.1 is not a proper step size for the α value. In contrast, the convergence curve in Fig. 4.13b grows rapidly at first, and as generation increases, the curves stabilise and eventually converge. We can see some fluctuations above the convergence curve. This is expected as when α is too large we need to decrease it, while α is too small we need to increase it. In the process of finding the proper α value, the α value will fluctuate around the proper value, and eventually converge to the proper α value. This indicates that the new α adaptation scheme has better capability to find the proper α value. At the same time, Fig. 4.13b also confirms our hypothesis that different α values are required for different instances.

4.5.7.3 Analysis of Evolved Routing Policies

We extract an effective routing policy evolved by αMOGP-a on Ugdb6 instance to make some further understanding of the behavior of the GP evolved routing policy. To interpret the routing policy evolved by αMOGP-a, we rewrite it by the following equations,

$$RP = S_1 + S_2 + S_3, \tag{4.20}$$

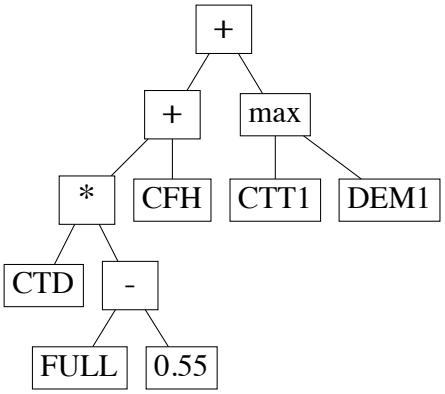


Figure 4.14: A Routing Policies Evolved by αMOGP-a.

where

$$S_1 = \mathsf{CFH}, \tag{4.21}$$

$$S_2 = (\mathsf{FULL} - 0.55) * \mathsf{CTD}, \tag{4.22}$$

$$S_3 = \max(\mathsf{CTT1}, \mathsf{DEM1}). \tag{4.23}$$

We can interpret the above routing policy as follows.

- The policy always prefers tasks close to the current location. This is consistent with the well-known nearest neighbour heuristic.
- If $\mathsf{FULL} \geq 0.55$, i.e., the vehicle is more than half full, the coefficient of CTD in S_2 is non-negative. The policy prefers tasks close to the depot. The focus on CTD increases as the vehicle fills up. This is consistent with our intuition that as the vehicle becomes more full, it is more likely to return to the depot to refill. If the route failure occurs on a task near the depot, the recourse cost is minimised. If $\mathsf{FULL} < 0.55$, the route failure is unlikely to occur. The policy prefers tasks that are distant from the depot, to leave the tasks closer to the depot to the states where the route failure is more likely to occur (the vehicle is more full).

- If $\mathsf{CTT1} \geq \mathsf{DEM1}$, $S_3 = \mathsf{CTT1}$. In other words, if the cost from the candidate task to its nearest task is greater than the demand of the candidate task's nearest task, the policy tends to select tasks that are more isolated to the other remaining tasks. On the other hand, if $\mathsf{CTT1} < \mathsf{DEM1}$, then $S_3 = \mathsf{DEM1}$. In other words, if the candidate task is close to some other remaining tasks (i.e., the cost is smaller than the demand of the other remaining tasks), then the policy prefers tasks whose nearby remaining tasks have smaller demands. This way, after serving the candidate task, the vehicle can easily serve its nearby remaining task with low cost and probability of route failure (due to the small demand of the nearby task).

By observing the Pareto front found by αMOGP-a, we have some other interesting observations. Eq. (4.24) and Eq. (4.25) shows two non-dominated routing policies from same Pareto front of a single run on Ugdb6.

$$RP_1 = \mathsf{CFH} + \mathsf{CTT1} * \mathsf{FULL} + \frac{\mathsf{RQ}}{\mathsf{CTD}}, \tag{4.24}$$

$$RP_2 = \mathsf{CFH} + \mathsf{CTT1} * \mathsf{FULL}. \tag{4.25}$$

The mean total cost of RP_1 is 351.02, and the mean total cost of RP_2 is 362.75. It can be seen that the only difference between RP_1 and RP_2 is the term $\frac{\mathsf{RQ}}{\mathsf{CTD}}$, but their effectiveness is much different. This indicates that $\frac{\mathsf{RQ}}{\mathsf{CTD}}$ is a useful building block. We analyse its behavior. RQ refers to the remaining capacity of the vehicle and CTD refers to the cost from the candidate task to the depot. We can transform $\frac{\mathsf{RQ}}{\mathsf{CTD}}$ to $\mathsf{RQ} * \frac{1}{\mathsf{CTD}}$. At each decision point, RQ is a constant for all the candidate tasks. Then, CTD becomes the only variable. Obviously, $\frac{1}{\mathsf{CTD}}$ decreases as CTD increases. Thus, $\frac{\mathsf{RQ}}{\mathsf{CTD}}$ tends to select tasks that are distant from the depot. Then, we analyse the impact of RQ. At the beginning of a vehicle route, its remaining capacity of the vehicle is the largest (equal to its capacity). As the service continues, RQ becomes smaller and smaller. This indicates that the impact of $\frac{\mathsf{RQ}}{\mathsf{CTD}}$ decreases when the vehicle serves more tasks and becomes more full. This

is similar to the observation in Eq. (4.22), except that RP_1 does not prefer tasks near the depot when the vehicle is more full. As a result, RP_1 still performs worse than RP shown in Eq. (4.20).

4.5.8 Summary

In summary, αMOGP-a applied two strategies, i.e., α dominance strategy and archive strategy, to handle objective selection bias issue and stochastic fitness evaluation issue simultaneously. The experimental results show that αMOGP-a evolves relatively complete Pareto front than previous MOGP algorithms for UCARP. In addition, it evolves compact and effective routing policies that is more interpretable to end users.

4.6 Chapter Summary

The main goal of this chapter is to evolve Pareto front that contains routing policies with degree of interpretability and effectiveness so that users can choose based on their preference. This goal is achieved by proposing a set of new MOGP algorithms. There are two main issues, i.e., objective selection bias issue and stochastic fitness evaluation issue, when designing MOGP algorithms for UCARP. Firstly, we investigate the utilisation of α dominance and three different α adaptation schemes to handle the objective selection bias issue. The results show that the objective selection bias issue can be partially handled by α dominance. By observing the results, we found that it is necessary self-adaptive α adaptation scheme so that α dominance can find proper α value to handle the objective selection bias issue. In this case, we design a new self-adaptive α adaptation scheme which can automatically adjust the α value based on the bias status. The results show that the new adaptation scheme outperforms manually designed schemes. Then, we propose a new archive strategy to handle the stochastic evaluation issue. The results show that

the archive can effectively maintain the potentially good individuals during the evolutionary process and improve the distribution of the Pareto front. Finally, the αMOGP-a is proposed to handle both issues simultaneously. The experimental results show that αMOGP-a outperforms the state-of-the-art algorithms for UCARP. The results showed that both the α adaptation scheme and the archive are effective. The HV and IGD become worse when either component is removed. We further analysed on several evolved routing policies reveals that CFH is a very important feature, in addition, the importance of the CTD and CTT1 are somewhat related to how fully loaded the vehicle is. Because the new αMOGP-a can provide a set of non-dominated solutions, we can gain some domain knowledge which building blocks are more useful for routing policy by observing these solutions in the same set of non-dominated solutions. Overall, αMOGP-a can obtain better Pareto front and potentially more interpretable routing policies than the state-of-the-art algorithms.

Chapter 5

Local and Global explanation methods of Genetic Programming-Evolved Routing Policies for Uncertain Capacitated Arc Routing Problem

5.1 Introduction

The methods to improve the interpretability of AI models can be normally divided into two categories [147], i.e., intrinsic methods and post-hoc methods. Intrinsic methods are mainly used *during* the training process of the model. They design specific learning/search mechanisms to learn both effective and interpretable models [35]. The post-hoc methods, on the other hand, are mainly used *after* the model training process to "explain" the trained models. Post-hoc methods can be applied to not only "black-box" models, but also transparent models (the model itself is easy to interpret), since post-hoc methods are usually decoupled from the main

model [35].

Previous chapters are mainly about intrinsic interpretability. These methods generally improve interpretability by evolving smaller (simpler) routing policies. However, even with the effort to evolve simpler routing policies, the final routing policy may still be difficult to understand in some cases, as the GP-evolved routing policies need to maintain a certain level of complexity in order to be effective [25, 95]. Once we have evolved an effective GP-evolved routing policy for a UCARP instance, it becomes difficult to improve its interpretability without sacrificing its effectiveness. Contrary to the intrinsic methods, the post-hoc methods can potentially improve interpretability without sacrificing effectiveness, since they do not change the trained model. Therefore, in this work we focus on developing post-hoc methods to explain GP-evolved UCARP routing policies.

Under a UCARP decision situation, e.g., when a vehicle completes the current task and becomes idle, it uses a routing policy to prioritise/rank the candidate tasks (those that can be served by the vehicle without violating the capacity constraint) and selects the top-ranked task to serve next. Thus, the behaviour of a routing policy can be characterised from two aspects: (1) its task ranking and (2) the selected task. To explain the behaviour of a routing policy in a decision situation, we need to explain why the routing policy ranks the candidate tasks in a particular way, and why it selects a certain task over others in this decision situation.

There are a number of different post-hoc explanation methods, such as text explanations, visualization or explanations by example [120]. Local explanation is one of explanations by example methods, it is suitable for UCARP as we can give explanation of the decision made by GP-evolved routing policies in each decision point. The local explanation aims to explain a single decision or prediction as opposed to explaining the entire decision or prediction space by the complex model. In general, it is quite difficult to describe the full mapping learned by a model, but it is much

easier to give explanations to less complex solution subspaces that are relevant for the whole model [120]. In UCARP, there are usually many decision situations, and one of many candidate tasks has to be selected to serve in each decision situation. The GP-evolved routing policies are complex and difficult to interpret as there are so many decisions to be made in so many situations. However, to our best knowledge, there is not any existing local explanation methods for GP-evolved routing policies. Thus, it is necessary to design novel local explanation methods for GP-evolved routing policies.

In addition to explaining the above *local* behaviour of routing policy in each decision situation, it is also important to explain its *global* behaviour in all the different decision situations occurred during the entire service process. However, the global explanation is much more difficult to obtain than the local explanation for each single decision situation [120]. A potential way to build a global explanation is to find patterns in all local explanations. Therefore, it is necessary to design novel global explanation method by grouping local explanations.

5.1.1 Chapter Goals

The overall goal of this paper is to develop new post-hoc methods to explain the local and global behaviours of GP-evolved rules for stochastic optimisation and machine learning problems, using UCARP as an example. We have the following specific research objectives:

1. To propose two metrics, namely *consistency* and *correlation*, to characterise the local behaviour of routing policy in each decision situation.

2. To investigate the feasibility of using a linear model to construct a local explanation.

3. To use PSO to evolve linear models for the local explanation, and

define a new fitness function that incorporates the consistency, correlation and number of used attributes in the linear model.

4. To develop clustering mechanisms to summarise the local explanations of all the decision situations occurred in the UCARP instance into a global explanation, in the form of a production rule, for example, "If the decision situation shows pattern X, then the local explanation shows pattern Y".

5. To verify the accuracy of the proposed LGRE method, in terms of consistency and correlation in different decision situations.

6. To show the interpretability of the LGRE method by giving case studies of the local and global explanations.

5.1.2 Chapter Organisation

The rest of this paper is organised as follows. Section 5.2 describes the investigation of the feasibility of using a linear model to construct a local explanation. Section 5.3 describes PSO-based Local and Global explanation method. Finally, Section 5.4 gives the summary of the chapter.

5.2 L1 Regression-Based Local Explanation Method

5.2.1 Overall Framework

This section describes the Local Ranking Explanation (LRE), a post-hoc intractability method based on local explanations used to help explain the decision made by GP-evolved routing policies. Note that, this section introduces the LRE under the context of the GPHH for UCARP, although it can be generalised to other problems, such as dynamic flexible job shop scheduling problem and dynamic cloud workflow scheduling problem. The LRE is illustrated in Algorithm 11. Fig. 5.1 shows

an example of LRE. In our research, we extend the application of L1 regression to address the unique challenges posed by the interpretability of GP-evolved routing policies in the context of UCARP. Our focus is on improving the interpretability of these policies within the realm of logistics and transportation, where they have practical implications for real-world decision-making. The Evolutionary Feature Synthesis (EFS) method [12] is a fast and efficient regression approach for creating interpretable nonlinear models from small to medium-sized datasets. EFS uses evolutionary computation to search for features and select subsets, making it one of the quickest tools in this domain. While our research and EFS both employ evolutionary algorithms, they address different problems. EFS focuses on feature synthesis and regression, whereas our work concentrates on improving the interpretability of routing policies for logistics, specifically the UCARP.

A routing policies is typically a priority function. At a specific decision situation that a set of candidate tasks are waiting to be served, a routing policy is applied to calculate priority values for each candidate task, and then the most prioritised task is selected to be served next.

Give a GP-evolved routing policy RP to be explained and a set of decision situations D, each decision situation $d \in D$ contains a set of candidate tasks T that need to be served. There is a set of attributes A in RP. Each $t \in T$ can be represented as a vector of the attribute values of the task. Then, we can use the set of attributes to calculate the corresponding priority value p of each candidate task in a specific decision situation. To explain the specific decision, we will collect all the attribute values of each candidate task in the decision situation as input X, and the corresponding priority values of each candidate task as output Y. These data will utilised to generate a linear regression.

Some pre-processing of the data is also necessary before training the linear regression. First, we will remove some useless attributes from the input data. In the specified decision situation, some attributes have the

Algorithm 11: Local Ranking Explanation (LRE).

```
Input: GP-evolved Routing Policy RP, A decision situation d
Output: An explanation for RP in d
Identify all attributes A used in RP;
Get all the candidate tasks T in d;
// feature value set
X is empty;
// prediction value set
Y is empty;
for t ∈ T do
    x = attributes in t;
    // calculate priority value of task using routing policy
    y = RP(t);
    add x to X;
    add y to Y;
end
Remove redundant attributes from X to get X';
Convert Y to rank value Y';
// When two tasks have same rank then remove the task with
   larger index
Remove tie from X and Y';
LinearModel(LM) = Run linear regression (X', Y');
for a ∈ A do
    Analyse the coefficient of a;
    Calculate importance of a using Eq. 5.3 ;
end
return model and attributes importance as an explanation for RP in d;
```

same value on all candidate tasks and these attributes are redundant for training linear regression. For example, FULL represents the fullness of a vehicle, and when we are deciding which task to service for a vehicle, all the candidate tasks have the same FULL values. Second, for a decision situation, we will rank the priority values of all the candidate tasks in this decision situation and then use these ranked values as the new output Y'. This is because the priority values usually do not follow the linear regression. Converting the priority values into ranked values will simplify the problem. For example, if we have three candidates with priority values 1, 1.1 and 1000, it is difficult to describe the behaviors of the routing policy, whereas if we transform them into ranked values 1, 2 and 3, it can better reflect the behaviors of the routing policy. Also, it will not change the behaviour of the routing policy in this way. The most prioritised candidate task will be ranked with value 1. Third, we will remove all the ties in each decision situation. A tie means that two candidate tasks have the same priority values. When a tie occurs, the routing policy will simply choose the task with the smallest index value. Thus, in the input data, if there is a tie in a decision situation, we can remove the tie by simply removing the task with larger index.

After the data preprocessing, we can use the input X and output Y to generate a linear model LM to represent the RP at a decision situation. The representation of LM is shown in Eq. 5.1. Because the intercept does not affect the ranking process of the linear model, the intercept is omitted in the linear mode. The loss function for LM is shown in Eq. 5.2. The loss function contains the least square error and a regularisation in it. This is to balance the regression error and model complexity (i.e., number of attributes used). When λ equals to zero, the loss function becomes the linear regression with least square error. Then, we will analyse the coefficients C in the LM. Attribute importance will be calculated based on the attribute value $a \in A$ and its corresponding coefficient $c \in C$. The attribute importance is defined as the mean squared error value between the LM and LM'

that sets the coefficient of the attribute to zero in LM in Alg. 11. This measurement of importance shows the extent to which an attribute affects the current decision when it changes. The measurement is shown in Eq. 5.3, where t refers to task and a refers to the attribute. Note that, we can also observe the coefficients found by LM directly to get some understanding. This LM is able to provide a local interpretation of the decisions made by RP in a decision situation. As a linear equation, the interpretability of LM is more straightforward than the interpretability of RP. There are of course exceptions, especially when the number of nodes in RP is small. This is the case where the method is used in situations where the original RP is difficult to read and understand by the end user.

$$LM = C \cdot A = c_1 * a_1 + c_2 * a_2 + ... + c_n * a_n \tag{5.1}$$

$$Loss = \sum_{t \in T} (LM(t) - RP(t))^2 + \sum_{c \in C} \lambda |c| \tag{5.2}$$

$$Importance(a) = \frac{1}{|T|} \sqrt{\sum_{t \in T} (LM(t) - LM'(t))^2} \tag{5.3}$$

5.2.2 Experimental Study

This section presents an experimental analysis of the interpretability of the proposed method. Firstly, we will perform an analysis of the performance of different method. The performance is evaluated based on how well a linear regression can fit the routing policy at a decision situation. We use the following three metrics: the mean error between the output of the linear regression and the actual ranked value on each candidate task over all decision situations, the correlation between the trend of the output of the linear regression and the trend of the actual ranked value, and the consistency between the prioritised task of the output of the linear regression and the actual prioritised task. Then, we analyse the coefficients in linear regressions to give a local explanation for the routing policy.

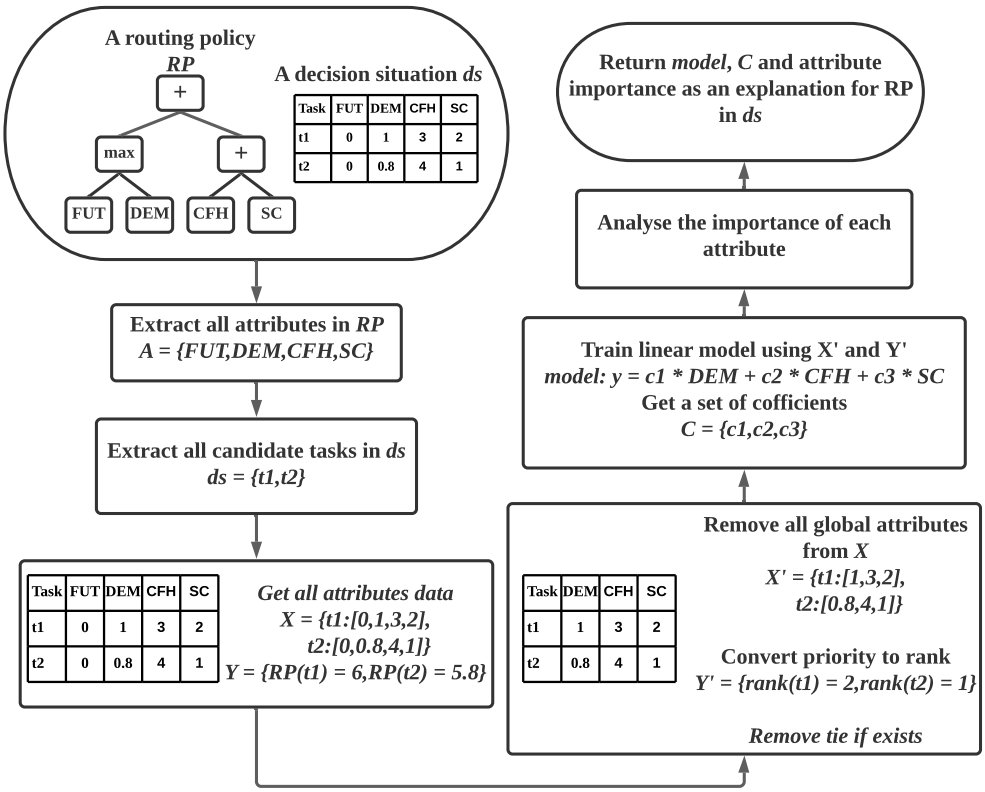


Figure 5.1: An example of LRE

We select 6 representative UCARP instances from the commonly used Ugdb and Uval datasets [122,143] for experiments which have been introduced in Table 2.1. To make the problem simpler, we use only one vehicle in the experiment. We use 30 full simulations for each GP-evolved routing policy for each instance.

For GP, the routing policies are evolved using the Evolutionary Computation Java (ECJ) package [131]. The number of generations is 100. The population size is 1000. The tournament selection size is 7. The maximal depth is 8. The crossover and mutation rates are 0.85 and 0.15.

