University of Nottingham
UK | CHINA | MALAYSIA

# Automated Design of Population-Based Algorithms: A Case Study in Vehicle Routing


**Wenjie Yi**

20187866

First Supervisor: Rong Qu
Second Supervisor: Dario Landa Silva


Computational Optimisation & Learning Lab

Thesis submitted to the University of Nottingham
for the degree of Doctor of Philosophy

January 27, 2023

# *Abstract*

Metaheuristics have been extensively studied to solve constraint combinatorial optimisation problems such as vehicle routing problems. Most existing algorithms require considerable human effort and different kinds of expertise in algorithm design. These manually designed algorithms are discarded after solving the specific instances. It is highly desirable to automate the design of search algorithms, thus to solve problem instances effectively with less human intervention.

This thesis develops a novel general search framework to formulate in a unified way a range of population-based algorithms. Within this framework, generic algorithmic components such as selection heuristics on the population and evolution operators are defined, and can be composed using machine learning to generate effective search algorithms automatically. This unified framework aims to serve as the basis to analyse algorithmic components, generating effective search algorithms for complex combinatorial optimisation problems. Three key research issues within the general search framework are identified: automated design of evolution operators, of selection heuristics, and of both.

To accurately describe the search space of algorithm design as a new task for machine learning, this thesis identifies new key features, namely search-dependent and instance-dependent features. These features are identified to assist effective algorithm design. With these features, a set of state-of-the-art reinforcement learning techniques, such as deep Q-network based and proximal policy optimisation based models and maximum entropy mechanisms have been developed to intelligently select and combine appropriate evolution operators and selection heuristics during different stages of the optimisation process. The effectiveness and generality of these algorithms automatically designed within the proposed general search framework are validated comprehensively across different capacitated vehicle routing problem with time windows benchmark instances. This thesis contributes to making a key step towards automated algorithm design with a general framework supporting fundamental analysis by effective machine learning.

# *Acknowledgements*

First of all, I would like to express my sincere gratitude to my supervisor Dr. Rong Qu for her invaluable advice, continuous support, and patience during my Ph.D study. Her immense knowledge and plentiful experience have encouraged me throughout my academic research and daily life. I would also like to thank my second supervisor Dr. Dario Landa-Silva for his suggestions on my Ph.D training.

My sincere thanks to many researchers and visiting scholars of the Computational Optimisation and Learning (COL) research group. It is their kind help and support that have made my study and life in the UK a wonderful time.

This Ph.D research would not have been possible without the financial support of the Vice-Chancellor's Scholarship provided by the School of Computer Science, University of Nottingham.

Finally, I would like to thank my family: my parents Mr. Keping Yi and Ms. Xiuzhu Chen, my sister Jieyun Yi, and my brother Xiaomin Yi. Without my family's tremendous understanding and encouragement in the past few years, I would not have been able to continue my studies during difficult time, especially during the COVID-19 pandemic.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

This chapter begins by introducing the vehicle routing problem with time windows, which is the case study in this thesis. Then, existing approaches for solving vehicle routing problems, including exact approaches, metaheuristics and automated design approaches, are discussed. Details of the objectives and major contributions are provided, followed by the organisation of this thesis.

## 1.1 Vehicle Routing Problems

The vehicle routing problem (VRP) is a type of combinatorial optimisation problems (COPs) which involve finding the best possible solution from a finite set of feasible solutions. It has gained significant attention from both academia researchers and industry professionals due to its practical applications in transportation [1] and logistic distribution [2], [3].

In the VRP, the main objective is to find the optimal set of routes to be performed by multiple vehicles to serve a given set of customers while satisfying various constraints [4]. These constraints typically include limitations on vehicle capacity, the time windows of customers, vehicle availability and depot visitation. The VRP is a computational challenging problem. It has been proved to be NP-hard [5], and for large-scale instances with more than 100 nodes (i.e customers), finding a general solution is practically infeasible [6].

Therefore, the VRP serves as a well-established and challenging benchmark problem for evaluating the performance of different algorithms. Consequently, this thesis selects the VRP as the case study, driven by the significance of its role as a benchmark problem.

The capacitated vehicle routing problem with time windows (CVRPTW) is considered one of the most extensively studied VRP variants within operations research and optimisation communities. The CVRPTW incorporates time-window constraints, making it particularly relevant to real-world logistics scenarios. By adopting the CVRPTW as a benchmark, researchers can effectively compare their proposed algorithms with existing approaches, facilitating a fair evaluation and benchmarking of the algorithm performance.

The focus of this thesis is on the automated algorithm design itself, and for this purpose, we have selected challenging CVRPTW instances with 100 customers as case studies. These instances serve as suitable problems to investigate the effectiveness of the designed algorithm.

## 1.2 Existing Approaches for Combinatorial Optimisation Problems

In the existing literature, there are numerous algorithms for solving complex combinatorial optimisation problems such as the vehicle routing problem. The two most commonly used categories are exact approaches and metaheuristics. The former focuses on finding the optimal solution, while the latter aims to obtain satisfactory solutions within a limited computational time. In addition, in recent years, with the advancement of machine learning algorithms, the research field has seen the emergence of using machine learning methods to automatically design new search algorithms to provide solutions in different scenarios. Note that VRP is selected as the case study for this thesis, this section will focus on reviewing existing approaches specifically designed for

solving the VRP.

**Exact Approaches:**

Exact approaches, such as cutting plane algorithm, column generation and branch-and-price, aim to find the optimal solution by exhaustively exploring all possible solutions. Exact approaches are able to guarantee the optimal of the produced solution, but their computational complexity increases exponentially with problem size, making them suitable for small or medium-sized instances. For large-scale vehicle routing problems which includes more than 100 customers, exact approaches require high computational time due to the exponential search space. Therefore, albeit many exact approaches exist to solve VRP efficiently, the fact remains that large-scale or more complicated types of VRP are not easily tackled by exact approaches. This can be observed in the literature that there are no known optimal solutions for most CVRPTW instances with 100 customers.

**Metaheuristics:**

Metaheuristics, which can be roughly categorised into single-solution based algorithms and population-based algorithms, are higher-level strategies that guide the search process in finding good-quality solutions in a reasonable amount of time but may not guarantee optimality.

Single-solution based algorithms such as simulated annealing [7], tabu search [8] and variable neighbourhood search [9], have been proposed to solve VRP variants effectively. Population-based algorithms such as the genetic algorithm [10], particle swarm optimisation [11] and ant colony optimisation [12], have also shown to be effective on different VRP variants. Although there exists many metaheuristics for solving VRP, they generally adopt a specific type of pre-defined metaheuristic to address the specific VRP variant, requiring considerable human effort and different kinds of expertise to make various algorithm design decisions such as choosing the suitable metaheuristic template and tuning the corresponding hyper-parameters. Furthermore, these customised metaheuristics are often discarded after solving a specific VRP in-

stance, resulting in a significant waste of human resources. Therefore, it is desirable to automate the process of algorithm design to reduce the burden on human experts.

**Automated Design Approaches:**

Towards automated algorithm design, the problem of designing metaheuristics itself is defined as a combinatorial optimisation problem in [13], upon a search space of different decision variables, e.g. algorithm parameters, portfolio of algorithms or algorithmic components. The research in this field therefore can be categorised into automated algorithm configuration, algorithm selection, and algorithm composition, based on the different types of decision variables considered in the search space of algorithms [13]. The first category aims to automatically configure the parameters of a specific type or a family of algorithms. The second category focuses on selecting a candidate algorithm or combining several existing algorithms against problem/instance characteristics. In contrast to these two categories, by combining the basic algorithmic components, automated algorithm composition aims to generate general-purpose algorithms, i.e. the algorithms generated do not belong to any specific search algorithms, e.g., genetic algorithm or particle swarm optimisation, etc.

Algorithm configuration can determine a well-performing parameter setting; however, it requires sufficient prior knowledge about which specific algorithm should be used. Algorithm selection addresses the limitation of the first category; however, it introduces the difficult and complex problem of identifying the key characteristics of the problems/instances. Automated algorithm composition aims to flexibly compose and generate new algorithms; however, some human expertise is still required to pre-select candidate heuristics in existing frameworks. This thesis falls into the third category, i.e. automated algorithm composition, to investigate the elementary and basic components to automatically design search algorithms.

**Summary:**

Exact approaches are usually not applicable to large-scale or complicated com-

binatorial optimisation problems. Metaheuristics show great potential for obtaining high-quality solutions within a reasonable computational time. However, when designing a highly specialised metaheuristic, human experts are required to make a large number of design decisions. To address this issue, automated algorithm design has recently attracted considerable attention from the research community. Automated algorithm design is still an under explored research area albeit some successful preliminary attempts .

## 1.3 Motivations and Objectives

Most evolutionary algorithms and metaheuristics in the existing literature have been manually designed by researchers of different expertise, many with ad hoc chosen algorithms for the specific problems in hand. With the recent rapid development of artificial intelligence, particularly in machine learning, for solving complex real-world problems have motivated the advances towards automated design of search algorithms. Within the domain of automated algorithm design, automated algorithm composition is gaining increasing attention due to its greater potential to generate more general search algorithms to effectively solve complex COPs, such as vehicle routing problems. It is not subjected to existing specific search algorithm templates such as the genetic algorithm [10] or the particle swarm optimisation algorithm [11]. The focus of this thesis is on the automated algorithm composition problem, leveraging advanced machine learning techniques for effective search algorithm design. This section aims to present existing frameworks, features, and learning techniques employed in the automated algorithm design process.

In automated algorithm composition, a set of heuristics is automatically combined to generate new search algorithms to solve instances across different problem domains. The most investigated technique is hyper-heuristics [14], which involves intelligently selecting or generating appropriate heuristics for a given situation. Frameworks developed include HyFlex [15], EvoHyp [16],

and SHH [17], etc. HyFlex explores a decision space of low-level heuristics (i.e a set of problem-specific rules or algorithms) or heuristic operators (e.g., taking search operators from ten well-known techniques as building blocks [18]) while EvoHyp adapts evolutionary algorithms as high-level strategies. SHH is specifically built for automatically combining different components of swarm intelligence algorithms [17]. In addition, some composition frameworks have been built within specific metaheuristic templates, such as CMA-ES [19] and PSO-DE [20].

In supporting effective automated algorithm composition using machine learning techniques, different types of features have been proposed in the existing literature. The most investigated feature types include search-dependent features and instance-dependent features. Search-dependent features encompass observations of the search process itself, such as the mean and standard deviation of the population fitness, and the average distance from the best individual [21]. On the other hand, instance-dependent features capture the fundamental characteristics of the problem instances. Other types of features, such as landmarking features [22] and image features [23], are not included in this thesis as they are specifically tailored to solution encoding scheme and therefore are not transferable for developing a general methodology; this would not serve the purpose of automated algorithm design.

Different machine learning techniques have been employed to support the automated algorithm composition as a new learning task in the literature. The most investigated one is Reinforcement Learning (RL) [24], which models the problem of algorithm design as a Markov Decision Process (MDP). RL is a learning technique, where an agent determines an optimal action at each state based on its interactions with the environment. At each new state of the environment, the agent selects an action from a set of actions. Based on the rewards or punishments after performing each selected action, the agent learns to intelligently select the action in the current state by forming the state-action pairs through trial and error [25]. In the context of automated algorithm composi-

tion, the state is defined by various features while the action is represented by the basic algorithmic components. Some RL methods, including tabular RLs such as SARSA [24] and Q-learning [26], deep RLs such as Deep Q-Network (DQN) [27] and Proximal Policy Optimisation (PPO) [28], have been used to support the automated algorithm design task in recent literature.

The research gaps in the field of automated design of search algorithms can be identified from three perspectives, i.e. framework establishment, feature identification and learning models development, as shown below.

- From the perspective of framework establishment, although existing automated algorithm composition frameworks (e.g. HyFlex, EvoHyp and SHH) have been successfully used for solving a variety of COPs, several limitations remain. HyFlex requires a set of pre-defined or problem-specific heuristics rather than basic algorithmic components to generate more general and powerful search algorithms for wider range of problems. EvoHyp predefines the selection operator and evolution operator, while SHH mixes these two types of operators. These frameworks thus build on the reduced search space of algorithm design, however, result in the loss of some advantageous combinations of basic components which may never be obtained or explored.

- From the perspective of feature identification, utilising machine learning techniques to assist automated algorithm design is still at a preliminary stage albeit some successful attempts across different disciplines. One of the important issues is on how to identify the key features to accurately characterise the search space for building successful machine learning. Although various features, such as search-dependent features and instance-dependent features, have been extracted for effective algorithm design in the literature, there is a lack of a systematic investigation analysing the extracted features within a consistent and general framework.

- From the perspective of learning models development, one research issue in applying tabular RL is concerned with the discretisation of the continuous state space, leading to unreliable results [29], [30]. Additionally, the number of identified features is limited and insufficient for effective learning. Furthermore, the use of simple positive/negative reward schemes may fail to accurately reflect the effects of the selected action. Moreover, it is often not clear how the RL techniques within the hyperheuristic framework have been devised, i.e. lack of clear definition on the three fundamental elements of RL, namely the state, action and reward scheme. There exists a significant scope and gap in this area of research, as it is often challenging to reimplement the exact same method and subsequently replicate the experiments.

The above raised research gaps motivate us to systematically investigate automated design of population-based algorithms to effectively solve COPs in different scenarios. Based on this, three research questions (RQ) are presented as follows:

- *RQ1:* what kind of framework can be established to serve as the basis of analysing algorithms for automated algorithm design?

- *RQ2:* what kind of features can be identified to capture useful and sufficient information for assisting effective algorithm design?

- *RQ3:* what kind of machine learning techniques can automatically design effective search algorithms with little human intervention?

Based on the motivations and research questions, the main aim of this thesis can be summarised as: To systematically investigate automated design of population-based algorithms for constraint combinatorial optimisation problems, taking the vehicle routing problem as a case study. To achieve this goal, three main research objectives are derived as follows:

- *Objective 1:* to develop a general search framework to serve as the basis of analysing algorithms for automated design and identify key research issues.

- *Objective 2:* to identify the key features to provide useful and sufficient information about the search space of algorithm design for building successful machine learning.

- *Objective 3:* to develop machine learning models with the identified features to address key research issues within the proposed general search framework.

## 1.4 Contributions

This overall objective of this thesis is to develop a general search framework (GSF), to support automated design of population-based algorithms. The capacitated vehicle routing problem with time windows (CVRPTW) is used as a case study in this thesis. A set of reinforcement learning based models are devised to address different key research issues within GSF, producing promising results. The contributions of this thesis include:

- **Establishment of a New General Search Framework**: a novel general search framework (GSF) is established to formulate different single-solution based and population-based algorithms. The unified GSF serves as the basis to analyse algorithmic components, generating effective search algorithms for CVRPTW automatically.

- **Feature Identification**: two groups of features, namely search-dependent and instance-dependent features, are identified to capture useful information for RL thus assisting effective algorithm design. A state-of-the-art reinforcement learning technique, proximal policy optimisation (PPO) [28], is employed to analyse the influence of different state representation (i.e. with different identified features).

- **Algorithmic Component Analysis**: Search patterns of the algorithms which are automatically designed by machine learning, consisting of the utilisation and transition of algorithmic components, are analysed to further provide insights into reusing knowledge extracted in algorithm design using machine learning.

- **Automated Composition of Evolution Operators**: the automated algorithm composition process is formulated as a MDP. Two advanced deep RL methods, deep Q-network (DQN) [27] and proximal policy optimisation (PPO) [28], have been investigated within the proposed GSF to address the key issue of automated selection and combination of the most efficient evolution operators during different stages of evolution. Results on CVRPTW benchmarks demonstrate the effectiveness of the trained policy compared to a search procedure without learning. The generality of the trained policy is further validated by applying it directly to new CVRPTW instances. In addition to the knowledge extracted and retained in the DQN and PPO models, the training time of RL-based techniques is also justified by the time and expertise needed to develop new models and algorithms from scratch to tackle new problem instances.

- **Automated Composition of Selection Heuristics and Evolution Operators**: the automated algorithm composition problem is systematically investigated by exploring the design space within different modules of the general search framework separately using controlled experiments, exploring the whole design space without fixing components manually. A state-of-the-art reinforcement learning method with a maximum entropy mechanism is developed to tackle the automated algorithm design problem with a continuous state space and a high-dimensional discrete action space. The analysis on comprehensive experiments on the CVRPTW benchmark instances demonstrate the effectiveness and generality of the proposed method transferring knowledge discovered into

solving new problem instances.

The aforementioned contributions are part of or included in the following list of works completed during the PhD studies:

- Wenjie Yi, Rong Qu, Licheng Jiao, Ben Niu. Automated Design of Metaheuristics Using Reinforcement Learning within a Novel General Search Framework. IEEE Transactions on Evolutionary Computation, 2022. Doi:10.1109/TEVC.2022.3197298.

  The content of this paper is covered in Chapter 3 and Chapter 5.

- Wenjie Yi, Rong Qu, Licheng Jiao. Automated Algorithm Design Using Proximal Policy Optimisation with Identified Features. Expert Systems with Applications, Volume 216, 15 April 2023,119461.

  The content of this paper is covered in Chapter 4.

- Wenjie Yi, Rong Qu. Automated Design of Search Algorithms based on Reinforcement Learning, paper under review.

  The content of this paper is covered in Chapter 6.

## 1.5 Organisation of the Thesis

This thesis is structured as follows:

- Chapter 2 presents the related work, including the optimisation models of the VRP, existing approaches for solving combinatorial optimisation problems such as the VRP, automated algorithm design and corresponding machine learning approaches. Details of the basic CVRPTW model are described. This chapter reviews and analyses the advantages and drawbacks of the current automated algorithm design research within three categories. This chapter also presents the existing studies on utilising machine learning techniques on the new task of automated algorithm design, especially those based on reinforcement learning.

- A novel general search framework to support effective algorithm design is presented in Chapter 3. Based on the existing literature, some generic and specific algorithmic components, including selection heuristics on the population and evolution operators, are defined within the general search framework. This chapter also presents three key research issues within the proposed framework: automated composition of evolution operators, of selection heuristics, and of both.

- Two types of key features for building successful machine learning, i.e. search-dependent and instance-dependent features, are identified in Chapter 4 to accurately describe the search space of algorithm design as a new task for machine learning. Then, an advanced reinforcement learning method is devised to automatically design search algorithms with the newly identified key features. The effectiveness of the identified features is validated and the component analysis of the best automatically designed algorithms is presented.

- Chapter 5 focuses on the first research issue within the proposed general search framework: automated composition of evolution operators. Two reinforcement learning based methods, deep Q-network based and proximal policy optimisation based methods, have been devised to automatically select suitable evolution operators during the optimisation process. The effectiveness and generality of the proposed methods are validated comprehensively across different CVRPTW instances.

- Chapter 6 focuses on the second and third key research issues within the proposed general search framework: automated composition of selection heuristics and of both selection heuristics and evolution operators. This chapter systematically investigates the impact of individual algorithmic components and the synergy between multiple algorithmic components. An advanced reinforcement learning method with adapted maximum entropy mechanisms is devised to tackle the automated algorithm problem

with a continuous state space and a high-dimensional action space. The effectiveness and generality of the proposed methods are validated across different CVRPTW instances.

- Chapter 7 summarises the contributions of this thesis. Limitations and suggestions for further improvements are also presented.

# Chapter 2

# Related Work

This chapter starts by firstly introducing the optimisation models of the vehicle routing problem. Second, the related work on search algorithms for solving the VRP is presented. Third, the existing scientific literature on automated algorithm design is presented. Fourth, this chapter reviews the existing machine learning techniques for automatically designing search algorithms. Finally, the related studies utilising reinforcement learning techniques for automated algorithm design are presented.

## 2.1 Optimisation Models of the Vehicle Routing Problem

The vehicle routing problem is arguably one of the most important transport scheduling problems. In the classic model CVRPTW, a fleet of vehicles are routed to serve the customers with the minimal distance, satisfying capacity and time windows constraints. Due to its generality, CVRPTW is used as a benchmark problem in evaluating the performance and general applicability of the methodologies proposed in the research of other VRP variants [31], [32], [33], [34], [35], [36]. This thesis will investigate the CVRPTW to better understand the proposed reinforcement learning based automated algorithm design approaches.

The CVRPTW can be mathematically formulated as follows [37]:

A fleet of $K$ vehicles are used to serve $n$ customers. To customer $v_i$, the service start time $b_i$ must fall within the time window $[e_i, f_i]$, where $e_i$ and $f_i$ represent the earliest and latest time to serve $q_i$ (i.e. the demand of $v_i$), respectively. If a vehicle arrives at $v_i$ at time $a_i < e_i$, a waiting time $w_i = \max\{0, e_i - a_i\}$ occurs. Consequently, the service start time $b_i = \max\{e_i, a_i\}$. Each vehicle with a capacity $Q$ travels on a route connecting a subset of customers starting from $v_0$ and ending within the schedule horizon $[e_0, f_0]$. $d_{ij}$ represents the distance from customer $v_i$ to customer $v_j$.

**Decision variables:**

$X_{ij}^k = 1$, if the edge from $v_i$ to $v_j$ is assigned in the route of vehicle $k$; otherwise $X_{ij}^k = 0$.

**Objective functions:**

$$\text{Minimise} \qquad K \tag{2.1}$$

$$\text{Minimise} \qquad \sum_{k\in K}\sum_{v_i\in V}\sum_{v_j\in V} X_{ij}^k d_{ij} \tag{2.2}$$

**Constraints:**

$$\sum_{k\in K}\sum_{v_i\in V} X_{ij}^k = 1, \forall v_i \in V\setminus\{v_o\} \tag{2.3}$$

$$\sum_{k\in K}\sum_{v_j\in V} X_{ij}^k = 1, \forall v_j \in V\setminus\{v_o\} \tag{2.4}$$

$$\sum_{k\in K}\sum_{v_i\in V}\sum_{v_j\in V\setminus\{v_o\}} X_{ij}^k = n \tag{2.5}$$

$$\sum_{v_j\in V} X_{oj}^k = 1, \forall k \in K \tag{2.6}$$

$$\sum_{v_i\in V} X_{ij}^k - \sum_{v_j\in V} X_{ji}^k = 0, \forall k \in K, v_j \in V\setminus\{v_o\} \tag{2.7}$$

$$\sum_{v_i} X_{io}^k = 1, \forall k \in K \tag{2.8}$$

$$e_i \leqslant b_i \leqslant f_i, \forall v_i \in V \tag{2.9}$$

$$\sum_{v_i \in V} \sum_{v_j \in V} X_{ij}^k q_i \leqslant Q, \forall k \in K \tag{2.10}$$

$$X_{ij}^k \in \{0,1\}, \forall v_i, v_j \in V, k \in K \tag{2.11}$$

The first objective is to minimise the number of vehicles (Equation (2.1)) while the second objective is to minimise the total travelled distance (Equation (2.2)). Constraints (2.3–2.5) limit every customer to be visited exactly once while ensuring that all customers are served. Constraints (2.6–2.8) define the route by vehicle $k$. Constraints (2.9) and (2.10) define the customer time windows constraint and vehicle capacity constraint, respectively. Constraint (2.11) defines the domain of the decision variables $X_{ij}^k$.

As shown in Equation (2.12), we adapt the mostly used evaluation function in the literature, where the two objectives are transformed into a single objective with a penalty weight factor $c$=1000 to assign a higher priority to the first objective [38]. Figure 2.1 provides an illustrative example of CVRPTW with three routes/vehicles and a single depot.

$$f = \sum_{k \in K}^{K} \sum_{v_i \in V} \sum_{v_j \in V} X_{ij}^k d_{ij} + c \times K \tag{2.12}$$

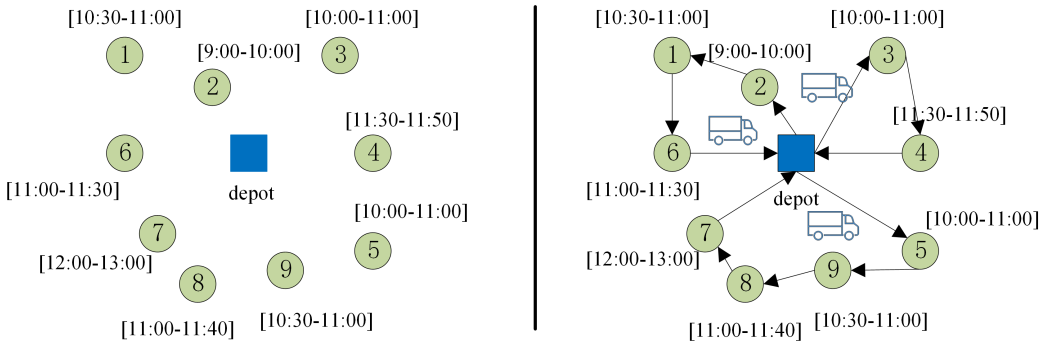



**Figure 2.1:** *Example of a CVRPTW instance and a solution (with one depot and nine customers)*

The Solomon benchmark dataset [39] consists of six sets of instances of different characteristics (C1, C2, R1, R2, RC1 and RC2). The instances differ with respect to the customers' geographical locations, vehicle capacity, den-

sity and tightness of the time windows. Customers in instance sets C1 and C2 are clustered geographically; while customers in instance sets R1 and R2 are randomly located. Instance sets RC1 and RC2 contain a mixture of random and clustered customers. The customer locations are identical for the same type of problem instances. The instances within one type differ with respect to the density and tightness of the time windows, i.e. the percentage of time-constrained customers and the width of the time windows. The Solomon benchmark dataset has been extended and applied to other VRP variants research due to its ability to effectively represent the characteristics of real-world VRP instances and demonstrate strong scalability. The CVRPTW remains a challenge to current state-of-the-art research since the CVRPTW instances with more than 100 nodes (i.e. customers) are unsolvable in general [6].

## 2.2 Search Algorithms for the Vehicle Routing Problem

Due to their non-convex problem structure and complex constraints, VRP and its variants are NP-hard [5]. Different algorithms have been extensively investigated and successfully applied to VRP. These algorithms can be roughly classified into exact approaches and metaheuristic algorithms. Exact approaches, such as branch-and-bound, branch-and-cut and Lagrangian relaxation, are designed in such a way that the solutions they find can be guaranteed to be optimal. However, exact approaches are usually not applicable to complicated or large-scale problems as they require exponential computational time. Therefore, albeit many approaches exist to solve VRP efficiently, the fact remains that large-scale or more complicated types of VRP are not easily tackled by exact approaches.

With the increasing difficulties arising from VRP, such as multiple objectives and complex constraints, this optimisation problem is likely to be intractable

for exact approaches, regardless of exponentially increasing computing power. Metaheuristic algorithms, which are able to find a good approximate solution within a reasonable computational time, have attracted considerable attention from the operational research and evolutionary computation communities. Metaheuristic algorithms can be categorised into single-solution based and population-based algorithms [40]. This section presents the related work on utilising different metaheuristic algorithms to investigate VRP.

### 2.2.1 Single-solution Based Algorithms

Single-solution based algorithms concern only one candidate solution during the whole search process. Simulated annealing (SA), tabu search (TS), and variable neighbourhood search (VNS) are currently among the most popular single-solution based algorithms for VRP in the literature.

#### (1) Simulated Annealing

Simulated annealing was first proposed by Metropolis et al.[7] and introduced into combinatorial optimisation by S.Kirkpatrick [38]. It is a stochastic optimisation algorithm based on Monte-Carlo iterative strategy and the similarity between the annealing process of solid substances in physics and combinatorial optimisation problems. Simulated annealing starts from a certain high initial temperature, with the continuous decrease of temperature parameters. This algorithm is able to escape from local optima by accepting worse solutions based on the temperature probabilistically and finally reaches the global optimal solution.

Intensive research has been conducted utilising SA as a VRP solver [41],[42], [43], [44], [45]. One research direction is on combining the standard SA with well-known heuristics to improve neighbourhood searching. For example, Alfa et al. [46] developed a hybrid SA combined with the 3-opt operator to solve the basic VRP and Ilhan et al. [47] proposed an improved SA with crossover

operator as a CVRP solver.

### (2) Tabu Search

Tabu search, proposed by Glover et.al [8], starts from an initial feasible solution, and chooses the move which improves the fitness function most. A worse solution can be accepted if there is no improvement in the solution. To prevent being trapped into local optima, a tabu list is built to forbid the selection of already visited solutions.

The research on applying TS to VRP can be classified into how to generate the initial solution and how to design the neighbourhood structure. In terms of the first category, for example, Wang et al. [48] proposed a modified TS considering time randomness to obtain better initial VRP solutions. Regarding the second research direction, Berbotto et al. [49] designed a tabu search algorithm based on granular neighbourhood structure, which performs well on many benchmark instances.

### (3) Variable Neighbourhood Search

The variable neighbourhood search utilises several different neighbourhoods to search systematically [9]. VNS has been extensively investigated for different types of VRP. In a study conducted by Braysy [50], a modified VNS based on a four-phase approach showed to be effective on the VRPTW. A new route construction heuristic was proposed to generate an initial solution, and a route elimination procedure was used to improve the solutions in terms of the number of vehicles in the second phase. Then, four novel local search procedures were designed to improve the solutions with respect to the total travelled distance and finally the objective function was modified to escape from local optima. Multiple-depot VRPTW was first tackled using VNS by Polacek et al. [51]. The corresponding competitive experiment with a TS algorithm confirmed the superiority of VNS. Later, several studies have investigated the utilisation of VNS for the VRPTW. For instance, Chen et al. [52] composed several

independent neighbourhood operators into compound neighbour operators to tackle the VRPTW which considers minimising the number of vehicles and the total travelled distance simultaneously. In addition to VRPTW, VNS has also been used to tackle the large-scale VRPs [53], multi-objective VRP [54], the open VRP [55] and other variants.

### 2.2.2 Population-based Algorithms

Unlike single-solution based algorithms, population-based algorithms are associated with a set of solutions rather than a single solution. The most investigated population-based algorithms are Evolutionary Algorithms (EAs) and Swarm Intelligence (SI).

#### (1) Evolutionary Algorithms

Evolutionary algorithms are motivated by Darwin's evolutionary theory, which simulates the evolution processes of selection, crossover and mutation to find near- or global-optimal solutions. Evolutionary algorithms consist of mainly genetic algorithm (GA) [10], evolution strategies (ES) [56], evolutionary programming (EP) [57], and genetic programming (GP) [58].

There are typically five algorithmic modules within EAs, including initialisation, selection for evolution, evolution, selection for replacement and termination. At the beginning of EAs, initialisation is used to produce the initial population. Then, individuals from the population are selected as parents through 'selection for evolution' heuristics. Evolution operators, including mutation and crossover, are applied to offspring and parents. Finally, 'selection for replacement' heuristics are used to update the current population with the offspring. The termination criteria is usually set as a predefined maximum number of iterations or the maximum computational time.

The most classical instance of evolutionary algorithms is the genetic algorithm [10] and it has been extensively investigated to certain types of VRP, e.g.,

CVRP [59], [60], [61], [62], VRPTW [63], [64], [65], [66] and VRPTWSD [67]. In the process of solving VRP by using GA, a common problem lies in the large number of infeasible solutions generated after applying crossover operators. To address this issue, many studies have attempted to design the chromosomes with trip delimiters and use a repair procedure to generate feasible solutions. However, such procedures weaken the genetic transmission of information from parents to offspring. To bridge the gap, Prins [68] firstly hybridised a special splitting procedure with GA to convert a chromosome into a feasible VRP solution without any repair procedures. This design can guarantee the feasibility of the solution after crossover without using repair mechanisms. GAs have also been successfully applied to different variants of multi-objective vehicle routing problems (MOVRP), such as MOVRP with time windows [69], multi-trip VRP [70], and multi-depot green VRP [71]. In these applications, most GAs are the variants of Non-dominated Genetic Algorithm II (NSGAII) [72] which utilises the fundamental concept in multi-objective optimisation, i.e. Pareto dominance.

**(2) Swarm Intelligent Algorithms**

Since the 1990's, swarm intelligent algorithms, as a new kind of computational intelligence, have attracted considerable attention from the optimisation research community [73]. These novel intelligent algorithms are general-purpose stochastic search approaches originated with the observation of social behaviour in biological systems.

A considerable amount of literature investigate novel swarm intelligent algorithms in the last few decades. Most of them are inspired by the behaviour of swarms. Ant colony optimisation (ACO) [12] is motivated from the foraging behaviour of ants; particle swarm optimisation (PSO) [11] is motivated from swarm behaviour of bird flocking or fish schooling; artificial bee colony algorithm (ABC) [74] is stimulated by social specialisation behaviour of bees; artificial fish swarm algorithm (AFS) [75] is originated from the swarming

behaviour of fish. Swarm intelligent algorithms have demonstrated high performance and great development potential in real-world applications such as VRP. Two commonly used swarm intelligent algorithms for VRP are PSO and ACO.