For linear regression, we use Scikit-learn python package [156] to implement the linear regression. The linear regression is running with no

regularization (LR-0), a linear regression with λ = 0.5 (LR-0.5) and a linear regression with λ = 1 (LR-1). These models were chosen as they are considered interpretable regression methods, since the coefficients give us a notion of attribute importance. Although multiple λ have tested, but the primary goal is not to find the optimal λ but to observe patterns as λ increases, focusing on these three values—no regularization, moderate regularization, and strong regularization—can indeed provide valuable insights. By examining these selected points along the regularization spectrum, we can gain a good understanding of how interpretability changes with varying levels of regularization, which aligns with our research objectives. This approach allows for a more focused investigation and simplifies the experimentation process while still yielding meaningful results.

5.2.3 Results

5.2.3.1 Performance of Linear Regression

We first analyze the how well a linear regression can fit the routing policy at a decision situation. We utilise three three metrics the evaluate the linear model.

- *Error*: the mean error between the output of the linear regression and the ranked value given by the routing policy on each candidate task over all decision situations. The error between the linear model $\boldsymbol{lm}$ and the routing policy $\boldsymbol{rp}$ is defined as

$$\texttt{err} = \frac{1}{|D|}\sum_{d\in D}\frac{1}{|T|}\sum_{t\in T}(\boldsymbol{lm}(t) - \boldsymbol{rp}(t))^2 \tag{5.4}$$

  where $|T|$ refers to the number of candidate tasks in a decision situation, and $|D|$ refers to the number of decision situations.

- *Consistency*: whether the linear model selects the same task (with rank 1) as the routing policy $\varphi(\cdot)$. A higher consistency indicates that

the linear model makes more consistent decisions with the routing policy. The consistency between two rank vectors γ and γ' is defined as

$$\varsigma(\gamma, \gamma') = \begin{cases} 1, & \text{if } \texttt{idx}(\gamma, 1) = \texttt{idx}(\gamma', 1), \\ 0, & \text{otherwise}, \end{cases} \tag{5.5}$$

where γ refers to the rank vector of routing policy and γ' refers to the rank vector of linear model, $\texttt{idx}(\gamma, 1)$ indicates the index of 1 in γ.

- *Correlation*: how close is the rank vector of the linear model to that obtained by the routing policy. A higher correlation indicates that the linear model behaves more closely related to the routing policy for all the candidate tasks in the decision situation. The correlation between two rank vectors γ and γ' is defined as follows:

$$\texttt{corr}(\gamma, \gamma') = \frac{\sum_{i=1}^{N}(\gamma_i - \bar{\gamma})(\gamma'_i - \bar{\gamma'})}{\sqrt{\sum_{i=1}^{N}(\gamma_i - \bar{\gamma})^2}\sqrt{\sum_{i=1}^{N}(\gamma'_i - \bar{\gamma'})^2}}. \tag{5.6}$$

  where $\bar{\gamma}$ and $\bar{\gamma'}$ refer to the average rank of the two rank vectors, respectively.

Table 5.1 shows the mean and standard deviation for three metrics that describe the quality of the linear regression of explaining the routing policy over 30 simulations. It can be seen that LR-0 performs best among three linear regression models in terms of error metric on all instances. LR-0 can always get the smallest mean error. This is as expected as LR-0 does not use any regularisation. We can see that the mean error increases progressively on LR-0, LR-0.5 and LR-1, which is also positively related to the complexity of the linear regression model. For some instances the error values are small, e.g. Ugdb1 and Ugdb2, but for some instances they are large, e.g. Uval10D. This is reasonable because linear regression is inherently difficult to fit perfectly for some complex data. Another reason for the large error is that we have only converted the priority values but

Table 5.1: The mean and standard deviation for three metrics of identifying the performance of the linear regression over 30 simulations.

Instance	Model	Error	Correlation	Consistency
Ugdb1	LR-0	3.11(0.07)	0.978(0.002)	0.81(0.02)
	LR-0.5	4.45(0.08)	0.969(0.004)	0.92(0.03)
	LR-1	4.84(0.1)	0.965(0.004)	0.92(0.03)
Ugdb2	LR-0	4.32(0.09)	0.974(0.001)	0.61(0.01)
	LR-0.5	5.89(0.12)	0.959(0.003)	0.58(0.03)
	LR-1	8.46(0.2)	0.943(0.004)	0.61(0.02)
Ugdb8	LR-0	8.88(1.06)	0.97(0.004)	0.76(0.04)
	LR-0.5	9.38(1.1)	0.954(0.008)	0.75(0.04)
	LR-1	9.67(1.11)	0.936(0.023)	0.72(0.04)
Ugdb23	LR-0	18.83(1.05)	0.825(0.019)	0.78(0.02)
	LR-0.5	19.52(1.07)	0.733(0.036)	0.73(0.04)
	LR-1	20.78(1.06)	0.677(0.038)	0.72(0.03)
Uval10A	LR-0	19.97(0.99)	0.995(0.001)	0.87(0.01)
	LR-0.5	20.12(0.99)	0.996(0.001)	0.85(0.01)
	LR-1	20.4(1.0)	0.995(0.001)	0.85(0.01)
Uval10D	LR-0	78.95(3.0)	0.969(0.003)	0.64(0.04)
	LR-0.5	79.34(3.0)	0.967(0.003)	0.65(0.04)
	LR-1	80.19(3.0)	0.964(0.005)	0.66(0.04)

not the attribute values, which leads to a mismatch between the attribute values and the values being converted. However, for our problem, the correlation of trends and behaviour between linear regression and routing policy are more important to the ranking decisions. From a correlation perspective, the three linear regression models maintain a high positive correlation with the routing policy on almost all instances. This is promising because a high positive correlation means that the linear regression models can be broadly consistent with the routing policy's ranking of candidate tasks. From a consistency perspective, the three linear regression models are guaranteed to make exactly the same decision as the routing policy in at least 55% of decision situations across all instances. Note that, on some instances, such as Ugdb1, LR-1 can achieve full consistency with

the routing policy in 92% of the decision situations.

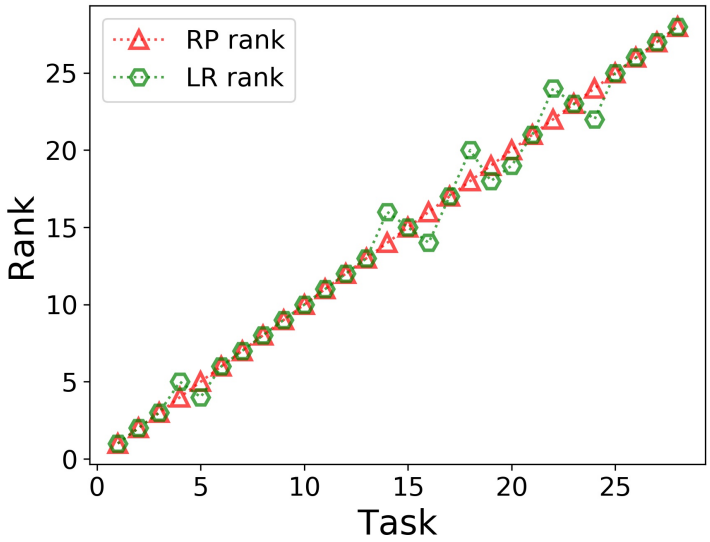


Figure 5.2: LR-0 and the rank value of routing policy on a decision situation in Ugdb1

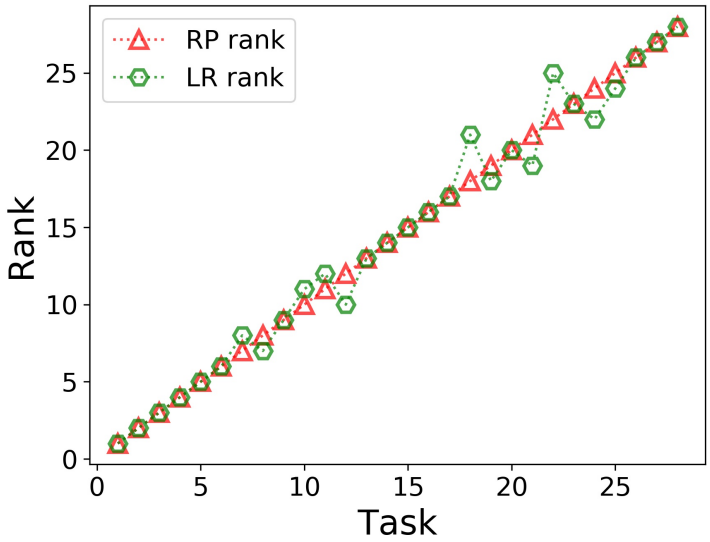


Figure 5.3: LR-0.5 and the rank value of routing policy on a decision situation in Ugdb1

Figs. 5.2, 5.3 and 5.4 show how well the candidate task ranking predicted by the linear regression model fits the routing policy's ranking val-

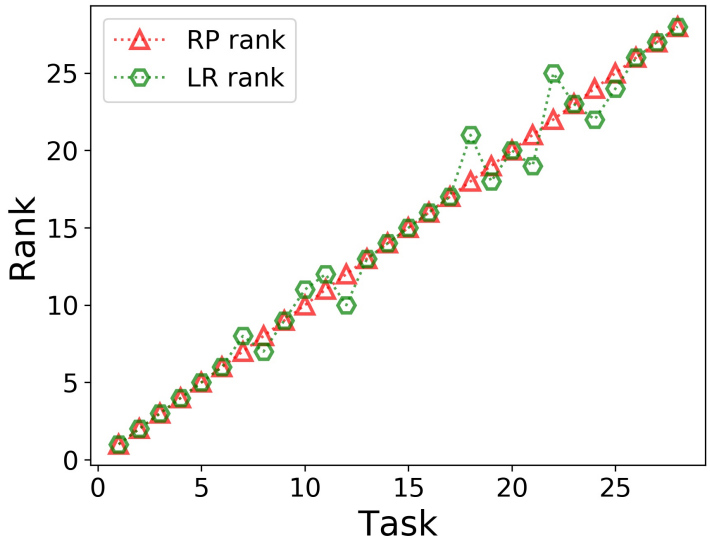


Figure 5.4: LR-1 and the rank value of routing policy on a decision situation in Ugdb1

ues for candidate tasks in a decision situation. Besides, their general trends are consistent with the corresponding routing policy. This answers the research question one that it is feasible to use a linear model to fit a routing policy locally.

5.2.3.2 Complexity of Linear Model

As we are interested in model interpretability, Table 5.2 shows the complexity of the functions obtained by all methods. For the GP-evolved routing policy (RP-GP), the complexity is given by the number of nodes present in the trees. To make a fair comparison, for the linear regression models, we convert each linear models with its non-zero coefficients to a tree format and report the mean number of nodes in it. Besides, we also analyse the number of leaf nodes (i.e., attributes and coefficients) in the routing policy and the linear models.

From Tables 5.2 and 5.3, We can see that the complexity of the RP-GP is the greatest, both in terms of the overall number of nodes and the number of leaf nodes. This is as expected since we set the maximum depth of the

Table 5.2: The mean and standard deviation of the number of nodes for routing policy and the linear regression models over 30 simulations.

Instance	RP-GP	LR-0	LR-0.5	LR-1
Ugdb1	103	16.59(0.0)	9.31(0.14)	8.19(0.09)
Ugdb2	55	13.88(0.0)	11.26(0.21)	9.92(0.32)
Ugdb8	59	10.24(0.18)	7.2(0.3)	6.76(0.21)
Ugdb23	53	7.99(0.16)	6.29(0.14)	5.64(0.06)
Uval10A	71	13.92(0.02)	10.82(0.11)	9.44(0.08)
Uval10D	71	14.0(0.0)	12.26(0.17)	11.21(0.22)

Table 5.3: The mean and standard deviation of the number of leaf nodes for routing policy and the linear regression models over 30 simulations.

Instance	RP-GP	LR-0	LR-0.5	LR-1
Ugdb1	53	11.73(0.0)	6.87(0.09)	6.13(0.06)
Ugdb2	28	9.92(0.0)	8.17(0.14)	7.28(0.21)
Ugdb8	30	7.49(0.12)	5.47(0.2)	5.17(0.14)
Ugdb23	27	5.99(0.11)	4.86(0.09)	4.43(0.04)
Uval10A	36	9.95(0.02)	7.88(0.07)	6.96(0.06)
Uval10D	36	10.0(0.0)	8.84(0.11)	8.14(0.15)

GP tree to 8. Thus, GP is able to evolve relatively complex routing policies. In contrast to RP-GP, all three linear regression models have much lower complexity. In this case, they can have better interpretability. By further observing the results, we can see that the complexity of the linear model is somewhat positively correlated with RP-GP, meaning that when the complexity of RP-GP is high, the linear regression model also becomes more complex. Although the complexity of all three linear regression models is low, we can observe that LR-1 performs the best among three models, it has the fewest coefficients. LR-1 allows for a further increase in interpretability by reducing the number of coefficients used as much as possible. These results can answer the research two that there is a tradeoff between the performance and the complexity of the linear regressions.

The analysis of the above results shows that to provide a local expla-

nation of the routing policy using a linear regression model is possible in most decision situations. Also, by using regularisation in linear model, we can further simplify the linear regression model to make it more user-friendly. Because LR-1 has a lower complexity and, at the same time, maintains good enough correlation and consistency with the routing policy, our attention will be focused mainly on LR-1 in the subsequent content. Fig. 5.5 shows a routing policy for Ugdb1, and Eq. 5.7 is a linear regression by LR-1 (Error is 3.46, Correlation is 0.96 and Consistency with routing policy) for a decision situation in Ugdb1. We can give an initial explanation by observing the coefficients in Eq. 5.7, under the decision situation where the vehicle is empty and all the tasks are not served, the routing policy focuses the most on small cost from the current location, some extent of small serving cost, and slightly consider small cost to depot.

$$y = 1.1493 * CFH + 0.1171 * CTD + 0.4604 * SC \tag{5.7}$$

5.2.4 Further Analysis

5.2.4.1 Coefficients Analysis

In this section, a particular routing policy, i.e. the routing policy in Fig. 5.5, and its corresponding linear models will be analysed. First, we will pick a representative decision situation, i.e. one that has good correlation and consistency with the routing decision. Then, the decision situations that are similar to it ((has similar FULL, RQ, FUT and FRT)) are identified across the 30 simulations. Two decision situations are considered similar when they differ by no more than 0.01 on FULL, FUT and FRT, and no more than 0.1 on RQ. We will observe the distribution of the coefficients of their linear regression models for each attribute in a boxplot format to check whether we can get any insight from it. After that, we will focus more on analysing the linear regression in a single simulation process and each decision situation within it, as opposed to analysing the results of the

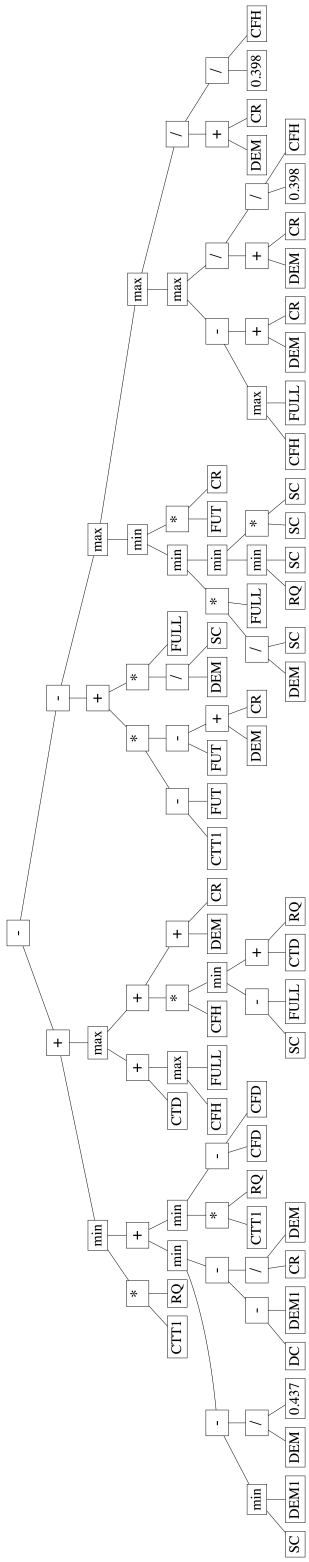
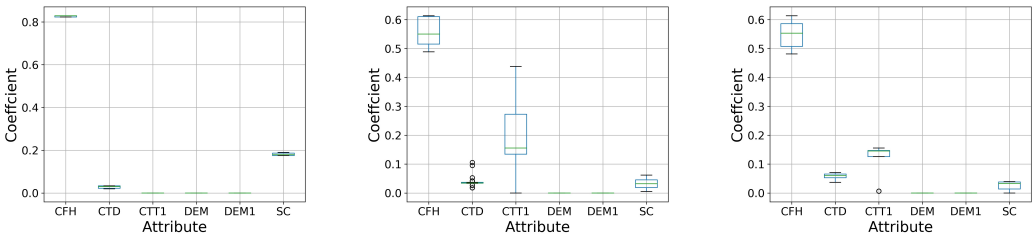


Figure 5.5: A routing policy for Ugdb1.



(a) DS1 (b) DS2 (c) DS3

Figure 5.6: BoxplotS of coefficients of linear regression models with similar decision situations.

thirty simulations as a whole. The attribute importance will be analysed based on a linear model at a representative decision situation. Finally, the linear model and the attribute importance analysis can be used as a local explanation for the routing policy at the decision situation.

Fig. 5.6 shows the Boxplots of coefficients of linear regression models with similar decision situations on three representative decision situations. For DS1 (FULL is 0.18, RQ is 4.06, FUT is 0.77), it represents the situation that the vehicle is almost empty, has lots of remaining capacity and most tasks are unserved. DS2 (FULL is 0.5, RQ is 2.46, FUT is 0.5) refers to the situation that the vehicle is half loaded and half of the tasks have already been serviced. DS3 (FULL is 0.69, RQ is 1.5, FUT is 0.3) represents the situation that the vehicle only has few remaining capacity and most candidate tasks have been served.

In Fig. 5.6a, each box in the figure represents the distribution of the coefficients for each attribute. It can be seen that the coefficients are 0 for attributes CTT1, DEM and DEM1. This means that they are useless in this kind of situations. Combining the above results with the global attributes, i.e. FULL, RQ and FUT, we can make a preliminary interpretation of the routing policy in this decision situation, i.e. with a large amount of spare capacity in the vehicle (FULL is 0.18, RQ is 4.06) and with more than half of the tasks remaining to be performed (FUT is 0.77), the routing policy will

only consider cost from the candidate task to the current location (CFH), cost from the candidate task to the depot(CTD) and the cost of serving the candidate task(SC), and will not consider the cost from the candidate to its closest remaining task (CTT1), the expected demand of the candidate task (DEM) and demand of unserved task that closest to the candidate task (DEM1). We can also observe that CFH, CTD and SC all have very small standard deviations, which suggests that their linear models are similar on these similar decision situations.

From Fig. 5.6b, it can be seen that the distribution of the coefficients is similar to that in Fig. 5.6a, the biggest difference being that the coefficient on CTT1 is no longer zero, meaning that the routing policy takes the cost from the candidate to its closest remaining task into account in such cases. CFH, CTD and SC still have small standard deviations, but CTT1 does not have a small standard deviation, due to the fact that global attributes still do not give a complete picture of a decision situation. The patterns in Fig. 5.6c is consistent with that in Fig. 5.6b, but the standard deviation of CTT1 has become smaller. At the same time, the standard deviations of CFH, CTD and SC are still very small.

By looking at the above results, the research question three can be answered that we can obtain similar linear models for similar decision situations.

5.2.4.2 Importance Analysis

To get some further understanding of these three linear regressions, we carried out an attribute importance analysis for three linear models. The equation to calculate the importance of an attribute has been already discussed in Eq. 5.3. To make it easier to understand, we transform the importance to a ranked value, which means that the most important attribute is ranked as 1. Table 5.4 show the attribute importance in ranked value for all three decision situations.

From Table 5.4, for all three decision situations, CFH is the most im-

Table 5.4: The attribute importance in ranked value for three representative decision situations.

DS	CFH	CTD	CTT1	SC
DS1	1	3	null	2
DS2	1	4	3	2
DS3	1	2	4	3

portant attributes since it always have the rank value 1. In DS1, SC plays a second important role and CTD is the third important attribute in this decision situation. Note that, attributes with coefficient 0 is not listed in the table. CTT1 is labelled $null$ since its coefficient is 0 in DS1. In DS2, SC still plays a second important role. CTT1 replaces CTD to become third most important attribute, and CTD becomes the fourth most important attribute. DS3 differs significantly from the previous two decision situations in that the CTD is no longer the least important attribute, in contrast to which it becomes the second most important attribute, in line with the conclusions of the previous article on explaining the routing policy, which takes more into account the distance to the depot when the vehicle is about to be fully loaded. The above observations answer the research question four that for different decision situations, different attributes play different roles.