PSO algorithm, which is inspired by the food-seeking behaviours of birds, has become the most extensively investigated swarm intelligence algorithm due to its simple implementation and strong global optimisation capability [76]. Every particle, i.e a candidate solution of the optimisation problem, is varied according to its own search experience and relationship with other particles within the population. Gradually the population moves into promising regions of the search space due to its learning mechanism.

Most applications of PSO have concentrated on continuous optimisation while some work has been done to discrete optimisation. When applying PSO to the VRP, algorithm designers are required to propose suitable encoding and decoding mechanisms. For example, a random-key based encoding and decoding mechanism for tackle VRP has been proposed [77]. Specifically, a VRP solution with $n$ customers and $m$ vehicles is represented as a $(n+2m)$-dimensional particle and the decoding method is based on the customer priority list and vehicle priority matrix. Encoding and decoding methods have been successfully applied to solve CVRP [77] and the vehicle routing problem with simultaneous pickup and delivery (VRPSPD) [78]. Another research direction to address the route coding issue is to hybridise PSO with local search methods. For example, a multiple phase neighbourhood search-greedy randomised adaptive search procedure (MPNS-GRASP), expanding neighbourhood search (ENS), and a path relinking (PR) strategy have been hybridised in PSO to solve large-scale VRP [79]. A new version of PSO based on tabu search has been proposed to tackle the encoding and decoding issues and it has showed to be efficient on the VRP [80]. To date, PSO has shown to be effective on several VRP variants, including VRP with capacity constraint [77], VRP with simultaneous pickup and delivery [78], VRP with time windows [52], and VRP

with stochastic demands [81].

Ant colony optimisation models the optimisation process upon path-finding behaviours of a colony of ants searching for food [12]. The basic idea can be summarised as: the path of ants is used to represent the feasible solution of the problem, and all the paths of the whole ant colony constitute the solution space of the problem to be optimised. Ants with shorter paths released more pheromones. As time went on, the concentration of pheromones accumulated on shorter paths increased gradually, and the number of ants choosing this path increased. Finally, the whole ant colony will converge to the optimal path.

However, certain drawbacks need to be addressed applying original ACO to real-world applications such as the VRP. The major drawbacks are its inability to escape from local optima and poor time performance. Several studies have attempted to adjust the pheromone approach and introduce new mutation/crossover operators to the original ACO. An improved ACO with an ant-weight strategy to update the increased pheromone and a mutation operation showed to be effective on the VRP [82]. An ACO has been hybridised with the saving algorithm and $\lambda$-interchange mechanism to solve VRPTW [83]. Besides, to escape from local optima, pheromone approach has been adjusted and a disaster operator has been introduced within the hybrid ACO method. Bin et al. proposed a modified ACO to tackle period vehicle routing problem with time windows (PVRPTW). A multi-dimension pheromone matrix is used to accumulate heuristic information on different days and two-crossover operations are introduced to improve the performance of ACO [84]. Another research direction involves the usage of a multiple ACO to solve the corresponding multiple objective vehicle routing problem. For example, a hierarchy of artificial ant colonies is designed where the first colony minimises the number of vehicles while the second colony minimises the total travelled distances. Colonies cooperate by exchanging information through pheromone updating [85]. There also exist works on ACO to solve a VRP with more than

two objectives. For example, a modified multiple ACO is designed to produce Pareto optimal solutions for the VRPTW with three objectives: the number of vehicles, the total travelled distance, and the total delivery time [86].

## 2.3 Automated Design of Search Algorithms

Metaheuristic algorithms, although shown to be effective on solving complex COPs such as VRP, usually work for particular problem instances and heavily rely on different and extensive human expertise. To address the limitations, automated algorithm design has received considerable research attention in recent years [87], [88].

The research in this field can be categorised into algorithm configuration, algorithm selection, and algorithm composition, based on the different types of decision variables considered in the search space of algorithms [13]. This section provides a detailed review and analysis of these different automated algorithm design techniques.

### 2.3.1 Automated Algorithm Configuration

Most metaheuristics in the literature are manually tuned by testing all possible or the most promising parameter values and then find the best setting for a specific problem instance, which is computationally expensive and problem-dependent. The same parameter configuration is not likely performing well on unseen problem instances. In addition, there may exist better parameter configurations which may never be explored. This motivates researchers to automate the configuration process, i.e. automated algorithm configuration.

Automated algorithm configuration aims to determine a well-performing parameter setting of a given algorithm across a given set of problem instances automatically. A basic algorithm configuration procedure [89] is shown in Figure 2.2. Note that algorithm configuration is considered as a black-box optimisation problem within this procedure. A target algorithm with a specific

parameter setting is executed on problem instances (configuration scenario). Then, the performance of the target algorithm on the configuration scenario is used to guide the selection of subsequent target algorithm.

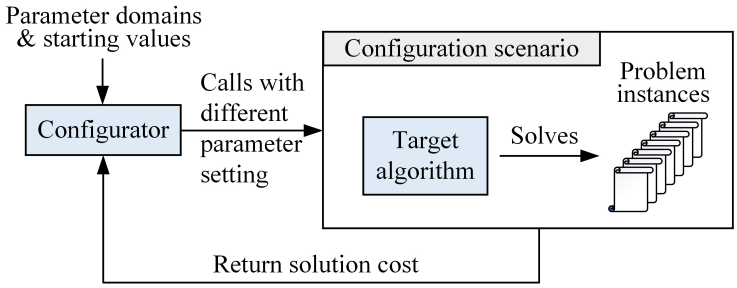



**Figure 2.2:** *A basic algorithm configuration procedure*

A number of frameworks have been established to automatically configure search algorithms. These frameworks include ParamILS [90], which utilises iterated local search, F-race [91] and irace [92], both using racing mechanism, and the surrogate-based methods such as SPOT [93], SMAC [94], MIP-EGO [95] and Hyperopt [96]. There are also other algorithm configuration frameworks based on a component-based view, such as multi-objective ant colony optimisation algorithms (MOACOs) [97], and the extended multi-objective evolutionary algorithms (MOEAs) framework [98].

### 2.3.2 Automated Algorithm Selection

Most existing studies usually select a specific type/family of metaheuristic algorithm manually to tackle a specific optimisation problem, requiring considerable human efforts choosing a candidate algorithm or combine several existing algorithms against problem/instance characteristics. To address this issue, researchers have attempted to investigate automated algorithm selection. A basic algorithm selection framework composed of four components [99] is illustrated in Figure 2.3.

- The first component, i.e. the problem space, needs to be properly defined using mathematical notation, but the fact remains that some problems

cannot be easily described mathematically.

- The second component, i.e the algorithm space, contains the smallest set of algorithms which perform well on a large subset of problems in the problem space. The included algorithms need to satisfy the requirements of complementation and robustness simultaneously. In other words, they should be able to solve different types of problems with a scientifically demonstrable accuracy.

- The third component is the performance space with indicators to measure the accuracy, speed, or other requirements of the algorithm. Due to the complexity of the problem space and algorithm space , it is not easy to obtain solutions to algorithm selection problem.

- The fourth component, i.e. the characteristics space, is introduced to simplify the problem [99]. Characteristics, which rely on problem domains, should be able to reflect the complexity of the problem space and assess the advantages and disadvantages of every algorithm within the algorithm space. In this regard, the fourth component is the most important one when utilising this framework to automate algorithm selection process.

Algorithm selection can be roughly categorised into one-algorithm selection, including per-instance and per-set algorithm selection, and multiple-algorithm selection, including algorithm schedule and algorithm portfolio.
In one-algorithm selection, one single algorithm is selected to solve a given problem instance (per-instance) or a group of problem instances (per-set). The basic algorithm selection framework shown in Figure 2.3 belongs to the per-instance one-algorithm selection. The major limitation of this framework is that it cannot be reused for other instances of the same problem, i.e. low generality. To address this limitation, researchers have further developed per-set algorithm selection methods aiming at solving a group of problem instances.

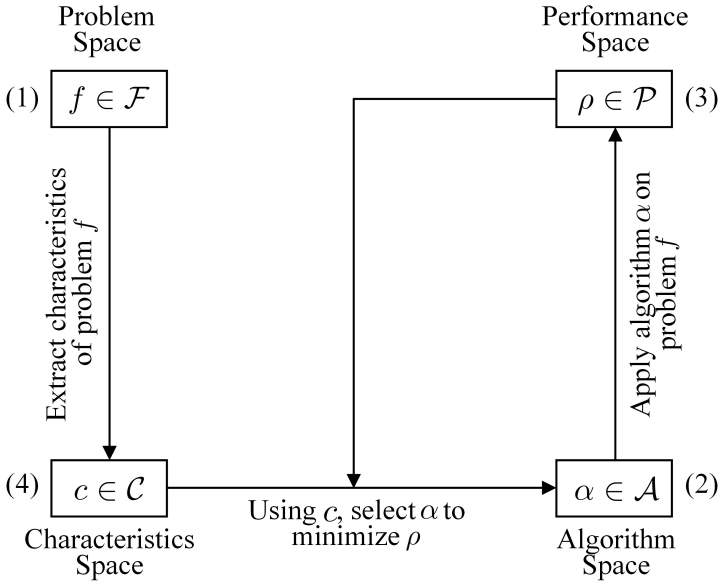



**Figure 2.3:** *A basic algorithm selection framework*

Multiple-algorithm selection aims to select and combine different search algorithms to effectively solve the given problem. Research can be mainly categorised into algorithm schedule and algorithm portfolio. Algorithm schedule arranges different algorithms at different stages of the problem-solving process [100] while algorithm portfolio runs the algorithms within the portfolio in parallel [101].

A number of frameworks have been established to automatically select search algorithms. These frameworks developed include PAP [102], which integrates different evolutionary algorithms to solve numerical optimisation problems, and Hydra [103], with a configuration technique for portfolio-based algorithm selection, and machine learning based algorithm selectors [104].

### 2.3.3 Automated Algorithm Composition

Unlike automated algorithm configuration and automated algorithm selection, automated algorithm composition aims to generate general-purpose search algorithms which can obtain high-quality solutions across different problem instances/domains rather than just one problem instance or a group of problem instances.

From the algorithmic perspective, a number of novel metaheuristic algorithms have been proposed in recent years, especially those based on new metaphors. However, the novelty of such metaheuristic algorithms has been questioned [105]. In some metaheuristics literature, researchers make a large amount of efforts in designing the search process based on the behaviour of different swarms but without fundamental and scientific contributions to the field. These proposed algorithms do not introduce any new ideas but simply rename existing methods with the same underlying concepts. Metaheuristics should be analysed from a component-based perspective to obtain truly innovative and fundamental ideas. These ideas consist of analysing the contribution of the common components within the existing metaheuristics to the solution quality and developing truly powerful methods based on the analytical results. The most investigated technique in this field is hyper-heuristics [14]. Corresponding frameworks, including HyFlex [15], EvoHyp [16] and SHH [17], have been established.

Hyper-heuristics [14], which are broadly concerned with intelligently selecting or generating appropriate heuristics in a given situation, present to be one of those automated algorithm composition techniques. The basic selection hyper-heuristic framework is illustrated in Figure 2.4 [106]. The core motivation behind hyper-heuristics is to increase the generality level of different heuristics by drawing on the advantages and recognising the disadvantages of low-level heuristics. Due to the generality of hyper-heuristics, they have been extensively investigated and successfully applied to a wide range of applications such as timetabling [107], [108], graph colouring [109], and VRPTW [110].

HyFlex [15] and EvoHyp [16] are the two most popular hyper-heuristics platforms in the existing literature. Provided with problem-specific low-level heuristics to specific optimisation problems, HyFlex enables algorithm designers to develop cross-domain search algorithms. With HyFlex, algorithm designers can concentrate on the design of high-level strategies rather than putting a large amount of efforts on understanding problem domain knowledge. Com-

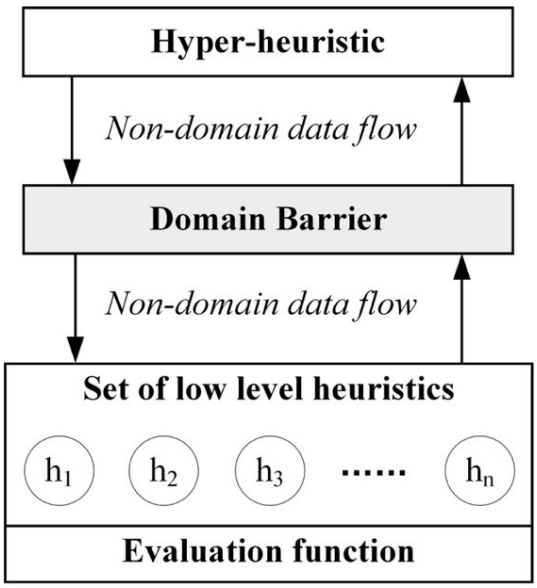



**Figure 2.4:** *Selection hyper-heuristics framework*

pared to HyFlex, EvoHyp tends to be more concerned with the utilisation of evolutionary computation techniques as high-level strategies to combine low-level heuristics.

Regarding the integration of population-based algorithms with hyper-heuristics frameworks, there are mainly two ways. First, a specific population-based algorithm can be used as a high-level strategy in hyper-heuristics. An individual within the population is encoded as a low-level heuristic or a sequence of low-level heuristics. The objective is to maximise the performance of the heuristic combination that generates the solution. Previously studied techniques in this category include particle swarm optimisation (PSO) [111], bacterial foraging optimisation (BFO) [112], [113], ant colony optimisation (ACO) [114], and cuckoo search (CS) [115]. The second way is to utilise a hyper-heuristic for local/neighbourhood search within a specific population-based algorithm. Every individual is encoded as a candidate solution of the problem. Previously studied techniques in this category include PSO [116] and ABC [117], [118]. Another hyper-heuristics framework, named swarm hyper-heuristic (SHH), is specifically built to automatically combine different components of swarm intelligent algorithms [17].

## 2.4 Machine Learning for Automated Algorithm Design

This section reviews the existing literature on integrating machine learning into automated algorithm design for combinatorial optimisation problems, which are a challenging subset of optimisation problems due to their discrete nature and the potential combinatorial explosion of the search space. Section 2.4.1 shows the existing features employed in the context of automated algorithm design, since feature selection showed to be one of the current key research issues in developing successful machine learning. Section 2.4.2 reviews different categories of machine learning for dealing with the learning task of automated algorithm design.

### 2.4.1 Existing Features for Automated Algorithm Design

To support automated design of metaheuristics, the first important step is to identify key features which can comprehensively capture the characteristics of the search space of algorithms and the problem instances. Existing features used for automated algorithm design can be roughly categorised into search-dependent features and instance-dependent features. Other types of features, such as landmarking features [22] and image features [23], are not included in this study as they are specifically associated with the solution encoding scheme and therefore are not transferable for developing a general methodology; this would not serve the purpose of automated algorithm design.

Search-dependent features aim to describe the search process itself accurately. These features can be extracted during the search process of metaheuristics, and roughly divided into two categories. The first category is stage-based, such as the current iteration or stage of the search process [119]. The second category is solution-based which are usually derived from the solutions obtained

by metaheuristics, such as the best fitness, the fitness growth [22], the number of feasible solutions, the number of feasible solutions that are better than the initial solution [23], and the mean/standard deviation of explored solutions [22], [21].

Instance-dependent features consist of specific characteristics extracted from the problem instance definition, which can be divided into two categories. For VRP, the first category includes customer-based features, e.g., the mean/maximum /median and the skewness of the customer demand and the service time of customer, etc [23]. The second category of constraint-related features include the vehicle capacity, the average time-window size and the average time-window overlap [22].

Albeit various features have been proposed in the literature, it remains challenging to represent the search space of algorithms and to characterise problem instances, resulting in the low generality of existing search algorithms. Although showed to perform well on some selected benchmark datasets, existing features cannot always be extended to solve other benchmark datasets or real-world problem instances. It is necessary to conduct a systematic investigation on the impact of different feature sets on the performance of the metaheuristics, thus to support automated algorithm design with effective learning.

### 2.4.2 Existing Machine Learning Techniques for Automated Algorithm Design

Machine learning is an area of artificial intelligence that can learn and improve from experience automatically. Machine learning tasks can be generally classified by learning mode into three categories [120]: supervised learning, unsupervised learning, and reinforcement learning. A number of techniques within these different categories have been adopted in the literature to support the learning task of automated algorithm design including automated algorithm configuration, selection and composition.

**Supervised and Unsupervised Learning:**

In supervised learning, the values of both input variables and output variables are known. Given the correct label of a set of training data, supervised learning techniques can learn the mapping relationship to predict the labels of unseen data. In the context of automated algorithm configuration, supervised learning techniques, including linear regression, logistic regression and random forest, have been employed to automatically determine parameter settings of search algorithms. Results on tuning search algorithms for travelling salesman problem and vehicle routing problem have demonstrated the superiority of this automated method [121], [122]. Regarding the applications on automated algorithm selection, supervised learning techniques such as k-nearest neighbours and ridge regression have shown to be effective in selecting suitable algorithms for travelling salesman problem [123] and SAT [124], [125]. Compared to automated algorithm configuration and selection, there has been relatively less work on utilising supervised learning methods to assist automated algorithm composition although there has been some successful attempts within the framework of hyper-heuristics. These attempts include neural networks to classify the algorithmic compositions generated by a graph-based hyper-heuristics proposed for exam timetabling problem [126] and extract hidden patterns for open VRP [127], and associative classification algorithms and decision trees to predict the behaviour of low-level heuristics used by a hybrid hyper-heuristic for the training scheduling problem [128]. Although these supervised learning methods show to be effective on the automated algorithm design task, especially on automated algorithm configuration, they heavily rely on the labelled training data. When solving complex real-world problems through automated algorithm design, obtaining labels for the data is not always feasible which limit the application of supervised learning methods.

Compared to supervised learning and reinforcement learning, the utilisation of unsupervised learning techniques in automated algorithm design has been

relatively limited. In unsupervised learning, the values of input variables are known while the values of output variables are unknown. Unsupervised learning techniques are used to describe structures and rules hidden in unlabelled data. In the context of automated algorithm design, unsupervised learning are mainly employed in automated algorithm configuration and selection. Regarding the applications on automated algorithm configuration, they are used to cluster features, aiming to enhance efficiency and avoid inappropriate classification. For instance, k-means has been employed to cluster instance-dependent features to improve the configuration performance [129] and automatically design neighbourhoods during the search process [130]. As for the application on automated algorithm composition, a clustering technique has been combined with a genetic algorithm to select low-level heuristics for solving large-scale vehicle routing problems [131], and $k$-means has been combined with Q-learning to produce a hyper-heuristic for job-shop scheduling problems [132]. Although unsupervised learning methods offer potential benefits for automated algorithm design, particularly in automated algorithm configuration and composition, they are highly sensitive to the properties of the input data used for training. Additionally, these algorithms can be computationally expensive, especially when dealing with large-scale or high-dimensional data.

**Reinforcement Learning:**

Reinforcement learning (RL) is a machine learning technique, where intelligent agents take actions based on a learned policy trained through trial and error interactions with the environment by maximising total reward. RL is often modelled as a Markov Decision Process (MDP), $M = (S, A, p, r)$, which consists of a set of possible states $S$ and a set of selectable actions $A$. An episode refers to a complete sequence of interactions between the agent and its environment, from the initial state to the termination state. A timestep, represents a discrete unit of time within an episode. In each timestep $t$, the agent interacts with the environment to obtain information of the current state

$s_t \in S$, where $S$ is the state space, and then chooses an action $a_t \in A$, according to the policy $\pi(a_t|s_t)$, where $A$ is the set of available actions. After that, the agent receives a reward $r_t(s_t, a_t)$ and the environment moves to the next state based on the policy $\pi$, i.e $s_{t+1} \sim \pi(s_{t+1}|s_t, a_t)$. The goal of the RL agent is to learn a policy that maximises the expected accumulated reward.

In recent literature on automated algorithm design, some RLs, such as SARSA [24], QL [26] and DQN [27], have been used to support the intelligent selection of the most appropriate evolution operators. They utilise feedback information on the performance of evolution operators during different stages of the search process. The research in this field can be classified into two categories based on how the action space is defined.

The first category of RL techniques in automated algorithm design defines the evolution operators in a specific type of search algorithms as the optional actions of the RL agent. In the literature, RL techniques are mostly applied to evolutionary algorithms such as the genetic algorithm, to select efficient mutation and crossover operators [29]. Results on the travelling salesman problem and the 0-1 knapsack problem have demonstrated the superiority of this automated method [133], [29], [134], [135].

However, due to the complexity of RL techniques, most studies in this field [133], [29], [134] have only focused on using the simplest tabular RL methods such as SARSA and QL. Few studies have investigated advanced techniques to handle the continuous state spaces when applying RL to select evolution operators [135]. There is a lack of research on advanced RL in effective and efficient automated algorithm design in evolutionary computation.

The second research category treats problem-specific heuristics as the optional actions of the RL agent. RL techniques are used as the high-level strategy to automatically combine different low-level heuristics in hyper-heuristics. Results of these RL-based approaches on unmanned aerial vehicles [136] and different COPs within the HyFlex software framework [25] demonstrated the effectiveness of these methods.

In these studies, several search-dependent features, i.e. the observations of the search process itself such as the current iteration and the total improvement over the initial solution [119], have been used to represent the state. The number of features identified, however, is limited and insufficient for learning. More advanced RL techniques are required to handle the continuous state space represented by key features involving sufficient information. Also, simple positive/negative reward schemes are used, which cannot accurately reflect the effects of the selected action. Furthermore, it is often not clear how the RL techniques within the hyper-heuristic framework have been devised, i.e. lack of clear definition on the three fundamental elements of RL, namely the state, action and reward scheme. There is still a large scope and gap in this research area, as it remains challenging to reimplement the exact same method and subsequently replicate the experiments.

To the best of our knowledge, no work has investigated RL for the task of automated composition of selection heuristics on the population, and both evolution operators and selection heuristics. In particular, the expanded high-dimensional action space is discrete, on which RL techniques are difficult to converge.

In this thesis, advanced RL techniques with a neural network function approximator have been applied to automatically compose algorithms to tackle the issue to represent appropriately the continuous state space. A maximum entropy mechanism is further designed to handle the high-dimensional action space issue in this thesis. The state space with sufficient features for effective learning is carefully defined. The action space is defined as the basic algorithmic components (i.e. evolution operators, selection heuristics, or both) to learn reusable knowledge in automated design of general search algorithms. Also, an effective reward scheme is defined to encourage the RL system to find efficient search policies. Note that this thesis adopts an offline RL framework, in which the policy is trained offline but used in an online fashion for new instances. This is different from most of RL-based automated algorithm design

methods in the literature.

## 2.5 Summary

This chapter reviews the literature related to vehicle routing problems and corresponding search algorithms. The capacitated vehicle routing problem with time windows model is introduced first. The CVRPTW has been shown to be an NP-hard problem, which cannot be easily tackled by exact approaches. Therefore, more recent research attention has focused on the utilisation of metaheuristics, which can return near-optimal solutions within a shorter time frame. These metaheuristic approaches, i.e. single-solution based algorithms and population-based algorithms, perform well on solving a specific type of VRP. A number of studies have begun to explore the automated algorithm design from a component-based view with the hope to generate more powerful search algorithms.

Automated algorithm design can be categorised into algorithm configuration, algorithm selection, and algorithm composition based on the difference in the search space of algorithms. Automated algorithm configuration can effectively determine the parameter settings of search algorithms, but these methods are usually limited due to the demand of sufficient prior knowledge about which specific algorithm should be used. Automated algorithm selection addresses this limitation of automated algorithm configuration by automatically selecting a specific existing algorithm or combining several existing algorithms based on the analysis of problem characteristics; however, it is very difficult to determine the key problem characteristics. Unlike the other two categories, automated algorithm composition can generate more general or even new algorithms by combining the most basic algorithmic components. The methods developed/proposed in this thesis falls into the third category, as it involves the usage of the most generic algorithmic components to design search algorithms automatically.

# Chapter 3

# A Novel General Search Framework to Support Effective Algorithm Design

The literature has identified a number of limitations of existing search frameworks, to support automated design of effective metaheuristic algorithms. In this chapter, a novel general search framework is developed to serve as the basis of future research to analyse algorithms for automated design, and thus generate effective search algorithms for vehicle routing, especially CVRPTW. Generic algorithmic components are defined within the proposed framework, and three key research issues are presented.

## 3.1 Introduction

Addressing highly complex combinatorial optimisation problems with various real-world constraints, such as CVRPTW, has shown to be one of the current research challenges in evolutionary computation although there have been some successful attempts at utilising metaheuristics as solvers.

The present research has been conducted with the following two motivations. From an algorithmic perspective, there is a growing body of literature that in-

troduces novel metaheuristics. However, researchers have questioned whether some of the newly proposed metaheuristics indeed present new ideas or have just renamed the concepts of already existing methods [105]. Several attempts have been made to develop metaheuristic libraries, such as HeuristicLab [137], Opt4J [138] and jMetal [139], with the aim of facilitating the reuse of existing metaheuristics. The components implemented in one specific library are not easily transferable for reuse in other frameworks. Recognising this issue, early attempts such as EvoSpec [140] and PISA [141] have focused on achieving the interoperability across frameworks. However, the main limitation of these existing libraries or frameworks is that they include a set of complete algorithms as framework components, rather than focusing on the basic generic algorithmic components that have the potential to generate more effective algorithms. There is a lack of standard or framework to include the basic generic algorithmic components, making it challenging to distinguish the differences between various metaheuristics and identify truly innovative and fundamental ideas. Secondly, from an application perspective, existing metaheuristics have been carefully designed for specific problems or instances, thus are difficult to be adapted to solve other problems or instances. The reason is that they are limited to a specific template of metaheuristic algorithms and they rely highly on the problem-domain knowledge. There is a lack of research into clear definition of general basic algorithmic components with little or no problem-domain knowledge. These general basic algorithmic components could be used to support effective algorithm design, and thus produces metaheuristic algorithms which could be easily adapted to solve different problems or instances. More specifically, we aim to answer the following research questions (RQ):

- *RQ1* What kind of framework we can establish to formulate in a unified way a range of metaheuristics to show the fundamental difference between metaheuristics?

- *RQ2* What key research issues we can define based on the established

search framework?

The goal of this chapter is to develop a novel general search framework, within which machine learning can be applied to the design space of algorithms and thus support automated algorithm design. Specifically, this chapter aims to address the following research objectives:

1. To develop a novel general search framework with clearly defined generic algorithmic components, including selection heuristics on the population and evolution operators. (RQ1)

2. To identify the key research issues within the proposed general search framework: automated composition of selection heuristics on the population, of evolution operators and of both. (RQ2)

The rest of this chapter is organised as follows. Detailed descriptions of the proposed framework are given in Section 3.2. The key research issues are presented in Section 3.3. Finally, Section 3.4 concludes this chapter.

## 3.2 Proposed General Search Framework

Evolutionary algorithms and metaheuristics in the literature follow a similar underlying philosophy of artificial evolution driven by selection and reproduction. The evolution and search process of a specific metaheuristic is distinguished and mainly depends on the selection heuristics and evolution operators.

Based on the analysis of the basic schemes of metaheuristic algorithms, a general search framework (GSF) has been developed, as illustrated in Figure 3.1. The framework is composed of five modules as shown in Table 3.1 for updating the individuals and four archives as shown in Table 3.3 for storing the individuals. The heuristic/operator sets for the main modules are presented in Tables 3.4. From these heuristic/operator sets, different settings, heuristics or parameters can be chosen, as shown in Tables 3.5-3.7, to automatically

compose and design different general search algorithms within the GSF. Algorithms represented by the combination of selection heuristics and evolution operators are set as the output. Note that GSF is a high-level template for existing evolutionary algorithms. The modules and workflow within GSF are the same as those in traditional evolutionary algorithms. However, the set of algorithmic components in the modules, including heuristics and operators, is different. In other words, any type of evolutionary algorithms, such as GA, is a specific configuration of GSF with predefined algorithmic components.

With respect to Initialisation, although some problem-specific heuristics ($h_p$) have been developed, the majority of existing studies generally adopt a 'purely at random' ($h_r$) strategy. The two most common criteria for Termination are computation time ($h_t$) and population convergence ($h_c$). Of the five modules presented in Figure 3.1, Selection for Evolution, Evolution, and Selection for Replacement contribute more to the search performance. Therefore, they are discussed in detail in the following sections.

Table 3.1: Modules within GSF

| Module | Different heuristics, operators or parameters |
|---|---|
| Initialisation | random ($h_r$), problem-specific ($h_p$) |
| Selection for Evolution | probability-based operators ($h_1, h_2, h_3$), deterministic operators ($h_4, h_5, h_6$) |
| Evolution | mutation ($O_{mutation}$), crossover ($O_{crossover}$) |
| Selection for Replacement | comma-selection ($h_7$), plus-selection ($h_8$) |
| Termination | computation time ($h_t$), convergence ($h_c$) |

The proposed GSF is able to formulate in a unified way a range of single-solution based algorithms and population-based algorithms by setting different parameters for the modules and archives, as shown in Table 3.2, e.g. different population size, the four archives and heuristic sets in the Selection for Evolution module. Note that a single-solution based algorithm can be seen as a special case of a population-based algorithm when population size is set to 1. This thesis focuses on automated design of population-based algorithms, as

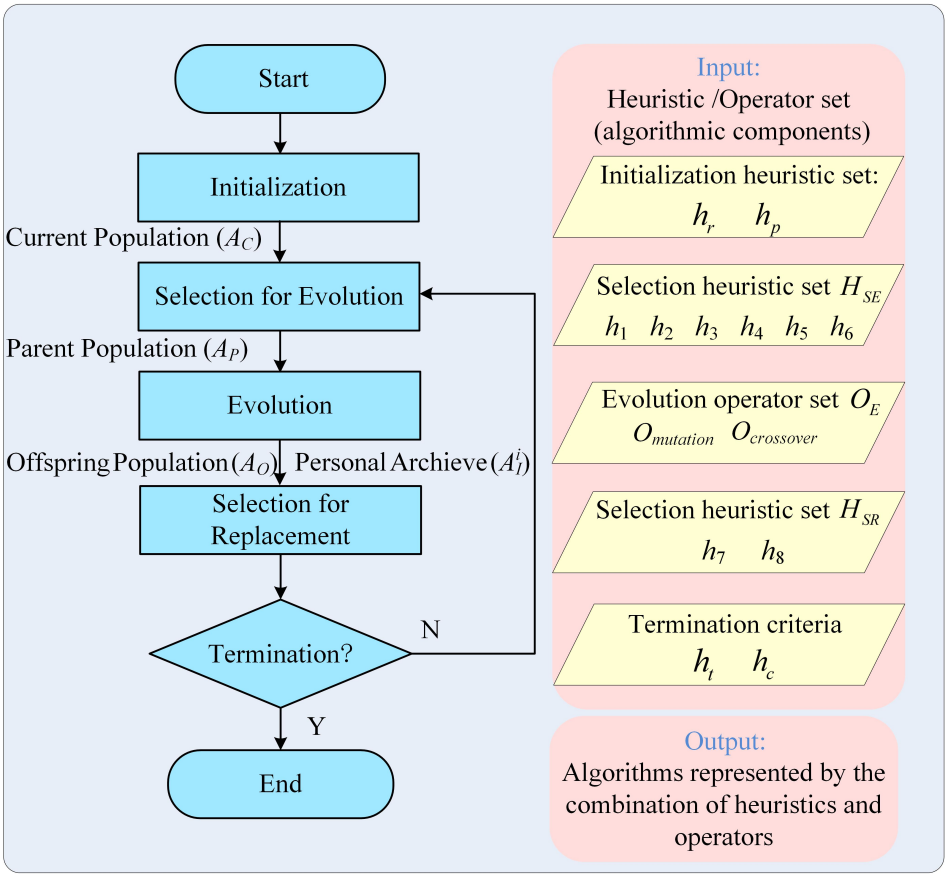



**Figure 3.1:** *General search framework*

Table 3.2: Single-solution based and population-based search algorithms defined with GSF

| Parameters/components | Single-solution based algorithm | Population-based algorithm |
|---|---|---|
| Population size | 1 | $> 1$ |
| Archive | $A_C = A_P = A_O = A_I^i$ | $A_C \neq A_P \neq A_O \neq A_I^i$ |
| Initialisation | $h_r, h_p$ | $h_r, h_p$ |
| Selection for Evolution | $h_4$ | $h_1, h_2, h_3, h_4, h_5, h_6$ |
| Evolution | $O_{mutation}$ | $O_{mutation}, O_{crossover}$ |
| Selection for Replacement | $h_7, h_8$ | $h_7, h_8$ |
| Termination | $h_t, h_c$ | $h_t, h_c$ |

they offer notable advantages in terms of exploration, robustness, parallelisation and adaptability. Firstly, population-based algorithms demonstrate proficiency in conducting global search and exploring extensive search spaces by maintaining a diverse set of solutions. Secondly, they are capable of handling noisy or uncertain objective functions, which is facilitated by the presence of diversity within the population. Thirdly, population-based algorithms can be

parallelised to efficiently utilise computational resources, enabling the simultaneous evaluation or evolution of multiple individuals within the population. Lastly, population-based algorithms possess the capability to adapt the population over time, making them well-suited for dynamic or evolving environments.