Based on the above observations and analysis, we can give a local explanation for each of the three representative decision situations. For DS1, the routing policy only considers the distance between the current location and the candidate task (CFH), the service cost of the candidate task (SC) and the distance from the candidate task to the depot (CTD), on the basis of which CFH is the most important, followed by SC and finally CTD. For DS2, the routing policy considers the distance between the current location and the candidate task (CFH), the service cost of the candidate task (SC), the distance from the candidate task to the depot (CTD) and the cost of the candidate task to its nearest task (CTT1), on the basis of which CFH is

the most important, followed by SC, then CTT1 and finally CTD. For DS3, as with DS2, the routing policy only considers the distance between the current location and the candidate task (CFH), the service cost of the candidate task (SC), the distance from the candidate task to the depot (CTD) and the cost of the candidate task to its nearest task (CTT1). However, CTD becomes second in importance, SC third and finally CTT1.

5.2.5 Summary

In summary, we have investigated the feasibility of using linear model to fit the GP-evolved routing policies in a single decision situation and use the linear model as a local explanation. This goal has been achieved by proposing a Local Ranking Explanation method (LRE). LRE explains the behaviours of a routing policy over a decision situation. The experimental results give the following conclusions: firstly, it is possible to use linear regression to fit a routing policy in a decision situation. Secondly, there is a trade-off between the complexity and interpretability of linear models. Thirdly, we can obtain similar linear models in similar decision situations. Fourth, the importance of the same attribute varies across decision situations. By analysing a number of examples, we find that it is simpler to use LRE to explain the routing policy compared to a direct analysis of routing policies. However, LRE method cannot give good local explanations on all instances as the consistency and correlation cannot be optimised directly in linear regression optimisation.

5.3 Local and Global Explanation Method

In this section, we will introduce a new Local-Global Ranking Explanation (LGRE) method. Previous LRE the local ranking explanation task as a normal regression task (linear model directly predicting the ranks). However, in this work, the model is changed from a normal regression task to

a learning-to-rank task and used PSO to learn the model. Second, LRE is only a local explanation method. In contrast, this work extended it to both local and global explanation methods. The approach has also moved from a single method of providing the local explanation to a two-layer structured explanation framework. The first layer of the framework generates a series of local ranking explanations by any learning-to-rank method such as PSO in this work, and the second layer of the framework employs the local explanations obtained from the first layer to generate the global explanation.

5.3.1 Overall Framework

This section describes Local-Global Ranking Explanation (LGRE), a post-hoc interpretability method to help explain the decision made by routing policies. LGRE contains two layers. The first layer is a PSO-based searching algorithm to find the approximation of the original routing policy in a local decision situation. The second layer uses k-means clustering to cluster the large number of local explanations obtained in the first layer, and extract patterns from clusters to generate a global explanation.

The process of the proposed local ranking explanation of a *routing policy* $\varphi(\mathbf{x})$ under a *decision situation* $\mathbf{X}$ is shown in Algorithm 12, and is illustrated in Fig. 5.7 (colors in the figure are independent to decision situation). Specifically, a routing policy $\varphi(\mathbf{x})$ is a priority function of the attributes $\mathbf{x}$, e.g., the serving cost the candidate task (SC) and demand of the candidate task (DEM). A decision situation $\mathbf{X} = [x_{ij}]_{M \times N}$ is a matrix with M candidate tasks and N attributes, where x_{ij} stands for the attribute j value of candidate task i in this decision situation. In the decision situation, a vehicle aims to select the next task to serve from the M candidate tasks. To explain the local behaviour of the routing policy in this decision situation, the local ranking explanation learns a *linear model* with as few features as possible because linear models are considered easy to con-

struct, widely applied and "highly interpretable" [147,165].

Algorithm 12: Local Ranking Explanation

Input: A routing policy to be explained $\varphi(\mathbf{x})$, a decision situation $\mathbf{X} = [x_{ij}]_{M \times N}$
Output: A linear model to explain $\varphi(\mathbf{x})$ on $\mathbf{X}$
Identify the decision-specific attributes $\mathbf{x}^{\langle local \rangle} \subseteq \mathbf{x}$;
Let the decision-specific attribute columns in $\mathbf{X}$ be $\mathbf{X}^{\langle local \rangle}$;
`// Priority and rank calculation`
for $i = 1 \rightarrow M$ **do**
  Calculate priority $\varphi(\mathbf{x}_i)$ for task $\mathbf{x}_i := \mathbf{X}_{i,:}$;
end
Let $\varphi(\mathbf{X}) := [\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_M)]$;
`// Remove tie`
for $i_1 = 1 \rightarrow M - 1$ **do**
  for $i_2 = i_1 + 1 \rightarrow M$ **do**
    if $\varphi(\mathbf{x}_{i_1}) == \varphi(\mathbf{x}_{i_2})$ **then**
      Remove $\mathbf{x}_{i_2}$ from $\mathbf{X}$;
      Remove $\varphi(\mathbf{x}_{i_2})$ from $\varphi(\mathbf{X})$;
    end
  end
end
`// Convert priority value to rank`
$\boldsymbol{\gamma} = \texttt{sortedIdx}(\varphi(\mathbf{X}))$;
`// Linear model learning`
$\boldsymbol{\beta} = \texttt{PSOLE}(\mathbf{X}^{\langle local \rangle}, \boldsymbol{\gamma})$;
`// Feature selection`
$\boldsymbol{\beta} = \texttt{LEFS}(\boldsymbol{\beta}, \mathbf{X}^{\langle local \rangle})$;
return the linear model $\mathbf{x}^{\langle local \rangle} \cdot \hat{\boldsymbol{\beta}}$;

To explain the local behaviour accurately, the linear model is required to (1) select the same task as the routing policy (consistency), and (2) have the same task ranking as the routing policy (correlation). To this end, the local ranking explanation consists of *decision-specific attribute extraction*, *priority and rank calculation*, *linear model learning*, and *feature selection*, which will be described next.

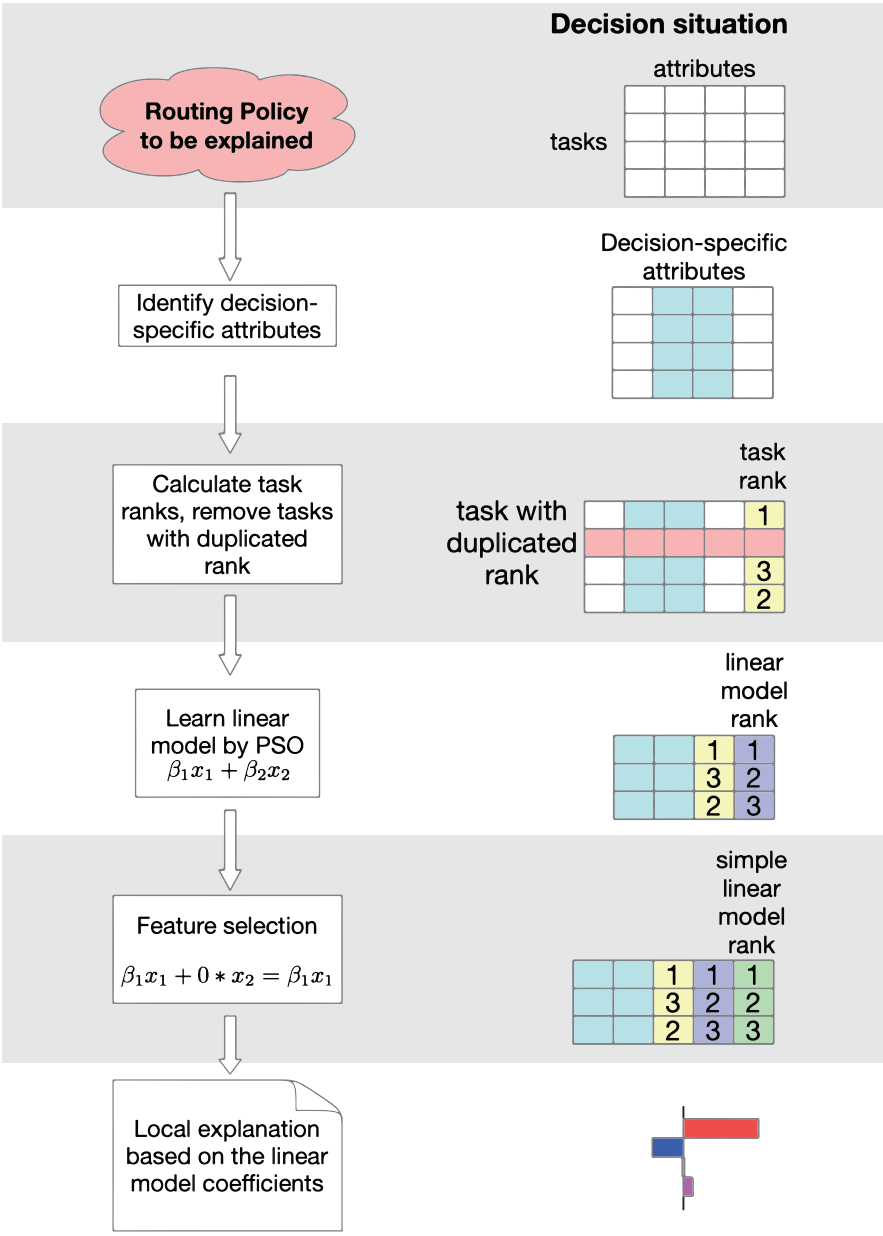


Figure 5.7: The process of the local ranking explanation of a routing policy on a decision situation.

5.3.1.1 Decision-Specific Attribute Extraction

First, the *decision-specific attributes* $\mathbf{x}^{\langle local \rangle} \subseteq \mathbf{x}$ among the attributes appear in the GP tree to be explained. A decision-specific attribute is an attribute that is specific to the decision (i.e., candidate task in this case), such as the demand and location of the candidate task. On the contrary, a *decision-independent attribute* represents the global environment that is independent of the decision (candidate task), such as the remaining capacity and location of the vehicle. The corresponding decision-specific attribute columns in $\mathbf{X}$ are also extracted, and the resultant matrix is denoted as $\mathbf{X}^{\langle local \rangle}$. We have normalised the data and each decision-specific attribute has been normalised to 0 to 1 so that we can analyse the coefficients of the linear models directly.

5.3.1.2 Priority and Rank Calculation

Second, for each candidate task i ($i = 1, \ldots, M$), we calculate its priority value by $\varphi(\mathbf{x}_i)$, where $\mathbf{x}_i := \mathbf{X}_{i,:}$ is the feature vector of the task, i.e., row i of $\mathbf{X}$. Otherwise, use the routing policy to calculate the priority of each candidate task, and select the best-priority task to serve next. Ties are broken by selecting the task with the smallest ID, which is the simplest and most naive deterministic tie breaker. We do not consider random tie breaking that is less interpretable in practice, nor heuristic tie breaking, which may reduce the effect of GP-evolved policies (e.g., GP will learn policies to generate many ties and use the heuristic tie breaker to make decisions). Note that we use this tie breaker as an example (since the routing policies were trained under this tie breaker). However, our approach can theoretically be applied to any policy with any tie breaker, as long as we know how to break ties. We can simply change the task removal process by removing the tasks except for the one selected by the tie breaker. After removing the tasks with the same priority, all the remaining tasks have distinct priority values. Then, we calculate the task ranks $\boldsymbol{\gamma}$ by the `sortedIdx`$(\cdot)$ method.

For example, the task with the minimal priority value has rank 1, and that with the second minimal priority value has rank 2.

5.3.1.3 Linear Model Learning

Third, we learn a linear model $\mathbf{x}^{\langle local \rangle} \cdot \boldsymbol{\beta}$ to fit the task ranks $\boldsymbol{\gamma}$. Different from traditional linear regression, which directly compares between the predicted values and the target values, we compare between the *ranks of the tasks* based on the predicted values given by the linear model and the target ranks. To this end, there are three steps to fit the task ranks:

- Calculate the predicted task priority values $\mathbf{X}^{\langle local \rangle} \cdot \boldsymbol{\beta}$;
- Convert the task priority values to task ranks by $\boldsymbol{\gamma}_{\boldsymbol{\beta}} = \texttt{sortedIdx}(\mathbf{X}^{\langle local \rangle} \cdot \boldsymbol{\beta})$;
- Compare between $\boldsymbol{\gamma}_{\boldsymbol{\beta}}$ and the target task ranks $\boldsymbol{\gamma}$. Ideally, they should be the identical.

Due to the conversion from the raw priority value to the rank, the normal linear regression methods such as least squares and Lasso are no longer applicable. Therefore, we employ Particle Swarm Optimisation (PSO) [158], which is a promising gradient-free continuous optimisation method, to find the best $\boldsymbol{\beta}$ values.

The PSO-based Local Explanation, $\texttt{PSOLE}(\cdot)$, is described in Algorithm 13. It is a standard PSO process that evolves a swarm of particles $\mathbf{B} = [\beta_{ij}]_{N \times P}$, where P is the number of particles, and N is the number of attributes/variables in the linear model. The jth particle is represented by the jth column of $\mathbf{B}$, i.e., $\boldsymbol{\beta}_j = \mathbf{B}_{:,j}$.

For evaluating a linear model $\mathbf{x}^{\langle local \rangle} \cdot \boldsymbol{\beta}$, in addition to the consistency and correlation that are discussed in Section 5.2.3.1, we consider the one more aspect that affect the quality of explanation:

- *Model complexity*: How complex the model is. A less complex linear model tends to be easier to interpret. Here, the model complexity is

measured by the L1 regularisation term $\sum_{i=1}^{N} |\beta_i|$. In case two models predict the same ranks (e.g., f(x) and 2*f(x)), the one with less complexity is considered better.

To take the above aspects into account, the fitness function of PSO is defined as follows:

$$\texttt{fit}(\boldsymbol{\beta}) = 1 - \frac{\varsigma(\boldsymbol{\gamma}_{\boldsymbol{\beta}}, \boldsymbol{\gamma}) + \texttt{corr}(\boldsymbol{\gamma}_{\boldsymbol{\beta}}, \boldsymbol{\gamma})}{2} + \sum_{i=1}^{N} |\beta_i|. \tag{5.8}$$

The goal of PSO is to find the best linear model coefficients $\boldsymbol{\beta}^*$ that minimises the above fitness function. Note that PSO can be replaced by any other learning-to-rank methods [119] that is suitable to optimise the loss function as Eq. (5.8).

Remark. *The linear model only considers the decision-specific attributes. In other words, it does not consider the decision-independent attributes, and has no constant term. First, the task ranks are unchanged by adding any constant. Therefore, we can ignore the constant term in the linear model without changing the ranking. Second, in a decision situation, all the candidate tasks have the same decision-independent attribute values, thus the decision-independent attributes are constants for the task ranking and can be ignored.*

5.3.1.4 Feature Selection

Due to the sensitivity of the performance on the regularisation term $\sum_{i=1}^{N} |\beta_i|$, the $\boldsymbol{\beta}$ value obtained by PSO might still have some small non-zero β coefficients. Intuitively, simply changing a tiny β value to zero can greatly improve the interpretability of the linear model, as we can remove the corresponding feature. However, it is important to ensure that such feature removal does not degrade the accuracy of the model. To address this issue, we develop a local explanation feature selection $\texttt{LEFS}(\cdot)$ method to remove features without decreasing the model accuracy in terms of consistency and correlation.

Algorithm 13: `PSOLE`$(\mathbf{X}, \boldsymbol{\gamma})$

Input: A decision situation $\mathbf{X} = [x_{ij}]_{M \times N}$, a rank vector $\boldsymbol{\gamma}$
Output: The coefficients $\boldsymbol{\beta}$ of linear model $\mathbf{x} \cdot \boldsymbol{\beta}$

```
// Initialisation
```

Randomly initialise a population of weights $\mathbf{B} = [\beta_{ij}]_{N \times P}$ and velocities $\mathbf{V} = [v_{ij}]_{N \times P}$;
while *true* **do**
  Calculate output matrix $\mathbf{Y} = [y_{ij}]_{M \times P} = \mathbf{X} \cdot \mathbf{B}$;
  `// Fitness evaluation`
  for $j = 1 \to P$ **do**
    Get the particle output $\mathbf{y}_j = [y_{1j}, \ldots, y_{Mj}]$;
    $\boldsymbol{\gamma}_j = \texttt{sortedIdx}(\mathbf{y}_j)$;
    Calculate $\varsigma(\boldsymbol{\gamma}_j, \boldsymbol{\gamma})$ by Eq. (5.5);
    Calculate $\texttt{corr}(\boldsymbol{\gamma}_j, \boldsymbol{\gamma})$ by Eq. (5.6);
    Calculate the fitness of $\boldsymbol{\beta}_j := \mathbf{B}_{:,j}$ by Eq. (5.8);
    Update pbest $\boldsymbol{\beta}_j^{\langle pb \rangle}$ and gbest $\boldsymbol{\beta}^{\langle gb \rangle}$;
  end
  if *stop* **then return** $\boldsymbol{\beta}^{\langle gb \rangle}$;
  `// Particle update`
  for $j = 1 \to P$ **do**
    for $i = 1 \to N$ **do**
      $v_{ij} = wv_{ij} + r_1 c_1 \beta_{ij}^{\langle pb \rangle} + r_2 c_2 \beta_i^{\langle gb \rangle}$;
      $\beta_{ij} = \beta_{ij} + v_{ij}$;
    end
  end
  return $\boldsymbol{\beta}$;
end

The LEFS method is described in Algorithm 14. It repetitively identifies a feature so that after removing the feature, the selected task (i.e., $\texttt{idx}(\boldsymbol{\gamma}_{\boldsymbol{\beta}}, 1)$) is unchanged, and the correlation obtained by the new model is not much worse (within a predefined threshold ε). If multiple features meet the condition, then the one with the least correlation decrease is removed (i.e., the corresponding β is set to 0). The feature selection process continues until no feature can be removed without deteriorating the model accuracy. Finally, the remaining weights are normalised, i.e., they are divided by the maximal weight, to make it easy to do clustering later. Note that the weight normalisation does not change the model accuracy, since the task ranks are not affected.