Table 3.3: Four population archives within GSF

| Archive | Description |
|---|---|
| $A_C$: Current population, $nPop = \|A_C\|$. | The individuals chosen in the current population before evolution. |
| $A_P$: Parent population, $\mu = \|A_P\|$. | The individuals chosen using heuristic in $H_{SE}$. |
| $A_O$: Offspring population, $\lambda = \|A_O\|$. | The offspring population after evolution $O_E$. |
| $A_I^i$: Personal archive, $i \in \{1, 2. \cdots, nPop\}$. | Personal archive $A_I^i$ for the individual $i^{th}$ to reserve individual trajectories. |

Table 3.3 shows the archives defined within GSF. In the Selection for Evolution module, some individuals within the Current Population archive ($A_C$) are selected and stored in the Parent Population archive ($A_P$) using selection heuristics ($H_{SE}$). The population is updated or evolved by using the evolution operators ($O_E$) in the Evolution module and stored in the Offspring Population archive ($A_O$). The Current Population archive is then generated by adopting the selection heuristics ($H_{SR}$) in the Selection for Replacement module. In addition, every individual has a Personal Archive ($A_I^i$) to reserve individual trajectories.

In GSF, the Selection for Evolution and Selection for Replacement modules select individuals using various heuristics based on the fitness of individuals in the population archives. Without loss of generality, all selection heuristics are set for solving optimisation problems where the aim is to minimise the objective value.

Table 3.4: The heuristic/operator set for the modules within GSF in Figure 3.1

| Heuristic/operator set | Description |
|---|---|
| $H_{SE}$ for module Selection for Evolution | Various heuristics as defined in Table 3.5 to select the parent population $A_P$ from the current population $A_C$. |
| $H_{SR}$ for module Selection for Replacement | Various heuristics as defined in Table 3.6 to update the current population $A_C$ based on $A_P$. |
| $O_E$ for module Evolution | Various operators as defined in Table 3.7 to generate the offspring population $A_O$ based on $A_P$ selected by $H_{SE}$. |

### 3.2.1 Selection for Evolution Module

There are two types of heuristics in the Selection for Evolution module. As Table 3.5 shows, $h_1$, $h_2$ and $h_3$ are probability-based, where individuals are selected as parents according to a probability related to their fitness. $h_4$, $h_5$ and $h_6$ select an individual to be a parent in a deterministic way instead of by probability.

Table 3.5: $H_{SE}$: heuristics in selection for evolution module

| Heuristic | Description |
|---|---|
| $h_1$ | $h_{1_b}^t/h_{1_w}^t$ : tournament selection of the best/worst of $v \in \{1, \cdots, nPop\}$ individuals as parent candidates from $A_P$. The probability of selecting each individual $i$ as parent candidate $p_i' = 1/nPop$. When $v = 1$: random selection. When $v = nPop$: greedy selection of the best/worst individual. |
| $h_2$ | Proportionate roulette wheel selection of an individual $i$ as parent from $A_P$ with a probability proportional to its fitness. |
| $h_3$ | Ranking selection of an individual $i$ as parent according to the probability proportional to the its rank (ascending order based on the fitness function). |
| $h_4$ | Select the current individual itself as parent. |
| $h_5$ | Rank selection of the best previous position as parent based on individual's personal archive $A_I^i$. |
| $h_6$ | Selection of all the individual(s) with a lower fitness than the current individual as parent(s) from $A_P$. |

### 3.2.2 Selection for Replacement Module

After evolution, the population is updated by using the selection heuristic $(h_7, h_8)$ in the Selection for Replacement module, as shown in Table 3.6.

Table 3.6: $H_{SR}$: heuristics in selection for replacement module

| Heuristic | Description |
|---|---|
| $h_7$ | Comma-selection $(nPop, \lambda)$. Select $nPop$ individuals only from the offspring population $A_O$, $\lambda \geqslant nPop, nPop = \|A_C\|, \lambda = \|A_O\|$. |
| $h_8$ | Plus-selection $(nPop, \mu + \lambda)$. Select individuals from both the parent population $A_P$ and the offspring population $A_O$, $nPop = \|A_C\|$, $\mu = \|A_P\|, \lambda = \|A_O\|$. |

### 3.2.3 Evolution Module

Evolution operators $(O_E)$ in the Evolution module include $O_{mutation}$, which operates upon one individual, and $O_{crossover}$, which operates on multiple individuals. Regarding the capacitated vehicle routing problem with time windows (CVRPTW), crossover operators are prone to infeasible solutions. Therefore, in this thesis, we focus on investigating various mutation operators, which are defined in Table 3.7, for solving CVRPTW. Figure 3.2 presents the solution encoding of a CVRPTW with nine customers and three vehicles, and Figure 3.3 further provides the illustration of the evolution operators listed in Table 3.7. Note that these general basic operators (exchange, insert and remove, etc.) can be adapted and adjusted accordingly to automatically design algorithms for different COPs. In this study, CVRPTW is used as a case study to demonstrate the effectiveness of the proposed approach. The generality of RL within the framework for other COPs will be investigated in the future research.

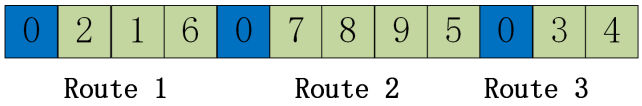



**Figure 3.2:** *Solution encoding of a CVRPTW with nine customers and three vehicles*

Table 3.7: $O_E$: evolution operators for CVRPTW

| Operator | Description |
|---|---|
| $o_{chg\_in}$ | Exchange $m$ and $n$ nodes in the same route in a solution |
| $o_{chg\_bw}$ | Exchange $m$ and $n$ nodes from different routes in a solution |
| $o_{ins\_in}$ | Insert $m$ nodes to the same route in a solution |
| $o_{ins\_bw}$ | Insert $m$ nodes to other positions of different routes in a solution |
| $o_{ruin\_recreat}$ | The $m$ nodes within a pre-determined distance $d$ to the base customer are removed from the solution. The value of $d$ is set based on the distance between the base node and the furthest node from the base node. If there exist feasible routes which can accommodate the removed nodes, the insertion position with the minimum waiting time is selected. Otherwise, a new route is created. |
| $o_{two\_opt}$ | Exchange two nodes in the same route in a solution |
| $o_{two\_opt*}$ | Take the end sections of two routes in a solution and swap them to create two new routes |

## 3.3 Key Research Issues within GSF

Based on the design space on algorithmic components, the automated algorithm design problem can be defined into the following three key research issues:

- **Automated Composition of Evolution Operators:**

  This research issue is to automatically select and apply suitable evolution operators in the Evolution module during the optimisation process while fixing components in other modules within GSF. The decision variables in the search space of algorithms are evolutionary operators.

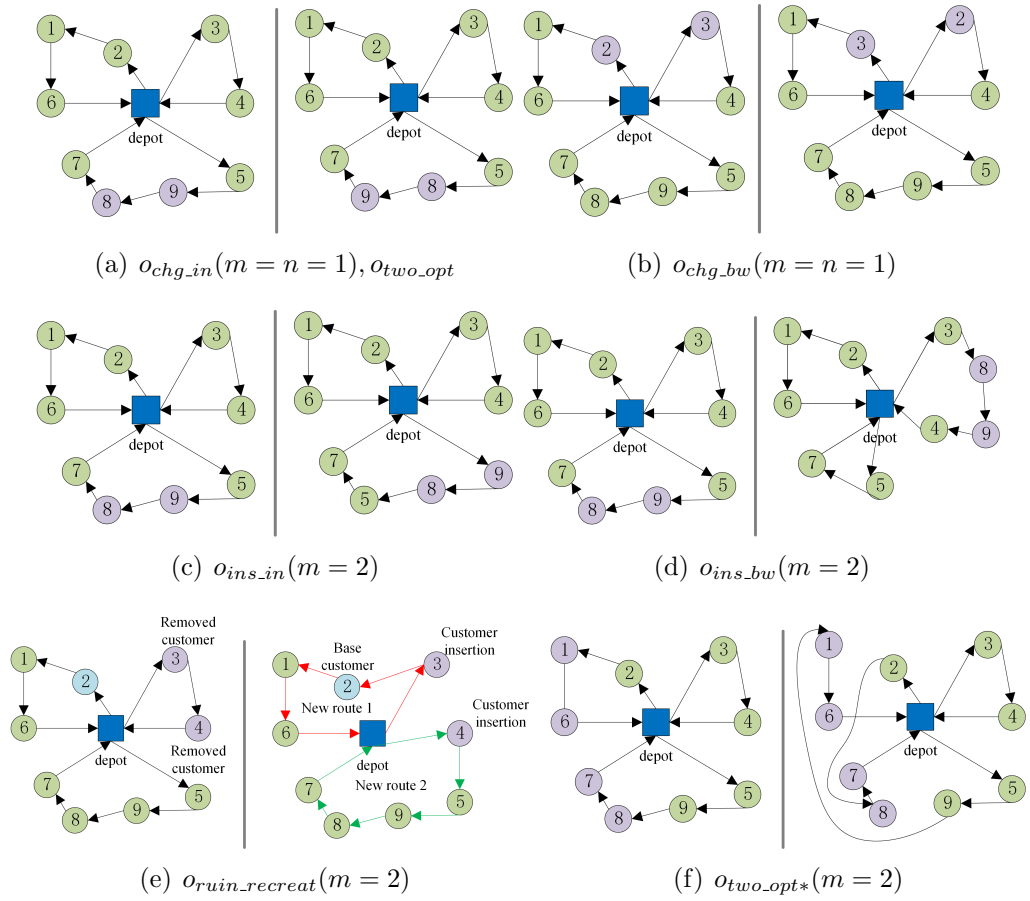
- **Automated Composition of Selection Heuristics:**

  This research issue is to automatically select and apply suitable selection heuristics on the population (decision variables in the search space) in

the Selection for Evolution/Replacement modules within GSF during the optimisation process while fixing components in the Evolution module.

- **Automated Composition of Evolution Operators and Selection Heuristics Simultaneously:**

  The decision variables are defined as pair of evolution operators and selection heuristics, automatically selected and applied during the optimisation process.




(a) $o_{chg\_in}(m = n = 1), o_{two\_opt}$ (b) $o_{chg\_bw}(m = n = 1)$

(c) $o_{ins\_in}(m = 2)$ (d) $o_{ins\_bw}(m = 2)$

(e) $o_{ruin\_recreat}(m = 2)$ (f) $o_{two\_opt*}(m = 2)$

**Figure 3.3:** *Illustration of evolution operators in Table 3.7*

## 3.4 Summary

In this chapter, a general search framework (GSF) is proposed to formulate in a unified way a range of different metaheuristics, including single-solution based algorithms and population-based algorithms. This framework defines generic

algorithmic components, including selection heuristics on the population and evolution operators. The unified general search framework aims to serve as the basis of analysing algorithmic components for automated algorithm design. Three key research issues within GSF have been defined: automated composition of evolution operators, of selection heuristics on the population, and of both. In the remaining chapters of this thesis, Chapter 4 identifies the key features for building successful machine learning to tackle the above-mentioned research issues. With the newly identified features, a set of reinforcement learning techniques have been proposed to investigate the first research issue in Chapter 5 and the other two research issues in Chapter 6.

# Chapter 4

# Feature Identification for Automated Algorithm Design

In Chapter 3, a general search framework to serve as the basic of automatically design search algorithms using machine learning was developed. This chapter focuses on identifying the key features to provide useful and sufficient information about the search space of algorithm design for building successful machine learning approaches.

## 4.1 Introduction

Identifying effective features to characterise the search space of algorithm design and the problem instance plays an important role in the automated algorithm design process. A suitable feature set can help to distinguish different states in reinforcement learning, which is the key to performance improvement. For this purpose, different features have been proposed in the literature to support the new task of automated algorithm design.

In supporting effective automated algorithm design, we identified and categorised existing features for designing general search algorithms into two groups. The first group, namely search-dependent features, is composed of features observing the search, such as the mean and standard deviation of

the population fitness, and the average distance from the best individual [21]. The second group, namely instance-dependent features, consists of the basic characteristics of the problem instances. Another group of common features, namely the solution-dependent features, are associated with the solution encoding scheme, take TSP as an example, the encoding of a complete tour can be directly defined as the state. In many cases, the solution-dependent features cannot be used to develop a general methodology since they are problem-specific. Therefore, it is not included in this thesis.

The present research has been conducted with the following two motivations based on the literature review. From the application perspective, the CVRPTW instances share the same problem structure but differ in instance-dependent features such as the vehicle capacity and customer time windows. However, existing search algorithms treat each problem instance independently, highly depending on extensive and different human expertise in algorithm design. From the methodology perspective, utilising machine learning techniques to assist effective algorithm design is still at a preliminary stage, albeit with some successful attempts. One of the important issues is how to identify the key features that accurately characterise the search space for building successful machine learning approaches. Although various features have been extracted for effective algorithm design in the literature, there is a lack of a systematic investigation analysing the extracted features within a consistent and general framework. The aim of this chapter is therefore to identify and analyse feature sets which provide helpful and sufficient information on the state of the evolutionary search, and to verify the effectiveness of the identified features in assisting algorithm design with the support of machine learning. More specifically, we aim to answer the following research questions (RQ):

- *RQ1* What kind of features can be identified to capture useful information for assisting effective algorithm design?

- *RQ2* Is machine learning effective to utilise such information and to au-

tomatically design effective search algorithms with less human involvement?

- *RQ3* Are there search patterns that we can observe from the automatically designed algorithms to derive new knowledge in evolutionary computation?

The goal of this chapter is to identify key features to assist algorithm design within the general search framework. Specifically, this chapter addresses the following research objectives:

1. Identify two groups of features, i.e. search-dependent and instance-dependent features, to capture useful information for assisting effective algorithm design. (RQ1)

2. Devise a reinforcement learning based model to extract useful knowledge hidden in the data collected during the optimisation process and investigate the impact of identified features on the proposed learning model. (RQ2)

3. Analyse the search patterns of the automatically designed algorithms, i.e. the utilisation and transition of algorithmic components, to further provide insights into reusing knowledge extracted in algorithm design using machine learning in solving new problem instances. (RQ3)

The rest of this chapter is organised as follows. Detailed descriptions of the identified features are presented in Section 4.2. The proposed reinforcement learning method is shown in Section 4.3. The experiments and discussions are shown in Section 4.4. Finally, Section 4.5 concludes this chapter.

## 4.2 Identified Features

Identifying appropriate features is a crucial step to conduct effective reinforcement learning for automated algorithm design. Different features provide

different aspects of essential information, having a significant impact on the performance of the automated algorithm design methods. In this section, two groups of features (i.e. search-dependent and instance-dependent features) are identified for developing a general methodology to solve different CVRPTW instances.

### 4.2.1 Search-dependent Features

To assist algorithm design, ten search-dependant key features have been extracted from the data collected during the optimisation process. Related symbols are shown in Table 4.1 for the definitions of search-dependent features listed in Table 4.2.

Table 4.1: Symbols used for defining search-dependent features in Table 4.2

| Symbol | Description |
|---|---|
| $f_i$ | The fitness value of the $i^{th}$ individual |
| $\bar{f}$ | The average fitness value of the population |
| $N$ | The size of the population |
| $P$ | Population |
| $I$ | The initial population |
| $C$ | The current population |

Features in Table 4.2 capture the characteristics of the search process from different aspects. With this information on the intensification and diversification of the search, the learning agent can be guided toward making better decisions on selection of algorithmic components during the search, considering the balance between searching beyond the area currently being explored (i.e. exploration) and focusing on the already explored area (i.e. exploitation). Such balance has a significant impact on the performance of the designed search algorithms. Note that the focus of this section is not to propose new features to state representation, but rather to identify effective features from existing literature and then systematically verify the impact of these features on the automated algorithm design task. The majority of the features in Table 4.2 are derived from existing studies. For example, $f_1$ is derived from [119], $f_2$ is

Table 4.2: Search-dependent features

| Features | Description |
|---|---|
| $f_1$: search stage $S$ | indicates which stage the learning agent is in |
| $f_2$: fitness improvement $FI = \frac{\sum_{i \in I} f_i - \sum_{j \in C} f_j}{\sum_{i \in I} f_i}$ | evaluates the quality of fitness value between the current population and the initial population |
| $f_3$: std of fitness $std\,(f) = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} \left(f_i - \bar{f}\right)^2}$ | evaluates the bumpiness of the fitness space by measuring each value's deviation |
| $f_4$: mean of fitness $\bar{f} = \frac{1}{N} \sum_{i=1}^{N} f_i$ | evaluates the general fitness value of the fitness space |
| $f_5$: skewness of fitness $\gamma_1\,(f) = E\left\{\left[\frac{\left(f_i - \bar{f}\right)}{std(f)}\right]^3\right\}$, $i = 1, \cdots, N$ | measures the lack of symmetry of the fitness space |
| $f_6$: kurtosis of fitness $\gamma_2\,(f) = \frac{E\left[\left(f_i - \bar{f}\right)^4\right]}{\left(E\left[\left(f_i - \bar{f}\right)^2\right]^2\right)}$, $i = 1, \cdots, N$ | measures the fitness space relative to the normal distribution |
| $f_7$: amplitude of fitness $Amp\,(P) = \frac{N \cdot \left(\max\limits_{i \in I} f_i - \max\limits_{i \in P} f_i\right)}{\sum_{i \in P} fi}$, $\triangle Amp = \frac{\lvert Amp(C) - AMP(I) \rvert}{Amp(I)}$ | evaluates the degree of the search space altitude on the basis of difference between the upper and lower bound of fitness values |
| $f_8$: Q1 of fitness (25%) | represents the lower quartile of fitness value |
| $f_9$: Q2 of fitness (50%) | represents the median quartile of fitness value |
| $f_{10}$: Q3 of fitness (75%) | represents the upper quartile of fitness value |

derived from [21] [119], $f_3$ and $f_4$ are derived from [21] [23].

### 4.2.2 Instance-dependent Features

Instance-dependent features assist algorithm design with the basic information on problem instances. Values of these features are determined and will not change in algorithm design once an instance is given. Related symbols are shown in Table 4.3 for the definitions of instance-dependent features.

Table 4.3: Symbols for instance-dependent features in Table 4.4

| Symbol | Description |
|---|---|
| $i$ | Customer $i$ |
| $n$ | The number of customers |
| $q_i$ | The demand for customer $i$ |
| $s_i$ | The service time for customer $i$ |
| $r_i$ | The ready time for customer $i$ |
| $d_i$ | The due date for customer $i$ |

Table 4.4: Instance-dependent features

| Features | Description |
|---|---|
| $f_{11}$: vehicle number $V$ | the number of available vehicles |
| $f_{12}$: capacity $Q$ | vehicle capacity |
| $f_{13}$: demand $\sum_{i=1}^{n} \frac{q_i}{n}$ | average customer demand |
| $f_{14}$: service $\sum_{i=1}^{n} \frac{s_i}{n}$ | average service time |
| $f_{15}$: time-window $TW = \frac{\left(\sum_{i=1}^{n} d_i - \sum_{i=1}^{n} r_i\right)}{n}$ | average time-window size |
| $f_{16}$: time-window overlaps $\sum_{i=1}^{n} \sum_{j=1}^{n} \frac{O_{ij}}{n}$, $O_{ij} = \frac{inter(i,j)}{union(i,j)}$ | average time-window overlaps between customers |
| $f_{17}$: time-window density $D$ | the percentage of time-constrained customers (25%, 50%, 75%, 100%) |

In Table 4.4, average time-window overlaps are calculated based on Equation (4.1) and Equation (4.2) as proposed in [22] to measure the relationships between the time-windows of all customer in a given instance.

$$inter\left(i,j\right)=\begin{cases} I=\min\left\{d_i,d_j\right\}-\max\left\{r_i,r_j\right\},I>0 \\ 0,otherwise \end{cases} \tag{4.1}$$

$$union\left(i,j\right)=\begin{cases} \max\left\{d_i,d_j\right\}-\min\left\{r_i,r_j\right\},inter\left(i,j\right)>0 \\ \left(d_i-r_i\right)+\left(d_j-r_j\right),otherwise \end{cases} \tag{4.2}$$

## 4.3 Proposed Reinforcement Learning Method

Figure 4.1 depicts the overall research framework of the proposed reinforcement learning method in the context of automated algorithm design. This chapter involves the use of a reinforcement learning method to automatically generate search algorithms for solving different CVRPTW instances. Specifically, the overall research framework has been defined in five steps. The first three steps are related to the definition of three key components of the RL method (i.e. state, action and reward scheme), shown in Sections 4.3.1-4.3.3. Note that one of the focuses of this study is on identifying the key features to capture and characterise the search space of algorithm design, details of these identified features are discussed in Section 4.2. This presents new contributions to the existing literature. The fourth step defines the main optimisation process, i.e. automatically generating search algorithms based on RL. The final step is related to solving the CVRPTW instances using the generated search algorithms in Step 4.

This chapter aims to address the key issue of automated algorithm design rather than considering the whole design space. Based on the general search framework [142], the key focus of algorithm design is on selecting and composing evolution operators at different stages during the search process. An actor-critic proximal policy optimisation method is used to determine the suitable evolution operators of the search algorithm.

For the task of automated algorithm design, due to the random characteristics, the stochastic policy generated by policy-based RL is better than the

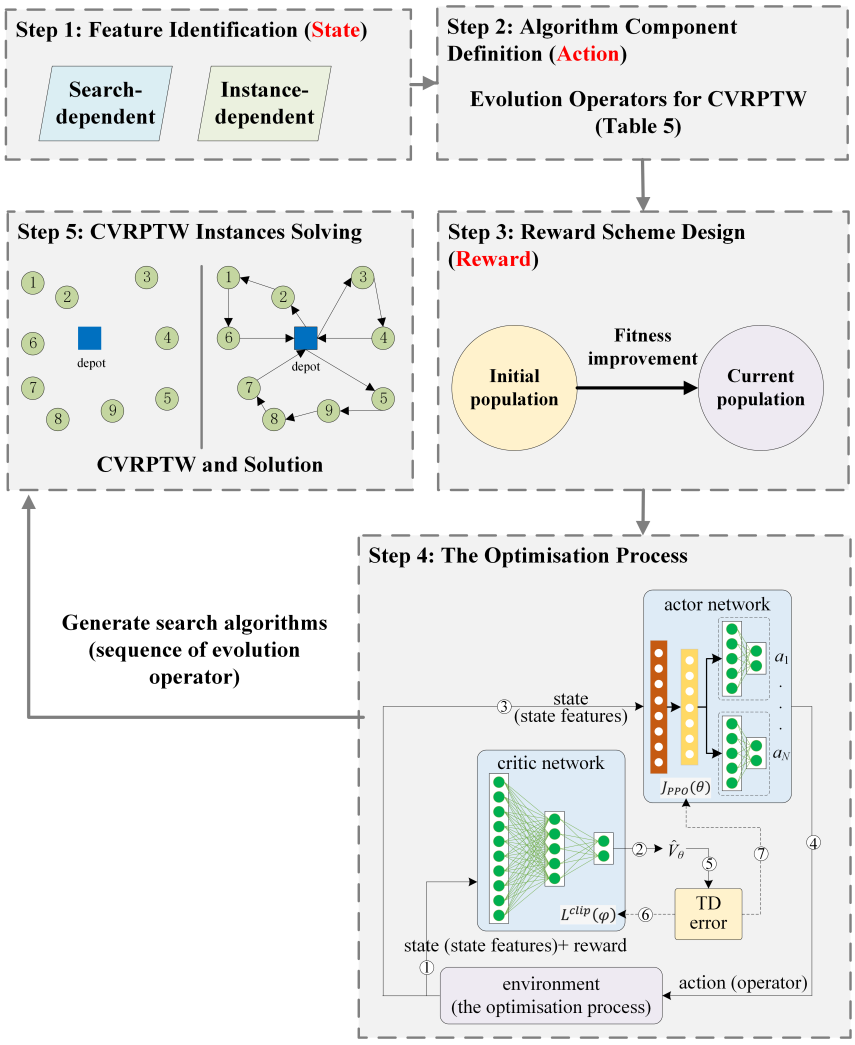



**Figure 4.1:** *Research framework of the proposed reinforcement learning method in the context of automated algorithm design*

deterministic policy generated by value-based RL. The Proximal Policy Optimisation (PPO), which is a policy-based RL method, is used to obtain the optimal stochastic policy. However, the policy of the original PPO is trained per episode, resulting in a slow convergence. Therefore, the PPO method in this section is modified with an Actor-Critic architecture to generate a stochastic policy trained per timestep rather than per episode with a clipped objective to determine evolution operators for automated algorithm design.

Table 4.5: Notations used in PPO methods in Figure 4.1

| Notation | Description |
|---|---|
| $t$ | current timestep |
| $s_t$ | state at timestep $t$ |
| $a_t$ | selected action at timestep $t$ |
| $r_t$ | reward value at timestep $t$ |
| $\lambda$ | parameter of $\lambda$-step return |
| $V$ | state value function |
| $\theta$ | parameter of the actor neural network |
| $\varphi$ | parameter of the critic neural network |

The overview of the actor-critic PPO method is shown in Figure 4.1 (Step 4). Related notations are given in Table 4.5.

The reward and state (defined by the identified features) (①) obtained from the environment is taken as the input of the critic neural network, the output of which is the state-action value $\hat{V}_\theta$ (②). At the same time, state (③) is also taken as the input of the actor neural network. The corresponding output is the probability of each action (i.e. operator) (④). The chosen action is executed (④) in the environment (i.e. the optimisation process), and the next loop is started.

The temporal difference error (TD-error) is calculated (⑤) by the state-action value $\hat{V}_\theta$ (②). The actor neural network is updated by $J_{PPO}(\theta)$ (⑦), as shown in Equation (4.3), and the critic neural network is updated based on the calculation of $L^{clip}(\varphi)$ (⑥), as shown in Equation (4.6).

The output of the learned policy is a probability distribution of actions (i.e. operators). The sequential sample results of one episode are the composition

operators in the automated algorithm design. The action (i.e. operator) obtained (④) by the actor neural network is executed in the environment (i.e. the optimisation process).

The objective function of the actor neural network $J_{PPO}(\theta)$ is shown in Equation (4.3).

$$J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta\left(a_t \left| s_t\right.\right)}{\pi_{old}\left(a_t \left| s_t\right.\right)} A_t \tag{4.3}$$

where $A_t$ is the advantage function representing the gap of the currently selected action relative to the average of all actions, such that:

$$A_t = \delta_t + (\gamma\lambda)\,\delta_t + \cdots + (\gamma\lambda)^{T-t+1}\,\delta_{T-1} \tag{4.4}$$

$$\delta_t = r_t + \gamma V\left(s_{t+1}\right) - V\left(s_t\right) \tag{4.5}$$

$A_t$ is obtained by performing a complete episode when there is only an actor neural network. In the critic neural network, the value function can be estimated at every timestep, so the update process (⑦) of the actor neural network is accelerated compared to that in the original PPO.

The critic neural network $L^{clip}(\varphi)$ aims at minimising the loss function, as shown in Equation (4.6).

$$L^{clip}\left(\varphi\right) = E_t\left\{\min\left[r_t\left(\varphi\right)\cdot A_t, clip\left[r_t\left(\varphi\right), 1-\epsilon, 1+\epsilon\right]\cdot A_t\right]\right\} \tag{4.6}$$

Here, $\epsilon$ is the clipping probability ratio. With the lower bound, the 'min' operator, the trained policy increases monotonically with a low computing requirement.

### 4.3.1 State Representation

The search-dependent features in Table 4.2 and instance-dependent features in Table 4.4, are used to define the state space.

### 4.3.2 Action Representation

The action space is defined by the set of evolution operators in Table 3.7, which are the algorithmic components of the Evolution module in the proposed general search framework (GSF). Refer to Chapter 3 for more information of the GSF and its basic modules. The main task of the proposed PPO method is to address the key issue of automated selection and combination of the most efficient evolution operators during different optimisation stages.

### 4.3.3 Reward Scheme

A reward scheme is used for the learning agent to determine whether the selected action is appropriate, thus, is very important for an RL method. As shown in Equation (4.7) and Equation (4.8), the reward in the proposed PPO model is calculated based on the fitness improvement of the current population over the initial population. A higher reward is assigned to the same fitness improvement when population fitness is optimised above a certain threshold $a$.

$$r = \frac{f_{current}}{f_{initial}} \tag{4.7}$$

$$reward = \begin{cases} -r, & if\ r > a \\ -r - \log_{10}(r), & if\ r \leqslant a \end{cases} \tag{4.8}$$

Two methods are used in setting the reward: normalise $r$ to increase the training efficiency; assign a larger reward by using a log function to the same fitness improvement in the later stage of the optimisation process.

Many of the simple positive/negative reward schemes in the literature track the fitness improvement by counting the number of steps achieved successfully. The proposed reward scheme is designed to instead maximise the total fitness improvement itself, which is what really needs to be optimised in the newly defined machine learning task. Compared to the simple positive/negative re-

ward schemes, the proposed reward scheme not only reflects but also measures the positive/negative impact of the selected action. Moreover, the proposed reward scheme assigns a larger reward to the actions that lead to fitness improvements at the later stage of the optimisation process, to address the issue that such improvements are usually very small at the final stage of evolution.

### 4.3.4 Episode Setting

An episode is defined as the whole optimisation process. Since the time-based stopping criteria is used in this thesis, the period of each episode equals to the given optimisation time $t_{\max}$. An episode is divided into $NoT$ timesteps, so the period of each timestep equals to $t_{\max}/NoT$.

For training purposes, the proposed RL-based method is executed for $NoE$ episodes. For testing purposes, the designed RL-based method is executed for one episode.

## 4.4 Experiments and Discussions

The experimental analysis aims to address two research issues: (1) verifying the impact of the identified features on the reinforcement learning model; (2) analysing the search patterns of the best designed algorithm compositions. In assessing the impact of the identified features, two learning models with different features are trained and tested. In the analysis of search patterns, the utilisation and transition of algorithmic components are included. The results of the proposed method are compared with the best-known results in the literature. The details of the selected VRPTW benchmark dataset are shown in Table 4.6.

Noted that it is usually not possible to compare design time of automated methods and manually design methods since this information is usually not reported in the published papers. Some of them only published their results and did not provide the application time. As shown in Table 4.7, a direct

Table 4.6: The selected VRPTW benchmark dataset

| Instance | Vehicle capacity | Density of the time windows | Best-known solution identified by heuristics [143] |
|---|---|---|---|
| C101 | 200 | 100% | 10828.94 [144] |
| C102 | 200 | 75% | 10828.94 [144] |
| C103 | 200 | 50% | 10828.94 [144] |
| C104 | 200 | 25% | 10824.78 [144] |
| C201 | 700 | 100% | 3591.56 [144] |
| C202 | 700 | 75% | 3591.56 [144] |
| C203 | 700 | 50% | 3591.17 [144] |
| C204 | 700 | 25% | 3590.6 [144] |
| R101 | 200 | 100% | 20645.79 [145] |
| R102 | 200 | 75% | 18486.12 [144] |
| R103 | 200 | 50% | 14292.68 [146] |
| R104 | 200 | 25% | 10007.24 [147] |
| R201 | 1000 | 100% | 5252.37 [148] |
| R202 | 1000 | 75% | 4191.7 [149] |
| R203 | 1000 | 50% | 3939.54 [150] |
| R204 | 1000 | 25% | 2825.52 [151] |
| RC101 | 200 | 100% | 15696.94 [152] |
| RC102 | 200 | 75% | 13554.75 [152] |
| RC103 | 200 | 50% | 12261.67 [153] |
| RC104 | 200 | 25% | 12135.487 [154] |
| RC201 | 1000 | 100% | 5406.91 [147] |
| RC202 | 1000 | 75% | 4367.09 [155] |
| RC203 | 1000 | 50% | 4049.62 [155] |
| RC204 | 1000 | 25% | 3798.41 [147] |

comparison on the computational expenses between the proposed automated methods and the manually design methods which produce the best-known solutions is unfair due to the different computing platforms and implementation languages. Furthermore, the termination condition and the number of independent runs differs from methods to methods in most of the published algorithms. The proposed methods require extra computation time on the training and testing process compared to other methods. However, the aim is not to develop a fast method but rather to automatically develop search algorithms that can produce state-of-the-art results with a higher degree of generality. The extra time can potentially be compensated by solving different problem instances without redesigning or fine-tuning algorithms in the long-term.