5.3.1.5 An Example Illustration

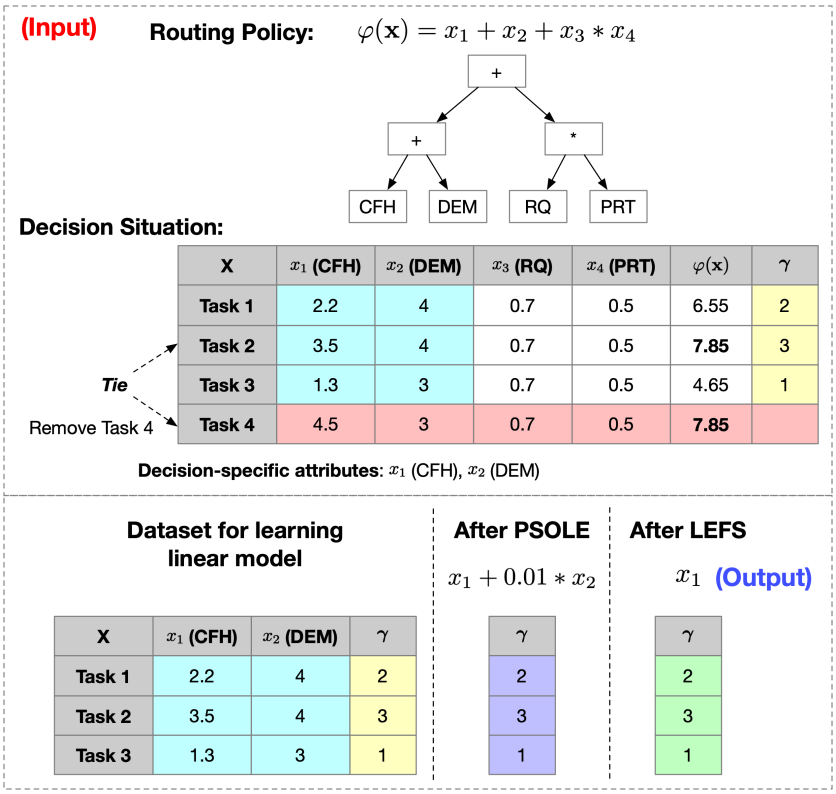
Fig. 5.8 shows an example to explain the routing policy $x_1 + x_2 + x_3 * x_4$, where the meaning of the four attributes are as follows:

- x_1 (CFH): the cost from the vehicle's current location to the candidate task
- x_2 (DEM): the demand of the candidate task
- x_3 (RQ): the remaining capacity of the vehicle
- x_4 (PRT): the percentage of remaining (unserved) tasks

It can be seen that x_1 (CFH) and x_2 (DEM) are decision-specific attributes, while RQ and PRT are decision-independent attributes. After calculating the priority, we found that tasks 2 and 4 have the same priority value, thus task 4 is removed as it has a larger index than task 2. The resultant target rank vector is $[2, 3, 1]$ for the linear model learning. After applying the PSOLE method, we might obtain a linear model $x_1 + 0.01 * x_2$, with perfect consistency and correlation. Finally, after the feature selection, we can further reduce β_2 to zero, and simplify the model to x_1 as

Algorithm 14: LEFS($\boldsymbol{\beta}, \mathbf{X}, \boldsymbol{\gamma}, \varepsilon$)

```
Input: A weight vector β = [β_i]_{N×1}, a decision situation X = [x_ij]_{M×N}, the
       target rank vector γ, a correlation threshold ε
Output: An updated weight vector β̂
Calculate output y_β = [y_i]_{M×1} = X · β;
The ranks γ_β = sortedIdx(y);
Calculate the correlation corr(γ_β, γ);
while true do
    Initialise i* = 0, δ* = ε;
    for i = 1 → N do
        if β_i = 0 then continue;
        // Try to remove feature i
        Set β' = β, and change β'_i = 0;
        Calculate new output y_β' = X · β';
        γ_β' = sortedIdx(y_β');
        if idx(γ_β, 1) = idx(γ_β', 1) then
            Calculate correlation loss δ = corr(γ_β, γ) − corr(γ_β', γ);
            if δ < δ* then
                Update i* = i, δ* = δ;
            end
        end
        if i* = 0 then break;
        Set β_i = 0;                                  // Remove feature i
    end
end
Normalise the weights β̂ = β / max(β);
return β̂;
```



X	x_1 (CFH)	x_2 (DEM)	x_3 (RQ)	x_4 (PRT)	$\varphi(\mathbf{x})$	γ
Task 1	2.2	4	0.7	0.5	6.55	2
Task 2	3.5	4	0.7	0.5	**7.85**	3
Task 3	1.3	3	0.7	0.5	4.65	1
Task 4	4.5	3	0.7	0.5	**7.85**	

X	x_1 (CFH)	x_2 (DEM)	γ
Task 1	2.2	4	2
Task 2	3.5	4	3
Task 3	1.3	3	1

γ
2
3
1

γ
2
3
1

Figure 5.8: An example of the local ranking explanation.

there is no impact on the correlation and consistency when β_2 is removed. In other words, we explain that the routing policy selects the closest task to the current vehicle location in this decision situation.

5.3.1.6 Global Explanation

Intuitively, the global behaviour of a routing policy contains its local behaviour in each possible decision situation. Therefore, we proposed a global explanation method based on a collection of various decision situations and their corresponding local explanations, which is described in Algorithm 15. Given a large number of L decision situations $\mathbf{X}_i$ and their local explanations (i.e., the linear models $\boldsymbol{\beta}_i$), We first cluster the decision situations into a small number of K clusters using K-means so that each

Algorithm 15: Global Explanation Method

Input: A set of local explanations $\langle \mathbf{X}_1, \boldsymbol{\beta}_1 \rangle, \ldots, \langle \mathbf{X}_L, \boldsymbol{\beta}_L \rangle$
$(\mathcal{B}_1, \ldots, \mathcal{B}_K) = \texttt{kmeans}(\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_L)$;
for $i = 1 \rightarrow K$ **do** Set $\mathcal{X}_i = \emptyset$;
for $i = 1 \rightarrow L$ **do**
  Find the cluster index c_i of $\boldsymbol{\beta}_i$;
  Add the first row of $\mathbf{X}_i^{\langle global \rangle}$ into $\mathcal{X}_{c_i}$;
end
Explain the global behaviour based on $\langle \mathcal{X}_1, \mathcal{B}_1 \rangle, \ldots, \langle \mathcal{X}_K, \mathcal{B}_K \rangle$;

cluster contains similar local explanations (i.e., coefficients of the linear models). Then, we analyse the distribution of decision-independent attributes (that represent the global environment) of the decision situations in each cluster. Finally, we can explain the global behaviour of the routing policy based on its local behaviours in each cluster (e.g., in the format of "if (the environment (decision-independent features) is in cluster k), then (the local explanation for cluster k)").

5.3.2 Experimental Study

In this section, we verify the proposed LGRE method on the commonly used UCARP datasets and promising routing policies evolved by GP. Specifically, we use the Ugdb and Uval datasets [141, 201] which have been introduced in Table 2.1.

For each UCARP instance, we run a standard GP and select the routing policy with the best test performance to be explained. The terminal (attribute) set of GP is shown in Table 5.5, where the local (global) attributes are shown as "L" ("G"). All decision-specific attributes are stochastic because all decision-specific attributes are calculated based on stochastic deadheading cost and stochastic demand. The function set is set to $\{+, -, \times, /, \min, \max\}$ where "/" operator returns 1 if divided by 0. The population size is set to

1000. The tournament selection size is set to 7. The maximal GP tree size is set to 8. The crossover and mutation rates are 0.85 and 0.15, respectively. The total number of generations is set to 100. Based on previous studies [122,201], such a GP setting can evolve a promising routing policy. However, they are too large (e.g., with more than 50 nodes and complex combinations) and hard to interpret directly.

For each UCARP instance and its GP-evolved routing policy to be explained, we run 30 simulations with different random seeds (different sampled values of the random demands and deadheading costs). Each simulation consists of a number of decision situations (when a vehicle becomes idle and selects the next task to serve). Then we conduct two steps of experiments:

1. Use the *local ranking explanation* to explain the local behaviour of the routing policy in each decision situation.

2. Use the *global explanation* method to summarise the local explanations in all these decision situations into a global explanation.

For the local ranking explanation, we use the standard PSO with star topology to learn the linear model in Algorithm 13. We set 200 particles and 200 iterations. The parameters c_1 and c_2 are set to 1.49618. The inertia weight w is set to 0.729843788 [57,149,191].

For the global explanation, we use the K-means method with $K = 5$ to cluster the linear models for all the instances.

5.3.3 Results

5.3.3.1 Local Explanation Results

First we investigate the accuracy of the local explanations in terms of consistency and correlation, and the complexity of the learned linear models in terms of number of features used (with non-zero coefficients). To verify the effectiveness of the proposed model learning method (Algorithms 2

Table 5.5: The GP terminal set with local (L) and global (G) attributes.

Terminal	G/L	Description
SC	L	cost of serving the candidate task
CFD	L	cost from the depot to the candidate task
CTT1	L	cost from the candidate to its closest remaining task
CTD	L	cost from the candidate task to the depot
CFH	L	cost from the current vehicle location to the candidate task
DEM	L	expected demand of the candidate task
DEM1	L	demand of the closest unserved task to the candidate task
CR	G	cost to refill, i.e., from the current location to the depot
FULL	G	fullness (served demand over capacity) of the vehicle
PRT	G	percentage of unserved tasks
RQ	G	remaining capacity of the vehicle
ERC	G	a random constant value

and 3), we compare with the Lasso regression method [182], which considers both regression error and feature selection (i.e., better interpretability) in the model learning. Specifically, to fit the rank vector $\boldsymbol{\gamma}$, it minimises the following objective function:

$$\min_{\boldsymbol{\beta}} \frac{1}{M} ||\boldsymbol{\gamma} - \mathbf{X}^{\langle local \rangle} \boldsymbol{\beta}||_2^2 + \lambda ||\boldsymbol{\beta}||_1. \tag{5.9}$$

Note that Lasso can only use the raw linear aggregation $\mathbf{X}^{\langle local \rangle}\boldsymbol{\beta}$ instead of the converted ranks, since it requires continuity and smooth gradient.

Tables 5.6 and 5.7 show the mean and standard deviation of the consistency, correlation and number of used features obtained by Lasso and the proposed LRE over all the decision situations of the 30 simulations for the Ugdb and Uval dataset, respectively.

In terms of accuracy, we clearly see that the proposed LRE achieved much higher consistency and correlation than Lasso. We conducted Wilcoxon rank sum test with significance level of 0.05, and the results show that LRE significantly outperformed Lasso for all the instances in terms of both

consistency and correlation. Furthermore, LRE managed to consistently achieve consistency of 1.0 (i.e., always select the same task as the explained routing policy) for 16 out of the 23 Ugdb instances and 24 out of the 34 Uval instances. Overall, the mean consistency is no worse than 0.91 (Ugdb5). LRE also achieved very high correlation, which is as high as 0.98 in most cases. There are only 3 Ugdb instances where LRE achieved less than 0.9 correlation. In summary, the linear models learned by LRE can not only make the same decision (select the same task), but also achieve the same preference relationship among the candidate tasks as the explained routing policy. This verifies the effectiveness of LRE.

In comparison, Lasso obtained much lower consistency (mostly ranging from 0.6 to 0.8) and correlation (mostly between 0.9 and 0.98) than LRE. The consistency is much lower than correlation. This is because Lasso treats the tasks equally important, and cannot focus on a particular task selected by the routing policy. By minimising the mean square error, Lasso can achieve high correlation but makes more mistakes on the selected tasks. The advantage of LRE over Lasso, especially in consistency, verifies the effectiveness of the new fitness function in the proposed $\texttt{PSOLE}(\cdot)$ algorithm that considers consistency and correlation separately.

In terms of number of features (interpretability), we can see that both LRE and Lasso can obtain interpretable linear models with a small number of features (less than 4 features in almost all the cases). Lasso obtained much fewer features than LRE for some instances (e.g., Ugdb15 and Ugdb17). However, this is at the cost of lower accuracy (i.e., consistency and correlation). LRE obtained fewer features than Lasso on a number of Uval instances, which are larger and more complex than the Ugdb instances. This is because LRE converts the priority values to the ranks, and is more tolerant to the raw priority values. In other words, a simple model that produces very different priority values but still converts to the same ranks can still achieve high consistency and correlation. However, Lasso minimises the error based on the raw priority values, thus is much harder

to achieve high consistency and correlation with a simple model.

Table 5.6: The mean and standard deviation for three metrics of identifying the performance of the local explanation over 30 simulations on *Ugdb* dataset.

	Consistency		Correlation		Complexity	
Instance	Lasso	LRE	Lasso	LRE	Lasso	LRE
Ugdb1	0.9(0.03)	1.0(0.0)	0.97(0.004)	0.99(0.001)	3.1(0.0)	3.0(0.1)
Ugdb2	0.6(0.02)	1.0(0.0)	0.94(0.004)	0.94(0.003)	3.6(0.1)	3.0(0.1)
Ugdb3	0.8(0.05)	1.0(0.02)	0.97(0.004)	0.99(0.002)	2.5(0.1)	1.5(0.1)
Ugdb4	0.7(0.04)	0.9(0.0)	0.98(0.011)	0.99(0.001)	3.4(0.2)	2.8(0.2)
Ugdb5	0.6(0.02)	0.9(0.03)	0.55(0.036)	0.76(0.007)	3.3(0.1)	3.5(0.2)
Ugdb6	0.6(0.02)	1.0(0.0)	0.87(0.006)	0.89(0.018)	3.8(0.1)	3.6(0.1)
Ugdb7	0.9(0.02)	1.0(0.0)	0.77(0.02)	0.94(0.004)	3.3(0.0)	3.4(0.1)
Ugdb8	0.7(0.04)	1.0(0.0)	0.94(0.023)	1.0(0.0)	2.6(0.1)	2.2(0.1)
Ugdb9	0.7(0.04)	1.0(0.01)	0.92(0.012)	0.99(0.001)	1.7(0.1)	1.3(0.0)
Ugdb10	0.7(0.02)	1.0(0.0)	0.95(0.002)	0.98(0.001)	2.0(0.0)	1.8(0.1)
Ugdb11	0.7(0.05)	1.0(0.0)	0.97(0.003)	0.99(0.001)	3.4(0.1)	3.6(0.1)
Ugdb12	0.9(0.04)	1.0(0.0)	0.96(0.004)	0.98(0.002)	3.9(0.1)	3.4(0.2)
Ugdb13	0.5(0.08)	0.9(0.03)	0.91(0.021)	0.99(0.005)	2.7(0.1)	2.5(0.1)
Ugdb14	0.6(0.05)	1.0(0.0)	0.96(0.015)	0.99(0.002)	2.3(0.1)	3.0(0.1)
Ugdb15	0.9(0.0)	1.0(0.0)	0.67(0.009)	0.98(0.0)	1.0(0.0)	3.7(0.0)
Ugdb16	0.8(0.01)	1.0(0.0)	0.87(0.003)	0.98(0.001)	2.2(0.1)	3.5(0.1)
Ugdb17	0.8(0.0)	1.0(0.0)	0.76(0.008)	0.99(0.003)	1.7(0.0)	3.1(0.0)
Ugdb18	0.8(0.01)	1.0(0.0)	0.83(0.008)	1.0(0.0)	2.5(0.0)	2.9(0.1)
Ugdb19	0.6(0.06)	1.0(0.0)	0.91(0.018)	1.0(0.0)	2.5(0.2)	3.7(0.4)
Ugdb20	0.8(0.0)	1.0(0.0)	0.97(0.001)	0.99(0.0)	2.7(0.0)	2.9(0.2)
Ugdb21	0.8(0.02)	1.0(0.0)	0.94(0.013)	0.98(0.001)	3.0(0.1)	3.6(0.1)
Ugdb22	0.7(0.04)	1.0(0.0)	0.82(0.039)	0.98(0.002)	3.0(0.1)	4.0(0.1)
Ugdb23	0.7(0.03)	1.0(0.0)	0.7(0.038)	0.86(0.011)	2.2(0.0)	2.9(0.1)

5.3.4 Further Analysis: Case Studies On Local Explanations

In this section, we provide examples of the local explanations in different decision situations. Specifically, we consider the following scenarios:

- Small-sized instance (Ugdb2 with 26 tasks),
- Medium-sized instance (Ugdb9 with 51 tasks),

Table 5.7: The mean and standard deviation for three metrics of identifying the performance of the local explanation over 30 simulations on *Uval* dataset.

	Consistency		Correlation		Complexity	
Instance	Lasso	LRE	Lasso	LRE	Lasso	LRE
Uval1A	0.8(0.0)	1.0(0.0)	0.95(0.0)	0.99(0.0)	3.3(0.0)	3.3(0.1)
Uval1B	0.5(0.01)	1.0(0.0)	0.85(0.001)	0.91(0.001)	3.2(0.0)	3.2(0.1)
Uval1C	0.9(0.03)	1.0(0.0)	0.97(0.006)	1.0(0.001)	4.0(0.1)	3.9(0.1)
Uval2A	0.7(0.0)	1.0(0.0)	0.98(0.0)	0.99(0.0)	3.9(0.0)	2.0(0.1)
Uval2B	0.8(0.02)	1.0(0.01)	0.98(0.005)	0.99(0.001)	2.7(0.1)	1.6(0.1)
Uval2C	0.6(0.07)	1.0(0.01)	0.92(0.009)	0.97(0.003)	4.3(0.1)	4.1(0.1)
Uval3A	1.0(0.0)	1.0(0.0)	0.99(0.0)	1.0(0.0)	2.2(0.0)	1.4(0.1)
Uval3B	0.9(0.01)	1.0(0.0)	0.87(0.006)	0.95(0.001)	3.3(0.0)	3.8(0.1)
Uval3C	0.8(0.04)	1.0(0.0)	0.92(0.013)	0.99(0.001)	3.3(0.2)	3.3(0.2)
Uval4A	0.8(0.03)	1.0(0.0)	0.99(0.002)	1.0(0.0)	3.1(0.1)	1.9(0.1)
Uval4B	0.7(0.02)	1.0(0.0)	0.96(0.006)	0.97(0.003)	4.6(0.1)	2.7(0.2)
Uval4C	0.8(0.02)	1.0(0.0)	0.97(0.005)	0.98(0.002)	3.8(0.1)	2.7(0.1)
Uval4D	0.8(0.03)	1.0(0.0)	0.96(0.003)	1.0(0.0)	4.1(0.1)	2.8(0.1)
Uval5A	0.7(0.01)	1.0(0.01)	0.95(0.001)	0.99(0.001)	5.4(0.1)	3.1(0.1)
Uval5B	0.8(0.04)	1.0(0.0)	0.99(0.001)	1.0(0.0)	3.2(0.1)	1.2(0.1)
Uval5C	0.8(0.03)	1.0(0.0)	0.96(0.007)	0.98(0.003)	4.1(0.1)	2.8(0.1)
Uval5D	0.6(0.03)	1.0(0.0)	0.97(0.002)	0.99(0.0)	3.6(0.1)	2.3(0.1)
Uval6A	0.6(0.0)	1.0(0.0)	0.97(0.0)	0.99(0.0)	4.6(0.0)	3.2(0.1)
Uval6B	0.7(0.0)	1.0(0.0)	0.92(0.0)	0.94(0.0)	3.4(0.0)	3.2(0.1)
Uval6C	0.6(0.04)	0.9(0.02)	0.93(0.011)	0.97(0.004)	3.2(0.1)	2.3(0.1)
Uval7A	0.8(0.0)	1.0(0.0)	0.98(0.0)	1.0(0.0)	3.9(0.0)	2.3(0.1)
Uval7B	0.7(0.01)	1.0(0.0)	0.98(0.0)	1.0(0.0)	3.5(0.0)	2.3(0.1)
Uval7C	0.7(0.03)	1.0(0.0)	0.96(0.012)	0.99(0.001)	2.9(0.1)	2.3(0.1)
Uval8A	0.7(0.01)	1.0(0.0)	0.95(0.003)	0.97(0.002)	4.3(0.0)	2.3(0.1)
Uval8B	0.8(0.04)	1.0(0.01)	0.98(0.006)	0.99(0.003)	4.3(0.1)	2.1(0.2)
Uval8C	0.7(0.05)	1.0(0.0)	0.97(0.003)	0.99(0.001)	4.4(0.1)	2.8(0.1)
Uval9A	0.7(0.01)	1.0(0.0)	0.99(0.0)	1.0(0.0)	3.7(0.0)	1.5(0.0)
Uval9B	0.8(0.03)	1.0(0.0)	0.99(0.004)	1.0(0.001)	3.0(0.1)	1.3(0.1)
Uval9C	0.8(0.02)	1.0(0.01)	0.98(0.008)	0.98(0.003)	1.9(0.0)	1.3(0.0)
Uval9D	0.7(0.03)	1.0(0.0)	0.98(0.004)	0.99(0.001)	3.4(0.1)	1.7(0.1)
Uval10A	0.9(0.01)	1.0(0.0)	1.0(0.001)	1.0(0.0)	3.5(0.0)	1.2(0.0)
Uval10B	0.8(0.02)	1.0(0.0)	0.99(0.002)	1.0(0.0)	3.1(0.1)	1.2(0.0)
Uval10C	0.7(0.03)	1.0(0.01)	0.94(0.005)	0.98(0.002)	3.7(0.1)	2.6(0.1)
Uval10D	0.7(0.04)	1.0(0.0)	0.96(0.005)	0.98(0.002)	4.1(0.1)	2.7(0.1)

- Large-sized instance (Uval10C with 97 tasks),

For each instance, we consider decision situations where the vehicle is (a) empty; (b) half-loaded; or (c) full.

5.3.4.1 Ugdb2

The GP-evolved routing policy for Ugdb2 has the test performance of 361.78 (vs 414.50 of the path scanning heuristic [113]). It contains 55 nodes and 9 unique features.

In the first decision situation, the decision-independent feature FULL $= 0$, i.e., the vehicle is completely empty. The linear model obtained by LRE is as follows. Its consistency is 1, and correlation is 0.97.

$$\text{priority}^{\text{Ugdb2}}_{\text{empty}} = 0.1 \cdot \text{CFD} + 0.46 \cdot \text{CFH} - 0.07 \cdot \text{CTD} - \text{SC}. \tag{5.10}$$

Explanation. *Note that when the vehicle is empty, it must be at the depot. In this case, CFD and CFH are essentially the same. From the linear model, the vehicle prefers the tasks close to it ($0.1 \cdot$ CFD $+ 0.46 \cdot$ CFH) and have large serving costs ($-$SC). It also slightly prefers the tasks whose tail nodes (the end node of the edge) are far away from the depot ($-0.07 \cdot$ CTD). This is consistent with our intuition that when a vehicle is empty, its route should be in an outgoing direction, and the closer tasks are more preferred to minimise the total cost. Furthermore, the vehicle prefers the heavier tasks with larger serving costs, since leaving them to the end of the route can lead to large recourse cost if a route failure occurs then.*

In the second decision situation, FULL $= 0.58$, i.e., the vehicle is a bit over half-loaded. The linear model is as follows. Its consistency is 1, and correlation is 0.97.

$$\text{priority}^{\text{Ugdb2}}_{\text{half}} = \text{CFH} + 0.1 \cdot \text{CTD} + 0.03 \cdot \text{SC}. \tag{5.11}$$

Explanation. *The vehicle prefers the tasks close to it (CFH) and with small serving cost ($0.03 \cdot$ SC), and the emphasis on serving cost decreases as the vehicle becomes more loaded. However, it prefers the tasks whose tail node is closer to the depot ($0.1 \cdot$ CTD) now. This is because after half-loaded, the vehicle should be on the way back to the depot.*

In the last decision situation, FULL $= 0.71$. The linear model is shown below. Its consistency is 1, and correlation is 0.99.

$$\text{priority}_{\text{full}}^{\text{Ugdb2}} = 0.6 \cdot \text{CFD} + \text{CFH} + 0.33 \cdot \text{CTD}. \tag{5.12}$$

Explanation. *When the vehicle is nearly full, the serving cost becomes redundant, and the vehicle prefers the tasks close to it (CFH) and also to the depot ($0.6 \cdot$ CFD $+ 0.33 \cdot$ CTD). Compared to Eq. (5.11), the coefficient of CTD is smaller in Eq. (5.12), but the coefficient of CFD is larger, which indicates that although the vehicle is concerned with similar characteristics after the vehicle is half-loaded. The focus on CTD and CFD vary according to the vehicle load.*

Overall, the local explanations of the three different decision situations show that the routing policy for Ugdb2 always prefers the tasks close to the vehicle. In addition, the serving cost and the costs from and to the depot play different roles in different planning stages. All the patterns are consistent with our intuition.

5.3.4.2 Ugdb9

The GP-evolved routing policy for Ugdb9 has the test performance of 357.15 (vs 419.23 of the path scanning heuristic). It has 79 nodes and 6 unique features.

We select three decision situations for Ugdb9 with different vehicle loads. The first decision situation has Full $= 0$, i.e., the vehicle is completely empty. The linear model is as follows. Its consistency is 1, and correlation is 1.

$$\text{priority}_{\text{empty}}^{\text{Ugdb9}} = 0.65 \cdot \text{CFH} - \text{CTD}. \tag{5.13}$$

Explanation. *The vehicle simply prefers the tasks close to it ($0.65 \cdot$CFH) and with tail nodes far away from the depot ($-$CTD). Both features are very important for the decision, since their coefficients are both large.*

In the second decision situation, FULL $= 0.49$, i.e., the vehicle is half-loaded. The linear model is shown as follows. Its consistency is 1, and

correlation is 0.99.

$$\text{priority}_{\text{half}}^{\text{Ugdb9}} = \text{CFH}. \tag{5.14}$$

Explanation. *This is a surprisingly simply linear model. The vehicle simply selects the task that is closest to it (ties are broken by selecting the one with the lowest index). This indicates that for Ugdb9, the nearest neighbour heuristic can work just well in the middle of the routing stage.*

The last decision situation has FULL $= 0.93$, i.e., the vehicle is almost full. The corresponding linear model is shown below. Its consistency is 1, and correlation is 1.

$$\text{priority}_{\text{full}}^{\text{Ugdb9}} = 0.46 \cdot \text{CFH} + \text{CTD}. \tag{5.15}$$

Explanation. *The vehicle prefers the tasks close to it ($0.46 \cdot$ CFH and also close to the depot CTD. This is consistent with our intuition, since when the vehicle is almost full, it should go back to the depot, and serve the tasks on its way back. Compared with the model Eq. (5.12) for Ugdb2, the cost to the depot is more important than the cost from the vehicle (CTD has a larger coefficient). This could be due to the different graph topology between Ugdb2 and Ugdb9.*

Overall, the linear models obtained for the three different decision situations for Ugdb9 are also easy to interpret, and their patterns are consistent with our intuition.

5.3.4.3 Uval10C

The GP-evolved routing policy for Uval10C has the test performance of 480.36 (vs 504.44 of the path scanning heuristic). It contains 59 nodes and 8 unique features.

For Uval10C, the first decision situation has FULL $= 0$, i.e., the vehicle is completely empty. The linear model is shown below. Its consistency is 1, and correlation is 0.95.

$$\text{priority}_{\text{empty}}^{\text{Uval10C}} = 0.12 \cdot \text{CFD} + 0.82 \cdot \text{CFH} - \text{CTD}. \tag{5.16}$$

Explanation. *Note that CFD = CFH when vehicle is empty. Thus, the vehicle prefers the tasks close to it* $0.82 \cdot$ *CFH and tail nodes far away from the depot (−CTD). It also slightly prefers the tasks whose head node (the start node of the edge) is close to the depot (where the vehicle is when empty). This pattern is similar to that in the linear models for Ugdb2 Eq. (5.10) and Ugdb9 Eq. (5.13) when the vehicle is empty, and is consistent with our intuition.*

The second decision has FULL $= 0.47$, i.e., the vehicle is a bit less than half-loaded. The linear model is as follows. Its consistency is 1, and correlation is 0.99.

$$\text{priority}_{\text{half}}^{\text{Uval10C}} = -0.53 \cdot \text{CFD} + \text{CFH}. \tag{5.17}$$

Explanation. *The vehicle prefers the tasks close to it (CFH) and whose tail nodes are far away from the depot (*$-0.53 \cdot$ *CFD). The preference on small CFH is consistent with the other cases and our intuition. The preference on large CFD when half-loaded, although not considered in other cases, can be explained if the graph contains more tasks that are far away from the depot. This suggests that the vehicle still needs to travel outwardly when half loaded.*

In the last decision situation, FULL $= 0.96$, which means that the vehicle is almost full. The corresponding linear model is shown as follows. Its consistency is 1, and correlation is 1.

$$\text{priority}_{\text{full}}^{\text{Uval10C}} = \text{CTD}. \tag{5.18}$$

Explanation. *This linear model is surprisingly simple. It simply selects the tasks whose tail node is closest to the depot. This makes sense since toward the end of a route, it is highly likely to have route failures, and tasks with small CTD can reduce the recourse cost. Interestingly, the vehicle does not even consider CFH, which is known to be the most important routing feature. In other words, this local explanation is specific to this decision situation where all the candidate tasks have the same CFH. It also suggests the need for a more general global explanation.*

Overall, we found some different patterns for the local explanations of Uval10C from that of Ugdb2 and Ugdb9. However, these patterns are

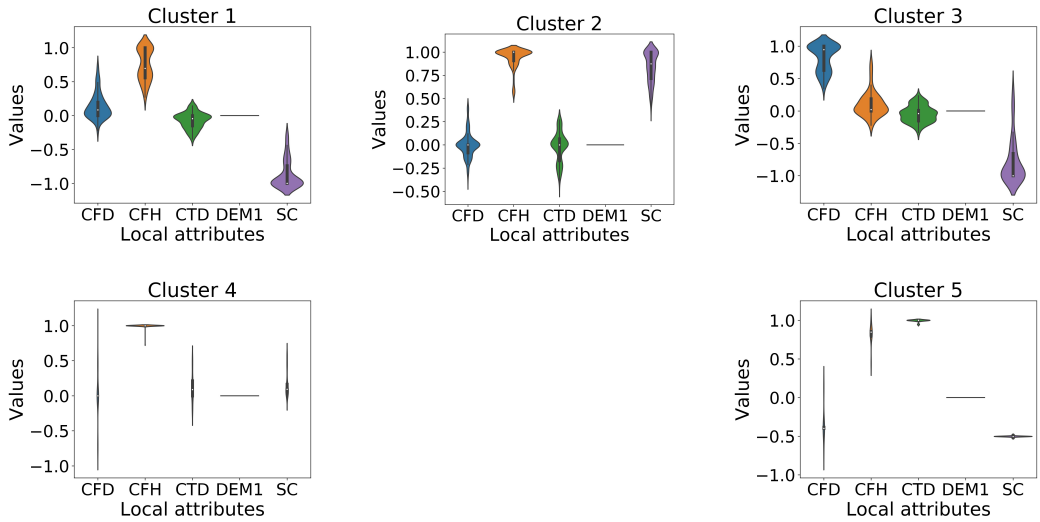


Figure 5.9: The distribution of the the coefficients of decision-specific attributes for each cluster in Ugdb2.

explainable and dependent on the instance characteristics such as graph topology. The local explanations seem to be over-specific to instance and decision situation. Therefore, it is desirable to have a more general global explanation that is independent of decision situation.

5.3.5 Further Analysis: Case Studies On Global Explanations

In this section, we show two examples of the global explanations obtained by Algorithm 15 (with $K = 5$) for Ugdb2 and Uval10C. Specifically, for each instance, we first collect the local explanations (linear models) of all the decision situations encountered in the 30 simulations (excluding those with only a single candidate task). Then, we cluster the $\boldsymbol{\beta}$ vectors of the local explanations into 5 clusters. The presence of too many clusters in the global explanation would make it difficult for the user to understand the whole global explanation. However, too few clusters can again lead to a lack of clarity in the global explanation. Here, to retain good interpretabil-

ity, we limit the maximal number of clusters to 5. We have tried 2, 3, 4, 5 clusters and found that 5 clusters showed the best clustering performance on $\boldsymbol{\beta}$. Then, we observe the distribution of the decision-independent attributes in each cluster and explain the relationship between the decision-independent attribute distribution (decision situation properties) and the $\boldsymbol{\beta}$ vector (routing policy behaviour).

5.3.5.1 Ugdb2

Fig. 5.9 shows the distribution (violin plot) of the $\boldsymbol{\beta}$ vectors of each cluster in Ugdb2. We can see that different clusters have different distributions of the $\boldsymbol{\beta}$ vectors. In each cluster, the relative importance (coefficient magnitude) of different decision-specific attributes are very clear. For example, in cluster 1, CFH and SC are much more important than others (their coefficients have much larger magnitude).

Fig. 5.10 shows the distribution of the decision-independent attributes of the decision situations in each cluster of Ugdb2, where each point stands for a decision situation. Note that there are four decision-independent attributes: CR, FULL, PRT and RQ. In the figure, we show only two decision-independent attributes FULL and PRT, and omit CR and RQ to facilitate visualisation. By definition, we have $\text{RQ} = (1 - \text{FULL}) \cdot Q$, thus RQ is redundant if FULL is already shown. Second, CR is omitted because it has essentially no effect on the clustering results. Other decision-independent attributes are sufficient to represent a decision situation. In addition, it is easier to observe the clustering results in two dimensions.

From Figs. 5.9 and 5.10, we can see the following patterns of the routing policy for Ugdb2.

- **(Cluster 1.)** The cluster 1 contains the red circle points on the left of Fig. 5.10 with $\text{FULL} = 0$, which are overlapped with the blue points (cluster 3). ***IF*** the vehicle is empty, ***THEN*** the vehicle tends to select the tasks close to its current location (large positive coef-

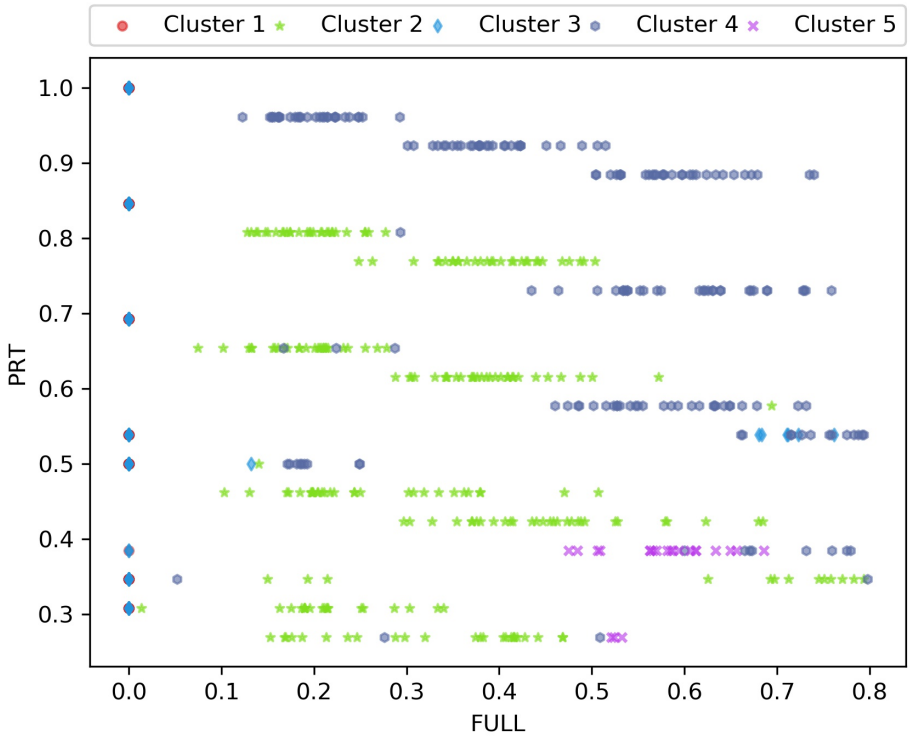


Figure 5.10: The distribution of decision-independent attributes of the decision situations in each cluster of Ugdb2.

ficient of CFH) and large serving cost (large negative coefficient of SC). It slightly prefers the tasks whose head nodes are close to the depot (slight positive coefficient of CFD) and tail nodes are far away from the depot (slight negative coefficient of CTD), i.e., the tasks with outward directions. However, the importance of CFD and CTD are much less than CFH and SC. DEM1 is completely redundant, as its coefficient is always 0.

- **(Cluster 2.)** ***IF*** toward the later service process (PRT < 0.5) or in the early stage ($0.5 <$ PRT < 0.8) and the vehicle is less than half full (FULL < 0.5, ***THEN*** the vehicle prefers the tasks close to it (large positive coefficient of CFH) and with small serving cost (large pos-

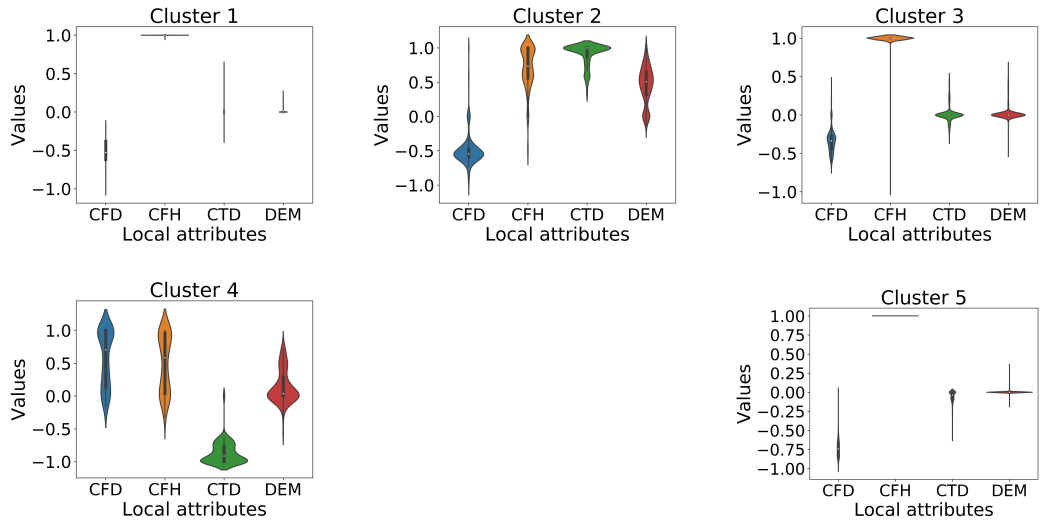


Figure 5.11: The distribution of the coefficients of **decision-specific attributes** for each cluster in **Uval10C**.

itive coefficient of SC). All the other decision-specific attributes are much less important, as their coefficients are close to 0.

- **(Cluster 3.)** This points in cluster 3 heavily overlap with that in cluster 1 in Fig. 5.10 with FULL $= 0$. By comparing the decision-specific attribute distributions of these two clusters in Fig. 5.9, we can see that the clusters 1 and 3 have similar distributions, except that the coefficients of CFD and CFH are swapped. Note that when the vehicle is empty, it is essentially at the depot, and CFD is essentially equal to CFH. Therefore, these two clusters essentially represent the same situation, i.e., ***IF*** the vehicle is empty and at the depot, ***THEN*** the vehicle prefers the tasks close to it and with a large serving cost.

- **(Cluster 4.)** ***IF*** at the very beginning of the service process (PRT $>$ 0.9) or during the first half of the service process ($0.5 <$ PRT < 0.9) and the vehicle is more than half full (FULL > 0.5), ***THEN*** the vehicle highly prefers the tasks close to it (largest positive coefficient of CFH). The importance of other decision-specific attributes are almost

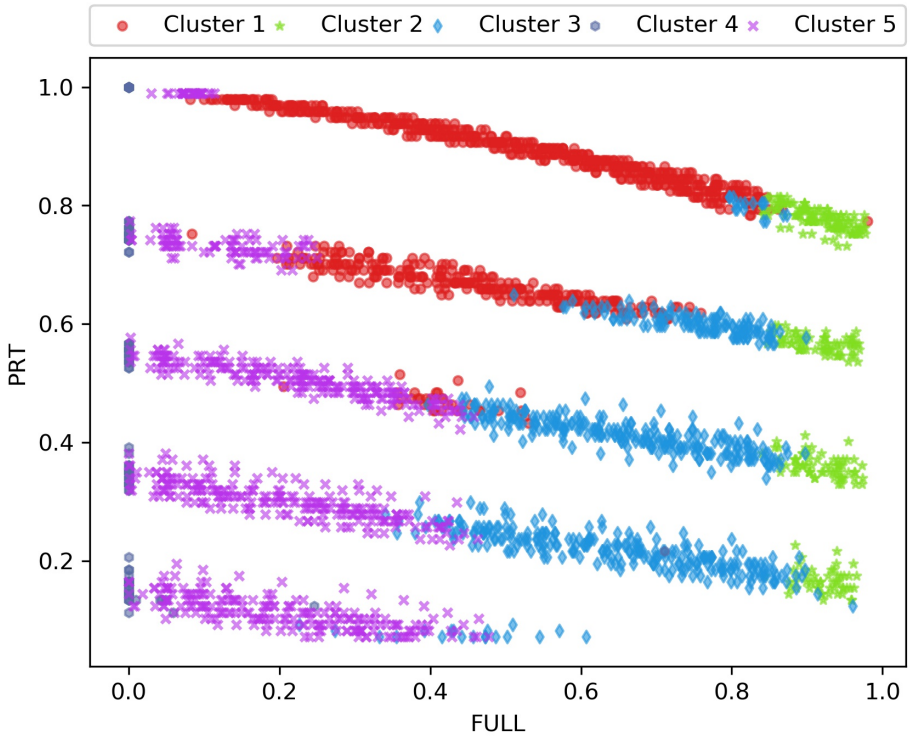


Figure 5.12: The distribution of **decision-independent attributes** of the decision situations in each cluster of **Uval10C**.

negligible (their coefficients are close to 0).

- **(Cluster 5.)** ***IF*** in the later stage of the service process (PRT < 0.5) and when the vehicle is substantially loaded ($0.5 <$ FULL < 0.7), ***THEN*** the vehicle tends to select the close tasks (large positive CFH coefficient) and close to the depot (large positive CTD coefficient). In addition, it prefers the tasks whose head nodes are far away from the depot (its direction is towards the depot) and with small serving cost (small recourse cost if route failure occurs).

Overall, the global explanation of the routing policy for Ugdb2 is as follows. The vehicle always prefers the tasks close to it. When the vehicle is empty, it prefers the tasks withe outward directions from the depot

and large serving cost. However, toward the later stage of the service process and when the vehicle is more loaded, the vehicle gradually switches its focus on the tasks with smaller serving cost, and whose directions are towards the depot.

5.3.5.2 Uval10C

Fig. 5.11 shows the distribution of the decision-specific attribute coefficients for each cluster in Uval10C. Note that the routing policy for Uval10C is different from that for Ugdb2, and contains a different set of decision-specific attributes. Specifically, the routing policy for Uval10C considers DEM, while ignores DEM1 and SC. The corresponding distribution of decision-independent attributes (FULL and PRT) of the decision situations in each cluster of Uval10C is shown in Fig. 5.12.

From the figures, we can observe the following patterns.

- **(Cluster 1.)** ***IF*** in the early stage of the service process (PRT > 0.6, ***THEN*** the vehicle prefers the tasks close to it (coefficient of CFH close to 1) and whose head node is far away from the depot (negative CFD coefficient). This indicates that the vehicle aims to travel to the regions far away from the depot as early as possible.