All experiments have been conducted using a computer with Intel(R) Xeon(R) W-2123 CPU@ 3.60 GHz processors, and with 32.0 GB of memory. The proposed methods are implemented in Java environment with IntelliJ IDEA

2020.3.3 as the development tool. Java is chosen since it is also used in other widely adopted frameworks in relevant literature, e.g. HyFlex for hyper-heuristics, thus supports flexible further extensions in future work. In the proposed RL-based method, the population size, the number of timesteps $NoT$ and pre-defined maximum running time of one episode $t_{\max}$ are set to 100, 50, and 600s, respectively. For training the policy, the number of episodes $NoE$ is set to 500.

Table 4.7: Setting of the comparison algorithms

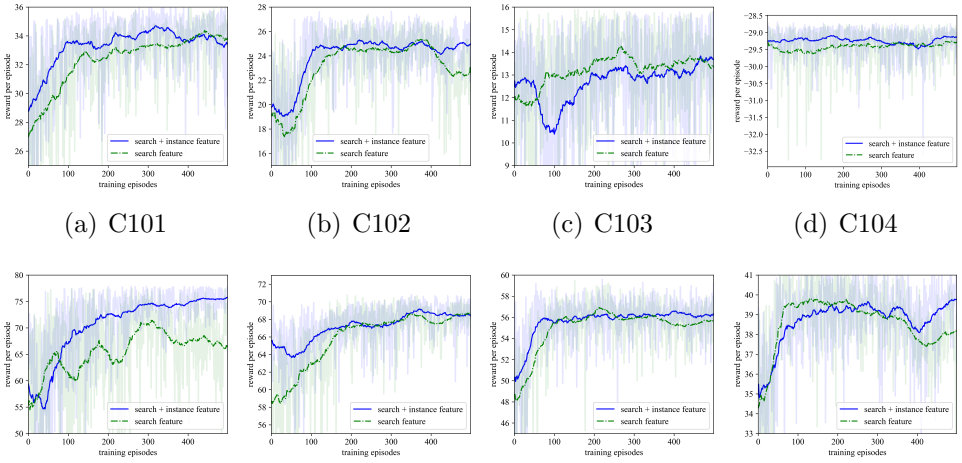
| References | Computer Platform | Language | Termination condition | Time/Iterations |
|---|---|---|---|---|
| RT [144] | Silicon Graphics Indigo 100 | - | maximum iterations | 3200 sec (type-C1)<br>7200 sec (type-C2)<br>2700 sec (type-R1) |
| H [145] | Pentium 400 | C | maximum iterations | - |
| HG [148] | Pentium 200 | C | maximum time | 10 runs<br>13 min |
| RGP [149] | Sun Ultra10 | - | - | 11000 sec |
| TBGGP [152] | Sun Sparc 10 | - | maximum iterations | 11264 sec (type-RC1) |
| BBB [156] | Pentium 400 | C++ | maximum time | 1800 sec |
| MBD [147] | Pentium 800 | Visual Basic | - | 1 run, 43.8 min |
| GCC [155] | Origin 2000 and<br>Sun Enterprise 6500 | C | maximum iterations | 1000 iterations |
| Proposed methods | Intel Xeon W-2123<br>CPU@ 3.60 GHz | Java | maximum time | 6000 sec |

### 4.4.1 Effectiveness of the Identified Features

To verify the effectiveness of instance-dependent features, we train the PPO model with only search-dependent features. It is meaningless to train the PPO model with only instance-dependent features, due to the fact that the values of instance-dependent features are determined once an instance is given and do not change during the training process. The performance of the PPO model with both search-dependent features and instance-dependent features is recorded for comparison.

Influence of different feature sets on the PPO model during the training process on different types of CVRPTW instances is shown in Figures 4.2-4.4, respectively. The green line represents the training results using only search-dependent features as inputs, while the blue line represents the training results using both search-dependent and instance-dependent features.

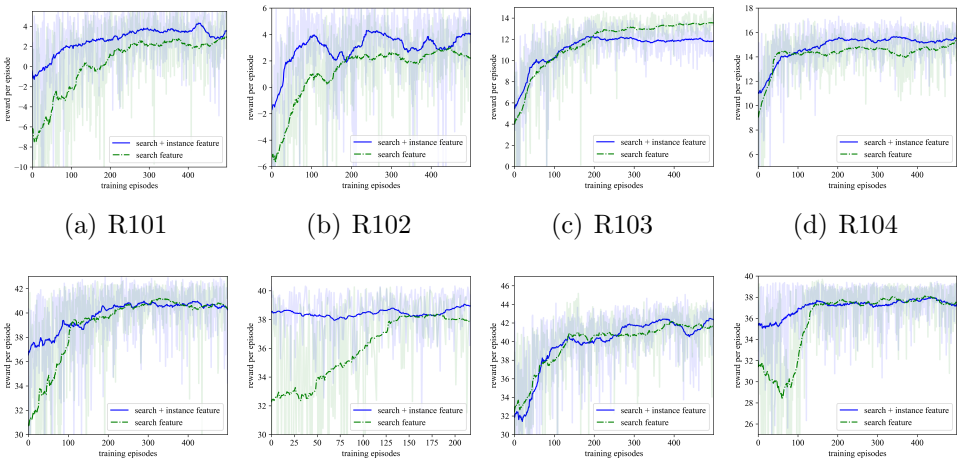
In most instances, the performance of the learning model during the training process, i.e. the expected accumulated reward of the policy, deteriorates without the instance-dependent feature set, except on instances C103, R103 and RC203. This indicates that the instance-dependent features can provide useful information to the learning process, by assisting the population to accurately determine the resulting state with better action choice.




(a) C101 (b) C102 (c) C103 (d) C104

(e) C201 (f) C202 (g) C203 (h) C204

**Figure 4.2:** *Influence of different feature sets on the learning model during training (type-C)*

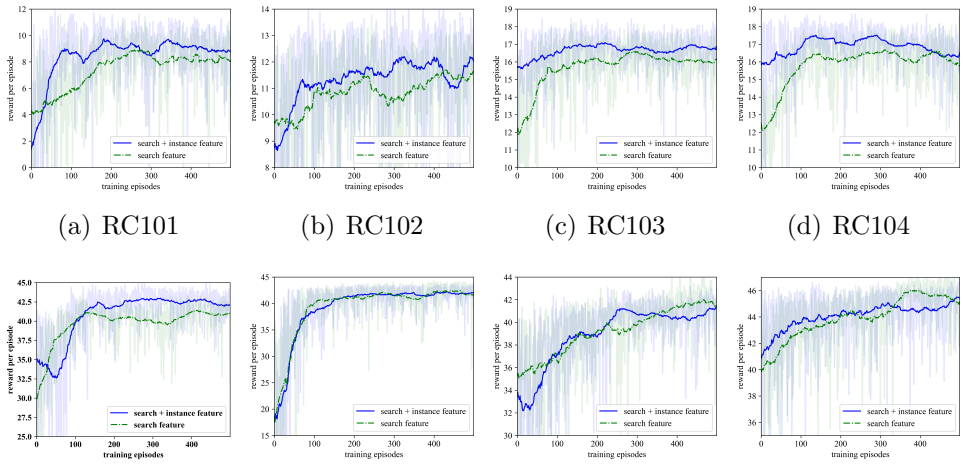



(a) R101 (b) R102 (c) R103 (d) R104

(e) R201 (f) R202 (g) R203 (h) R204

**Figure 4.3:** *Influence of different feature sets on the learning model during training (type-R)*

In the testing process, the same conclusion is reached that the instance-dependent feature set is effective for learning algorithm design, as shown in Table 4.8.




(a) RC101 (b) RC102 (c) RC103 (d) RC104

(e) RC201 (f) RC202 (g) RC203 (h) RC204

**Figure 4.4:** *Influence of different feature sets on the learning model during training (type-RC)*

When instance-dependent features are included, the four performance indicators, i.e. the average fitness value (AVG), the standard deviation of fitness value (SD), the best fitness value within 10 runs (BEST), and the gap between BEST and the best-known solution in the literature (GAP), achieve better values in most instances. Both learning models (i.e. one with both feature sets, and one with only search-dependent features) achieve quite similar BEST to the current best-known results in the literature (i.e. the GAP values are less than 5% in most instances), which verifies the effectiveness of the proposed PPO models. Noted that the aim of automated algorithm design is not trying to beat all the other manually designed metaheuristics but to develop an effective search algorithm without too much human involvement.

### 4.4.2 Search Pattern Analysis of the Best Automatically Designed Algorithms

**Utilisation of Algorithmic Components:**

Figures 4.5-4.7 show the proportion of each operator called in the PPO model with search-dependent and instance-dependent features during the training process, while Figures 4.8-4.10 show the utilisation of operators in the PPO

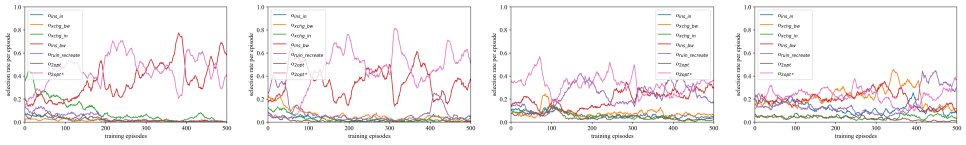
Table 4.8: Performance of the algorithms with different features during testing

| Instance | Search-dependent + Instance-dependent features | | | | Search-dependent features | | | |
|---|---|---|---|---|---|---|---|---|
| | AVG | SD | BEST | GAP | AVG | SD | BEST | GAP |
| C101 | **10828.94** | **0** | **10828.94** | **0** | **10828.94** | 1.8E-12 | **10828.94** | **0** |
| C102 | **10829.40** | **0.93** | **10828.94** | **0** | 10829.93 | 2.011 | **10828.94** | **0** |
| C103 | **10856.97** | **12.07** | **10837.15** | **0.08%** | 10858.79 | 17.80 | 10839.29 | 0.1% |
| C104 | **10916.01** | 20.64 | **10887.99** | **0.58%** | 10921.54 | 16.28 | 10896.9 | 0.67% |
| C201 | **3591.56** | **4.55E-13** | **3591.56** | **0** | **3591.56** | **4.55E-13** | **3591.56** | **0** |
| C202 | **3591.56** | 4.55E-13 | **3591.56** | **0** | **3591.56** | **2.4E-10** | **3591.56** | **0** |
| C203 | **3592.67** | **4.48** | **3591.17** | **0** | 3592.90 | 3.66 | **3591.17** | **0** |
| C204 | 3613.71 | 9.35 | 3599.58 | 0.25% | **3613.05** | **8.95** | **3598.93** | **0.23%** |
| R101 | **20658.64** | **3.60** | **20653.64** | **0.04%** | 20659.79 | 3.85 | 20654.3 | 0.04% |
| R102 | **18519.28** | **18.02** | **18499.30** | **0.07%** | 18621.61 | 293.55 | 18511.18 | 0.14% |
| R103 | 15048.02 | 416.20 | **14365.01** | **0.51%** | 15256.21 | **11.75** | 15228.24 | 6.5% |
| R104 | **11090.22** | 23.09 | **11051.22** | **10.43%** | 11097.74 | **15.93** | 11063.67 | 10.56% |
| R201 | **5320.51** | **13.56** | 5290.00 | 0.72% | 5320.84 | 23.52 | **5288.58** | **0.69%** |
| R202 | **5003.77** | **341.68** | **4307.68** | **2.77%** | 5163.65 | 11.86 | 5141.74 | 22.66% |
| R203 | 4036.05 | **10.23** | 4016.02 | 1.94% | **4035.76** | 16.50 | **4010.36** | **1.8%** |
| R204 | **3732.08** | 270.78 | **2920.27** | **3.35%** | 3828.59 | **5.09** | 3822.51 | 35.29% |
| RC101 | **16839.69** | **300.13** | 16703.71 | 6.41% | 16896.18 | 396.64 | **16677.99** | **6.25%** |
| RC102 | **15431.22** | **294.17** | **14550.96** | **7.35%** | 15545.00 | 13.85 | 15528.14 | 14.56% |
| RC103 | **13152.01** | 294.17 | **12356.41** | **0.77%** | 13359.17 | 15.04 | 13329.82 | 8.71% |
| RC104 | 12233.65 | 388.17 | **11248.41** | **1.01%** | 12131.09 | 281.68 | 11287.67 | 1.37% |
| RC201 | **5492.96** | 18.48 | **5455.76** | **0.90%** | 5505.30 | 12.14 | 5482.79 | 1.40% |
| RC202 | 5287.89 | 29.74 | **5226.54** | **19.68%** | **5261.77** | **12.14** | 5242.94 | 20.06% |
| RC203 | 4191.46 | 22.89 | 4165.96 | 2.87% | 4180.07 | 18.61 | 4137.23 | 2.16% |
| RC204 | **3877.75** | **17.78** | **3851.04** | **1.39%** | 3879.58 | **9.79** | 3863.21 | 1.71% |

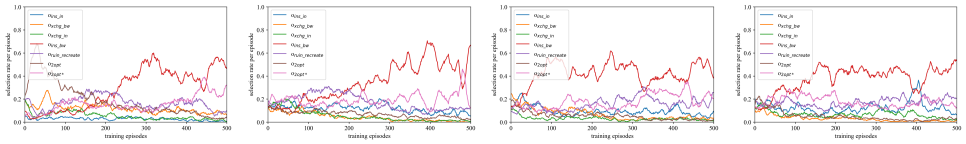
models with only search-dependent features. Both PPO models identify *ins_bw* and 2*opt*∗ as the most frequently selected operators in the best designed algorithms. *ins_bw* is selected most often by both PPO models, although this phenomenon is more obvious in the type-R and type-RC instances. Although the operators with a high frequency of combination are quite similar in both PPO models, the specific utilisation rates of each operator during each episode are different, indicating that the algorithm compositions obtained by these two PPO models (i.e one with only search-dependent features and one with both search-dependent and instance-dependent features) are different.

**Transition of Algorithmic Components:**

An analysis of the best designed algorithms that are automatically designed by the PPO models with different feature sets is analysed in this section. Figures 4.11-4.13 show the transition pattern of operators in the best designed algorithm compositions obtained by the PPO model with both feature sets, including the number of operators, the proportion of each operator and the number of transitions between operators during 50 timesteps. For example,

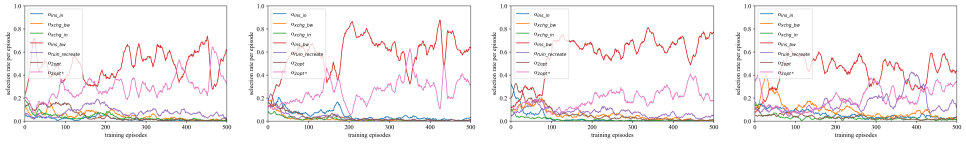


(a) C101 (b) C102 (c) C103 (d) C104

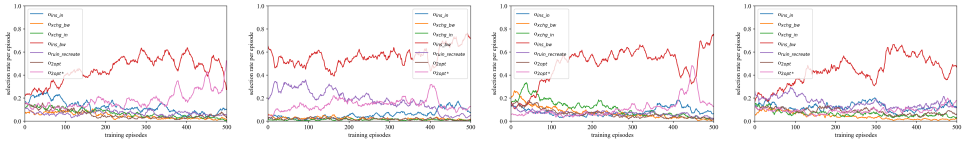


(e) C201 (f) C202 (g) C203 (h) C204

**Figure 4.5:** *Utilisation of operators during training (type-C, with search-dependent and instance-dependent features)*

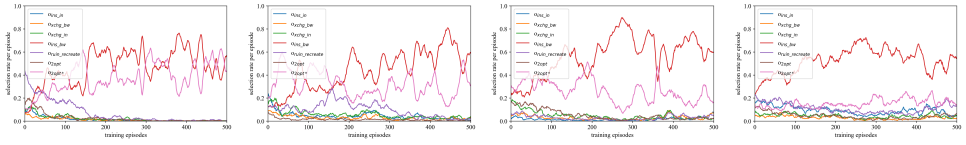


(a) R101 (b) R102 (c) R103 (d) R104

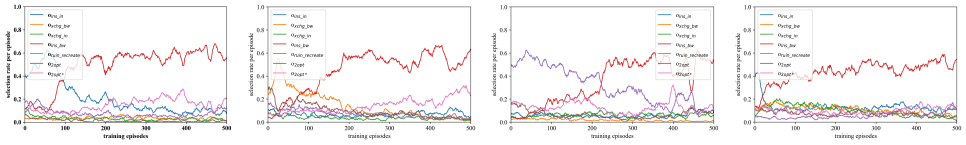


(e) R201 (f) R202 (g) R203 (h) R204

**Figure 4.6:** *Utilisation of operators during training (type-R, with search-dependent and instance-dependent features)*



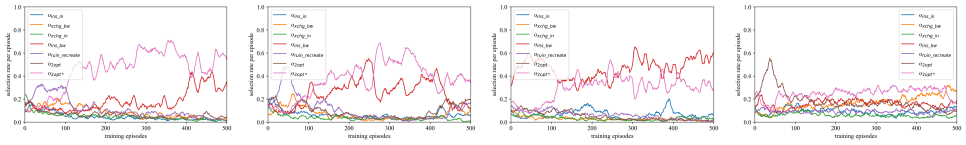
(a) RC101 (b) RC102 (c) RC103 (d) RC104



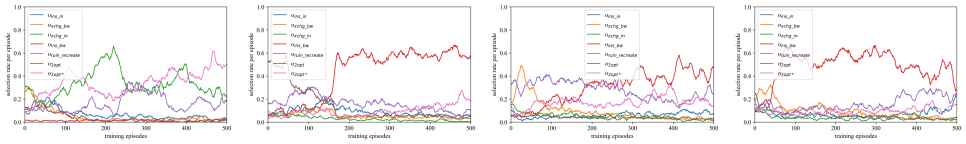
(e) RC201 (f) RC202 (g) RC203 (h) RC204

**Figure 4.7:** *Utilisation of operators during training (type-RC, with search-dependent and instance-dependent features)*

for C101 instance in Figure 4.11, only two out of seven operators are called, namely 54.0% for the $2opt*$ operator and 46% for the *ins_bw* operator with the corresponding number of transitions between operators (28). Figures 4.14-4.16

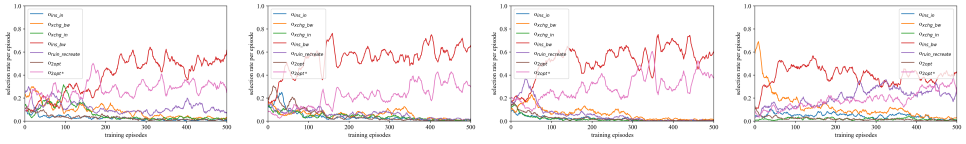


(a) C101 (b) C102 (c) C103 (d) C104

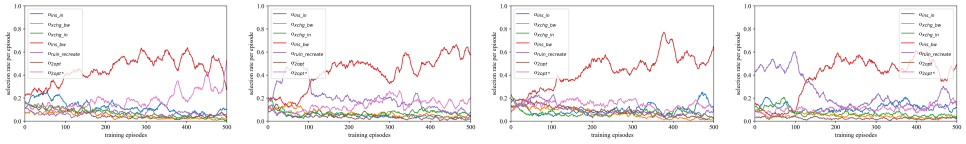


(e) C201 (f) C202 (g) C203 (h) C204

**Figure 4.8:** *Utilisation of operators during training (type-C, with only search-dependent features)*

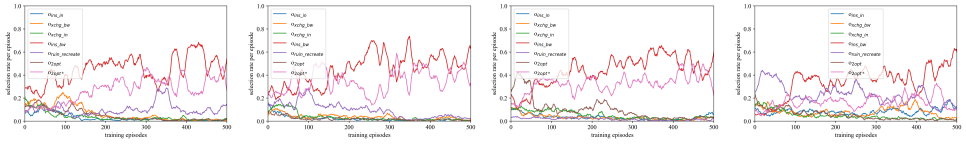


(a) R101 (b) R102 (c) R103 (d) R104

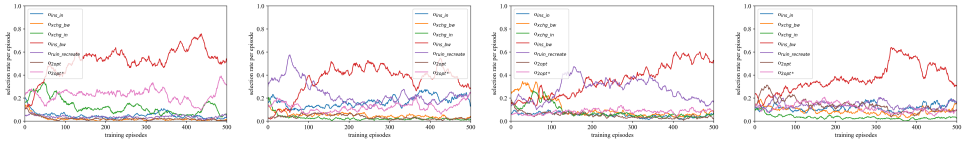


(e) R201 (f) R202 (g) R203 (h) R204

**Figure 4.9:** *Utilisation of operators during training (type-R, with only search-dependent features)*



(a) RC101 (b) RC102 (c) RC103 (d) RC104

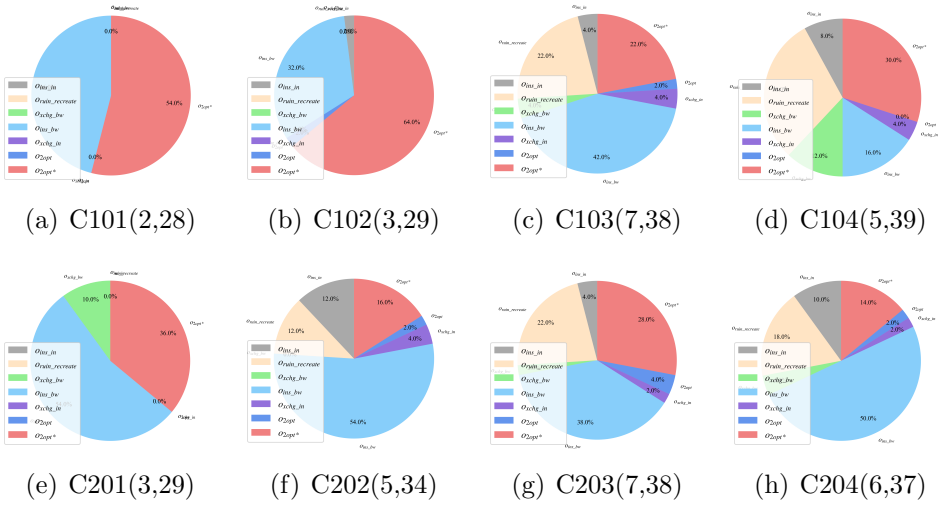


(e) RC201 (f) RC202 (g) RC203 (h) RC204

**Figure 4.10:** *Utilisation of operators during training (type-RC, with only search-dependent features)*

present the transition patterns of the best designed algorithms obtained by the PPO model with only search-dependent features.

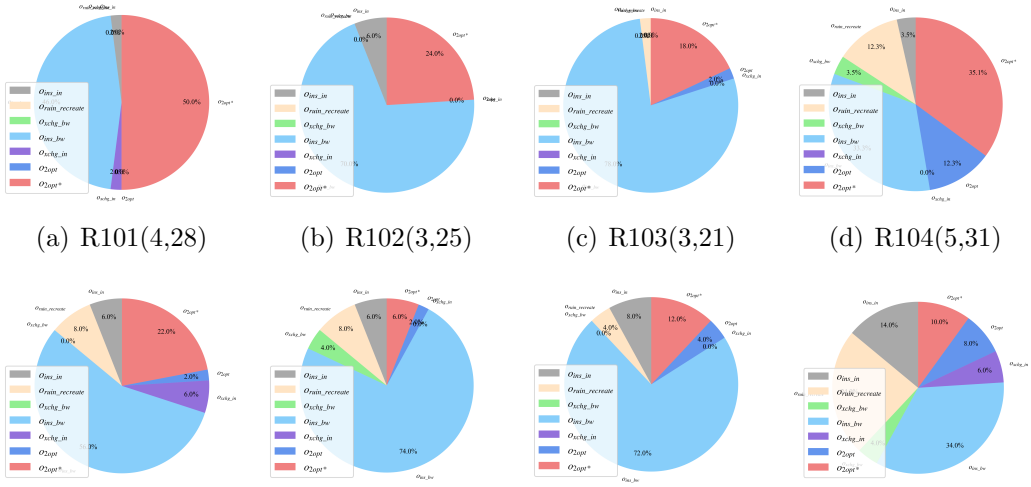
As can be seen from all the figures, the diversity of the operators in the best



(a) C101(2,28) (b) C102(3,29) (c) C103(7,38) (d) C104(5,39)

(e) C201(3,29) (f) C202(5,34) (g) C203(7,38) (h) C204(6,37)

**Figure 4.11:** *Transition of operators in the best designed algorithm (type-C, with search-dependent and instance-dependent features)*

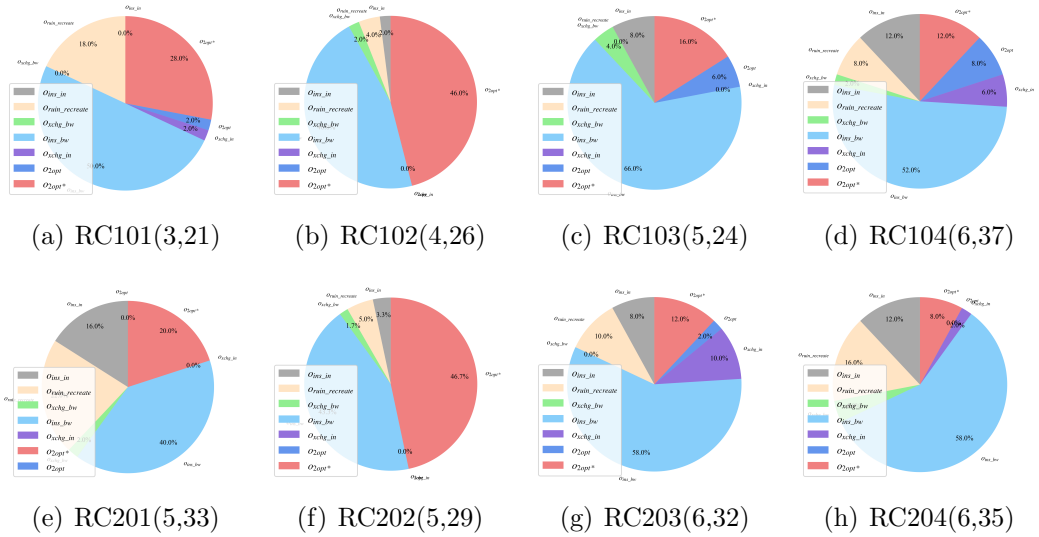


(a) R101(4,28) (b) R102(3,25) (c) R103(3,21) (d) R104(5,31)

(e) R201(7,35) (f) R202(6,22) (g) R203(5,18) (h) R204(7,36)

**Figure 4.12:** *Transition of operators in the best designed algorithm (type-R, with search-dependent and instance-dependent features)*

designed algorithm compositions (i.e. the number of operators) increases when the capacity of the vehicle increases. Taking type-C instances as examples, Figure 4.11 shows that the number of called operators in the best designed algorithm compositions (obtained by the PPO model with two feature sets) for solving the type-C2 instances with a larger vehicle capacity (i.e. 700), is higher than that of the type-C1 instances with a smaller vehicle capacity (i.e. 200). The same phenomenon can be observed in type-R and type-RC instances, details of which are shown in Figure 4.12 and Figure 4.13. Similar

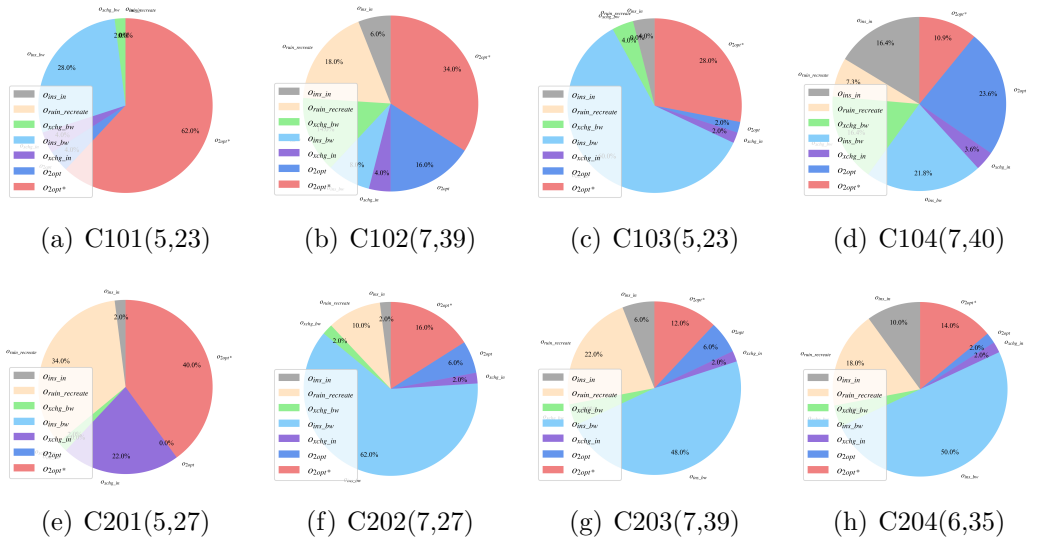


(a) RC101(3,21) (b) RC102(4,26) (c) RC103(5,24) (d) RC104(6,37)

(e) RC201(5,33) (f) RC202(5,29) (g) RC203(6,32) (h) RC204(6,35)

**Figure 4.13:** *Transition of operators in the best designed algorithm (type-RC, with search-dependent and instance-dependent features)*

conclusions can be reached regarding the diversity of operators in the best designed algorithm compositions obtained by the PPO model with only search-dependent features, as shown in Figures 4.14-4.16.

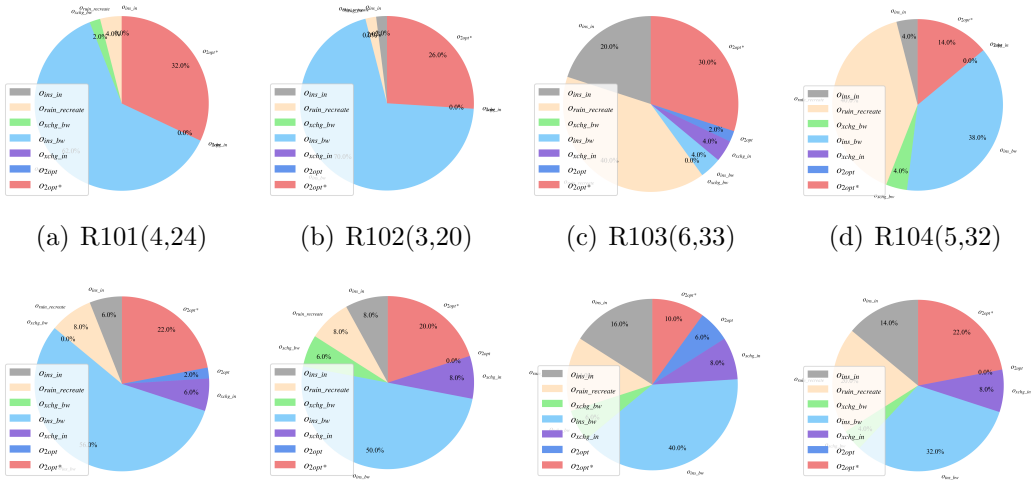


(a) C101(5,23) (b) C102(7,39) (c) C103(5,23) (d) C104(7,40)

(e) C201(5,27) (f) C202(7,27) (g) C203(7,39) (h) C204(6,35)

**Figure 4.14:** *Transition of operators in the best designed algorithm (type-C, with only search-dependent features)*

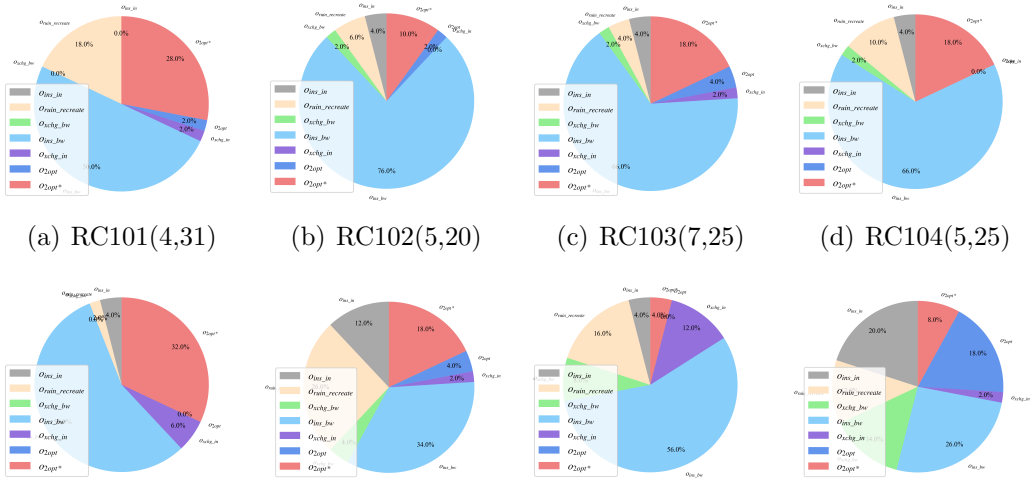
There is a negative correlation between the diversity of operators and the density of the time-window (i.e. the percentage of time-constrained customers), as shown in Figures 4.11-4.16. Taking type-RC instances as examples, Figure 4.13 shows that with the decrease of time window density, dropping from 100%



(a) R101(4,24) (b) R102(3,20) (c) R103(6,33) (d) R104(5,32)

(e) R201(5,29) (f) R202(6,32) (g) R203(7,38) (h) R204(6,35)

**Figure 4.15:** *Transition of operators in the best designed algorithm (type-R, with only search-dependent features)*



(a) RC101(4,31) (b) RC102(5,20) (c) RC103(7,25) (d) RC104(5,25)

(e) RC201(4,28) (f) RC202(6,36) (g) RC203(6,36) (h) RC204(6,44)

**Figure 4.16:** *Transition of operators in the best designed algorithm (type-RC, with only search-dependent features)*

in RC101 to 25% in RC104, the diversity of operators increases, rising from 3 to 6. The same phenomenon can be observed in the other types of instances as well.