- **(Cluster 2.)** ***IF*** the vehicle is close to full (route failures are more likely to occur), ***THEN*** the vehicle tends to select the task close to it (positive CFH coefficient), whose directions are towards the depot (negative CFD coefficient and positive CTD coefficient), and with small expected demand (positive DEM coefficient) to reduce the likelihood of route failures.

- **(Cluster 3.)** ***IF*** the vehicle is substantially loaded but not very full ($0.4 <$ FULL < 0.8), ***THEN*** the vehicle highly focuses on the tasks close to it (CFH coefficient close to 1), and slightly considers the tasks whose head nodes are far away from the depot (slightly negative CFD coefficient).

- **(Cluster 4.)** ***IF*** the vehicle is empty and at the depot, ***THEN*** the vehicle tends to select the tasks close to it (positive coefficients for CFH and CFD) and whose directions are outward from the depot (positive CFD coefficient and negative CTD coefficient).

- **(Cluster 5.)** ***IF*** the vehicle is less than half loaded, ***THEN*** the vehicle prefers the task close to it (CFH coefficient is 1) and whose head nodes are far away from the depot (negative CFD coefficient).

Overall, we can give the following global explanation of the routing policy for Uval10C. The vehicle always prefers close tasks. When the vehicle is empty and at the depot, it prefers the tasks with outward directions from the depot. When there are many unserved tasks and the vehicle is at the beginning of its route, it aims to travel to the regions far away from the depot as early as possible. Then, towards the later stage of the route, the vehicle prefers the tasks pointing towards the depot and with small demand to reduce possible route failures.

5.3.6 Summary

In summary, the results and analysis showed that the new proposed approach can get better consistency, correlation and complexity, which indicates that it can better explain the corresponding routing policy than the approach in [200]. In addition, the new proposed approach not only provides local explanations for the corresponding routing policy but can also provide a global explanation to users to understand the whole routing policy, which cannot be achieved in previous local ranking explanation work.

5.4 Chapter Summary

This chapter aims to explain the complex GP-evolved routing policies for UCARP. This goal has been successfully achieved by the proposed local-

global ranking explanation method. The local ranking explanation can obtain a simple linear model that accurately explains the ranking behaviour of the routing policy in each decision situation. The global explanation method can summarise the local linear models into an if-then explanation. We provide a number of case studies to show how a complex GP-evolved routing policy for UCARP can be explained by the proposed method. Our method is not restricted to UCARP, but can be easily generalised to other optimisation and machine learning problems where decisions are made based on ranking.

In this chapter, this work firstly propose two metrics that can evaluate the quality of a local explanation for GP-evolved routing policies. Then, this work investigate the feasibility of using a linear model to construct a local explanation. After that, this work further improve the local explanation method and finally this work extend the local explanation method to global explanation method.

The techniques presented in this chapter offer novel contributions to the field of interpretable machine learning, with a specific focus on Genetic Programming (GP)-evolved routing policies for the Uncertain Capacitated Arc Routing Problem (UCARP). These contributions stand out in several key ways compared to the existing literature in interpretable machine learning. Firstly, the chapter introduces the concept of local explanations tailored for the intricate routing policies in UCARP, acknowledging that numerous decisions are made in diverse decision situations, which require micro-level understanding. The chapter also innovatively introduces two novel metrics, "consistency" and "correlation," designed to evaluate the quality of local explanations, catering specifically to the complexity of UCARP's routing policies.

Furthermore, the exploration of the feasibility of employing linear models for local explanations represents a distinctive approach. While linear models are common in interpretable machine learning, applying them to elucidate GP-evolved routing policies in combinatorial optimization prob-

lems like UCARP presents unique challenges and opportunities. This investigation adds a new dimension to the field.

Another significant departure from the conventional literature is the extension of local explanations to global explanations by identifying patterns in local explanations across different decision situations. This transformation allows for the development of overarching insights into routing policy behavior, bridging the gap between micro-level understanding and macro-level insights.

Perhaps the most critical differentiation lies in the problem-specific application of these techniques. UCARP is a complex, real-world problem with applications in logistics and transportation, where interpreting GP-evolved routing policies is essential for practical deployment. The proposed techniques are explicitly tailored to meet the interpretability needs of UCARP, distinguishing them from more generic applications in interpretable machine learning.

The innovations presented in this chapter, including problem-specific adaptation, novel metrics, and the focus on local and global explanations, set these techniques apart from the existing interpretable machine learning literature. They offer a unique approach to address the complex challenges posed by UCARP and have the potential to significantly enhance the understanding and adoption of GP-evolved routing policies in real-world applications. Although the global explanation is not perfect yet (overlapping among clusters), we will investigate more to develop more effective way to generate global explanation.

Chapter 6

Conclusions

This thesis aims at improving the interpretability of GP-evolved routing policies for the UCARP. This goal has been successfully achieved by proposing both intrinsic and post-hoc methods. From the intrinsic aspect, we have proposed a GP program simplification method to automatically simplify the routing policies during the evolutionary process to get simpler thus more interpretable routing policies. We have also designed several multi-objective GP algorithms to optimise the interpretability along with the effectiveness of routing policies. New MOGP methods produce a Pareto front a routing policies with different tradeoff between the effectiveness and interpretability so that end-users from different domains can choose based on their preference. From the post-hoc aspect, we have proposed local and global explanation methods to further improve the interpretability of existing complex GP-evolved routing policies. Furthermore, the local and global explanation methods not only produce text explanations but also visual explanations.

The rest of this chapter highlights the achieved objectives in this thesis, followed by the main conclusions. Then, insightful discussions are provided to help understand the key issues in this research area. Finally, this chapter presents some potential research directions which are motivated by the studies in this thesis.

6.1 Achieved Objectives

The following research objectives have been fulfilled by this thesis.

1. This thesis has proposed an effective online simplification method for GP to evolve simpler routing policies for UCARP (Chapter 3). The new simplification approach use niching technique to group the individuals in the population based on their phenotypic behaviour (each group contains the individuals with the same behaviour), and simplify all the individuals in each group by replacing it with the smallest individual in that group. In this case, complex routing policies can be simplified to its simple version without losing effectiveness. This approach also contains an archive to maintain the diversity loss caused by the simplification. This work not only effectively reduces the complexity of GP-evolved routing policies but also improves the effectiveness of GP-evolved routing policies. This work motivates the study of explainable GP method for uncertain capacitated arc routing problems.

2. This thesis has developed different multi-objective GP algorithms to produce Pareto front of routing policies with different degrees of interpretability (Chapter 4). First, we investigate different strategies to adapt the α parameter to handle the objective selection bias issue. The α adaptation schemes are manually designed. The objective selection bias issue is partially addressed by manually designed schemes. Second, we design new self-adaptive α adaptation scheme to automatically adjust the α during the evolutionary process. The results show that self-adaptive α scheme can better handle the objective selection bias issue. Third, we develop an archive strategy to handle the stochastic evaluation issue. The archive is utilised to maintain the potentially good individuals evolved during the evolutionary process. The results show that the archive strategy effectively maintain the diversity and produce relatively complete Pareto

front. Finally, this thesis has developed a new multi-objective GP algorithm to handle both objective selection bias issue and stochastic evaluation issue simultaneously. The new MOGP algorithm contains both the α dominance strategy and the archive strategy. The results show that the proposed algorithms can evolve more complete Pareto front than common used multi-objective GP algorithms. In addition, it evolves more interpretable routing policies than single-objective GPHH algorithms that only optimise effectiveness. This thesis also shows how multi-objective optimisation can be used to improve the interpretability of the evolved routing policies.

3. This thesis makes several significant contributions to the field of post-hoc interpretability methods for GP-evolved routing policies in UCARP (Chapter 5). Firstly, the chapter proposes two novel metrics that evaluate the quality of a local explanation for GP-evolved routing policies. Secondly, the feasibility of constructing local explanations using a linear model is investigated, which suggests the effectiveness of linear models in generating simple and interpretable explanations. The chapter then introduces an improved method that combines linear models with feature selection algorithms for constructing local explanations. Moreover, the chapter extends the local explanation method to a global explanation method, providing insights into the overall behavior of GP-evolved routing policies. Lastly, the chapter recognises the importance of presenting explanations in different formats and emphasises the flexibility of providing local and global explanations in both text and visual formats. These contributions offer a comprehensive framework for interpreting and understanding the decision-making processes of GP-evolved routing policies, which can assist in enhancing their performance and real-world adoption.

6.2 Main Conclusions

This section describes the main conclusions for this thesis drawn from the five major contribution chapters, i.e., Chapter 3 to Chapter 5.

6.2.1 Interpretability Improvement with Simplification

To improve the interpretability of GP-evolved routing policies for UCARP, simplifying the evolved GP trees are the most straightforward approach. However, two main challenges must be addressed: identifying redundant materials in GP trees and removing them without negatively impacting the evolutionary process.

Chapter 3 presents an effective simplification method for simplifying GP trees during the evolutionary process. The proposed method identifies redundant materials based on the phenotypic behaviors of GP trees. Specifically, the niching technique is used to group individuals in the population based on their phenotypic behavior, with each niche containing individuals with the same behavior. The method then simplifies all individuals in each group by replacing them with the smallest individual in that group. To address the loss of population diversity resulting from niching simplification, the original population is preserved, and an archive is established to store niche representatives. Additionally, this thesis proposes a multi-source breeding mechanism to generate offspring from both the original population and the representative archive.

The proposed method is evaluated and compared to existing methods on 57 UCARP instances. Results indicate that the proposed method outperforms all compared approaches in terms of test effectiveness. Furthermore, the effects of each component of the proposed method are analysed through a set of controlled experiments, revealing that all new components contribute to evolving smaller and better routing policies. This work broadens the scope of GP simplification to UCARP and uses it to enhance the interpretability of GP-evolved routing policies. Moreover, the work

proposes a new simplification method that can replace algebraic and numerical simplification. The proposed method has higher effectiveness and efficiency, particularly for complex problems requiring large trees, such as UCARP.

In real-world applications, end-users will not adopt routing policies they cannot comprehend and trust. Using the proposed simplification method can aid in evolving more interpretable routing policies, making it more feasible to apply GP-evolved routing policies in practical settings.

6.2.2 Interpretability Improvement with Multi-Objective GP

To consider the interpretability in GPHH, there are several challenges. First, there is a trade-off between interpretability and effectiveness. GP trees that are designed to be more interpretable may sacrifice effectiveness, while more effective routing policies may be less interpretable. Finding the right balance between the two can be challenging. Second, The definition of "interpretable" can differ based on the targeted domain and audience. It can be a difficult task to develop a GP tree that is easily understandable and meets the requirements and inclinations of a particular user group.

Chapter 4 presents several Multi-Objective GP (MOGP) algorithms to evolve interpretable routing policies for UCARP. MOGP algorithms have shown great potential in solving UCARP because they can generate Pareto-optimal solutions that provide a trade-off between multiple objectives.

Two main issues arise when designing MOGP algorithms for UCARP. The first issue is the objective selection bias issue, which refers to the phenomenon that some objectives are more likely to be selected by the MOGP algorithm than others, leading to suboptimal or incomplete Pareto fronts. To address this issue, Chapter 4 proposes an MOGP algorithm using α dominance, a popular dominance relation that considers a trade-off between the objectives. The proposed algorithm incorporates manually de-

signed α adaptation schemes to dynamically adjust the α values, allowing for a more balanced representation of the objectives. The experimental results demonstrate that the proposed approach can evolve smaller routing policies without sacrificing the test performance. However, the manual design of α adaptation schemes can be time-consuming and requires domain knowledge. Thus, Chapter 4 further improves the MOGP algorithm using α dominance by designing a new self-adaptive α scheme. The self-adaptive α scheme adjusts the α values dynamically based on the bias status of the objectives, allowing for a more efficient and effective search for Pareto-optimal solutions. The experimental results show that the self-adaptive α scheme generates much better Pareto fronts in terms of Hypervolume (HV) and Inverted Generational Distance (IGD) than manually designed schemes, and can find different α value for different instances.

The second issue encountered when designing MOGP algorithms for UCARP is the stochastic fitness evaluation issue, which refers to the random rotation in the fitness values of the individuals due to the use of a limited number of training samples. To handle this issue, Chapter 4 proposes an archive strategy that stores a wide range of potentially effective individuals, in case they are lost from the population due to the training sample rotation. The archive strategy provides a more distributed Pareto front of routing policies, making it easier for the end-users to select routing policies based on their preferences.

Finally, Chapter 4 proposes an MOGP algorithm that handles both the objective selection bias issue and the stochastic evaluation issue simultaneously. The proposed MOGP algorithm involves both α dominance strategy and archive strategy, allowing for a more robust and reliable search for Pareto-optimal solutions. The experimental results show that the proposed algorithm, αMOGP-a, evolves a relatively complete Pareto front compared with previous work, providing end-users with a diverse set of routing policies with different levels of interpretability.

Regarding the practical application of GP-evolved routing policies, end-

users may come from different groups and have different domain knowledge. Some users can understand very complex routing policies, while others may only understand very simple routing policies. Different users may have different needs for interpretability. Therefore, providing only a single routing policy is not a good option. The proposed MOGP algorithm addresses this problem by providing a set of routing policies with varying interpretability, enabling end-users to select routing policies based on their preferences. Thus, the MOGP algorithm proposed in this study advances the practical application of GP-evolved routing policies.

6.2.3 Interpretability Improvement with Local and Global Explanation

Improving interpretability of GP-evolved routing policies faces a challenge due to the application of complex nonlinear transformations to input variables by GP trees, which makes it difficult to understand the input-output relationship. Post-hoc interpretability methods are gaining popularity in providing insights into complex machine learning models, particularly through text explanation and visualization techniques, which show promise in the context of GP-evolved routing policies for UCARP. These methods aim to explain the model's decision-making process, factors influencing the policy's design, and performance under different scenarios. However, despite their potential, empirical studies are currently lacking in investigating the effectiveness of such methods for GP-evolved routing policies in UCARP. Therefore, it is essential to design novel interpretability methods tailored to the specific characteristics and requirements of this domain. Chapter 5 presents both text explanation and visualization techniques to enhance the interpretability of GP-evolved routing policies for UCARP.

Chapter 5 investigates the feasibility of using a simple linear regression method to generate linear models as local explanations for GP-evolved

routing policies. The results show that it is possible to use a linear model to fit a routing policy in a decision situation. However, a new mechanism is required to produce more accurate linear models. Therefore, Chapter 5 proposes a PSO-based local ranking explanation method to evolve linear models. The results show that the PSO-based local ranking explanation method generates simpler and more effective linear models that can fit GP-evolved routing policies. Moreover, Chapter 5 extends local explanations to global explanations using a clustering technique. Finally, Chapter 5 produces local and global text and visual explanations for GP-evolved routing policies for UCARP.

In real-world applications, providing users with a complex model is sometimes not a good choice. Even when a relatively simple model is provided, it often takes a long time for users to fully understand the model's internal mechanism. In such cases, post-hoc methods can come in handy, as they provide a more user-friendly explanation that enables users to understand the model's decision-making process more clearly. This article presents a series of post-hoc methods for GP-evolved routing policies, providing the basis for the real-world application of GP-evolved routing policies.

While our research primarily focuses on enhancing the interpretability of GP-evolved routing policies for UCARP, the techniques and methodologies we have developed hold promise for tackling a broader spectrum of challenging combinatorial optimization problems (with suitable domain knowledge adaptation). Combinatorial optimization problems often share common characteristics, such as the need for effective yet interpretable solutions, and the trade-off between complexity and performance. Our intrinsic methods, which include simplification techniques and multi-objective optimization algorithms, can be adapted to various combinatorial optimization domains. For instance, problems involving resource allocation, scheduling, or network design could benefit from simplified yet effective solutions. Moreover, our post-hoc interpretability methods, which encom-

pass local and global explanations, are not limited to UCARP but can also shed light on complex decision-making processes in other domains. By applying these techniques to a wider range of combinatorial optimization problems, we can democratise the use of AI-driven solutions, making them more accessible and comprehensible to decision-makers across different industries.

This thesis not only advances the field of GP and its application to solving complex combinatorial optimization problems but also makes significant contributions to the broader AI literature, particularly in the domains of Responsible AI (RAI) and AI ethics. In the context of RAI, our work emphasises the importance of interpretability in AI-driven decision-making, especially in domains where human lives and resources are at stake, such as logistics and supply chain management. By proposing both intrinsic and post-hoc interpretability methods, we provide a framework for AI model transparency and accountability, aligning with the principles of RAI. Furthermore, our research addresses the ethical dimensions of AI by enabling users and decision-makers to understand the decision processes of GP-evolved routing policies. This transparency not only enhances trust in AI systems but also allows for the identification and mitigation of biases or unfair decision-making, promoting fairness and ethical AI practices. Our work underscores the significance of considering not only the performance but also the ethical and social implications of AI solutions, fostering a responsible and ethical AI ecosystem.

6.3 Future Work

6.3.1 Further Improvements to the Proposed Simplification Method

While the proposed method for simplifying GP trees during the evolutionary process is effective, further research can be done to improve its

efficiency and effectiveness. One possible direction is to explore new niching techniques or breeding mechanisms to improve the diversity of the population and the representative archive. Another possible direction is to investigate how the proposed method can be applied to other combinatorial optimization problems beyond UCARP.

6.3.2 Further Improvements to the Proposed Multi-Objective Methods

Our research has revealed that valuable knowledge can be extracted from the final Pareto front. For instance, we noticed that even though two individuals located next to each other on the front may have only minor differences in their genotype, the interpretation and validity of their routing policies can vary significantly. By closely analyzing these differences, we gain a deeper understanding of the internal mechanism of GP-evolved routing policies. To take our research to the next level, we propose using a multi-objective algorithm as a post-hoc method to explain the routing policy above in Pareto front.

6.3.3 Hybrid Approaches

A hybrid approach can be developed by combining the proposed simplification method and multi-objective GP algorithms to evolve more interpretable routing policies. This approach can overcome the trade-off between interpretability and effectiveness by evolving routing policies that are both small and effective. The proposed MOGP algorithms can generate Pareto-optimal solutions that provide a trade-off between multiple objectives, including interpretability and effectiveness. The proposed simplification method can further simplify the GP trees of the evolved routing policies, making them more interpretable. The combination of these two methods can lead to a more interpretable and effective set of routing policies.

6.3.4 Counterfactual Explanations

Counterfactual explanations can be used to explain why a particular decision was made and how it might have been different under different circumstances. They can provide valuable insights into the decision-making process and help to build trust and transparency. In the context of GP-evolved routing policies, counterfactual explanations could be used to explain why a particular policy was evolved and how it might have been different if different parameters or objectives were used. This can help to improve the interpretability of the policies and make them more understandable to end-users. However, it should be noted that developing effective counterfactual explanations can be challenging and requires careful consideration of the relevant factors and assumptions.

6.3.5 Developing Domain-Specific GP-Evolved Routing Policies

Developing GP-evolved routing policies that are specific to a particular domain or audience can be a useful direction for future research. This can involve customizing the GP trees to match the preferences and inclinations of a particular user group or developing domain-specific objectives that are relevant to the domain.

6.3.6 GP-Evolved Routing Policies in Real-World Settings

To apply GP-evolved routing policies in practical settings, it is necessary to investigate how end-users perceive and trust the evolved policies. Future research can explore how end-users interact with GP-evolved routing policies, and how the interpretability of the policies affects end-user adoption and acceptance.

6.3.7 Human in the Loop

Human-in-the-Loop (HITL) can be a direction to consider for future research in the context of improving the interpretability of GP-evolved routing policies for UCARP. HITL refers to the integration of human feedback and decision-making into the machine learning process. In the case of GP-evolved routing policies, HITL can be used to evaluate and refine the evolved policies in terms of their interpretability and usefulness in real-world applications. HITL can be implemented in several ways, such as providing feedback on the performance of evolved policies, ranking policies based on their interpretability, or even actively participating in the evolution process. By incorporating HITL, the interpretability and effectiveness of GP-evolved routing policies can be improved, and they can be tailored to meet the needs of different user groups. However, incorporating HITL into the evolution process can also present some challenges. For example, it can be time-consuming and costly to collect and incorporate human feedback. Also, human biases and preferences can impact the evolution process and lead to suboptimal results. Therefore, future research should consider how to effectively integrate HITL into the evolution process while mitigating these potential challenges. Singh et al. [176] incorporated human expert into AI route planning process and demonstrated that human insight can be used in collaborative planning for resilience.