Considering the above two findings, in the best designed algorithm compositions, more types of operators are called when the problem constraints are relaxed (i.e. a larger vehicle capacity and a smaller time-window density). A possible reason may be that the feasible solution space expands with the relaxation of problem constraints.

The number of operator transition is more than 25 over 50 timesteps in most instances although the number of operators is relatively small. This indicates that operators in the best designed algorithm compositions are frequently called interchangeably during the optimisation process, although some types of operators (i.e. $ins\_bw$ and $2opt*$) are called much more frequently than others. When a continuous selection of an operator fails to trigger a shift in state and an increase in reward, switching to another operator brings unexpected results. This means that the operators which are called less frequently are also useful. One possible explanation is that the search space of COPs is a non-stationary environment containing a variety of search regions with different characteristics. Different operators with different search behaviours, only perform well in some regions. Therefore, solving COPs using a search algorithm with only a single operator is less effective. Hence, it is reasonable to expect that the search algorithms which are automatically designed based on combinations of algorithm components (e.g. operators) will produce better performance.

## 4.5 Summary

In this chapter, two groups of features, namely search-dependent features and instance-dependent features, are identified to provide the key information to assist learning on automated algorithm design. Search-dependent features describe the search space of algorithm design, while instance-dependent features characterise the problem instances. Using the identified features to represent the state, a state-of-the-art reinforcement learning technique, namely proximal policy optimisation, is devised to automatically combine different evolution operators during different stages of the evolutionary process. Search patterns of the best designed algorithms which are obtained by the reinforcement learning models resulting from different state representation schemes are analysed.
With controlled experiments on the state representation, the impact of the identified features on the reinforcement learning model is verified on the bench-

mark instances of the capacitated vehicle routing problem with time windows. The results show that both search-dependent and instance-dependent features can provide useful information to the learning process by assisting the population to accurately detect the resulting state with better action choice.

Regarding the search patterns of the best designed algorithms, utilisation and transition of evolution operators are analysed. The analysis shows that two reinforcement learning models with different features identify $ins\_bw$ and $2opt*$ as the most frequently employed algorithmic components. Different operators are frequently called interchangeably during the optimisation process. This indicates the importance of adaptive operator selection for designing effective search algorithms.

This chapter explores feature identification for automated algorithm design, which is one of the key issues in developing successful machine learning for effective algorithm design. With the identified features in this chapter, a set of reinforcement learning techniques, have been devised to automatically select and combine evolution operators in Chapter 5, selection heuristics and pairs of evolution operators and selection heuristics in Chapter 6.

# Chapter 5

# Automated Composition of Evolution Operators

In Chapter 3, a novel general search framework was developed and three key research issues within the proposed framework were identified. In Chapter 4, the key features for building successful machine learning to undertake new task of automated algorithm design were identified. This chapter will focus on investigating the first key issue: automated composition of evolution operators, which have the greatest impact on an algorithm's performance. Reinforcement learning models with the identified features are devised in automated algorithm design to reward or penalise combinations of key evolution operators based on their performance.

## 5.1 Introduction

The research in this chapter has been conducted with the following motivations. Firstly, there has been growing research interest in utilising machine learning to support the newly defined learning task of automated algorithm design in recent years. This kind of research is generally conducted in the literature with a template of a specific metaheuristic algorithm such as genetic algorithm or particle swarm optimisation. However, this limited the scope of

algorithms under consideration. Secondly, existing studies focus more on improvement of the solution quality after applying machine learning to the new task of algorithm design. However, they pay little attention to the reusability and generality of the automatically designed algorithms, which are equally important to the solution quality.

With the novel general search framework and newly identified features, in this chapter, instead of studying all the algorithmic components, we only focus on investigating the first key issue: automated composition of key evolution operators, which have the greatest impact on algorithm's performance. More specifically, we aim to answer the following research questions (RQ):

- *RQ1* What kind of machine learning techniques would be useful to automatically select and combine the evolution operators to develop effective search algorithms?

- *RQ2* Is the performance of the learning based methods better than that of non-learning procedure?

- *RQ3* Do the automatically designed algorithms perform well on new problem instances, regarding reusability and generality?

The goal of this chapter is to develop effective reinforcement learning methods with the identified features that will automatically select and combine the suitable evolution operators to be applied during the evolutionary process. Specifically, this chapter addresses the following research objectives:

1. To develop two reinforcement learning based methods to address the key issue of automated selection and combination of the most efficient evolution operators during different stages of the evolutionary process. (RQ1)

2. To investigate the effectiveness of the trained policies compared to a search procedure without learning. (RQ2)

3. To assess the reusability and generality of the trained policies by directly applying them to new problem instances. (RQ3)

The rest of this chapter is organised as follows. Detailed descriptions of the proposed method are given in Section 5.2. The experiments and discussions are shown in Section 5.3. Finally, Section 5.4 concludes this chapter.

## 5.2 Proposed Reinforcement Learning Method

Reinforcement learning is a machine learning technique, where intelligent agents take actions based on the learned policy trained through trial and error interactions with the environment by maximising total reward. The environment of RL is considered as an MDP, which is composed of a set of possible states and a set of selectable actions. Each state-action pair is assigned a total reward value (Q-value). With the established general search framework (GSF) in Chapter 3, RL is used for automated algorithm design as shown in Figure 5.1. The actions are the selectable combinations of algorithmic components (i.e. evolution operators). The states are defined by different features of the search process and the instance, as shown in Table 5.2. The automated algorithm design process starts with the observation of the agent's current situation (a state) and the selection of a combination of algorithmic components (an action). The execution of the resulting algorithmic component (selected action) leads to a new state of the optimisation process (environment) by the chosen selection heuristic and evolution operator to the current state. A reward (or penalty) is assigned to the selected action with respect to the current state.

Tabular RL techniques, such as SARSA [24] and QL [26], have been used to select evolution operators in the literature. However, a Q-table cannot handle continuous state space, leading to unreliable results. To address this issue, in this chapter, RL techniques with a neural network as value-function approximator have been adopted.

RL techniques can be roughly divided into value-based methods and policy-

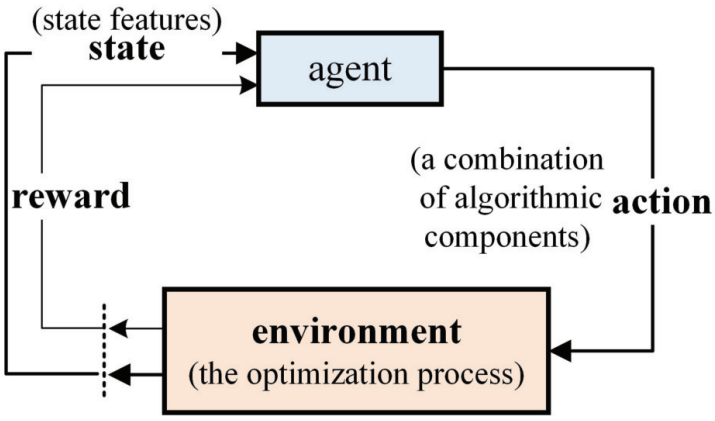



**Figure 5.1:** *Reinforcement learning in the context of automated algorithm composition in GSF*

based methods based on their policy update mechanism [24]. To comprehensively verify the effectiveness of RL on automated algorithm design, a typical value-based method and a typical policy-based method are investigated within GSF in this chapter.

In value-based RL, DQN [27], the first deep reinforcement learning method, is selected. The DQN-based method to automatically design an algorithm within the GSF is named DQN-GSF. In policy-based RL, PPO [28], which outperforms other policy gradient methods, is selected, named PPO-GSF.

Table 5.1 shows the notations used in this chapter. The pseudocodes of DQN-GSF and PPO-GSF are shown in Algorithm 1 and Algorithm 2, respectively. Note that $h_1$ and $h_8$ are fixed in the Selection for Evolution and Replacement modules to address our key research issues, i.e. how to automatically design algorithms with evolution operators which have the most impact on evolutionary algorithms. With the newly established GSF, in this chapter, the focus is on the key modules of evolution, rather than on determining all the components in all modules simultaneously to find the best results within a reasonable computational time. With controlled experiments on the key module while fixing the other modules, we can focus on examining the results only due to different settings in the Evolution module. The most commonly used components in the existing metaheuristic algorithms, i.e. $h_1$ in Selection for Evolution and $h_8$ in Replacement, are chosen for focused investigations.

Table 5.1: Notations used in DQN-GSF and PPO-GSF

| Notation | Description |
|---|---|
| $s_0$ | The initial state |
| $s_t$ | The state at timestep $t$ |
| $a_t$ | The selected action at timestep $t$ |
| $r_t$ | The reward value at timestep $t$ |
| $NoE$ | The number of episodes |
| $NoT$ | The number of timesteps in one episode |

**Algorithm 1** Pseudocode of DQN-GSF

Initialise memory buffer $D$
Initialise evaluation action-value function $Q$ network and target action-value function $\hat{Q}$ network
Generate initial population, record the initial state $s_0$
**for** episode $k = 1$ to $NoE$ **do**
  initialise the state $s_0$
  **for** timestep $t = 1$ to $NoT$ **do**
    observe the current state $s_t$ by calculating values of different state features in Table 5.2
    with probability $\varepsilon$ select a random action $a_t$, with probability $1 - \varepsilon$ select an action that has a maximum Q-value: $a_t = arg\max Q(s_t, a_t)$
    select parents using a selection heuristic $h_i\,(i = 1, 2, ..., 6)$ from $H_{SE}$ (fixed as $h_1$ in this study)
    generate offspring population by performing the selected action $a_t$ to state $s_t$
    update the population using a selection heuristic $h_i\,(i = 7, 8)$ from $H_{SR}$ (fixed as $h_8$ in this study)
    observe reward $r_t$ based on Equation (4.7) and Equation (4.8), and next state $s_{t+1}$, store experience $(s_t, a_t, r_t, s_{t+1})$ in $D$
    sample random mini-batch of experiences $[s_j, a_j, r_j, s_{j+1}]_J$ ($J$ denotes the size of the sampled mini-batch) from memory buffer $D$ and calculate the loss: $\left[r_j + \gamma \max_{a_{j+1}} \hat{Q}(s_{j+1}, a_{j+1}) - Q(s_j, a_j)\right]^2$, $\gamma$ denotes the discount factor.
    perform gradient descent with respect to $Q$ network in order to minimise the loss
    every $N$ timesteps reset $\hat{Q} = Q$
  **end for**
**end for**

**Algorithm 2** Pseudocode of PPO-GSF

1: Initialise memory buffer $D$
2: Initialise policy parameters $\theta_0$, value function parameters $\Phi_0$
3: Generate initial population, record the initial state $s_0$
4: **for** episode $k = 1$ to $NoE$ **do**
5:  **for** timestep $t = 1$ to $NoT$ **do**
6:  observe the current state $s_t$ by calculating values of different state features in Table 5.2
7:  select parents using a selection heuristic $h_i\,(i = 1, 2, ..., 6)$ from $H_{SE}$ (fixed as $h_1$ in this study)
8:  generate offspring population by performing the selected action $a_t$ based on policy $\pi_k = \pi_{\theta_k}$.
9:  update the population using a selection heuristic $h_i\,(i = 7, 8)$ from $H_{SR}$ (fixed as $h_8$ in this study)
10:  observe reward $r_t$ based on Equation (4.7) and Equation (4.8)
11:  collect experience $(s_t, a_t, r_t)$ and save it in $D$
12:  **end for**
13:  update the policy by maximising the PPO objective $\theta_{k+1}$ based on Equation (5.1)
14:  fit value function $\Phi_{k+1}$ based on Equation (5.2)
15:  empty memory buffer $D$
16: **end for**

In DQN-GSF and PPO-GSF, the two RL techniques, DQN and PPO, are firstly applied in multiple episodes to train the policy within the GSF. After that, the trained policy is used to design the search algorithm online. The training process is the key research issue, and described in details as follows.

As shown in Algorithm 1, the DQN-GSF is trained on every timestep. Specifically, an action (an evolution operator) is deterministically selected with the largest Q-value for exploitation or randomly selected for exploration (Line 8, Algorithm 1). The designed search algorithm with predefined selection heuristics is executed for one timestep (Line 9-11, Algorithm 1). The next state and reward are identified, and this experience $(s_t, a_t, r_t, s_{t+1})$ is stored in the memory buffer (Line 12, Algorithm 1). After that, a mini-batch of experiences is randomly sampled from the memory buffer to train the evaluation network (Line 13-14, Algorithm 1). The process is iterated at each timestep until the end of the episode. In the process, the target network parameters are periodically synchronised with the evaluation network parameters (Line

15, Algorithm 1).

Unlike value-based DQN-GSF, policy-based PPO-GSF is trained on every episode rather than each timestep. As shown in Algorithm 2, firstly, a series of actions are selected based on the probability of the policy $\pi_{\theta_k}, k = 1, 2, ..., NoE$, and then the designed search algorithm with predefined selection heuristics is correspondingly executed for one episode (Line 5-12, Algorithm 2). Then, the policy is updated by maximising the PPO objective based on Equation (5.1) (Line 13, Algorithm 2) and the value function is fitted by time differential error based on Equation (5.2) (Line 14, Algorithm 2). Finally, the memory buffer is set to be empty (Line 15, Algorithm 2). A series of actions is selected based on the updated policy to perform the next episode of optimisation (Line 8, Algorithm 2).

$$\begin{aligned}\theta_{k+1} = & arg\ \max_{\theta} \frac{1}{|D_c|\, NoT} \sum_{\tau \in D_c} \sum_{t=0}^{NoT} \min\left(r_t\left(\theta\right)\right. \\ & \left. A^{\pi_{\theta_k}}\left(s_t, a_t\right), clip\left(r_t\left(\theta\right), 1-\epsilon, 1+\epsilon\right)\right)\end{aligned} \tag{5.1}$$

$$\Phi_{k+1} = arg\min_{\Phi} \frac{1}{|D_c|\, NoT} \sum_{\tau \in D_c} \sum_{t=0}^{NoT} \left(V_{\Phi}\left(s_t\right) - \hat{R}_t\right)^2 \tag{5.2}$$

$r_t\left(\theta\right)$ denotes the probability ratio $r_t\left(\theta\right) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$. $A^{\pi_{\theta_k}}\left(s_t, a_t\right)$ is an estimator of the advantage function at timestep $t$. $\epsilon$ is a hyperparameter. $clip\left(r_t\left(\theta\right), 1-\epsilon, 1+\epsilon\right)$ denotes the modified surrogate objective by clipping the probability ratio. The rewards-to-go $\hat{R}_t$ is calculated according to trajectory $\tau : [(s_1, a_1, r_1), (s_2, a_2, r_2), \cdots, (s_t, a_t, r_t)]$. Please refer to [28] for more detail about these two equations.

### 5.2.1 State Representation

Different state features, including search-dependent and instance-dependent features, are distinguished in this section.

Search-dependent features observe the search process, such as the total improvement over the initial solution. Instance-dependent features refer to the instance-specific characteristics, such as the vehicle number or the vehicle capacity of VRP.

When search-dependent or instance-dependent features are used to define the state space, the learned information can be transferred to other instances of the same problem, or even to other problems. In this section, as shown in Table 5.2, four search-dependent features ($f_1$-$f_4$) and four instance-dependent features ($f_5$-$f_8$) are used to define the state space of algorithm design as a new task for machine learning. Note that the focus of this section is on automated algorithm composition rather than feature selection. No systematic feature selection methods are employed in this section. These 8 features are selected from 17 features in the previous chapter to conduct preliminary experiments on the automated algorithm composition of evolution operators since they are commonly used in the literature.

Table 5.2: Definition of the state space

| Feature | Description |
|---|---|
| $f_1$ | The total fitness improvement over the initial population fitness |
| $f_2$ | The diversity of the population, measured by the standard deviation of the population fitness |
| $f_3$ | The current algorithmic stage, calculated as $i/NoT$ where $i$ is the index of the current timestep |
| $f_4$ | The altitude of the search space which is based on the difference between the upper and lower bounds of the fitness values, i.e. of the current best and worst solutions found within the episode |
| $f_5$ | The number of vehicles |
| $f_6$ | The capacity of the vehicle |
| $f_7$ | The density of the time windows, i.e. the percentage of time-constrained customers |
| $f_8$ | The tightness of the time windows, i.e. the width of the time windows |

### 5.2.2 Action Representation

In DQN-GSF and PPO-GSF, the set of possible actions in each state is defined by the set of evolution operators ($O_E$) . Once an action is selected, it is applied to the whole population.

### 5.2.3 Reward Scheme

Reward scheme, which encourages the RL system to find efficient search policies, is very important for an RL method. In DQN-GSF and PPO-GSF, the reward is calculated based on the improvement of the fitness of the current population over the initial population. When population fitness is optimised above a certain threshold, a larger reward is assigned to the same fitness improvement. Same reward scheme is adopted as that in Section 4.3.3, which is shown in Equations (4.7)-(4.8).

## 5.3 Experiments and Discussions

The experimental investigations aim to address two research issues: (1) the effectiveness of the new RL techniques to automatically generate a search algorithm to tackle the benchmark Solomon CVRPTW dataset; (2) the generality of the trained policies to new problem instances. To analyse the influence of the Q-value function approximator on learning models, two value-based RL-GSF methods with fitness improvement as the state definition, namely QL-GSF with a Q-table and DQN-GSF with a neural network function approximator, are compared. To analyse the influence of the policy update mechanism on learning models, DQN-GSF and PPO-GSF, are assessed. The generality of the trained policies across the same-type and different-type of problem instances are assessed by directly applying the trained policies to new instances. The experimental settings in this chapter, such as the machine information and the computational time, are identical as those in Chapter 4.

### 5.3.1 Effectiveness of the Learning Models

This section investigates the effectiveness of the proposed RL-GSF models from two aspects: influence of the Q-value function approximator; and influence of the policy update mechanism.

**Influence of Q-value Function Approximator:**

QL, representing the tabular RL methods, and DQN, representing the function approximation RL methods, are applied in the established GSF in this section. The random algorithm is chosen as the baseline algorithm to demonstrate the performance of the RL methods. The Random-GSF method randomly selects algorithmic components within the established GSF during different stages of the optimisation process without any learning, i.e. each algorithmic component has the same probability of being selected.

The state space is defined by the total fitness improvement over the initial

population fitness, i.e. $f_1$ in Table 5.2. CVRPTW is an NP-hard problem with a continuous fitness search space, however, Q-learning methods are typically designed to handle discrete state spaces. Therefore, to enable the application of Q-learning to continuous state spaces, an approximation technique based on the concept of the state aggregation [157], [158], [159] is employed within the QL-GSF. This technique aggregates the state space into several disjoint categories.

From the preliminary experimental observations, in type-R1 and type-RC1 instances, the values obtained fall into the range [0.4,0.6]. The range is slightly different in type-C, type-R2 and type-RC2 instances, observed as [0.3,0.5] from experiments. Therefore, the state space of type-R1 and type-RC1 instances is divided as: $f_1 \in [0, 0.4), [0.41, 0.42), ..., [0.59, 0.6), [0.6, 0.7), ..., [0.9, 1]$; for type-C, type-R2 and type-RC2 instances, the state space is divided as: $f_1 \in [0, 0.3), [0.31, 0.32), ..., [0.49, 0.5), [0.5, 0.7), [0.7, 0.8), ..., [0.9, 1]$. In DQN-GSF with a neural network function approximator, there is no need to aggregate state; instead, $f_1$ can be simply used as the sole input of the neural network.

Apart from the Q-value function approximator, the experimental environment and parameters settings are identical for these three algorithms. For testing purposes, as shown in Tables 5.3-5.5, by running each learning algorithm 10 times, we collected the average best fitness (AVG), standard deviation (SD), the best fitness (BEST) and the GAP between BEST and the best-known solution in the literature [143]. The published results (BEST) of two state-of-the-art manually designed algorithms, i.e. RT [144] and HG [148], are listed in all tables to the comparisons. These two manually designed algorithms are selected as the comparison since they have most of the results and information in the published paper.

On the type-C instances, the results of BEST and GAP in Table 5.4 demonstrate that these three methods can produce the current best-known solutions [143]. This type of instances can be solved by evolutionary search without any learning techniques. The different AVG and SD indicate that the proposed

RL-GSF methods, especially DQN-GSF, are more stable to automatically design a search algorithm for solving type-C instances with statistical significance (measured by Wilcoxon rank sum test with $p < 0.05$), and indicated by $*$ in all the tables of results.

On the type-R and type-RC instances, as shown in Table 5.4 and Table 5.5, DQN-GSF achieves the best results among these three algorithms in most instances. QL-GSF is the second best, with a higher AVG and a smaller GAP than Random-GSF in most instances. It indicates that learning based models are more effective than the non-learning search procedure.

In conclusion, a neural network function approximator outperforms the simple Q-table. With more features to define the state space, the effectiveness of the learning methods is likely to be further improved. However, the memory required by a simple Q-table to handle multiple features will increase and the amount of time required to explore each state to create the required Q-table becomes unrealistic. In comparison to a Q-table, a neural network is able to handle multiple features.

It can also be observed that Random-GSF shows comparable performance with the other two methods on type-C instances but poorer performance on type-R and type-RC instances. This indicates that learning mechanisms can help to find better combination of the algorithmic components, obtaining better solutions. In the next section, two neural network based RL-GSF methods, DQN-GSF and PPO-GSF, will be investigated further.

**Influence of Policy Update Mechanisms in the Learning Models:**

In the value-based method DQN-GSF, and policy-based method PPO-GSF, apart from the policy update mechanism, the other parameters, such as the population size and the maximum running time are all identical to conduct fair comparison.

The policies of PPO-GSF and DQN-GSF are gradually improved during the training process. However, a certain degree of randomness must be maintained

Table 5.3: Comparisons on selected type-C instances (influence of Q-value function approximator). ♯ and ∗ indicate DQN-GSF is significantly different from Random-GSF and QL-GSF, respectively, i.e. $p < 0.05$

| Instance | | C101 | C102 | C105 | C201 | C202 | C205 |
|---|---|---|---|---|---|---|---|
| Best-known solutions | | 10828.94 [144] | 10828.94 [144] | 10828.94 [144] | 3591.56[144] | 3591.56[144] | 3588.88[144] |
| RT [144] | | 10828.94 | 10828.94 | 10828.94 | 3591.56 | 3591.56 | 3588.88 |
| HG [148] | | 10828.94 | 10828.94 | 10828.94 | 3591.56 | 3591.56 | 3588.88 |
| Random-GSF | AVG | 11061.1 | 10928.73 | 10832.97 | 3604.44 | 3616.24 | 3599.05 |
| | SD | 483.51 | 110.36 | 12.75 | 16.64 | 24.9 | 15.82 |
| | BEST | **10828.94** | **10828.94** | **10828.94** | **3591.56** | **3591.56** | **3588.88** |
| | GAP | **0** | **0** | **0** | **0** | **0** | **0** |
| QL-GSF | AVG | **10828.94** | 10860.36 | **10828.94** | **3591.56** | 3618.87 | 3591.75 |
| | SD | **0** | 45.11 | **0** | **0** | **17.32** | 8.62 |
| | BEST | **10828.94** | **10828.94** | **10828.94** | **3591.56** | **3591.56** | **3588.88** |
| | GAP | **0** | **0** | **0** | **0** | **0** | **0** |
| DQN-GSF | AVG | 10831.47 ♯ | **10840.84** ♯ ∗ | 10833.93 | **3591.56** | **3616.15** | **3588.88** ♯ |
| | SD | 7.61 | **24.65** | 14.99 | **0** | 17.44 | **0** |
| | BEST | **10828.94** | **10828.94** | **10828.94** | **3591.56** | **3591.56** | **3588.88** |
| | GAP | **0** | **0** | **0** | **0** | **0** | **0** |

Table 5.4: Comparisons on selected type-R instances (influence of Q-value function approximator). ♯ and ∗ indicate DQN-GSF is significantly different from Random-GSF and QL-GSF, respectively, i.e. $p < 0.05$

| Instance | | R101 | R102 | R105 | R201 | R202 | R205 |
|---|---|---|---|---|---|---|---|
| Best-known solutions | | 20645.79 [145] | 18486.12 [144] | 15377.11 [144] | 5252.37[148] | 4191.7[149] | 3994.42[149] |
| RT[144] | | 20650.80 | 18486.12 | 15377.11 | 5281.58 | 5088.07 | 4063.24 |
| HG[148] | | 20656.79 | 18490.39 | 15377.11 | 5252.37 | 4198.45 | 4162.06 |
| Random-GSF | AVG | 21213.58 | 19541.53 | 16777.28 | 5445.83 | 5285.64 | 4193.57 |
| | SD | 525.00 | 25.30 | 503.17 | 75.07 | 50.04 | 37.23 |
| | BEST | 21637.7 | 19510.7 | 16421.4 | **5318.79** | 5183.26 | 4139.7 |
| | GAP | 4.8% | 5.5% | 6.8% | **1.3%** | 23.66% | 3.6% |
| QL-GSF | AVG | 21893.56 | **19505.09** | 16938.17 | 5384.10 | 5213.21 | 4179.57 |
| | SD | **397.57** | **19.62** | 500.57 | 40.39 | **22.08** | **27.64** |
| | BEST | 21665.47 | 19486.77 | 16438.95 | 5336.10 | 5183.86 | 4131.79 |
| | GAP | 4.94% | 5.41% | 6.91% | 1.59% | 23.67% | 3.44% |
| DQN-GSF | AVG | **21171.67** ♯ ∗ | 19516.86 ♯ | **16736.59** ∗ | **5366.27** ♯ ∗ | **5191.93** ♯ | **4168.28** ♯ ∗ |
| | SD | 500.27 | 20.15 | **459.63** | **27.17** | 31.23 | 36.63 |
| | BEST | **20655.81** | **19481.96** | **16414.47** | 5325.66 | **5137.16** | **4086.29** |
| | GAP | **0.05%** | **5.39%** | **6.75%** | 1.40% | **22.56%** | **2.30%** |

Table 5.5: Comparisons on selected type-RC instances (influence of Q-value function approximator). ♯ and ∗ indicate DQN-GSF is significantly different from Random-GSF and QL-GSF, respectively, i.e. $p < 0.05$

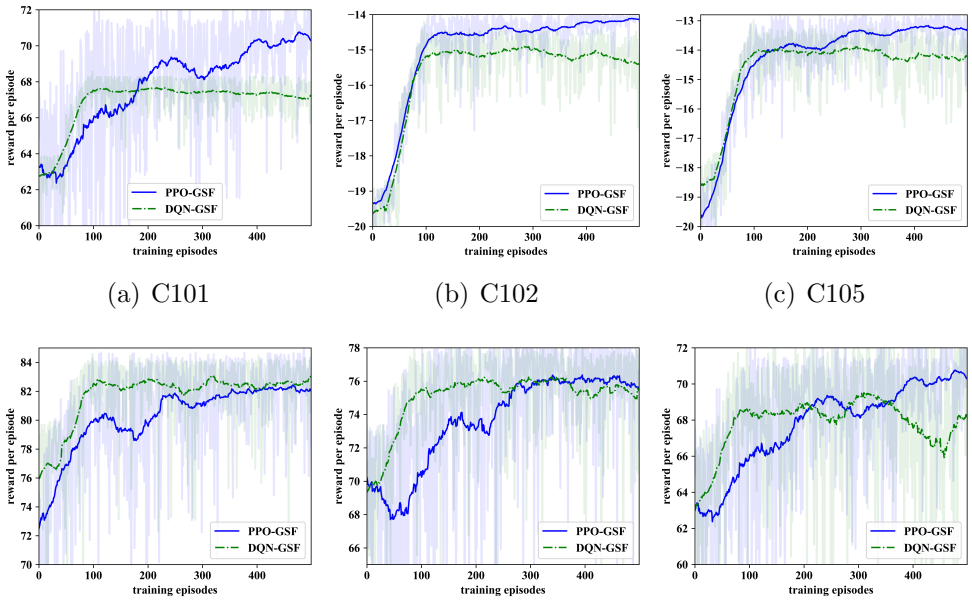
| Instance | | RC101 | RC102 | RC105 | RC201 | RC202 | RC205 |
|---|---|---|---|---|---|---|---|
| Best-known solutions | | 15696.94 [152] | 13554.72 [152] | 14628.44[156] | 5406.91[147] | 4367.09[155] | 5297.19[147] |
| RT[144] | | 16623.58 | 14477.54 | 14733.56 | 5438.89 | 5165.57 | 5333.71 |
| HG[148] | | 15697.43 | 13558.07 | 14637.15 | 5418.86 | 4665.56 | 5302.42 |
| Random-GSF | AVG | **17340.05** | 16196.35 | 16971.02 | 5631.78 | 5439.67 | 5533.55 |
| | SD | **537.58** | 543.03 | 508.91 | 73.60 | 53.03 | 62.02 |
| | BEST | 16687.5 | **14527.04** | 16627.61 | 5561.31 | 5315.21 | 5435.81 |
| | GAP | 6.3% | 7.2% | 13.7% | 2.9% | 21.7% | 2.6% |
| QL-GSF | AVG | 17840.39 | 15967.39 | **16932.14** | **5595.50** | 5464.09 | **5430.87** |
| | SD | 716.74 | 499.91 | **463.88** | **30.34** | **31.41** | **33.28** |
| | BEST | 16686.71 | 15502.68 | 16613.85 | 5553.28 | 5446.44 | 5396.74 |
| | GAP | 6.31% | 14.37% | 13.56% | 2.71% | 24.72% | **1.87%** |
| DQN-GSF | AVG | 17523.59 ♯ ∗ | **15653.58** ♯ ∗ | 16936.21 ♯ | 5659.06 ∗ | **5338.71** ♯ ∗ | 5547.81 ∗ |
| | SD | 621.65 | **294.00** | 468.76 | 296.92 | 33.30 | 305.41 |
| | BEST | **16662.50** | 15490.86 | **16589.13** | **5473.00** | **5251.44** | **5396.45** |
| | GAP | **6.15%** | **14.28%** | **13.4%** | **1.22%** | **20.25%** | 1.88% |

to avoid being trapped in a local optima. As shown in Figures 5.2-5.4, the original reward curve (depicted in the background of these figures) exhibits a rising trend with some fluctuations as a result of this. To present the training effects more clearly, the reward curve is smoothed by using a sliding window filter method (moving average), as shown in Equation (5.3).

$$l^{smoothed} = \frac{convolve\left(x_{buff}, y_{buff}\right)}{convolve\left(z_{buff}, y_{buff}\right)} \tag{5.3}$$

where $x_{buff}$ is the raw reward per episode, $y_{buff} = [1, \cdots, 1]_q$ is a vector with the length of the smooth factor $q$, $z_{buff} = [1, \cdots, 1]_{NoE}$ is a vector with the length of the whole training data. $q$ is set to 5 in the experiment.

On the type-C instances, as illustrated in Figure 5.2, PPO-GSF performs better than DQN-GSF in most instances. As shown in Table 5.6, the average best fitness and standard deviation also demonstrate the superiority of PPO-GSF over DQN-GSF. Again both learning methods can produce the current best-known solutions [143].




(a) C101 (b) C102 (c) C105

(d) C201 (e) C202 (f) C205

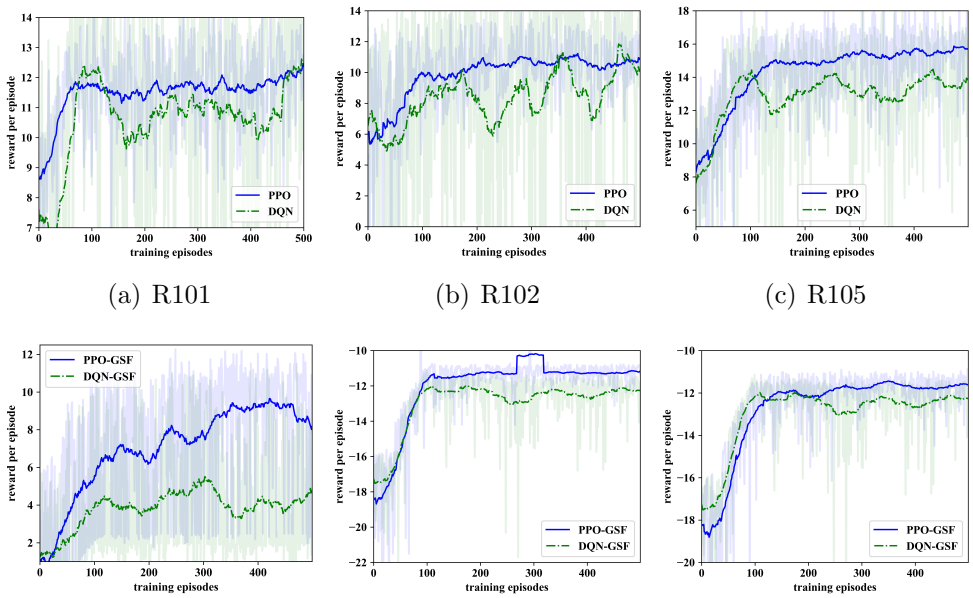
**Figure 5.2:** *Influence of policy update mechanisms on the learning models (type-C problem instances)*

On the type-R instances, as illustrated in Figure 5.3, PPO-GSF outperforms

Table 5.6: Comparisons on selected type-C instances (influence of policy update mechanisms). $*$ indicates PPO-GSF is significantly different from DQN-GSF, i.e. $p < 0.05$

| Instance | | C101 | C102 | C105 | C201 | C202 | C205 |
|---|---|---|---|---|---|---|---|
| Best-known solutions | | 10828.94 [144] | 10828.94 [144] | 10828.94[144] | 3591.56[144] | 3591.56[144] | 3588.88[144] |
| RT[144] | | 10828.94 | 10828.94 | 10828.94 | 3591.56 | 3591.56 | 3588.88 |
| HG[148] | | 10828.94 | 10828.94 | 10828.94 | 3591.56 | 3591.56 | 3588.88 |
| DQN-GSF | AVG | 11148.36 | 10900.96 | 10832.46 | **3591.56** | 3612.52 | **3588.88** |
| | SD | 487.93 | 112.83 | 10.57 | **0** | 13.95 | **0** |
| | BEST | **10828.94** | **10828.94** | **10828.94** | **3591.56** | **3591.56** | **3588.88** |
| | GAP | **0** | **0** | **0** | **0** | **0** | **0** |
| PPO-GSF | AVG | **10834.01** $*$ | **10855.22** $*$ | **10831.72** | **3591.56** | **3595.29** $*$ | **3588.88** |
| | SD | **10.15** | **50.24** | **8.36** | **0** | **11.19** | **0** |
| | BEST | **10828.94** | **10828.94** | **10828.94** | **3591.56** | **3591.56** | **3588.88** |
| | GAP | **0** | **0** | **0** | **0** | **0** | **0** |

DQN-GSF in terms of algorithm convergence and solution quality. Furthermore, Table 5.7 reveals that PPO-GSF achieves better results in terms of all four indicators AVG, SD, BEST and GAP in most instances. The GAPs of PPO-GSF are less than 3% in most instances except on the R202 instance.




(a) R101 (b) R102 (c) R105

(d) R201 (e) R202 (f) R205

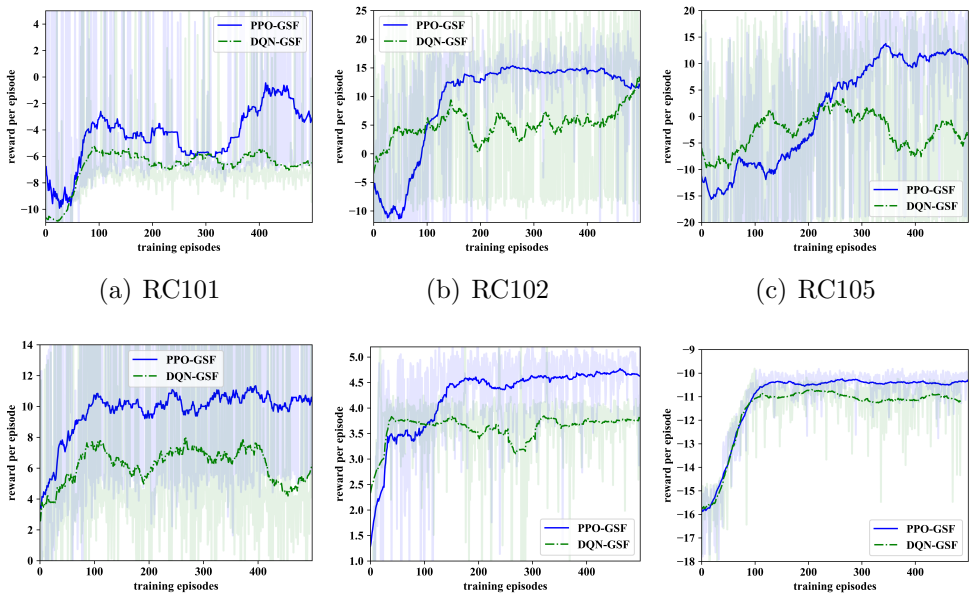
**Figure 5.3:** *Influence of policy update mechanisms on the learning models (type-R problem instances)*

On the type-RC instances as illustrated in Figure 5.4, PPO-GSF clearly outperforms DQN-GSF in all instances. In Table 5.8, in most type-RC instances, the solutions obtained by PPO-GSF and DQN-GSF are non-dominated solutions to the best-known solution identified by all the other metaheuristics methods in the literature [143]. Taking RC101 as an example, DQN-GSF

Table 5.7: Comparisons on selected type-R instances (influence of policy update mechanisms). $*$ indicates PPO-GSF is significantly different from DQN-GSF, i.e. $p < 0.05$

| Instance | | R101 | R102 | R105 | R201 | R202 | R205 |
|---|---|---|---|---|---|---|---|
| Best-known solutions | | 20645.79 [145] | 18486.12 [144] | 15377.11 [144] | 5252.37 [148] | 4191.7[149] | 3994.42[149] |
| RT[144] | | 20650.80 | 18486.12 | 15377.11 | 5281.58 | 5088.07 | 4063.24 |
| HG[148] | | 20656.79 | 18490.39 | 15377.11 | 5252.37 | 4198.45 | 4162.06 |
| DQN-GSF | AVG | 20981.19 | 19910.38 | 16434.78 | **5378.77** | 5201.13 | 4280.21 |
| | SD | 454.50 | 452.07 | 489.72 | 44.67 | 34.34 | 308.16 |
| | BEST | 20656.49 | 19495.43 | **15410.25** | **5304.48** | 5159.01 | 4131.92 |
| | GAP | **0.05%** | 5.5% | **0.02%** | **0.99%** | 23.1% | 3.4% |
| PPO-GSF | AVG | **20665.46** $*$ | **18708.26** $*$ | **16329.39** $*$ | 5382.98 | **5200.721** | **4145.18** $*$ |
| | SD | **10.16** | **413.91** | **321.27** | **39.15** | **29.83** | **29.52** |
| | BEST | **20655.81** | **18493.03** | 15418.00 | 5318.35 | **5144.32** | **4094.81** |
| | GAP | **0.05%** | **0.04%** | 0.03% | 1.3% | **22.7%** | **2.5%** |

can obtain a solution with 15 vehicles travelling a total distance of 1570.05. PPO-GSF can obtain a solution with 15 vehicles travelling a total distance of 1679.80. The best-known solution for RC101 is a total travelled distance of 1696.94 by 14 vehicles.




(a) RC101 (b) RC102 (c) RC105

(d) RC201 (e) RC202 (f) RC205

**Figure 5.4:** *Influence of policy update mechanisms on the learning models (type-RC problem instances)*

In conclusion, the experimental results show that both the PPO-GSF and DQN-GSF methods can support effective learning in GSF to automatically generate evolutionary algorithms for solving different types of CVRPTW instances. In particular, with a neural network approximator, PPO-GSF, the policy-based model, is more effective than DQN-GSF, the value-based model.

Table 5.8: Comparisons on selected type-RC instances (influence of policy update mechanisms). $*$ indicates PPO-GSF is significantly different from DQN-GSF, i.e. $p < 0.05$

| Instance | | RC101 | RC102 | RC105 | RC201 | RC202 | RC205 |
|---|---|---|---|---|---|---|---|
| Best-known solutions | | 15696.94 [152] | 13554.72 [152] | 14628.44[156] | 5406.91[147] | 4367.09[155] | 5297.19[147] |
| RT[144] | | 16623.58 | 14477.54 | 14733.56 | 5438.89 | 5165.57 | 5333.71 |
| HG[148] | | 15697.43 | 13558.07 | 14637.15 | 5418.86 | 4665.56 | 5302.42 |
| DQN-GSF | AVG | 17586.44 | 16146.86 | 17041.4 | 5661.65 | **5351.60** | 5535.85 |
| | SD | 728.65 | **486.43** | 499.10 | 269.55 | **45.33** | 284.21 |
| | BEST | **16570.05** | 15526.71 | 16603.84 | 5507.77 | 5293.64 | 5361.69 |
| | GAP | **5.56%** | 14.55% | 13.5% | 1.87% | 21.2% | **1.2%** |
| PPO-GSF | AVG | **17521.35** $*$ | **16005.04** $*$ | **16543.15** $*$ | **5567.00** $*$ | 5359.51 | **5450.50** $*$ |
| | SD | **410.16** | 525.97 | **309.87** | **78.01** | 57.68 | **36.60** |
| | BEST | 16679.80 | **15511.07** | **15617.44** | **5467.35** | **5246.94** | **5359.96** |
| | GAP | 6.3% | **14.4%** | **6.8%** | **1.12%** | **20.1%** | **1.2%** |

There are mainly two reasons. Firstly, policy-based methods can learn stochastic policies while value-based methods can only learn deterministic policies. Policy-based methods are more capable of better environmental exploration. Secondly, PPO-GSF can ensure that the learned policy is monotonically increasing due to its effective value function optimisation method, leading to better exploitation.

Table 5.9: Generality across the same-type of instances. $*$ indicates PPO-GSF is significantly different from DQN-GSF, i.e. $p < 0.05$

| Instance | | R102 | R105 | R201 | R202 | R205 |
|---|---|---|---|---|---|---|
| Best-known solutions | | 18486.12 [144] | 15377.11 [144] | 5252.37[148] | 4191.7[149] | 3994.42[149] |
| RT[144] | | 18486.12 | 15377.11 | 5281.58 | 5088.07 | 4063.24 |
| HG[148] | | 18490.39 | 15377.11 | 5252.37 | 4198.45 | 1462.06 |
| DQN-GSF | AVG | 19198.08 | 16446.11 | 5369.89 | 5180.19 | 4165.78 |
| | SD | **455.54** | 438.62 | 39.99 | **31.71** | 58.63 |
| (trained policy) | BEST | **18499.58** | 15475.15 | 5325.66 | 5131.13 | **4069.41** |
| | GAP | **0.07%** | 0.64% | 1.40% | 22.41% | **1.88%** |
| PPO-GSF | AVG | **18918.03** $*$ | **16343.12** $*$ | **5344.44** | **5103.13** $*$ | **4135.23** |
| | SD | 486.78 | **297.87** | **24.74** | 273.80 | **26.08** |
| (trained policy) | BEST | 18507.80 | **15451.57** | **5311.65** | **4286.01** | 4096.00 |
| | GAP | 0.12% | **0.48%** | **1.13%** | **2.25%** | 2.54% |

Table 5.10: Generality across different-type of instances (type-C). $*$ indicates PPO-GSF is significantly different from DQN-GSF, i.e. $p < 0.05$

| Instance | | C101 | C102 | C105 | C201 | C202 | C205 |
|---|---|---|---|---|---|---|---|
| Best-known solutions | | **10828.94** [144] | **10828.94** [144] | **10828.94**[144] | **3591.56**[144] | **3591.56**[144] | **3588.88**[144] |
| RT[144] | | **10828.94** | **10828.94** | **10828.94** | **3591.56** | **3591.56** | **3588.88** |
| HG[148] | | **10828.94** | **10828.94** | **10828.94** | **3591.56** | **3591.56** | **3588.88** |
| DQN-GSF | AVG | **10828.94** | 10856.95 | **10828.94** | **3591.56** | **3608.60** | **3588.88** |
| | SD | **0** | 52.99 | **0** | **0** | 25.82 | **0** |
| (trained policy) | BEST | **10828.94** | **10828.94** | **10828.94** | **3591.56** | **3591.56** | **3588.88** |
| | GAP | **0** | **0** | **0** | **0** | **0** | **0** |
| PPO-GSF | AVG | **10828.94** | 10867.18 | 10836.85 | **3591.56** | 3615.97 $*$ | **3588.88** |
| | SD | **0** | 57.35 | 15.84 | **0** | **12.53** | **0** |
| (trained policy) | BEST | **10828.94** | **10828.94** | **10828.94** | **3591.56** | **3591.56** | **3588.88** |
| | GAP | **0** | **0** | **0** | **0** | **0** | **0** |

Table 5.11: Generality across different-type of instances (type-RC). $*$ indicates PPO-GSF is significantly different from DQN-GSF, i.e. $p < 0.05$

| Instance | | RC101 | RC102 | RC105 | RC201 | RC202 | RC205 |
|---|---|---|---|---|---|---|---|
| Best-known solutions | | 15696.94[152] | 13554.72 [152] | 14628.44[156] | 5406.91[147] | 4367.09[155] | 5297.19[147] |
| RT[144] | | 16623.58 | 14477.54 | 14733.56 | 5438.89 | 5165.57 | 5333.71 |
| HG[148] | | 15697.43 | 13558.07 | 14637.15 | 5418.86 | 4665.56 | 5302.42 |
| DQN-GSF | AVG | 17402.55 | **15552.85** | 16933.63 | 5561.91 | 5332.53 | 5441.44 |
| | SD | 662.31 | **22.76** | 473.08 | **38.81** | 45.89 | **41.46** |
| (trained policy) | BEST | **16650.43** | 15524.71 | 16596.77 | 5505.18 | **5243.20** | **5381.83** |
| | GAP | **6.07%** | 14.53% | 13.45% | 1.82% | **20.06%** | **1.60%** |
| PPO-GSF | AVG | **17221.51** $*$ | 15562.96 | **16633.88** $*$ | **5552.57** | **5323.74** | **5440.90** |
| | SD | **501.87** | 425.14 | **28.28** | 42.40 | **44.29** | 42.06 |
| (trained policy) | BEST | 16671.46 | **14678.64** | **16592.69** | **5478.44** | 5251.08 | 5396.30 |
| | GAP | 6.20% | **8.29%** | **13.42%** | **1.30%** | 20.2% | 1.68% |

### 5.3.2 Generality of the Learning Models

The training process of RL-GSF models is very time-consuming. This section investigates the generality of the policies trained by the proposed RL-GSF models, potentially reducing the time and reusing policies learned on automated algorithm design in solving new problem instances. Analysis has been conducted from two aspects: generality across the same-type instances and generality across different-type instances.

**Generality across the Same-type Instances:**

The policies trained on instance R101 by DQN-GSF and PPO-GSF are used to validate their generality to other type-R instances. Results in Table 5.9 of applying these policies to other five instances demonstrate a good degree of generalisation. Policies trained by DQN-GSF lead to a GAP less than 2% apart from instance R202. With PPO-GSF, the GAP is less than 3% in all instances, obtaining comparable results to the best-known results in the literature [143].

**Generality across Different-type Instances:**

Generality of the policies trained on instance R101 by DQN-GSF and PPO-GSF are validated by directly applying them to type-C and type-RC instances with different features. Results in Tables 5.10-5.11 for twelve other instances again demonstrate the generality of the trained policies. For type-C instances, all the GAP values are equal to 0, which means the trained policies

of DQN-GSF and PPO-GSF can produce the current best-known solutions. On the type-RC instances, in most cases, the trained policies can obtain non-dominated solutions to the best-known solutions. For example, on RC101, the trained policy of PPO-GSF can obtain a solution of 15 vehicles travelling a total distance of 1671.46, while the best-known solution is of a total travelled distance of 1696.94 by 14 vehicles.

In conclusion, the experimental results show that the algorithms designed automatically by DQN-GSF/PPO-GSF are able to produce high-quality solutions for different problem instances, of the same and also different types. This indicates that the proposed framework is reliable for different scenarios, which is the aim of the automated algorithm design.

## 5.4 Summary

In this chapter, reinforcement learning methods, Deep Q-Network (DQN) and Proximal Policy Optimisation (PPO), are devised within the established unified GSF to automatically design population-based algorithms by intelligently selecting appropriate combinations of the algorithmic components (i.e. evolution operators) during different stages of the optimisation process. The proposed models showed to be able to effectively design algorithms within GSF, by learning from interactions with the environment (optimisation process).

The performance of the proposed two reinforcement learning models has been evaluated on different benchmark instances of the capacitated vehicle routing problem with time windows to investigate their effectiveness and generality. Regarding the effectiveness of the learning models, investigations on the Q-value function approximator and policy update mechanism show that the policy-based models with a neural network function approximator (i.e. PPO) are more suitable to automatically design search algorithms. Regarding the generality, the policies learned on one instance are applied across the same-type and different-type instances. The results validate the generality of the

trained policies of DQN-GSF and PPO-GSF models. This provides promising evidence in learning reusable new knowledge in designing algorithms based on the basic algorithmic components within the unified general search framework.

# Chapter 6

# Automated Composition of Evolution Operators and Selection Heuristics

Chapter 3 identified three key research issues within the proposed general search framework and Chapter 5 aimed to address the first key research issue, i.e. automated composition of evolution operators, with the key features identified in Chapter 4. This chapter focuses on the other two research issues: automated composition of selection heuristics and of both evolution operators and selection heuristics together.

## 6.1 Introduction

A general combinatorial optimisation problem (GCOP) model has recently been built to define the design of search algorithms itself as a COP, to which the solutions are new general-purpose algorithms composed of basic algorithmic components [13]. With the basic algorithmic components in the GCOP model, the AutoGCOP framework has been built for automated design of local search algorithms [160]. In this thesis, another framework, GSF, has been further developed to support the automated design of both local search algo-

rithms and population-based algorithms in Chapter 3. This thesis focuses on automatically designing population-based algorithms within GSF. Three key research issues are defined in Chapter 3.

The first issue, i.e. automated composition of evolution operators, has been extensively investigated in the literature, as evolution operators are considered as the most important algorithmic components in designing evolutionary algorithms. Examples include genetic algorithm for the travelling salesman problem (TSP) [133], hyper-heuristics for unmanned aerial vehicles [136], the set covering problem [161], the set packing problem [162] and several COPs within the HyFlex framework [25].

Automated composition of selection heuristics, including selection for evolution heuristics (e.g., tournament selection and roulette wheel selection) and selection for replacement heuristics (e.g., comma-selection and plus-selection), has attracted less attention compared to that of evolution operators. However, they are also important algorithmic components for designing successful search algorithms, i.e. determining how individuals should be selected as parents to produce new candidate solutions. Example algorithms include the genetic programming based hyper-heuristics for the NK-landscape benchmark problem [163] and the grammatical evolution based hyper-heuristics for the 0-1 Knapsack problem [164].

Furthermore, there has been limited literature regarding automated composition of evolution operators and selection heuristics simultaneously within a general framework of algorithms. This expands the design space into a high-dimensional one, thus posing new challenges to machine learning.

Overall, the present research has been conducted with the following two motivations. Firstly, while some research has been carried out on automated composition of evolution operators, there has been limited amount of systematic investigation into the automated design of search algorithms through composition of different basic algorithmic components, namely selection heuristics and evolution operators. Note that the focus of this thesis is on automated algo-

rithm composition. Although extensive literature and surveys exist regarding the other two types of automated algorithm design, namely automated algorithm configuration [165] [166] and automated algorithm selection [167], they are beyond the scope of this thesis. Therefore, based on the unified GSF with defined basic algorithmic components, this chapter seeks to systematically investigate different design spaces, aiming to acquire transferable or reusable knowledge in automated algorithm design. Secondly, few studies have integrated machine learning to systematically investigate the new learning task of automated algorithm design. One of the key issues is on how to cope effectively with the high-dimensional algorithm design space. More specifically, we aim to answer the following three research questions (RQ):

- *RQ1* How can the design space of algorithms (considering different design decisions) be systematically explored to acquire transferable or reusable knowledge?
- *RQ2* How can successful machine learning be built to effectively deal with the high-dimensional search space of the automated algorithm design problem? What kind of learning mechanism would be useful to handle such search space?
- *RQ3* Can the discovered knowledge be transferred into solving new problem instances?

The goal of this chapter is to devise an effective learning method to systematically investigate the impact of individual algorithmic components and the synergy of multiple algorithmic components. Specifically, this chapter addresses the following research objectives:

1. To systematically explore the design spaces of algorithms within different modules of the GSF with controlled experiments against that of ad hoc design space without a framework, which is usually considered in the literature. (RQ1)

2. To propose an advanced reinforcement learning technique with a maximum entropy mechanism to tackle the above-mentioned problem of automated algorithm design which is defined as a new learning task with a continuous state space and a high-dimensional discrete action space. (RQ2)

3. To evaluate the performance of the devised learning models in terms of their effectiveness and generality when transferring knowledge discovered into solving new problem instances.(RQ3)

The rest of this chapter is organised as follows. Detailed descriptions of the devised learning models are given in Section 6.2. The experiments and discussions are shown in Section 6.3. Finally, Section 6.4 concludes this chapter.

## 6.2 Proposed Maximum Entropy Reinforcement Learning Method

Before introducing the proposed Actor-Critic with Entropy (ACE) method, a modified soft actor critic (SAC) method [168] for discrete action space, the problem of algorithm design is defined as a new reinforcement learning task within GSF, as shown in Figure 6.1. The states are defined by search-dependent and instance-dependent features which are identified in Chapter 4. The actions are defined by evolution operators and/or selection heuristics. Different from other RLs, the reward scheme is designed to maximise the expected accumulated reward $r$ and the expected entropy of the policy $\mathcal{H}$ as shown in Equation (6.1).

$$r_{\pi}\left(s_t, a_t\right) = r\left(s_t, a_t\right) + \alpha \cdot \mathcal{H}\left[\pi\left(a_t \mid s_t\right)\right] \tag{6.1}$$

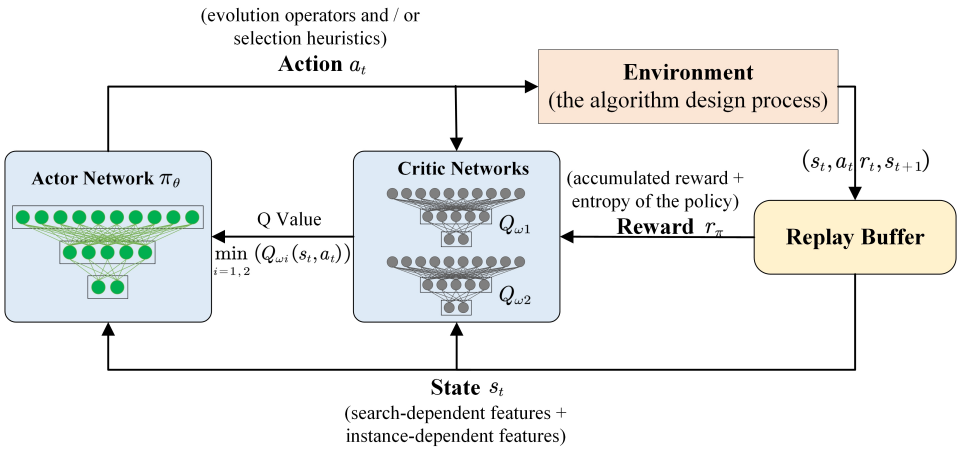



**Figure 6.1:** *Proposed reinforcement learning for automated composition within GSF*

The entropy $\mathcal{H}\left[\pi\left(a_t \mid s_t\right)\right]$ essentially introduces a noise to reinforcement learning weighted by a coefficient $\alpha$. Higher values of $\alpha$ tend to give more chances to actions which are rarely or never selected; while lower values of $\alpha$ emphasise on utilising the actions with good historical performance. As $\alpha$ decreases, more emphasis is given on the accumulated reward $r$ compared to the entropy of the policy $\mathcal{H}$. In other words, as $\alpha$ decreases, the learning process focuses

more on exploitation than exploration.

Three entropy coefficient adjustment schemes are proposed as follows, and evaluated in Section 6.3 to strike a balance between exploration and exploitation in the learning.

- fixed scheme (FS): the entropy coefficient $\alpha$ is set to a fixed value, i.e. $\alpha = 0.5$
- linear adaptive scheme (LAS): decrease $\alpha$ linearly, i.e. $\alpha_{t+1} = \alpha_t \cdot 0.9998$
- non-linear adaptive scheme (NLAS): decrease $\alpha$ nonlinearly, i.e. with a neural network

Apart from the maximum entropy mechanism to balance exploration and exploitation, ACE is enhanced with two mechanisms as shown in Figure 6.1. Firstly, an actor-critic architecture, $\pi$ is devised with one policy network (Actor), and two separate critic networks $Q_{\omega_1}$ and $Q_{\omega_2}$ (with the same structure) are used simultaneously to eliminate overestimation. The minimum of $Q_{\omega_1}$ and $Q_{\omega_2}$ is chosen exporting the $Q$ value. Secondly, the experience replay buffer is utilised to break the correlations between the stored experience $(s_t, a_t, r_t, s_{t+1})$ and reuse collected experience multiple times.

The pseudocode of ACE-GSF is shown in Algorithm 3. Specifically, an action is selected based on the current policy network (i.e. Actor network) (Line 9, Algorithm 3) from the set of three key components of search algorithms, namely: selection for evolution heuristic, evolution operator, and selection for replacement heuristic (Line 10-12, Algorithm 3). The reward and next state are observed and the corresponding experience $(s_t, a_t, r_t, s_{t+1})$ is stored in the replay buffer (Line 13, Algorithm 3). A random mini-batch of experiences is sampled to train the critic evaluation networks using a temporal difference algorithm (Line 14-15, Algorithm 3). After that, the actor network is updated by minimising the loss function (Line 16, Algorithm 3). The entropy coefficient $\alpha$ is then updated based on the pre-defined adjustment scheme. The process

is iterated at each timestep until the end of the episode. In the process, the target networks are updated using a soft update strategy (Line 18, Algorithm 3).

**Algorithm 3** Pseudocode of ACE-GSF

1: Randomly initialise critic networks $Q_{\omega_1}$, $Q_{\omega_2}$ with parameters $\omega_1$ and $\omega_2$, actor network $\pi_\theta$ with parameter $\theta$
2: Initialise target critic network $Q_{\omega_1{}^-}$, $Q_{\omega_2{}^-}$ with parameters $\omega_1{}^- \leftarrow \omega_1, \omega_2{}^- \leftarrow \omega_2$
3: Initialise replay buffer $D$, the number of episode $NoE$, the number of timesteps in one episode $NoT$
4: Generate initial population, record the initial state $s_0$
5: **for** episode $k = 1$ to $NoE$ **do**
6: Initialise the state $s_0$
7: **for** timestep $t = 1$ to $NoT$ **do**
8: Observe the current state $s_t$ by calculating values of different state features
9: Select an action based on the current policy $a_t = \pi_\theta(s_t)$
10: Select parents using a selection heuristic $h_i\,(i = 1, 2, ..., 6)$ from $H_{SE}$ by performing the selected action $a_t$
11: Generate offspring population using an evolution operator from $O_E$ by performing the selected action $a_t$
12: Update the population using a selection heuristic $h_i\,(i = 7, 8)$ from $H_{SR}$ by performing the selected action $a_t$
13: Observe reward $r_t$ based on Equation (6.5) and Equation (6.6), and next state $s_{t+1}$, store experience $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $D$
14: Sample random mini-batch of experiences $[s_j, a_j, r_j, s_{j+1}]_J$ ($J$ denotes the size of the sampled mini-batch) from replay buffer $D$
15: Update two critic networks $Q_{\omega_i}, i = 1, 2$ by minimising the loss function shown in Equation (6.2).
16: Update the actor network $\pi_\theta$ by minimising the loss function shown in Equation (6.4).
17: Update entropy coefficient $\alpha$ based on the selected adjustment scheme (i.e FS/LAS/NLAS)
18: Every $N$ timesteps, update the target critic networks with a soft strategy:
($\tau$ is a parameter that is typically chosen to be close to 1)
$\omega_1^- \leftarrow \tau\omega_1 + (1-\tau)\,\omega_1^-$
$\omega_2^- \leftarrow \tau\omega_2 + (1-\tau)\,\omega_2^-$
19: **end for**
20: **end for**

The critic networks $Q_{\omega_1}$, $Q_{\omega_2}$ are updated using temporal difference algorithm. The loss function is shown in Equation (6.2) and the temporal difference target $y_j$ is shown in Equation (6.3). The actor network $\pi_\theta$ is updated by minimising

the loss function shown in Equation (6.4).

$$L = \frac{1}{J}\sum_{j=1}^{J}\left(y_j - Q_{\omega_i}\left(s_j, a_j\right)\right)^2 \tag{6.2}$$

$$\begin{aligned} y_j = r_j + \gamma(\min_{i=1,2} Q_{\omega_i^-}\left(s_{j+1}, a_{j+1}\right) - \alpha \log \pi_\theta\left(a_{j+1}\,|s_{j+1}\right)), \\ a_{j+1} \sim \pi_\theta\left(\cdot\,|s_{j+1}\right) \end{aligned} \tag{6.3}$$

$$L_\pi\left(\theta\right) = \frac{1}{J}\sum_{j=1}^{J}\left(\alpha \log \pi_\theta\left(a_j\,|s_j\right) - \min_{i=1,2} Q_{\omega_i}\left(s_j, a_j\right)\right) \tag{6.4}$$

### 6.2.1 State Representation

The state should provide sufficient information on the environmental status to support accurate selection of the action. In this chapter, two groups of features, namely search-dependent features, which identify the key attributes of the search process itself, and instance-dependent features, which accurately describe the properties of the problem instance, are employed to represent the state. Refer to Chapter 4 for more details.

The search-dependent features include the search stage, fitness improvement, the standard deviation/mean/skewness/kurtosis/amplitude of fitness, and the lower(Q1) median(Q2)/upper(Q3) quartile of fitness.

The instance-dependent features include the number of available vehicles, vehicle capacity, average customer demand, average service time, average time-window size, average time-window overlaps between customers, and the percentage of time-constrained customers.

### 6.2.2 Action Representation

With regards to automated design of selection heuristics and evolution operators, the set of possible actions in each state is defined in Tables 3.5-3.7, respectively. Regarding automated design of both, a pair of evolution operators in Table 3.7 and selection heuristics in Tables 3.5-3.6 is defined as the

action. Refer to Chapter 3 for more details.

### 6.2.3 Reward Scheme

ACE is designed to learn a policy of a high reward while acting as randomly as possible. Note that the reward scheme in ACE is different from that in previous chapters since it aims to maximise both the expected accumulated reward and policy entropy simultaneously, whereas the reward schemes in previous chapters only focus on maximising the expected accumulated reward.