Bibliography

[1] AL-SAHAF, H., BI, Y., CHEN, Q., LENSEN, A., MEI, Y., SUN, Y., TRAN, B., XUE, B., AND ZHANG, M. A survey on evolutionary machine learning. *J. Roy. Soc. New Zeal. 49*, 2 (2019), 205–228.

[2] ALBA, E., COTTA, C., AND TROYA, J. M. Evolutionary design of fuzzy logic controllers using strongly-typed gp. *Mathware and Soft Computing 6*, 1 (1999), 109–124.

[3] ALFARO-CID, E., MERELO, J., DE VEGA, F. F., ESPARCIA-ALCÁZAR, A. I., AND SHARMAN, K. Bloat control operators and diversity in genetic programming: A comparative study. *Evolutionary Computation 18*, 2 (2010), 305–332.

[4] ALLEN, S., BURKE, E. K., HYDE, M., AND KENDALL, G. Evolving reusable 3d packing heuristics with genetic programming. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (2009), pp. 931–938.

[5] ALONSO, J. M., CORDÓN, O., QUIRIN, A., AND MAGDALENA, L. Analyzing interpretability of fuzzy rule-based systems by means of fuzzy inference-grams. In *World Congress on Soft Computing* (2011).

[6] ALONSO, J. M., AND MAGDALENA, L. Special issue on interpretable fuzzy systems, 2011.

[7] AMPONSAH, S., AND SALHI, S. The investigation of a class of capacitated arc routing problems: The collection of garbage in developing countries. *Waste Management 24*, 7 (2004), 711–721.

[8] ARDEH, M. A., MEI, Y., AND ZHANG, M. Genetic programming hyper-heuristic with knowledge transfer for uncertain capacitated arc routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2019), pp. 334–335.

[9] ARDEH, M. A., MEI, Y., AND ZHANG, M. A novel genetic programming algorithm with knowledge transfer for uncertain capacitated arc routing problem. In *Proceeding of PRICAI* (2019), Springer, pp. 196–200.

[10] ARDEH, M. A., MEI, Y., AND ZHANG, M. Genetic programming with knowledge transfer and guided search for uncertain capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation 26*, 4 (2021), 765–779.

[11] ARDEH, M. A., MEI, Y., ZHANG, M., AND YAO, X. Knowledge transfer genetic programming with auxiliary population for solving uncertain capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation 27*, 2 (2022), 311–325.

[12] ARNALDO, I., O'REILLY, U.-M., AND VEERAMACHANENI, K. Building predictive models via feature synthesis. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation* (2015), pp. 983–990.

[13] ARRIETA, A. B., DÍAZ-RODRÍGUEZ, N., DEL SER, J., BENNETOT, A., TABIK, S., BARBADO, A., GARCÍA, S., GIL-LÓPEZ, S., MOLINA, D., BENJAMINS, R., ET AL. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion 58* (2020), 82–115.

[14] ARULKUMARAN, K., CULLY, A., AND TOGELIUS, J. Alphastar: An evolutionary computation perspective. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2019), pp. 314–315.

[15] BABAEE TIRKOLAEE, E., ALINAGHIAN, M., BAKHSHI SASI, M., AND SEYYED ESFAHANI, M. Solving a robust capacitated arc routing problem using a hybrid simulated annealing algorithm: a waste collection application. *Journal of Industrial Engineering and Management Studies 3*, 1 (2016), 61–76.

[16] BADER-EL-DEN, M., AND POLI, R. Generating sat local-search heuristics using a gp hyper-heuristic framework. In *International Conference on Artificial Evolution (Evolution Artificielle)* (2007), Springer, pp. 37–49.

[17] BADER-EL-DEN, M., POLI, R., AND FATIMA, S. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing 1*, 3 (2009), 205.

[18] BELENGUER, J. M., AND BENAVENT, E. A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research 30*, 5 (2003), 705–728.

[19] BERLANGA, F. J., RIVERA, A., DEL JESÚS, M. J., AND HERRERA, F. Gp-coach: Genetic programming-based learning of compact and accurate fuzzy rule-based classification systems for high-dimensional problems. *Information Sciences 180*, 8 (2010), 1183–1200.

[20] BERNSTEIN, Y., LI, X., CIESIELSKI, V., AND SONG, A. Multiobjective parsimony enforcement for superior generalisation performance. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)* (2004), vol. 1, IEEE, pp. 83–89.

[21] BI, Y., XUE, B., AND ZHANG, M. Genetic programming with a new representation to automatically learn features and evolve ensembles for image classification. *IEEE transactions on cybernetics 51*, 4 (2020), 1769–1783.

[22] BLEULER, S., BADER, J., AND ZITZLER, E. Reducing bloat in gp with multiple objectives. In *Multiobjective Problem Solving from Nature*. Springer, 2008, pp. 177–200.

[23] BLEULER, S., BRACK, M., THIELE, L., AND ZITZLER, E. Multiobjective genetic programming: Reducing bloat using spea2. In *Proceeding of IEEE Congress on Evolutionary Computation (CEC)* (2001), IEEE, pp. 536–543.

[24] BRANDÃO, J., AND EGLESE, R. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research 35*, 4 (2008), 1112–1126.

[25] BRANKE, J., HILDEBRANDT, T., AND SCHOLZ-REITER, B. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary computation 23*, 2 (2015), 249–277.

[26] BRANKE, J., NGUYEN, S., PICKARDT, C. W., AND ZHANG, M. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation 20*, 1 (2015), 110–124.

[27] BURKE, E. K., GENDREAU, M., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND QU, R. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society 64*, 12 (2013), 1695–1724.

[28] BURKE, E. K., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. R. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.

[29] BURKE, E. K., HYDE, M., KENDALL, G., AND WOODWARD, J. A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. *IEEE Transactions on Evolutionary Computation 14*, 6 (2010), 942–958.

[30] BURKE, E. K., HYDE, M. R., AND KENDALL, G. Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature-PPSN IX*. Springer, 2006, pp. 860–869.

[31] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., OZCAN, E., AND WOODWARD, J. R. Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence*. Springer, 2009, pp. 177–201.

[32] BURKE, E. K., HYDE, M. R., KENDALL, G., AND WOODWARD, J. Automating the packing heuristic design process with genetic programming. *Evolutionary computation 20*, 1 (2012), 63–89.

[33] CANO, A., ZAFRA, A., AND VENTURA, S. An interpretable classification rule mining algorithm. *Information Sciences 240* (2013), 1–20.

[34] CARTON, S., HELSBY, J., JOSEPH, K., MAHMUD, A., PARK, Y., WALSH, J., CODY, C., PATTERSON, C. E., HAYNES, L., AND GHANI, R. Identifying police officers at risk of adverse events. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (2016), pp. 67–76.

[35] CARVALHO, D. V., PEREIRA, E. M., AND CARDOSO, J. S. Machine learning interpretability: A survey on methods and metrics. *Electronics 8*, 8 (2019), 832.

[36] CAVARETTA, M. J., AND CHELLAPILLA, K. Data mining using genetic programming: The implications of parsimony on generalization error. In *Proceedings of the Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)* (1999), vol. 2, IEEE, pp. 1330–1337.

[37] CHANG, J., GERRISH, S., WANG, C., BOYD-GRABER, J. L., AND BLEI, D. M. Reading tea leaves: How humans interpret topic models. In *Advances in neural information processing systems* (2009), pp. 288–296.

[38] CHEN, P.-C., KENDALL, G., AND BERGHE, G. V. An ant based hyper-heuristic for the travelling tournament problem. In *2007 IEEE Symposium on Computational Intelligence in Scheduling* (2007), IEEE, pp. 19–26.

[39] CHEN, Q., XUE, B., AND ZHANG, M. Rademacher complexity for enhancing the generalization of genetic programming for symbolic regression. *IEEE Transactions on Cybernetics 52*, 4 (2022), 2382–2395.

[40] CHEN, X., BAI, R., QU, R., AND DONG, H. Cooperative double-layer genetic programming hyper-heuristic for online container terminal truck dispatching. *IEEE Transactions on Evolutionary Computation* (2022), 10.1109/TEVC.2022.3209985.

[41] CHRYSSOLOURIS, G., AND SUBRAMANIAM, V. Dynamic scheduling of manufacturing job shops using genetic algorithms. *Journal of Intelligent Manufacturing 12*, 3 (2001), 281–293.

[42] COELLO COELLO, C. A., AND REYES SIERRA, M. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In *MICAI 2004: Advances in Artificial Intelligence: Third Mexican International Conference on Artificial Intelligence, Mexico City, Mexico, April 26-30, 2004. Proceedings 3* (2004), Springer, pp. 688–697.

[43] COWLING, P., KENDALL, G., AND HAN, L. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC'02.* (2002), vol. 2, IEEE, pp. 1185–1190.

[44] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In *International Conference on the Practice and Theory of Automated Timetabling* (2000), Springer, pp. 176–190.

[45] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the 4th metaheuristic international conference* (2001), vol. 1101, Citeseer, pp. 127–131.

[46] COWLING, P., KENDALL, G., AND SOUBEIGA, E. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Workshops on Applications of Evolutionary Computation* (2002), Springer, pp. 1–10.

[47] DAIDA, J. M., HILSS, A. M., WARD, D. J., AND LONG, S. L. Visualizing tree structures in genetic programming. *Genetic Programming and Evolvable Machines 6*, 1 (2005), 79–110.

[48] DE JONG, E. D., AND POLLACK, J. B. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines 4*, 3 (2003), 211–233.

[49] DE JONG, E. D., WATSON, R. A., AND POLLACK, J. B. Reducing bloat and promoting diversity using multi-objective methods. In *Proceeding of GECCO* (2001), Morgan Kaufmann Publishers Inc., pp. 11–18.

[50] DEB, K., AGRAWAL, S., PRATAP, A., AND MEYARIVAN, T. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *Proceeding of PPSN* (2000), Springer, pp. 849–858.

[51] DIJKSTRA, E. W., ET AL. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[52] DOERNER, K. F., HARTL, R. F., MANIEZZO, V., AND REIMANN, M. Applying ant colony optimization to the capacitated arc routing problem. In *International Workshop on Ant Colony Optimization and Swarm Intelligence* (2004), Springer, pp. 420–421.

[53] DOMINGOS, P. A few useful things to know about machine learning. *Communications of the ACM 55*, 10 (2012), 78–87.

[54] DOSHI-VELEZ, F., AND KIM, B. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608* (2017).

[55] DOSHI-VELEZ, F., WALLACE, B., AND ADAMS, R. Graph-sparse lda: a topic model with structured sparsity. *arXiv preprint arXiv:1410.4510* (2014).

[56] DOŠILOVIĆ, F. K., BRČIĆ, M., AND HLUPIĆ, N. Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)* (2018), IEEE, pp. 210–215.

[57] EBERHART, R. C., AND SHI, Y. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)* (2000), vol. 1, IEEE, pp. 84–88.

[58] EKÁRT, A., AND NEMETH, S. Z. Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines 2*, 1 (2001), 61–73.

[59] EMILIANO, I. Heuristic reasoning: Studies in applied philosophy, epistemology and rational ethics, 2015.

[60] EVANS, B. P., XUE, B., AND ZHANG, M. What's inside the black-box? a genetic programming method for interpreting complex ma-

chine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2019), pp. 1012–1020.

[61] FENG, L., ONG, Y.-S., TAN, A.-H., AND TSANG, I. W. Memes as building blocks: a case study on evolutionary optimization+ transfer learning for routing problems. *Memetic Computing 7*, 3 (2015), 159–180.

[62] FENG, L., ONG, Y.-S., TSANG, I. W.-H., AND TAN, A.-H. An evolutionary search paradigm that learns with past experiences. In *Proceeding of IEEE Congress on Evolutionary Computation (CEC)* (2012), IEEE, pp. 1–8.

[63] FERREIRA, L. A., GUIMARÃES, F. G., AND SILVA, R. Applying genetic programming to improve interpretability in machine learning models. *arXiv preprint arXiv:2005.09512* (2020).

[64] FLEURY, G., LACOMME, P., PRINS, C., AND RAMDANE-CHÉRIF, W. Improving robustness of solutions to arc routing problems. *Journal of the operational research society 56*, 5 (2005), 526–538.

[65] FOSTER, M. A., ET AL. *The program structure of genetic programming trees*. PhD thesis, Citeseer, 2005.

[66] FUKUNAGA, A. S. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary computation 16*, 1 (2008), 31–61.

[67] GAO, G., MEI, Y., XIN, B., JIA, Y.-H., AND BROWNE, W. N. Automated coordination strategy design using genetic programming for dynamic multipoint dynamic aggregation. *IEEE Transactions on Cybernetics* (2021).

[68] GEIGER, C. D., UZSOY, R., AND AYTUĞ, H. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling 9*, 1 (2006), 7–34.

[69] GILPIN, L. H., BAU, D., YUAN, B. Z., BAJWA, A., SPECTER, M., AND KAGAL, L. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)* (2018), IEEE, pp. 80–89.

[70] GOLDEN, B. L., DEARMON, J. S., AND BAKER, E. K. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research 10*, 1 (1983), 47–59.

[71] GOLDEN, B. L., RAGHAVAN, S., WASIL, E. A., ET AL. *The vehicle routing problem: latest advances and new challenges*, vol. 43. Springer, 2008.

[72] GOLDEN, B. L., AND WONG, R. T. Capacitated arc routing problems. *Networks 11*, 3 (1981), 305–315.

[73] GOODMAN, B., AND FLAXMAN, S. European union regulations on algorithmic decision-making and a "right to explanation". *AI magazine 38*, 3 (2017), 50–57.

[74] HAN, L., KENDALL, G., AND COWLING, P. An adaptive length chromosome hyper-heuristic genetic algorithm for a trainer scheduling problem. In *Recent Advances In Simulated Evolution And Learning*. World Scientific, 2004, pp. 506–525.

[75] HANDA, H., CHAPMAN, L., AND YAO, X. Robust route optimization for gritting/salting trucks: a cercia experience. *IEEE Computational Intelligence Magazine 1*, 1 (2006), 6–9.

[76] HEIN, D., UDLUFT, S., AND RUNKLER, T. A. Generating interpretable fuzzy controllers using particle swarm optimization and genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2018), pp. 1268–1275.

[77] HEIN, D., UDLUFT, S., AND RUNKLER, T. A. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence 76* (2018), 158–169.

[78] HERTZ, A., LAPORTE, G., AND MITTAZ, M. A tabu search heuristic for the capacitated arc routing problem. *Operations research 48*, 1 (2000), 129–135.

[79] HILDEBRANDT, T., AND BRANKE, J. On using surrogates with genetic programming. *Evolutionary computation 23*, 3 (2015), 343–367.

[80] HILDEBRANDT, T., HEGER, J., AND SCHOLZ-REITER, B. Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In *Proceedings of Genetic and Evolutionary Computation Conference* (2010), ACM, pp. 257–264.

[81] HOOPER, D. C., AND FLANN, N. S. Improving the accuracy and robustness of genetic programming through expression simplification. In *Proceeding of the conference on genetic programming* (1996), pp. 428–428.

[82] HU, T. Can genetic programming perform explainable machine learning for bioinformatics? In *Genetic Programming Theory and Practice XVII*. Springer, 2020, pp. 63–77.

[83] HU, T. Genetic programming for interpretable and explainable machine learning. In *Genetic Programming Theory and Practice XIX*. Springer, 2023, pp. 81–90.

[84] HUANG, T., GONG, Y.-J., KWONG, S., WANG, H., AND ZHANG, J. A niching memetic algorithm for multi-solution traveling salesman problem. *IEEE Transactions on Evolutionary Computation* (2019).

[85] HUANG, Z., MEI, Y., AND ZHANG, M. Investigation of Linear Genetic Programming for Dynamic Job Shop Scheduling. In *Proceed-*

ings of IEEE Symposium Series on Computational Intelligence (dec 2021), IEEE, pp. 1–8.

[86] HUGHES, M., GOERIGK, M., AND DOKKA, T. Automatic generation of algorithms for robust optimisation problems using grammar-guided genetic programming. *Computers & Operations Research 133* (2021), 105364.

[87] HUNT, R., JOHNSTON, M., AND ZHANG, M. Evolving" less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (2014), pp. 927–934.

[88] HUNT, R. J., JOHNSTON, M. R., AND ZHANG, M. *Evolving dispatching rules with greater understandability for dynamic job shop scheduling*. Citeseer, 2016.

[89] IBA, H., DE GARIS, H., AND SATO, T. Genetic programming using a minimum description length principle. *Advances in genetic programming 1* (1994), 265–284.

[90] IKEDA, K., KITA, H., AND KOBAYASHI, S. Failure of pareto-based moeas: Does non-dominated really mean near to optimal? In *Proceedings of the IEEE Congress on Evolutionary Computation* (2001), vol. 2, IEEE, pp. 957–962.

[91] JACOBSEN-GROCOTT, J., MEI, Y., CHEN, G., AND ZHANG, M. Evolving heuristics for dynamic vehicle routing with time windows using genetic programming. In *Proceeding of IEEE Congress on Evolutionary Computation (CEC)* (2017), IEEE, pp. 1948–1955.

[92] JACOVI, A., MARASOVIĆ, A., MILLER, T., AND GOLDBERG, Y. Formalizing trust in artificial intelligence: Prerequisites, causes and goals of human trust in ai. In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency* (2021), pp. 624–635.

[93] JAMES, G., WITTEN, D., HASTIE, T., AND TIBSHIRANI, R. *An introduction to statistical learning*, vol. 112. Springer, 2013.

[94] JIA, Y.-H., MEI, Y., AND ZHANG, M. A bilevel ant colony optimization algorithm for capacitated electric vehicle routing problem. *IEEE Transactions on Cybernetics* (2021).

[95] JIA, Y.-H., MEI, Y., AND ZHANG, M. Learning heuristics with different representations for stochastic routing. *IEEE Transactions on Cybernetics* (2022), 1–15.

[96] JOBIN, A., IENCA, M., AND VAYENA, E. The global landscape of ai ethics guidelines. *Nature machine intelligence 1*, 9 (2019), 389–399.

[97] KAWAGUCHI, K. Deep learning without poor local minima. In *Advances in neural information processing systems* (2016), pp. 586–594.

[98] KEIJZER, M., AND BABOVIC, V. Dimensionally aware genetic programming. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2* (1999), Morgan Kaufmann Publishers Inc., pp. 1069–1076.

[99] KELLER, R. E., AND POLI, R. Linear genetic programming of parsimonious metaheuristics. In *2007 IEEE Congress on Evolutionary Computation* (2007), IEEE, pp. 4508–4515.

[100] KENNEDY, J., AND EBERHART, R. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks* (1995), vol. 4, IEEE, pp. 1942–1948.

[101] KHOUADJIA, M. R., SARASOLA, B., ALBA, E., JOURDAN, L., AND TALBI, E.-G. A comparative study between dynamic adapted pso and vns for the vehicle routing problem with dynamic requests. *Applied Soft Computing 12*, 4 (2012), 1426–1439.

[102] KIM, B. *Interactive and interpretable machine learning models for human machine collaboration*. PhD thesis, Massachusetts Institute of Technology, 2015.

[103] KIM, B., KHANNA, R., AND KOYEJO, O. O. Examples are not enough, learn to criticize! criticism for interpretability. In *Advances in neural information processing systems* (2016), pp. 2280–2288.

[104] KIM, B., RUDIN, C., AND SHAH, J. A. The bayesian case model: A generative approach for case-based reasoning and prototype classification. In *Advances in neural information processing systems* (2014), pp. 1952–1960.

[105] KINZETT, D., JOHNSTON, M., AND ZHANG, M. How online simplification affects building blocks in genetic programming. In *Proceeding of GECCO* (2009), pp. 979–986.

[106] KINZETT, D., JOHNSTON, M., AND ZHANG, M. Numerical simplification for bloat control and analysis of building blocks in genetic programming. *Evolutionary Intelligence 2*, 4 (2009), 151.

[107] KINZETT, D., ZHANG, M., AND JOHNSTON, M. Using numerical simplification to control bloat in genetic programming. In *Proc.of SEAL* (2008), Springer, pp. 493–502.

[108] KINZETT, D., ZHANG, M., AND JOHNSTON, M. Investigation of simplification threshold and noise level of input data in numerical simplification of genetic programs. In *Proceeding of IEEE Congress on Evolutionary Computation (CEC)* (2010), IEEE, pp. 1–8.

[109] KOZA, J. R., AND KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992.

[110] KRENING, S., HARRISON, B., FEIGH, K. M., ISBELL, C. L., RIEDL, M., AND THOMAZ, A. Learning from explanations using sentiment

and advice in rl. *IEEE Transactions on Cognitive and Developmental Systems 9*, 1 (2016), 44–55.

[111] KRISHNAN, M. Against interpretability: a critical examination of the interpretability problem in machine learning. *Philosophy & Technology* (2019), 1–16.

[112] LACOMME, P., PRINS, C., AND RAMDANE-CHÉRIF, W. A genetic algorithm for the capacitated arc routing problem and its extensions. In *Workshops on Applications of Evolutionary Computation* (2001), Springer, pp. 473–483.

[113] LACOMME, P., PRINS, C., AND RAMDANE-CHERIF, W. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research 131*, 1-4 (2004), 159–185.

[114] LANGDON, W. B. Size fair and homologous tree genetic programming crossovers. *Genetic programming and evolvable machines 1*, 1/2 (2000), 95–119.

[115] LANGDON, W. B., SOULE, T., POLI, R., AND FOSTER, J. A. The evolution of size and shape. *Advances in genetic programming 3* (1999), 163–190.

[116] LE, T., MILLER, T., SINGH, R., AND SONENBERG, L. Improving model understanding and trust with counterfactual explanations of model confidence. *arXiv preprint arXiv:2206.02790* (2022).

[117] LE, T., MILLER, T., SINGH, R., AND SONENBERG, L. Explaining model confidence using counterfactuals. *arXiv preprint arXiv:2303.05729* (2023).

[118] LENSEN, A., XUE, B., AND ZHANG, M. Genetic programming for evolving a front of interpretable models for data visualization. *IEEE Transactions on Cybernetics* (2020).

[119] LI, H. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems 94*, 10 (2011), 1854–1862.

[120] LIPTON, Z. C. The mythos of model interpretability. *Queue 16*, 3 (2018), 31–57.

[121] LIU, J., TANG, K., AND YAO, X. Robust optimization in uncertain capacitated arc routing problems: Progresses and perspectives. *IEEE Computational Intelligence Magazine 16*, 1 (2021), 63–82.

[122] LIU, Y., MEI, Y., ZHANG, M., AND ZHANG, Z. Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem. In *Proceeding of GECCO* (2017), ACM, pp. 290–297.

[123] LIU, Y., MEI, Y., ZHANG, M., AND ZHANG, Z. A predictive-reactive approach with genetic programming and cooperative co-evolution for uncertain capacitated arc routing problem. *Evolutionary Computation* (2019).

[124] LIU, Y., MEI, Y., ZHANG, M., AND ZHANG, Z. A predictive-reactive approach with genetic programming and cooperative coevolution for the uncertain capacitated arc routing problem. *Evolutionary computation 28*, 2 (2020), 289–316.

[125] LIU, Y., WANG, J., ZHAO, J., AND LI, X. Route stability in the uncertain capacitated arc routing problem. *Frontiers in Energy Research* (2022), 965.

[126] LIU, Y., WANG, S., AND LI, X. A new cooperative recourse strategy for emergency material allocation in uncertain environments. *Frontiers in Physics 10* (2022), 26.

[127] LOU, Y., CARUANA, R., AND GEHRKE, J. Intelligible models for classification and regression. In *Proceedings of the 18th ACM SIGKDD*

international conference on Knowledge discovery and data mining (2012), pp. 150–158.

[128] LUKE, S., AND PANAIT, L. Fighting bloat with nonparametric parsimony pressure. In *Proceedings of International Conference on Parallel Problem Solving from Nature* (2002), Springer, pp. 411–421.

[129] LUKE, S., AND PANAIT, L. Lexicographic parsimony pressure. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* (2002), pp. 829–836.

[130] LUKE, S., AND PANAIT, L. A comparison of bloat control methods for genetic programming. *Evolutionary Computation 14*, 3 (2006), 309–344.

[131] LUKE, S., PANAIT, L., BALAN, G., PAUS, S., SKOLICKI, Z., BASSETT, J., HUBLEY, R., AND CHIRCOP, A. Ecj: A java-based evolutionary computation research system. *Downloadable versions and documentation can be found at the following url: http://cs. gmu. edu/eclab/projects/ecj* (2006).

[132] MAATEN, L. V. D., AND HINTON, G. Visualizing data using t-sne. *Journal of machine learning research 9*, Nov (2008), 2579–2605.

[133] MACLACHLAN, J., MEI, Y., BRANKE, J., AND ZHANG, M. An improved genetic programming hyper-heuristic for the uncertain capacitated arc routing problem. In *AJCAI* (2018), Springer, pp. 432–444.

[134] MACLACHLAN, J., MEI, Y., BRANKE, J., AND ZHANG, M. Genetic programming hyper-heuristics with vehicle collaboration for uncertain capacitated arc routing problems. *Evolutionary computation* (2019), 1–29.

[135] MACLACHLAN, J., MEI, Y., BRANKE, J., AND ZHANG, M. Genetic programming hyper-heuristics with vehicle collaboration for uncertain capacitated arc routing problems. *Evolutionary computation 28*, 4 (2020), 563–593.

[136] MAHFOUD, S. W. *Niching methods for genetic algorithms*. PhD thesis, Citeseer, 1995.

[137] MCAULEY, J., AND LESKOVEC, J. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems* (2013), pp. 165–172.

[138] MCPHEE, N. F., CASALE, M. M., FINZEL, M., HELMUTH, T., AND SPECTOR, L. Visualizing genetic programming ancestries using graph databases. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2017), pp. 245–246.

[139] MEI, Y., CHEN, Q., LENSEN, A., XUE, B., AND ZHANG, M. Explainable artificial intelligence by genetic programming: A survey. *IEEE Transactions on Evolutionary Computation* (2023), DOI:10.1109/TEVC.2022.3225509.

[140] MEI, Y., NGUYEN, S., AND ZHANG, M. Constrained dimensionally aware genetic programming for evolving interpretable dispatching rules in dynamic job shop scheduling. In *Proceeding of SEAL* (2017), Springer, pp. 435–447.

[141] MEI, Y., TANG, K., AND YAO, X. Capacitated arc routing problem in uncertain environments. In *Proceeding of IEEE Congress on Evolutionary Computation (CEC)* (2010), IEEE, pp. 1–8.

[142] MEI, Y., TANG, K., AND YAO, X. Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation 15*, 2 (2011), 151–165.

[143] MEI, Y., AND ZHANG, M. Genetic programming hyper-heuristic for multi-vehicle uncertain capacitated arc routing problem. In *Proceeding of GECCO* (2018), ACM, pp. 141–142.

[144] MILLER, T. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence 267* (2019), 1–38.

[145] MILLER, T. Are we measuring trust correctly in explainability, interpretability, and transparency research? *arXiv preprint arXiv:2209.00651* (2022).

[146] MOHRI, M., ROSTAMIZADEH, A., AND TALWALKAR, A. *Foundations of machine learning*. MIT press, 2018.

[147] MOLNAR, C. *Interpretable Machine Learning*. Lulu. com, 2020.

[148] MORDVINTSEV, A., OLAH, C., AND TYKA, M. Inceptionism: Going deeper into neural networks. *Google Research Blog* (2015).

[149] NGUYEN, H. B., XUE, B., ANDREAE, P., AND ZHANG, M. Particle swarm optimisation with genetic operators for feature selection. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (2017), IEEE, pp. 286–293.

[150] NGUYEN, S., MEI, Y., AND ZHANG, M. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems 3*, 1 (2017), 41–66.

[151] NGUYEN, S., ZHANG, M., ALAHAKOON, D., AND TAN, K. C. Visualizing the evolution of computer programs for genetic programming [research frontier]. *IEEE Computational Intelligence Magazine 13*, 4 (2018), 77–94.

[152] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Automatic design of scheduling policies for dynamic multi-objective job

shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation 18*, 2 (2013), 193–208.

[153] NORDIN, P., BANZHAF, W., ET AL. Complexity compression and evolution. In *ICGA* (1995), pp. 310–317.

[154] NORDIN, P., BANZHAF, W., AND FRANCONE, F. D. Efficient evolution of machine code for cisc architectures using instruction blocks and homologous crossover. *Advances in genetic programming 3* (1999), 275–299.

[155] OWEN, T. Heuristics: Intelligent search strategies for computer problem solving by judea pearl addison-wesley publishing company, massachusetts, usa, 101985 (£ 43.95). *Robotica 6*, 2 (1988), 165–165.

[156] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND ÉDOUARD DUCHESNAY. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research 12*, 85 (2011), 2825–2830.

[157] POHLHEIM, H. Visualization of evolutionary algorithms-set of standard techniques and multidimensional visualization. In *Proceedings of the Genetic and Evolutionary Computation Conference* (1999), vol. 1, San Francisco, CA., pp. 533–540.

[158] POLI, R., KENNEDY, J., AND BLACKWELL, T. Particle swarm optimization. *Swarm intelligence 1*, 1 (2007), 33–57.

[159] POLI, R., AND LANGDON, W. B. Genetic programming with one-point crossover. *Soft Computing in Engineering Design and Manufacturing* (1997), 180–189.

[160] POLI, R., AND LANGDON, W. B. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation 6*, 3 (1998), 231–252.

[161] POLI, R., LANGDON, W. B., AND MCPHEE, N. F. A field guide to genetic programming. *Published via http://lulu. com and freely available at http://www. gp-field-guide. org. uk.(With contributions by JR Koza)* (2008).

[162] POLI, R., MCPHEE, N. F., AND VANNESCHI, L. Elitism reduces bloat in genetic programming. In *Proceeding of GECCO* (2008), Citeseer, pp. 1343–1344.

[163] RACHLIN, H. Rational thought and rational behavior: A review of bounded rationality: The adaptive toolbox. *Journal of the Experimental Analysis of Behavior 79*, 3 (2003), 409–412.

[164] REBALA, G., RAVI, A., AND CHURIWALA, S. *An introduction to machine learning*. Springer, 2019.

[165] RIBEIRO, M. T., SINGH, S., AND GUESTRIN, C. Why should i trust you? explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (2016), pp. 1135–1144.

[166] RITZINGER, U., PUCHINGER, J., AND HARTL, R. F. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research 54*, 1 (2016), 215–231.

[167] ROSS, P., SCHULENBURG, S., MARÍN-BLÄZQUEZ, J. G., AND HART, E. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* (2002), pp. 942–948.

[168] RUDIN, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence 1*, 5 (2019), 206–215.

[169] SAMEK, W., WIEGAND, T., AND MÜLLER, K.-R. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296* (2017).

[170] SECOMANDI, N., AND MARGOT, F. Reoptimization approaches for the vehicle-routing problem with stochastic demands. *Operations research 57*, 1 (2009), 214–230.

[171] SILVA, S., AND ALMEIDA, J. Dynamic maximum tree depth: A simple technique for avoiding bloat in tree-based gp. In *Genetic and Evolutionary Computation—GECCO 2003: Genetic and Evolutionary Computation Conference Chicago, IL, USA, July 12–16, 2003 Proceedings, Part II* (2003), Springer, pp. 1776–1787.

[172] SILVA, S., AND COSTA, E. Dynamic limits for bloat control: Variations on size and depth. In *Genetic and Evolutionary Computation–GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004. Proceedings, Part II* (2004), Springer, pp. 666–677.

[173] SILVA, S., AND COSTA, E. Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines 10* (2009), 141–179.

[174] SILVER, D., HUANG, A., MADDISON, C. J., GUEZ, A., SIFRE, L., VAN DEN DRIESSCHE, G., SCHRITTWIESER, J., ANTONOGLOU, I., PANNEERSHELVAM, V., LANCTOT, M., ET AL. Mastering the game of go with deep neural networks and tree search. *nature 529*, 7587 (2016), 484–489.

[175] SIMONYAN, K., VEDALDI, A., AND ZISSERMAN, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034* (2013).

[176] SINGH, R., MILLER, T., AND REID, D. Collaborative human-agent planning for resilience. In *Coordination, Organizations, Institutions, Norms, and Ethics for Governance of Multi-Agent Systems XIV: International Workshop, COINE 2021, London, UK, May 3, 2021, Revised Selected Papers* (2022), Springer, pp. 28–43.

[177] SONG, A., CHEN, D., AND ZHANG, M. Contribution based bloat control in genetic programming. In *Proceeding of IEEE Congress on Evolutionary Computation (CEC)* (2010), IEEE, pp. 1–8.

[178] TAN, B., MA, H., AND MEI, Y. A hybrid genetic programming hyper-heuristic approach for online two-level resource allocation in container-based clouds. In *Proceeding of IEEE Congress on Evolutionary Computation (CEC)* (2019), IEEE, pp. 2681–2688.

[179] TAN, B., MA, H., MEI, Y., AND ZHANG, M. Evolutionary multi-objective optimization for web service location allocation problem. *IEEE Transactions on Services Computing* (2018).

[180] TAN, B., MA, H., MEI, Y., AND ZHANG, M. A cooperative coevolution genetic programming hyper-heuristics approach for on-line resource allocation in container-based clouds. *IEEE Transactions on Cloud Computing 10*, 3 (2020), 1500–1514.

[181] TANG, K., MEI, Y., AND YAO, X. Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation 13*, 5 (2009), 1151–1166.

[182] TIBSHIRANI, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological) 58*, 1 (1996), 267–288.

[183] TJOA, E., AND GUAN, C. A survey on explainable artificial intelligence (xai): towards medical xai. *arXiv preprint arXiv:1907.07374* (2019).

[184] TONG, H., MINKU, L., MENZEL, S., SENDHOFF, B., AND YAO, X. A hybrid local search framework for the dynamic capacitated arc routing problem. In *Proceeding of GECCO* (2021), pp. 139–140.

[185] TONG, H., MINKU, L. L., MENZEL, S., SENDHOFF, B., AND YAO, X. Towards novel meta-heuristic algorithms for dynamic capacitated arc routing problems. In *Proceeding of PPSN* (2020), Springer, pp. 428–440.

[186] TONG, H., MINKU, L. L., MENZEL, S., SENDHOFF, B., AND YAO, X. A novel generalised meta-heuristic framework for dynamic capacitated arc routing problems. *arXiv preprint arXiv:2104.06585* (2021).

[187] TSUTSUI, S., PELIKAN, M., AND GOLDBERG, D. E. Using edge histogram models to solve permutation problems with probabilistic model-building genetic algorithms. *IlliGAL Report 2003022* (2003).

[188] TVERSKY, A., AND KAHNEMAN, D. Judgment under uncertainty: Heuristics and biases. *science 185*, 4157 (1974), 1124–1131.

[189] ULNICANE, I. Artificial intelligence in the european union: Policy, ethics and regulation. In *The Routledge handbook of European integrations*. Taylor & Francis, 2022.

[190] ULUSOY, G. The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research 22*, 3 (1985), 329–337.

[191] VAN DEN BERGH, F., AND ENGELBRECHT, A. P. A study of particle swarm optimization particle trajectories. *Information sciences 176*, 8 (2006), 937–971.

[192] WALKER, J. D., OCHOA, G., GENDREAU, M., AND BURKE, E. K. Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. In *International conference on learning and intelligent optimization* (2012), Springer, pp. 265–276.

[193] WANG, J., TANG, K., LOZANO, J. A., AND 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 (2016), 96–109.

[194] WANG, J., TANG, K., AND YAO, X. A memetic algorithm for uncertain capacitated arc routing problems. In *2013 IEEE Workshop on Memetic Computing (MC)* (2013), IEEE, pp. 72–79.

[195] WANG, S., MEI, Y., PARK, J., AND ZHANG, M. Evolving ensembles of routing policies using genetic programming for uncertain capacitated arc routing problem. In *Proceeding of SSCI* (2019), IEEE, pp. 1628–1635.

[196] WANG, S., MEI, Y., PARK, J., AND ZHANG, M. A two-stage genetic programming hyper-heuristic for uncertain capacitated arc routing problem. In *Proceeding of SSCI* (2019), IEEE, pp. 1606–1613.

[197] WANG, S., MEI, Y., AND ZHANG, M. Novel ensemble genetic programming hyper-heuristics for uncertain capacitated arc routing problem. In *Proceeding of GECCO* (2019), pp. 1093–1101.

[198] WANG, S., MEI, Y., AND ZHANG, M. A multi-objective genetic programming approach with self-adaptive α dominance to uncertain capacitated arc routing problem. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)* (2021), IEEE, pp. 636–643.

[199] WANG, S., MEI, Y., AND ZHANG, M. Two-stage multi-objective genetic programming with archive for uncertain capacitated arc rout-

ing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2021), pp. 287–295.

[200] WANG, S., MEI, Y., AND ZHANG, M. Local ranking explanation for genetic programming evolved routing policies for uncertain capacitated arc routing problems. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2022), pp. 314–322.

[201] WANG, S., MEI, Y., ZHANG, M., AND YAO, X. Genetic programming with niching for uncertain capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation 26*, 1 (2022), 73–87.

[202] WEICK, K. Social psychology. *Psyccritiques 22*, 5 (1977).

[203] WEISE, T., DEVERT, A., AND TANG, K. A developmental solution to (dynamic) capacitated arc routing problems using genetic programming. In *Proceeding of GECCO* (2012), ACM, pp. 831–838.

[204] WØHLK, S. A decade of capacitated arc routing. In *The vehicle routing problem: latest advances and new challenges*. Springer, 2008, pp. 29–48.

[205] WONG, P., AND ZHANG, M. Algebraic simplification of gp programs during evolution. In *Proceeding of GECCO* (2006), pp. 927–934.

[206] XING, L.-N., ROHLFSHAGEN, P., CHEN, Y.-W., AND YAO, X. A hybrid ant colony optimization algorithm for the extended capacitated arc routing problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 41*, 4 (2011), 1110–1123.

[207] XU, M., MEI, Y., ZHANG, F., AND ZHANG, M. Genetic programming for dynamic workflow scheduling in fog computing. *IEEE Transactions on Services Computing* (feb 2023), Doi:10.1109/TSC.2023.3249160.

[208] XU, M., MEI, Y., ZHANG, F., AND ZHANG, M. Genetic programming with lexicase selection for large-scale dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation* (feb 2023), Doi:10.1109/TEVC.2023.3244607.

[209] YEUNG, K. Recommendation of the council on artificial intelligence (oecd). *International legal materials 59*, 1 (2020), 27–34.

[210] YSKA, D., MEI, Y., AND ZHANG, M. Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling. In *European Conference on Genetic Programming* (2018), Springer, pp. 306–321.

[211] ZEILER, M. D., AND FERGUS, R. Visualizing and understanding convolutional networks. In *European conference on computer vision* (2014), Springer, pp. 818–833.

[212] ZHANG, B.-T., AND MÜHLENBEIN, H. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation 3*, 1 (1995), 17–38.

[213] ZHANG, F., MEI, Y., NGUYEN, S., TAN, K. C., AND ZHANG, M. Multitask genetic programming-based generative hyperheuristics: A case study in dynamic scheduling. *IEEE Transactions on Cybernetics* (2021), 1–14.

[214] ZHANG, F., MEI, Y., NGUYEN, S., AND ZHANG, M. Correlation coefficient based recombinative guidance for genetic programming hyper-heuristics in dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation 25*, 3 (2021), 552–566.

[215] ZHANG, F., MEI, Y., NGUYEN, S., ZHANG, M., AND TAN, K. C. Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation 25*, 4 (2021), 651–665.

[216] ZHANG, F., MEI, Y., NGUYEN, S., ZHANG, M., AND TAN, K. C. Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation 25*, 4 (2021), 651–665.

[217] ZHANG, F., MEI, Y., AND ZHANG, M. Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics. In *Proceeding of IEEE Congress on Evolutionary Computation (CEC)* (2019), IEEE, pp. 1366–1373.

[218] ZHANG, M., AND WONG, P. Genetic programming for medical classification: a program simplification approach. *Genetic Programming and Evolvable Machines 9*, 3 (2008), 229–255.

[219] ZHANG, M., WONG, P., AND QIAN, D. Online program simplification in genetic programming. In *Proceeding of SEAL* (2006), Springer, pp. 592–600.

[220] ZHANG, M., ZHANG, Y., AND SMART, W. Program simplification in genetic programming for object classification. In *Proceeding of KES* (2005), Springer, pp. 988–996.

[221] ZHANG, Q., AND LI, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation 11*, 6 (2007), 712–731.

[222] ZHAO, H. A multi-objective genetic programming approach to developing pareto optimal decision trees. *Decision Support Systems 43*, 3 (2007), 809–826.

[223] ZHU, J., LIAPIS, A., RISI, S., BIDARRA, R., AND YOUNGBLOOD, G. M. Explainable ai for designers: A human-centered perspective on mixed-initiative co-creation. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)* (2018), IEEE, p. Doi:10.1109/CIG.2018.8490433.

[224] ZITZLER, E., LAUMANNS, M., AND THIELE, L. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report 103* (2001).

[225] ZITZLER, E., AND THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation 3*, 4 (1999), 257–271.