The first objective, as shown in Equations (6.5) and (6.6), is calculated based on the fitness improvement of the current population (i.e $f_{current}$) over the initial population (i.e $f_{initial}$). When population fitness is above a certain threshold (i.e $a$), which means that the search process enters the later stages of evolution, a larger reward is assigned for the same fitness improvement with a log function. The second objective of maximising the entropy of the policy is calculated based on the term $\mathcal{H}\left[\pi\left(a_t \mid s_t\right)\right]$. These two objectives are aggregated into a single objective function, i.e. minimising the sum of them.

$$r = \frac{f_{current}}{f_{initial}} \tag{6.5}$$

$$r_\pi\left(s_t, a_t\right) = \begin{cases} -r + \alpha \cdot \mathcal{H}\left[\pi\left(a_t \mid s_t\right)\right], if\ r > a \\ -r - \log_{10}\left(r\right) + \alpha \cdot \mathcal{H}\left[\pi\left(a_t \mid s_t\right)\right], if\ r \leqslant a \end{cases} \tag{6.6}$$

## 6.3 Experiments and Discussions

The experimental investigations aim to address two research issues: (1) the effectiveness of the learning models when tackling the design space of selection heuristics; (2) the effectiveness and generality of the learning models on the whole algorithm design space. To address the first issue, three ACE variants with different entropy coefficient adjustment schemes are validated in Section 6.3.1. On both selection heuristics and evolution operators, three ACE vari-

ants are assessed in Section 6.3.2. The experiments considering only evolution operators are conducted to the comparison to serve as the baseline. To analyse the generality of the learning models, the trained policies are directly applied to the same and different types of problem instances in Section 6.3.2. The experimental settings in this chapter, such as the machine information and the computational time, are identical as those in Chapter 4.

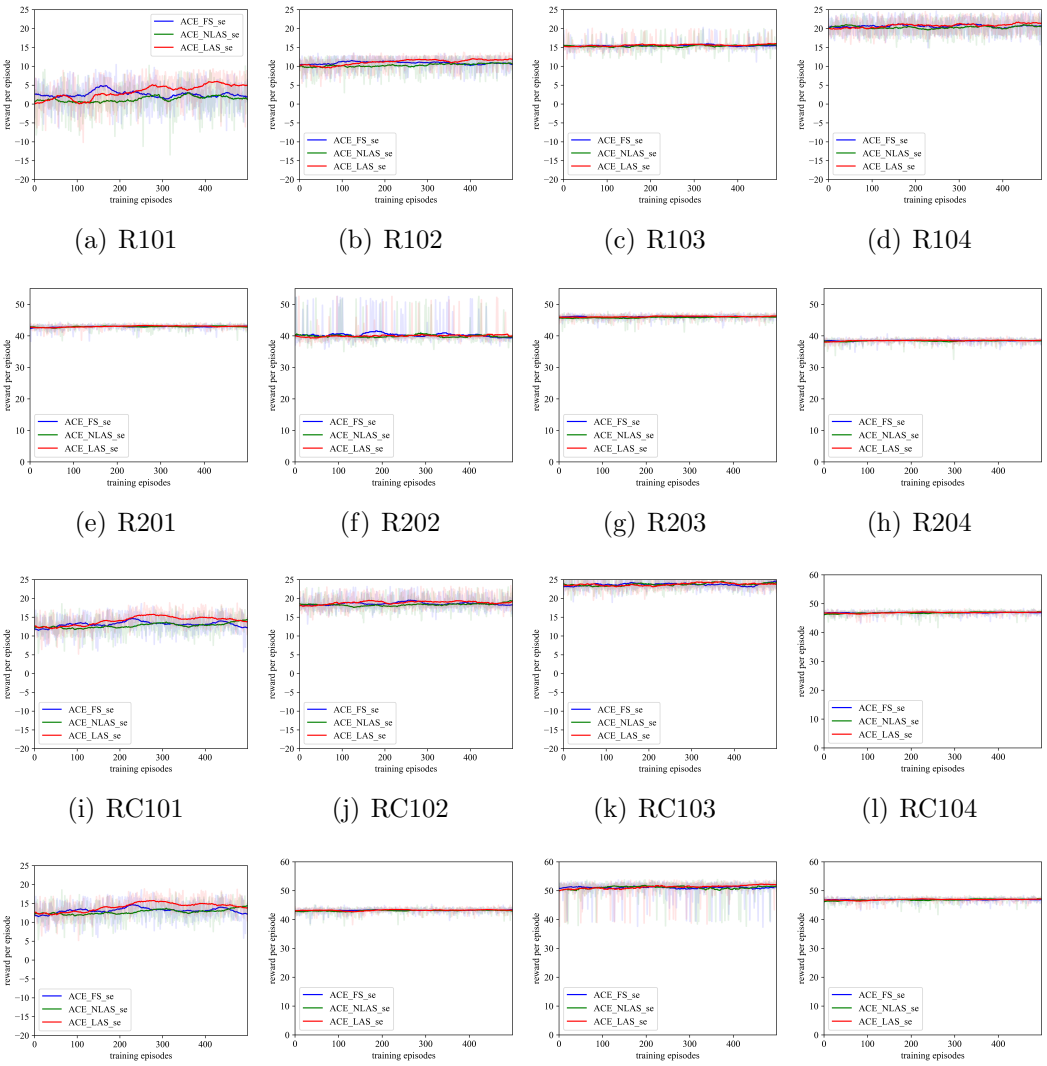
### 6.3.1 Automated Composition of Selection Heuristics

The proposed reinforcement learning method, namely ACE, is applied to automate the process of algorithm design by learning to adapt appropriate selection heuristics (including selection for evolution heuristics and selection for replacement heuristics).

#### Automated Composition of Selection for Evolution Heuristics:

With learning on the design space of only selection for evolution heuristics, three ACE variants with different entropy coefficient settings, namely ACE_FS, ACE_NLAS and ACE_LAS with a fixed/non-linear/linear entropy coefficient adjustment scheme respectively, are investigated. We fixed the algorithmic components in the other two modules as: $o_{ins\_bw}$ for the Evolution module and $h_8$ for Selection for Replacement module. These two components are the most frequently called evolution operator [169] and the most adopted selection for replacement heuristic in the literature.

As shown in Figure 6.2, results on different instances show that learning on selection for evolution heuristics alone has no significant impact on the performance of search algorithms throughout the training process. One possible reason is that although selection for evolution heuristics determine which individual should be combined to produce new solutions, it would not have so much impact if the algorithmic component in the Evolution module is fixed. In other words, there is limited scope for evolution. This supports human




(a) R101 (b) R102 (c) R103 (d) R104

(e) R201 (f) R202 (g) R203 (h) R204

(i) RC101 (j) RC102 (k) RC103 (l) RC104

(m) RC201 (n) RC202 (o) RC203 (p) RC204

**Figure 6.2:** *Performance comparison during the training process (learning on selection for evolution heuristics, i.e. se)*

experience in designing search algorithms and results reported in the literature, i.e. the selection heuristics have less impact on the performance of search algorithms and therefore there is no need to focus on the design of them.

As shown in Tables 6.1-6.4, the performance of three ACE variants is slightly but not significantly better than that of the non-learning method during the testing process. Note that "Non-learning" denotes the search algorithm with all fixed components in all modules ($h_1$, $o_{ins\_bw}$, $h_8$). This indicates that learning on selection for evolution heuristic has little impact on the algorithm performance, which is consistent with the findings shown in Figure 6.2.

Table 6.1: Performance comparison during the testing process (learning on selection for evolution heuristics, i.e. se), type-R1

| | | R101 | R102 | R103 | R104 |
|---|---|---|---|---|---|
| Best-known Solutions | | **20645.79** [145] | **18486.12** [144] | **14292.68** [146] | **10007.24** [147] |
| ACE_FS_se | AVG | 21019.62 | 18783.83 | 15336.91 | 11419.82 |
| | SD | 226.28 | 215.70 | 208.64 | 461.40 |
| | BEST | 20918.92 | 18706 | 14437.66 | 11083.57 |
| | GAP | 1.32% | 1.19% | 1.01% | 10.76% |
| ACE_NLAS_se | AVG | 21006.91 | 18761.66 | 15344.38 | 11333.54 |
| | SD | 230.78 | 213.22 | **205.63** | 405.13 |
| | BEST | **20887.4** | **18660.31** | 14459.78 | **11082.82** |
| | GAP | **1.17%** | **0.94%** | 1.17% | **10.75%** |
| ACE_LAS_se | AVG | **20937.17** | **18717.35** | **15269.27** | **11141.26** |
| | SD | **11.01** | **23.98** | 283.22 | **31.66** |
| | BEST | 20914.45 | 18669.80 | **14415.92** | 11089.68 |
| | GAP | 1.30% | 0.99% | **0.86%** | 10.82% |
| Non-learning | AVG | 20957.89 | 18769.97 | 15342.55 | 11373.85 |
| | SD | 30.37 | 215.69 | 207.39 | 443.38 |
| | BEST | 20901.02 | 18675.05 | 14448.89 | 11085.36 |
| | GAP | 1.24% | 1.02% | 1.09% | 10.77% |

Table 6.2: Performance comparison during the testing process (learning on selection for evolution heuristics, i.e. se), type-R2

| | | R201 | R202 | R203 | R204 |
|---|---|---|---|---|---|
| Best-known Solutions | | **5252.37** [148] | **4191.7** [149] | **3939.54** [150] | **2825.52** [151] |
| ACE_FS_se | AVG | 5479.97 | 5214.98 | 4168.84 | 3936.29 |
| | SD | 41.04 | 290.99 | 30.00 | 25.70 |
| | BEST | 5398.13 | 4317.59 | 4111.00 | 3891.49 |
| | GAP | 2.78% | 3% | 4.35% | 37.73% |
| ACE_NLAS_se | AVG | 5469.77 | 5171.22 | 4166.85 | 3932.16 |
| | SD | 25.33 | 335.45 | 32.48 | 22.07 |
| | BEST | **5392.77** | 4316.40 | **4087.81** | **3878.91** |
| | GAP | **2.67%** | 2.97% | **3.76%** | **37.28%** |
| ACE_LAS_se | AVG | **5444.77** | **5096.12** | **4159.70** | **3923.55** |
| | SD | **11.32** | 376.51 | **23.47** | **18.34** |
| | BEST | 5413.81 | **4307.61** | 4111.65 | 3891.72 |
| | GAP | 3.07% | **2.77%** | 4.37% | 37.73% |
| Non-learning | AVG | 5533.63 | 5384.61 | 4252.69 | 4020.82 |
| | SD | 24.47 | **49.37** | 23.49 | 19.65 |
| | BEST | 5481.47 | 5287.09 | 4184.90 | 3989.41 |
| | GAP | 4.36% | 26.13% | 6.23% | 41.19% |

Table 6.3: Performance comparison during the testing process (learning on selection for evolution heuristics, i.e. se), type-RC1

| | | RC101 | RC102 | RC103 | RC104 |
|---|---|---|---|---|---|
| Best-known Solutions | | **15696.94** [152] | **13554.75** [152] | **12261.67** [153] | **12135.487** [154] |
| ACE_FS_se | AVG | 17074.88 | 15223.6 | 13126.46 | 11926.51 |
| | SD | 432.50 | 481.39 | 469.69 | 486.41 |
| | BEST | **16734.77** | 14627.56 | 12342.64 | 11223.87 |
| | GAP | **6.61%** | 7.91% | 0.66% | 0.79% |
| ACE_NLAS_se | AVG | 17030.73 | 15258.4 | 12930.19 | 11968.47 |
| | SD | 420.36 | 471.00 | 511.73 | 477.55 |
| | BEST | 16758.36 | 14619.93 | 12364.56 | 11190.65 |
| | GAP | 6.76% | 7.86% | 0.84% | 0.50% |
| ACE_LAS_se | AVG | **16802.26** | **14617.32** | **12742.95** | **11433.27** |
| | SD | **15.93** | **27.55** | 458.11 | **400.82** |
| | BEST | 16766.4 | **14557.79** | 12347.93 | 11197.41 |
| | GAP | 6.81% | **7.4%** | 0.70% | 0.56% |
| Non-learning | AVG | 17105.48 | 15194.24 | 13210.88 | 12059.4 |
| | SD | 478.03 | 481.82 | **401.69** | 419.97 |
| | BEST | 16745.33 | 14597.72 | **12341.12** | **11171.26** |
| | GAP | 6.68% | 7.69% | **0.65%** | **0.32%** |

Table 6.4: Performance comparison during the testing process (learning on selection for evolution heuristics, i.e. se), type-RC2

| | | RC201 | RC202 | RC203 | RC204 |
|---|---|---|---|---|---|
| Best-known Solutions | | **5406.91** [147] | **4367.09** [155] | **4049.62** [155] | **3798.41** [147] |
| ACE_FS_se | AVG | 5682.11 | 5462.07 | 4345.10 | 3977.98 |
| | SD | 46.89 | 34.51 | 284.60 | 25.68 |
| | BEST | **5538.27** | 5400.92 | 4172.10 | 3934.86 |
| | GAP | **2.43%** | 23.71% | 3.02% | 3.59% |
| ACE_NLAS_se | AVG | 5689.10 | 5458.57 | 4247.98 | 3979.04 |
| | SD | 70.65 | 32.83 | 38.42 | 25.79 |
| | BEST | 5550.42 | 5382.51 | 4173.24 | 3928.85 |
| | GAP | 2.65% | 23.29% | 3.05% | 3.43% |
| ACE_LAS_se | AVG | **5617.92** | **5394.25** | **4211.85** | **3946.07** |
| | SD | **17.79** | **22.38** | **25.11** | **11.12** |
| | BEST | 5580.24 | **5332.49** | **4159.75** | 3924.74 |
| | GAP | 3.21% | **22.15%** | **2.72%** | 3.32% |
| Non-learning | AVG | 5683.60 | 5431.68 | 4238.98 | 3973.20 |
| | SD | 55.43 | 42.20 | 38.76 | 28.32 |
| | BEST | 5594.09 | 5347.23 | 4163.38 | **3905.85** |
| | GAP | 3.46% | 22.48% | 2.81% | **2.83%** |

**Automated Composition of Selection for Replacement Heuristics:**

Similarly, the effectiveness of the ACE method on the design space of only selection for replacement heuristics is validated with the ACE_FS, ACE_NLAS, ACE_LAS and non-learning method. The components in other two modules are fixed as: $o_{ins\_bw}$ for Evolution module and $h_1$ for Selection for Evolution module, which are the most frequently called evolution operator [169] and the most adopted selection for evolution heuristic in the literature.

As shown in Figure 6.3, the proposed ACE methods are able to learn on selection for replacement heuristics throughout the training process. However, as Tables 6.5- 6.8 show, the RL-based methods (i.e. three ACE variants) have slightly but not significantly better performance than the non-learning method on all instances during the testing process. This indicates that selection for replacement heuristic has little impact on the algorithm performance although ACE methods have a relatively good learning performance. One possible explanation is that the action space is relatively small thus presents limited scope for learning.

Table 6.5: Performance comparison during the testing process (learning on selection for replacement heuristics, i.e. sr), type-R1

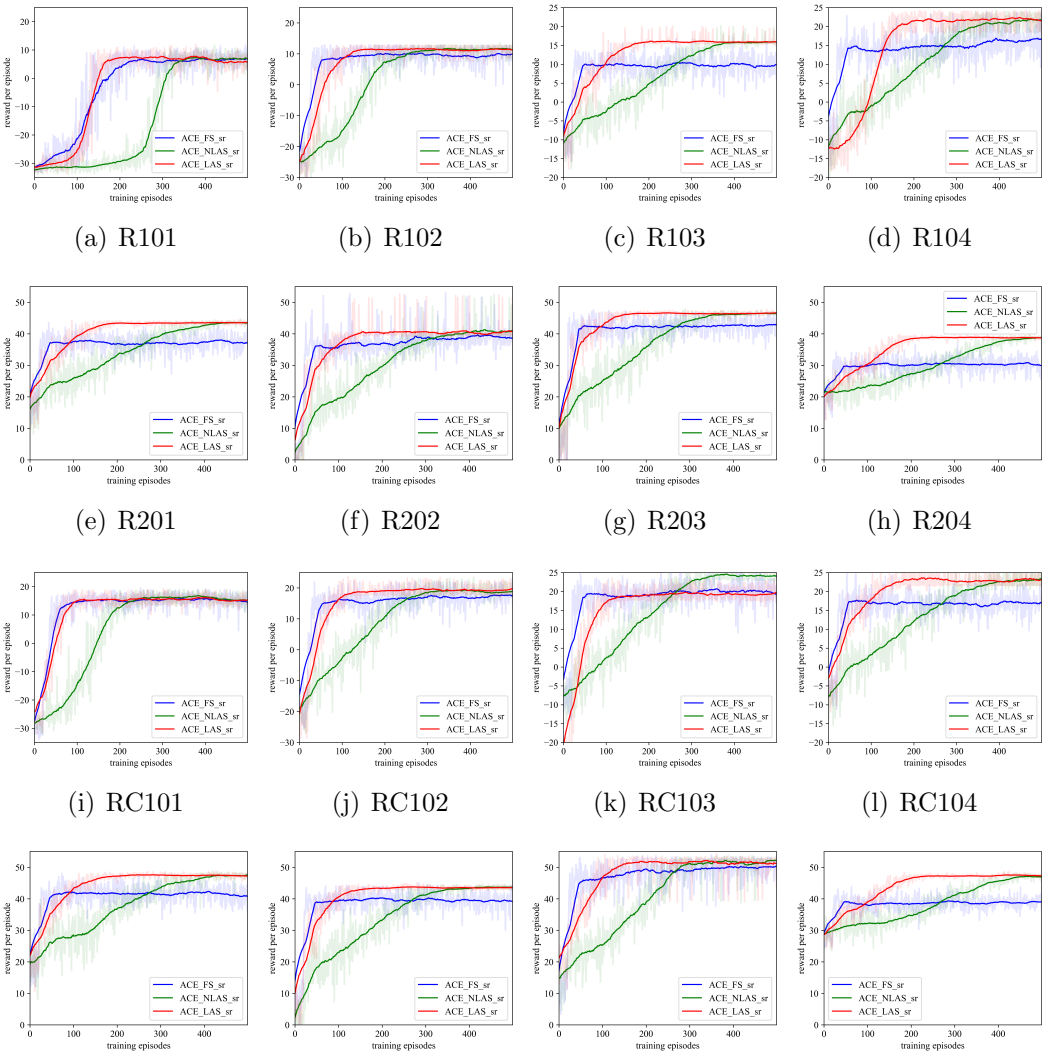
| | | R101 | R102 | R103 | R104 |
|---|---|---|---|---|---|
| Best-known Solutions | | **20645.79** [145] | **18486.12** [144] | **14292.68** [146] | **10007.24** [147] |
| ACE_FS_sr | AVG | 20953.99 | 18705.20 | **15253.01** | 11337.15 |
| | SD | 30.14 | 29.10 | 343.92 | 406.37 |
| | BEST | 20905.51 | 18659.77 | 14521.65 | 11097.61 |
| | GAP | 1.26% | 0.94% | 1.60% | 10.90% |
| ACE_NLAS_sr | AVG | 21039.63 | 18757.65 | 15296.03 | 11202.98 |
| | SD | 311.39 | 222.57 | 279.64 | 302.91 |
| | BEST | **20885.40** | 18669.78 | 14458.03 | 11070.86 |
| | GAP | **1.16%** | 0.99% | 1.16% | 10.63% |
| ACE_LAS_sr | AVG | **20941.11** | **18686.76** | 15287.74 | **11158.34** |
| | SD | **29.51** | **17.75** | 234.07 | **219.93** |
| | BEST | 20893.87 | **18628.8** | **14440.08** | **11057.46** |
| | GAP | 1.20% | **0.77%** | **1.03%** | **10.49%** |
| Non-learning | AVG | 20957.89 | 18769.97 | 15342.55 | 11373.85 |
| | SD | 30.37 | 215.69 | **207.39** | 443.38 |
| | BEST | 20901.02 | 18675.05 | 14448.89 | 11085.36 |
| | GAP | 1.24% | 1.02% | 1.09% | 10.77% |

Table 6.6: Performance comparison during the testing process (learning on selection for replacement heuristics, i.e. sr), type-R2

| | | R201 | R202 | R203 | R204 |
|---|---|---|---|---|---|
| Best-known Solutions | | **5252.37** [148] | **4191.7** [149] | **3939.54** [150] | **2825.52** [151] |
| ACE_FS_sr | AVG | 5485.78 | 5307.81 | 4184.56 | 3972.95 |
| | SD | **17.39** | **40.18** | 29.26 | 27.01 |
| | BEST | 5442.47 | 5220.64 | 4130.73 | 3928.06 |
| | GAP | 3.62% | 24.55% | 4.85% | 39.02% |
| ACE_NLAS_sr | AVG | 5532.74 | 5193.71 | 4155.69 | 3927.74 |
| | SD | 28.14 | 292.15 | 26.95 | 23.88 |
| | BEST | 5484.73 | **4290.74** | 4112.68 | **3884.60** |
| | GAP | 4.42% | **2.36%** | 4.4% | **37.48%** |
| ACE_LAS_sr | AVG | **5462.73** | **4860.06** | **4141.74** | **3912.04** |
| | SD | 41.66 | 432.75 | 29.79 | **16.42** |
| | BEST | **5391.18** | 4313.70 | **4079.64** | 3886.11 |
| | GAP | **2.64%** | 2.91% | **3.56%** | 37.54% |
| Non-learning | AVG | 5533.63 | 5384.61 | 4252.69 | 4020.82 |
| | SD | 24.47 | 49.37 | **23.49** | 19.65 |
| | BEST | 5481.47 | 5287.09 | 4184.90 | 3989.41 |
| | GAP | 4.36% | 26.13% | 6.23% | 41.19% |

Table 6.7: Performance comparison during the testing process (learning on selection for replacement heuristics, i.e. sr), type-RC1

| | | RC101 | RC102 | RC103 | RC104 |
|---|---|---|---|---|---|
| Best-known Solutions | | **15696.94** [152] | **13554.75** [152] | **12261.67** [153] | **12135.487** [154] |
| ACE_FS_sr | AVG | 17138.92 | 15075.5 | 13010.12 | 11932.74 |
| | SD | 495.57 | 489.15 | 480.12 | 487.54 |
| | BEST | 16729.1 | 14581.41 | 12380 | 11191.67 |
| | GAP | 6.58% | 7.57% | 0.97% | 0.50% |
| ACE_NLAS_sr | AVG | 17131.87 | 15335.74 | 13220.6 | 12009.46 |
| | SD | 504.41 | 434.11 | 382.15 | 453.92 |
| | BEST | **16695.41** | **14574.36** | 12381.99 | 11176.33 |
| | GAP | **6.36%** | **7.52%** | 0.98% | 0.37% |
| ACE_LAS_sr | AVG | **16789.31** | **14632.69** | **12408.88** | **11264.29** |
| | SD | **36.77** | **28.55** | **33.79** | **226.67** |
| | BEST | 16720.35 | 14577.61 | 12360.48 | 11188.16 |
| | GAP | 6.52% | 7.55% | 0.81% | 0.47% |
| Non-learning | AVG | 17105.48 | 15194.24 | 13210.88 | 12059.4 |
| | SD | 478.03 | 481.82 | 401.69 | 419.97 |
| | BEST | 16745.33 | 14597.72 | **12341.12** | **11171.26** |
| | GAP | 6.68% | 7.69% | **0.65%** | **0.32%** |




(a) R101 (b) R102 (c) R103 (d) R104

(e) R201 (f) R202 (g) R203 (h) R204

(i) RC101 (j) RC102 (k) RC103 (l) RC104

(m) RC201 (n) RC202 (o) RC203 (p) RC204

**Figure 6.3:** *Performance comparison during the training process (learning on selection for replacement heuristics, i.e. sr)*

### 6.3.2 Automated Composition of Selection Heuristics and Evolution Operators

**Effectiveness of the Devised Learning Models:**

Experimental results on the training process are shown in Figure 6.4 to investigate the effectiveness of extending the search space from evolution operators alone to both selection heuristics and evolution operators.

From Figure 6.4, we can clearly see that, despite having a worse starting point, all the ACE methods on the search space of both selection heuristics and evolution operators (both) outperform those on the search space of evolution

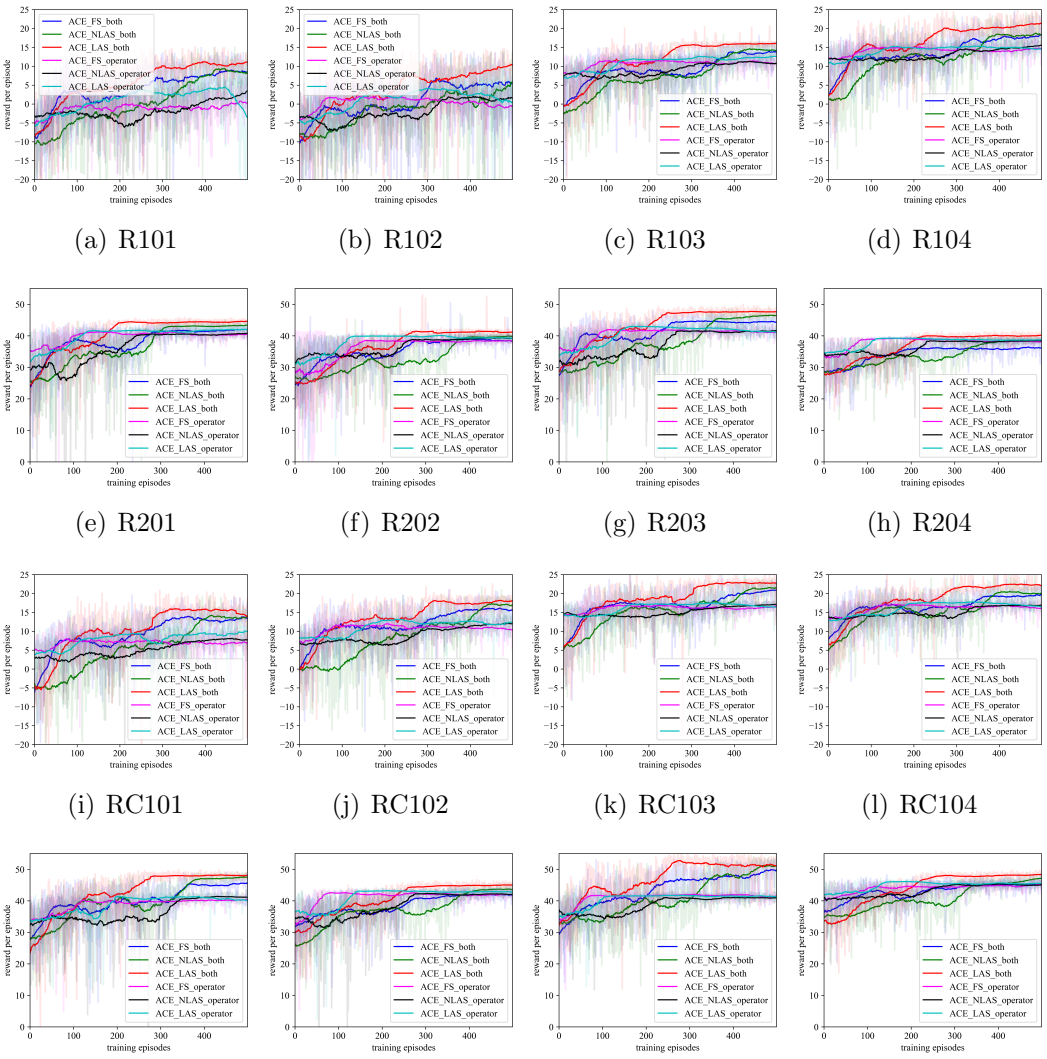
Table 6.8: Performance comparison during the testing process (learning on selection for replacement heuristics, i.e. sr), type-RC2

| | | RC201 | RC202 | RC203 | RC204 |
|---|---|---|---|---|---|
| Best-known Solutions | | **5406.91** [147] | **4367.09** [155] | **4049.62** [155] | **3798.41** [147] |
| ACE_FS_sr | AVG | 5664.16 | 5476.56 | 4252.20 | 4010.41 |
| | SD | 33.7 | **31.81** | 36.96 | 28.62 |
| | BEST | 5609.33 | 5422.93 | 4167.97 | 3962.13 |
| | GAP | 3.74% | 24.22% | 2.92% | 4.31% |
| ACE_NLAS_sr | AVG | 5663.75 | 5432.35 | 4308.48 | 3970.37 |
| | SD | 54.65 | 50.32 | 216.30 | 20.95 |
| | BEST | 5584.09 | 5341.78 | **4138.31** | 3929.21 |
| | GAP | 3.28% | 22.36% | **2.19%** | 3.44% |
| ACE_LAS_sr | AVG | **5624.67** | 5438.88 | **4217.56** | **3967.57** |
| | SD | **33.25** | 42.35 | **31.91** | **18.10** |
| | BEST | **5536.72** | **5331.38** | 4141.87 | 3925.76 |
| | GAP | **2.40%** | **22.12%** | 2.28% | 3.35% |
| Non-learning | AVG | 5683.60 | **5431.68** | 4238.98 | 3973.20 |
| | SD | 55.43 | 42.20 | 38.76 | 28.32 |
| | BEST | 5594.09 | 5347.23 | 4163.38 | **3905.85** |
| | GAP | 3.46% | 22.48% | 2.81% | **2.83%** |

operators alone (operator). This demonstrates the positive synergy between selection heuristics and evolution operators. In other words, proper collaboration between selection heuristics and evolution operators significantly improves the performance of search algorithms. More importantly, this indicates that human experience can help with algorithm design within a limited design space, i.e. using human expertise to fix the other components besides evolution operators leads to a better starting point, but machine learning outperforms within a larger algorithm design space and therefore has more potential to design better algorithms and attain better solutions.

Concerning learning on the search space of both selection heuristics and evolution operators, ACE_LAS_both performs better than the others, and ACE_FS_both and ACE_NLAS_both demonstrate competitive performance during the training process. This shows that the adaptive coefficient adjustment scheme can guide the agent to explore new actions at the early stage of the search process while also exploit the best actions at the later stage. The advantage of ACE_LAS_both over ACE_NLAS_both is that the simple linear coefficient adjustment is much less time-consuming, and therefore more computational time can be used to evolve the population and enhance the search performance.

Regarding the performance of ACE variants during the testing process, Tables




(a) R101 (b) R102 (c) R103 (d) R104

(e) R201 (f) R202 (g) R203 (h) R204

(i) RC101 (j) RC102 (k) RC103 (l) RC104

(m) RC201 (n) RC202 (o) RC203 (p) RC204

**Figure 6.4:** *Performance comparison during the training process (learning on evolution operators vs. learning on both selection heuristics and evolution operators)*

6.9-6.12 tabulate the statistical performances of ACE_FS_operator, ACE_NLAS_operator, ACE_LAS_operator concerning the search space of evolution operators, and ACE_FS_both, ACE_NLAS_both, ACE_LAS_both concerning the search space of both selection heuristics and evolution operators. Particularly, column "GAP" is the gap between the attained best fitness and the best-known results in the literature to demonstrate the overall performance of different ACE variants on the CVRPTW benchmarks.

In Tables 6.9-6.12, ACE_LAS_both attains better performance on solution quality than the other two ACE variants on most instances. Note that the only difference between ACE variants lies in the entropy coefficient adjust-

Table 6.9: Performance comparison during the testing process (learning on evolution operators vs. learning on both selection heuristics and evolution operators), type-R1

| | | R101 | R102 | R103 | R104 |
|---|---|---|---|---|---|
| Best-known Solutions | | **20645.79** [145] | **18486.12** [144] | **14292.68** [146] | **10007.24** [147] |
| ACE_FS_both | AVG | 20864.64 | 19267.41 | 15252.24 | 11630.38 |
| | SD | 395.65 | 516.22 | 150.05 | 483.08 |
| | BEST | 20653.99 | 18522.27 | 14613.02 | 11049.23 |
| | GAP | 0.04% | 0.20% | 2.24% | 10.41% |
| ACE_NLAS_both | AVG | 21119.96 | 19148.23 | 15222.18 | 11471.89 |
| | SD | 506.19 | 466.2739 | 200.952 | 482.93 |
| | BEST | 20653.76 | 18500.85 | 14350.38 | 11044.7 |
| | GAP | 0.04% | 0.08% | 0.40% | 10.37% |
| ACE_LAS_both | AVG | **20663.25** | 19193.66 | 15161.89 | **11069.15** |
| | SD | **7.22** | 446.35 | 269.76 | **19.38** |
| | BEST | **20652.27** | **18491.29** | **14348.23** | **11022.37** |
| | GAP | **0.03%** | **0.03%** | **0.39%** | **10.14%** |
| ACE_FS_operator | AVG | 20664.24 | 18653.01 | 15255.71 | 11092.17 |
| | SD | 13.08 | 339.90 | **7.28** | 20.39 |
| | BEST | 20653.76 | 18498.31 | 15243.27 | 11059.56 |
| | GAP | 0.04% | 0.07% | 6.65% | 10.52% |
| ACE_NLAS_operator | AVG | 20663.64 | **18512.82** | 15131.34 | 11126.13 |
| | SD | 9.26 | **12.75** | 293.55 | 205.91 |
| | BEST | 20653.99 | 18491.88 | 14400.16 | 11033.77 |
| | GAP | 0.04% | 0.03% | 0.75% | 10.26% |
| ACE_LAS_operator | AVG | 20788.71 | 18522.37 | **15129.87** | 11090.62 |
| | SD | 288.5524 | 14.02 | 305.60 | 35.27 |
| | BEST | 20653.76 | 18504.07 | 14353.07 | 11061.15 |
| | GAP | 0.04% | 0.097% | 0.42% | 10.53% |
| Non-learning | AVG | 20957.89 | 18769.97 | 15342.55 | 11373.85 |
| | SD | 30.37 | 215.69 | 207.39 | 443.38 |
| | BEST | 20901.02 | 18675.05 | 14448.89 | 11085.36 |
| | GAP | 1.24% | 1.02% | 1.09% | 10.77% |

ment scheme. From this, it is possible to infer that the linear entropy coefficient adjustment scheme is useful in striking the balance between exploration and exploitation during the learning process, particularly on the large search space of algorithm design. This observation is consistent with the results obtained during the training process.

Note that the only difference in the selected type-R1 instances lies in the customer time windows density, i.e. the percentage of customers with time windows of R101 and R102 is 100% and 75% respectively, and 50% and 25% for R103 and R104. The selected type-R2/type-RC1/type-RC2 instances have the same pattern. As shown in Figure 6.4, with the decrease of time windows density (i.e. with looser constraints), the starting points of all ACE variants increase. This observation indicates that the ACE methods learn better on instances with looser constraints. One possible reason is that the solution space of the instances with looser constraints has more feasible solution candidates,

Table 6.10: Performance comparison during the testing process (learning on evolution operators vs. learning on both selection heuristics and evolution operators), type-R2

| | | R201 | R202 | R203 | R204 |
|---|---|---|---|---|---|
| Best-known Solutions | | **5252.37** [148] | **4191.7** [149] | **3939.54** [150] | **2825.52** [151] |
| ACE_FS_both | AVG | 5351.79 | 5177.05 | 4065.49 | 3844.57 |
| | SD | 35.72 | **22.12** | 28.78 | 21.66 |
| | BEST | 5278.02 | 5146.30 | 4010.91 | 3796.73 |
| | GAP | 0.49% | 22.77% | 1.81% | 34.37% |
| ACE_NLAS_both | AVG | 5353.24 | 5169.60 | 4051.64 | 3836.50 |
| | SD | 36.00 | 22.24 | 30.32 | 25.38 |
| | BEST | 5284.13 | 5129.79 | 4006.80 | 3807.09 |
| | GAP | 0.60% | 22.38% | 1.71% | 34.74% |
| ACE_LAS_both | AVG | 5320.87 | **5050.10** | **4026.06** | 3825.20 |
| | SD | 15.84 | 266.36 | 14.00 | **8.44** |
| | BEST | **5277.71** | **4255.46** | 3999.29 | **3806.12** |
| | GAP | **0.48%** | **1.52%** | 1.52% | **34.71%** |
| ACE_FS_operator | AVG | 5318.49 | 5115.03 | 4033.03 | 3825.06 |
| | SD | **10.21** | 168.85 | **11.77** | 8.62 |
| | BEST | 5298.13 | 4361.74 | **3997.63** | 3782.99 |
| | GAP | 0.87% | 4.06% | **1.48%** | 33.89% |
| ACE_NLAS_operator | AVG | 5316.34 | 5065.46 | 4029.42 | **3816.46** |
| | SD | 10.57 | 258.14 | 16.08 | 8.64 |
| | BEST | 5298.88 | 4331.64 | 3998.09 | 3796.79 |
| | GAP | 0.89% | 3.34% | 1.49% | 34.38% |
| ACE_LAS_operator | AVG | **5314.33** | 5113.26 | 4077.97 | 3818.40 |
| | SD | 14.89 | 192.43 | 58.36 | 11.47 |
| | BEST | 5287.98 | 4265.06 | 4017.52 | 3795.08 |
| | GAP | 0.68% | 1.75% | 1.98% | 34.31% |
| Non-learning | AVG | 5533.63 | 5384.61 | 4252.69 | 4020.82 |
| | SD | 24.47 | 49.37 | 23.49 | 19.65 |
| | BEST | 5481.47 | 5287.09 | 4184.90 | 3989.41 |
| | GAP | 4.36% | 26.13% | 6.23% | 41.19% |

Table 6.11: Performance comparison during the testing process (learning on evolution operators vs. learning on both selection heuristics and evolution operators), type-RC1

| | | RC101 | RC102 | RC103 | RC104 |
|---|---|---|---|---|---|
| Best-known Solutions | | **15696.94** [152] | **13554.75** [152] | **12261.67** [153] | **12135.487** [154] |
| ACE_FS_both | AVG | 17550.06 | 15508.28 | 13474.65 | 12232.91 |
| | SD | 467.96 | 464.78 | 430.80 | **23.29** |
| | BEST | 16677.82 | 14521.81 | 12443.92 | 12176.53 |
| | GAP | 6.25% | 7.13% | 1.49% | 9.35% |
| ACE_NLAS_both | AVG | 17708.71 | 15444.12 | 13315.13 | 12235.36 |
| | SD | **312.92** | 511.43 | 223.25 | 24.31 |
| | BEST | 16705.59 | 14552.96 | **12321.2** | 12171.33 |
| | GAP | 6.43% | 7.36% | **0.49%** | 9.30% |
| ACE_LAS_both | AVG | 17555.36 | 15332.23 | 13273.83 | **12032.23** |
| | SD | 473.81 | 383.64 | 287.95 | 407.30 |
| | BEST | **16648.09** | **14516.89** | 12365.86 | **11168.88** |
| | GAP | **6.06%** | **7.10%** | 0.85% | **0.3%** |
| ACE_FS_operator | AVG | 17271.06 | 15480.25 | 13348.91 | 12072.58 |
| | SD | 478.25 | 205.12 | **17.19** | 333.02 |
| | BEST | 16681.82 | 14566.36 | 13302.72 | 11247.47 |
| | GAP | 6.27% | 7.46% | 8.49% | 1.01% |
| ACE_NLAS_operator | AVG | 17027.84 | 15479.54 | 13296.05 | 12065.08 |
| | SD | 462.77 | **201.48** | 211.14 | 344.85 |
| | BEST | 16674.02 | 14579.25 | 12354.11 | 11209.12 |
| | GAP | 6.22% | 7.56% | 0.75% | 0.66% |
| ACE_LAS_operator | AVG | **16951.76** | 15310.34 | 13302.72 | 12214.11 |
| | SD | 417.82 | 396.81 | 213.35 | 222.48 |
| | BEST | 16667.91 | 14561.73 | 12351.55 | 11225.98 |
| | GAP | 6.19% | 7.43% | 0.73% | 0.81% |
| Non-learning | AVG | 17105.48 | **15194.24** | **13210.88** | 12059.4 |
| | SD | 478.03 | 481.82 | 401.69 | 419.97 |
| | BEST | 16745.33 | 14597.72 | 12341.12 | 11171.26 |
| | GAP | 6.68% | 7.69% | 0.65% | 0.32% |

Table 6.12: Performance comparison during the testing process (learning on evolution operators vs. learning on both selection heuristics and evolution operators), type-RC2

| | | RC201 | RC202 | RC203 | RC204 |
|---|---|---|---|---|---|
| Best-known Solutions | | **5406.91** [147] | **4367.09** [155] | **4049.62** [155] | **3798.41** [147] |
| ACE_FS_both | AVG | 5534.21 | 5310.02 | 4306.63 | 3903.02 |
| | SD | 38.91 | 49.67 | 309.84 | 24.17 |
| | BEST | 5478.78 | **5204.87** | 4118.84 | **3842.72** |
| | GAP | 1.33% | **19.22%** | 1.71% | **1.17%** |
| ACE_NLAS_both | AVG | 5550.32 | 5288.01 | 4273.22 | 3910.53 |
| | SD | 48.62 | 46.29 | 264.60 | 30.43 |
| | BEST | **5449.35** | 5218.59 | 4128.85 | 3864.09 |
| | GAP | **0.78%** | 19.54% | 1.96% | 1.73% |
| ACE_LAS_both | AVG | 5510.25 | 5292.42 | **4161.03** | 3879.72 |
| | SD | 25.18 | 33.04 | 21.08 | 15.62 |
| | BEST | 5460.29 | 5209.88 | **4094.21** | 3843.16 |
| | GAP | 0.99% | 19.34% | **1.10%** | 1.18% |
| ACE_FS_operator | AVG | 5499.55 | 5272.72 | 4176.15 | 3877.98 |
| | SD | **12.66** | **18.38** | **20.04** | 13.64 |
| | BEST | 5474.01 | 5245.04 | 4131.80 | 3847.81 |
| | GAP | 1.24% | 20.14% | 2.03% | 1.30% |
| ACE_NLAS_operator | AVG | **5498.08** | 5276.13 | 4176.11 | 3885.27 |
| | SD | 20.45 | 19.19 | 27.09 | 13.30 |
| | BEST | 5459.43 | 5233.93 | 4133.47 | 3846.10 |
| | GAP | 0.97% | 19.89% | 2.07% | 1.25% |
| ACE_LAS_operator | AVG | 5558.41 | **5266.24** | 4202.15 | **3877.72** |
| | SD | 55.07 | 30.76 | 50.11 | **10.88** |
| | BEST | 5497.77 | 5214.28 | 4120.17 | 3855.23 |
| | GAP | 1.68% | 19.44% | 1.74% | 1.49% |
| Non-learning | AVG | 5683.60 | 5431.68 | 4238.98 | 3973.20 |
| | SD | 55.43 | 42.20 | 38.76 | 28.32 |
| | BEST | 5594.09 | 5347.23 | 4163.38 | 3905.85 |
| | GAP | 3.46% | 22.48% | 2.81% | 2.83% |

which is helpful for learning techniques to discover some knowledge or patterns.

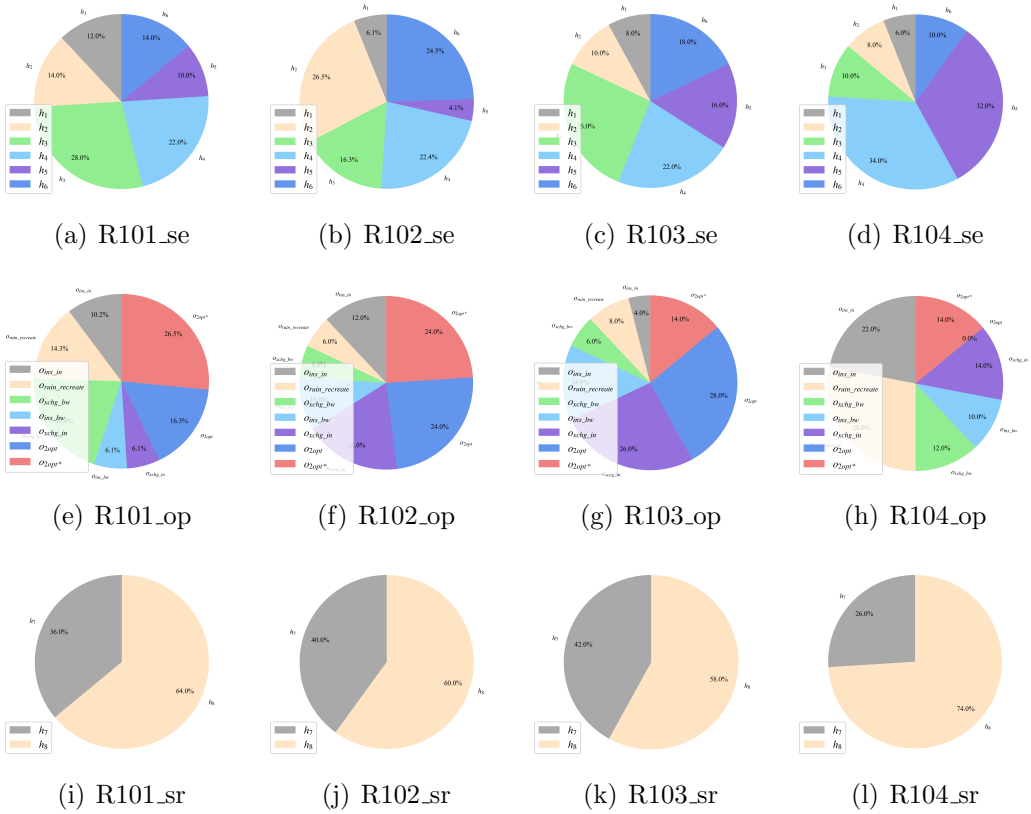
**Algorithmic Component Analysis of the Best Designed Search Algorithms:**

Taking R1 instances as examples, Figure 6.5 shows the most adapted algorithmic components of the best designed search algorithms learned by the ACE_LAS_both method on the Selection for Evolution heuristics (shown in Figure 6.5 (a)-(d)), Evolution Operators (shown in Figure 6.5 (e)-(h)) and Selection for Replacement heuristics (shown in Figure 6.5 (i)-(l)).

In the best designed search algorithms, all the Selection for Evolution heuristics are called during the optimisation process although the appearances of each heuristic are different. The same phenomenons can be observed in terms of Evolution Operators and Selection for Replacement heuristics. This indicates that using distinct algorithmic components (e.g., selection heuristics and evolution operators) can help to improve the performance of the search algorithms. Moreover, it should be noted that $h_8$ is identified as the most frequently called selection for replacement heuristic, which is consistent with the findings of manually designed algorithms in the literature.

Table 6.13: Generality of the trained policy R101 (20 runs)

| | | C101 | C201 | R105 | R201 | RC101 | RC201 |
|---|---|---|---|---|---|---|---|
| Best-known Solutions | | **10828.94** [144] | **3591.56** [144] | **15377.11** [144] | **5252.37** [148] | **15696.94** [152] | **5406.91** [147] |
| ACE_FS_both | AVG | **10828.94** | **3591.56** | 16367.11 | 5369.67 | 17451.46 | 5665.17 |
| | SD | **3.64E-12** | **4.44E-13** | 213.87 | 34.71 | 516.73 | 396.70 |
| | BEST | **10828.94** | **3591.56** | 15415.34 | 5295.14 | 16672.46 | 5524.74 |
| | GAP | **0** | **0** | 0.25% | 0.81% | 6.21% | 2.18% |
| ACE_NLAS_both | AVG | 10882.25 | 3593.05 | **16216.42** | 5361.80 | 17597.40 | 5551.16 |
| | SD | 226.77 | 6.37 | 385.46 | 30.95 | **301.94** | 30.14 |
| | BEST | **10828.94** | **3591.56** | **15383.19** | 5299.25 | 16687.34 | 5495.74 |
| | GAP | **0** | **0** | **0.04%** | 0.89% | 6.31% | 1.64% |
| ACE_LAS_both | AVG | **10828.94** | **3591.56** | 16252.23 | **5320.08** | **17031.94** | **5495.34** |
| | SD | 3.64E-12 | 4.55E-13 | 337.93 | **9.75** | 478.87 | **16.26** |
| | BEST | **10828.94** | **3591.56** | 15422.13 | **5288.70** | **16638.66** | **5450.61** |
| | GAP | **0** | **0** | 0.29% | **0.69%** | **6.0%** | **0.81%** |
| ACE_FS_op | AVG | **10828.94** | **3591.56** | 16354.52 | 5320.17 | 17243.77 | 5503.02 |
| | SD | **3.64E-12** | 4.55E-13 | **203.59** | 10.67 | 480.18 | 17.45 |
| | BEST | **10828.94** | **3591.56** | 15468.55 | 5301.74 | 16662.95 | 5465.95 |
| | GAP | **0** | **0** | 0.59% | 0.94% | 6.15% | 1.09% |
| ACE_NLAS_op | AVG | **10828.94** | **3591.56** | 16249.68 | 5323.86 | 17234.76 | 5511.80 |
| | SD | **3.64E-12** | 4.55E-13 | 350.09 | 19.90 | 492.75 | 30.22 |
| | BEST | **10828.94** | **3591.56** | 15393.51 | 5307.14 | 16653.41 | 5469.14 |
| | GAP | **0** | **0** | 0.11% | 1.04% | 6.09% | 1.15% |
| ACE_LAS_op | AVG | 11051.9 | **3591.56** | 16338.5 | 5383.6 | 17070.83 | 5564.74 |
| | SD | 441.29 | 4.55E-13 | 278.59 | 35.30 | 457.47 | 28.72 |
| | BEST | **10828.94** | **3591.56** | 15486.98 | 5332.277 | 16658.77 | 5506.94 |
| | GAP | **0** | **0** | 0.71% | 1.52% | 6.13% | 1.85% |



(a) R101_se (b) R102_se (c) R103_se (d) R104_se

(e) R101_op (f) R102_op (g) R103_op (h) R104_op

(i) R101_sr (j) R102_sr (k) R103_sr (l) R104_sr

**Figure 6.5:** *The most adapted algorithmic components of the best designed search algorithms obtained by ACE_LAS_both*

### Generality of the Devised Learning Models:

To investigate the generality of the policies trained by the ACE methods concerning different search spaces of algorithmic components for solving the same-type and different-type problem instances, the policies trained on instance R101 are employed to design search algorithms for solving different types of new problem instances.

Results in Table 6.13 demonstrate a good degree of generality of the reinforcement learning based models. The "GAP" is less than 3% on most selected instances apart from instance RC101. Note that ACE_LAS_both attained best "AVG", "BEST" and "GAP" on most instances, which is consistent with the experimental results regarding the effectiveness of the learning models in Section 6.3.2.

Table 6.14: Performance comparison during the testing process (learning on evolution operators), type-R

| | | R101 | R102 | R201 | R202 |
|---|---|---|---|---|---|
| Best-known Solutions | | **20645.79** [145] | **18486.12** [144] | **5252.37** [148] | **4191.7** [149] |
| DQN-GSF | AVG | 20981.19 | 19910.38 | 5378.77 | 5201.13 |
| | SD | 454.50 | 452.07 | 44.67 | 34.34 |
| | BEST | 20656.49 | 19495.43 | 5304.48 | 5159.01 |
| | GAP | 0.05% | 5.5% | 0.99% | 23.1% |
| PPO-GSF | AVG | 20665.46 | 18708.26 | 5382.98 | 5200.721 |
| | SD | 10.16 | 413.91 | 39.15 | **29.83** |
| | BEST | 20655.81 | 18493.03 | 5318.35 | 5144.32 |
| | GAP | 0.05% | 0.04% | 1.3% | 22.7% |
| ACE_FS_operator | AVG | 20664.24 | 18653.01 | 5318.49 | 5115.03 |
| | SD | 13.08 | 339.90 | **10.21** | 168.85 |
| | BEST | **20653.76** | 18498.31 | 5298.13 | 4361.74 |
| | GAP | **0.04%** | 0.07% | 0.87% | 4.06% |
| ACE_NLAS_operator | AVG | **20663.64** | **18512.82** | 5316.34 | **5065.46** |
| | SD | **9.26** | **12.75** | 10.57 | 258.14 |
| | BEST | 20653.99 | **18491.88** | 5298.88 | 4331.64 |
| | GAP | 0.04% | **0.03%** | 0.89% | 3.34% |
| ACE_LAS_operator | AVG | 20788.71 | 18522.37 | **5314.33** | 5113.26 |
| | SD | 288.5524 | 14.02 | 14.89 | 192.43 |
| | BEST | **20653.76** | 18504.07 | **5287.98** | **4265.06** |
| | GAP | **0.04%** | 0.097% | **0.68%** | **1.75%** |
| Non-learning | AVG | 20957.89 | 18769.97 | 5533.63 | 5384.61 |
| | SD | 30.37 | 215.69 | 24.47 | 49.37 |
| | BEST | 20901.02 | 18675.05 | 5481.47 | 5287.09 |
| | GAP | 1.24% | 1.02% | 4.36% | 26.13% |

Table 6.15: Performance comparison during the testing process (learning on evolution operators), type-RC

| | | RC101 | RC102 | RC201 | RC202 |
|---|---|---|---|---|---|
| Best-known Solutions | | **15696.94** [152] | **13554.75** [152] | **5406.91** [147] | **4367.09** [155] |
| DQN-GSF | AVG | 17586.44 | 16146.86 | 5661.65 | 5351.60 |
| | SD | 728.65 | 486.43 | 269.55 | 45.33 |
| | BEST | **16570.05** | 15526.71 | 5507.77 | 5293.64 |
| | GAP | **5.56%** | 14.55% | 1.87% | 21.2% |
| PPO-GSF | AVG | 17521.35 | 16005.04 | 5567.00 | 5359.51 |
| | SD | **410.16** | 525.97 | 78.01 | 57.68 |
| | BEST | 16679.80 | 15511.07 | 5467.35 | 5246.94 |
| | GAP | 6.3% | 14.4% | 1.12% | 20.1% |
| ACE_FS_operator | AVG | 17271.06 | 15480.25 | 5499.55 | 5272.72 |
| | SD | 478.25 | 205.12 | **12.66** | **18.38** |
| | BEST | 16681.82 | 14566.36 | 5474.01 | 5245.04 |
| | GAP | 6.27% | 7.46% | 1.24% | 20.14% |
| ACE_NLAS_operator | AVG | 17027.84 | 15479.54 | **5498.08** | 5276.13 |
| | SD | 462.77 | **201.48** | 20.45 | 19.19 |
| | BEST | 16674.02 | 14579.25 | **5459.43** | 5233.93 |
| | GAP | 6.22% | 7.56% | **0.97%** | 19.89% |
| ACE_LAS_operator | AVG | **16951.76** | 15310.34 | 5558.41 | **5266.24** |
| | SD | 417.82 | 396.81 | 55.07 | 30.76 |
| | BEST | 16667.91 | **14561.73** | 5497.77 | **5214.28** |
| | GAP | 6.19% | **7.43%** | 1.68% | **19.44%** |
| Non-learning | AVG | 17105.48 | **15194.24** | 5683.60 | 5431.68 |
| | SD | 478.03 | 481.82 | 55.43 | 42.20 |
| | BEST | 16745.33 | 14597.72 | 5594.09 | 5347.23 |
| | GAP | 6.68% | 7.69% | 3.46% | 22.48% |

### 6.3.3 Automated Composition of Evolution Operators

This section aims to investigate the effectiveness of different proposed reinforcement learning techniques: DQN-GSF and PPO-GSF from Chapter 5, and three ACE variants from Chapter 6. The investigation involves a direct comparison of the experimental results in the context of automated composition of evolution operators. Tables 6.14-6.15 present the performance comparison during the testing process of these five RL-based methods, while also including best-known solutions in the literature and random solutions (i.e. non-learning) without incorporating any learning techniques. Note that the training performance comparison is not included here, as the reward scheme differs between Chapter 5 and Chapter 6, rendering such comparisons meaningless.

Tables 6.14-6.15 indicate that ACE variants exhibit relatively better performance compared to DQN-GSF and PPO-GSF. This can be attributed to the fact that ACE variants place a stronger emphasis on exploration, whereas DQN-GSF and PPO-GSF primarily focus on exploiting the current policy, which may limit exploration in certain scenarios. Specifically, ACE variants encourage exploration by incorporating an entropy term in the reward scheme, i.e the policy objective. This helps the agent to explore a wider range of actions, ultimately facilitating the discovery of potentially better solutions.

## 6.4 Summary

In this chapter, we systematically investigate two key research issues in automated algorithm design with machine learning, namely the impact of individual algorithmic components and the synergy of multiple algorithmic components, within the unified general search framework. Extending the search space of algorithm design from individual components to multiple components results in a high-dimensional decision space of algorithm design. Therefore, an advanced reinforcement learning method with adapted maximum entropy mechanisms has been devised to address the automated algorithm design prob-

lem, with a continuous state space and a high-dimensional discrete action space.

The performance of the learning models, namely their effectiveness and generality, has been assessed on the capacitated vehicle routing problem with time windows. Results regarding the impact of individual components show that selection heuristics on the population have less impact on the performance of search algorithms, which supports human experience in designing search algorithms and findings reported in the literature. Learning on the synergy of multiple algorithmic components demonstrates that proper collaboration among selection heuristics and evolution operators can significantly improve the algorithm performance. The comparison experiments with the learning on evolution operators indicate that human design experience can help algorithm design to some extent, but machine learning techniques overtake human experience when dealing with a larger algorithm design space which human experts are not able to explore.

# Chapter 7

# Conclusions and Future Research

This thesis focuses on automated design of population-based algorithms within a novel general search framework for solving complex combinatorial optimisation problems efficiently. The vehicle routing problem is selected as a case study. This goal has been successfully achieved by investigating three key research issues within the proposed general search framework, i.e. automated design of evolution operators, of selection heuristics and of both together, with the support of reinforcement learning techniques. The effectiveness and generality of the machine learning based models are validated comprehensively across different benchmark instances of the capacitated vehicle routing problem with time windows.

## 7.1 Main Contributions

This section describes the main conclusions for this thesis drawn from the four major contribution chapters, i.e. Chapters 3 to 6.

### 7.1.1 Novel General Search Framework to Support Automated Algorithm Design

Metaheuristic algorithms have been investigated intensively to address highly complex combinatorial optimisation problems such as vehicle routing problems. However, most metaheuristic algorithms have been designed manually by researchers of different expertise without a consistent framework to support effective algorithm design. Therefore, they may produce good results for particular problem instances but perform poorly on other problem instances. Furthermore, knowledge learned is difficult to be transferred and reused, thus often discarded.

In Chapter 3, a novel general search framework was proposed to formulate different metaheuristic algorithms, including single-solution based and population-based algorithms. Generic algorithmic components such as selection heuristics and evolution operators were clearly defined within the proposed framework. This framework aims to serve as the basis of automated design of effective search algorithms for solving complex combinatorial optimisation problems with the support of machine learning. In addition, three key research issues within automated algorithm design, i.e automated composition of evolution operators, of selection heuristics and of both, were clearly defined.

### 7.1.2 Feature Identification for Automated Algorithm Design

Utilising machine learning techniques to assist effective algorithm design is still at a preliminary stage albeit some successful attempts. One of the key research issues is on how to identify key features to accurately describe the search space of algorithm design. Although various features have been identified in the literature, there is a lack of a systematic investigation analysing the identified features within a consistent and general framework.

In Chapter 4, two groups of key features, namely search-dependent and instance-

dependent features, were identified to capture useful information for assisting effective algorithm design. These identified features aim to serve as the basis of developing successful machine learning for automated design of general-purpose search algorithms. With these key features, a state-of-the-art reinforcement learning technique, namely proximal policy optimisation, was adapted to extract useful knowledge hidden in the data collected during the optimisation process. Search patterns of the best designed search algorithms, in particular the utilisation and transition of algorithmic components, were investigated to provide insights into reusing knowledge extracted in algorithm design using machine learning in solving new problem instances.

### 7.1.3 Automated Composition of Evolution Operators

Automated algorithm design, especially automated composition of evolution operators, has been investigated extensively in the literature. However, most existing work has been conducted within a template of a specific search algorithm such as a genetic algorithm, which limit the scope of algorithm design. Furthermore, existing studies pay more attention to the solution quality while neglecting other performance measures such as reusability and generality of the produced search algorithms.

In Chapter 5, with GSF and identified features, this thesis devised two reinforcement learning based methods, namely deep Q-network based and proximal policy optimisation based methods, to enable appropriate evolution operators to be intelligently selected and combined during different stages of the optimisation process. Experimental results validate the effectiveness and generality of the proposed methods. This provides promising evidence in learning reusable new knowledge in designing algorithms based on the basic algorithmic components within the unified general search framework.

### 7.1.4 Automated Composition of Selection Heuristics and Evolution Operators

Most existing studies on automated algorithm design have focused on the evolution operators, neglecting the automated composition of selection heuristics on the population, not to mention the automated composition of both decisions together.

Chapter 6 aimed to systematically investigate the automated design of search algorithms by exploring the impact of individual algorithmic components and the synergy among multiple algorithmic components. To tackle the high-dimensional search space of algorithms, an advanced deep reinforcement learning method with adapted maximum entropy mechanisms was devised. Comprehensive computational experiments were conducted on a range of benchmark instances to evaluate the effectiveness and generality of the proposed method. Regarding the impact of individual algorithmic components, selection heuristics have less impact than evolution operators on an algorithm's performance, which supports human experience in designing search algorithms and findings reported in the literature. Regarding the synergy among multiple algorithmic components, the results show that proper collaboration among selection heuristics and evolution operators can significantly improve the algorithm performance. The experiments comparing the individual impact and the synergy of algorithmic components indicated that human design experience can help algorithm design to some extent, but machine learning techniques can be more helpful when dealing with a larger algorithm design space.

## 7.2 Limitations and Future Works

The previous section provides a summary of the key finding in this thesis. Based on the summarised contributions, we have identified several limitations related to the research works conducted in this thesis. This section will high-

light a few of these limitations and then suggest some future research directions.

- In the first contribution, a general search framework has been built to formulate different metaheuristics. This thesis focuses specifically on the automated design of single-objective population-based algorithms. However, many combinatorial optimisation problems involve multiple objectives that needed to be addressed. For instance, the vehicle routing problems considered in this thesis include multiple objectives such as minimising the number of vehicles and the total distance travelled, but this thesis handles the multiple objectives by transforming them into a single objective and then applies single-objective approaches.

  For further work, it would be interesting to treat the vehicle routing problem as a multi-objective problem and then automatically design effective multi-objective evolutionary algorithms to address them. This would require further extension of the proposed general search framework to include additional algorithmic components related to handling multiple objectives, such as the performance metric, diversity maintenance and archive management. Another potential future direction could be adopt and adjust the basic algorithmic components within the Evolution module of the general search framework to other combinatorial optimisation problems.

  Besides, it would be interesting to further extend the current framework to enable the combination of local search algorithms and population-based algorithms, i.e., a high-level algorithmic template that take advantage of both types of algorithms. This kind of combination has been proved to be an effective strategy. One example is the memetic algorithm [170] which has demonstrated effectiveness in handling complex optimisation problems such as travelling salesman problems [171] [172], vehicle routing problems [173] [174], and job shop scheduling problems [175] [176] . The advantage of this kind of combination lies in its abil-

ity to combine global and local search mechanisms, allowing for efficient exploration and exploitation of the search space, leading to improved optimisation performance.

- In the second contribution, we identified two groups of features, i.e. search-dependent and instance-dependent features, to assist automated design of population-based algorithms. We also accessed the effectiveness of different types of feature sets. However, one limitation of this thesis is the lack of systematic feature selection or reduction among these identified features using approaches such as recursive elimination and principal component analysis. It would be valuable to conduct a systematic study to evaluate the impact of each specific feature, and then identify and select a subset of the most informative and discriminative features. By reducing the dimensionality of the feature space, the performance of the proposed automated algorithm design approaches can be enhanced. Another potential future direction could be identifying suitable features that accurately describe the search space of multi-objective algorithms to support the new learning task.

- In the third and fourth contributions, we developed different reinforcement learning approaches to address different research issues within the context of automated algorithm design. Both the effectiveness and generality of the proposed methods have been comprehensively validated on a challenging benchmark problem, namely the vehicle routing problem with time windows. One limitation of this thesis is that it has chosen CVRPTW as a case study for comprehensive research. It would be interesting to challenge the proposed automated design approaches by applying them to more complex VRP variants and real-world problems, such as the multi-depot vehicle routing problem [177], the vehicle routing problem with pickup and delivery [178], or the vehicle routing problem with time windows and split deliveries [179]. Further studies may also

investigate how to transfer the reusable knowledge in designing search algorithms for small-scale vehicle routing problems to large-scale vehicle routing problems, or even to other complex combinatorial optimisation problems such as permutation flow shop scheduling problem [180] and personnel scheduling problem [181].

# Bibliography


[1] S. P. Gayialis, G. D. Konstantakopoulos, and I. P. Tatsiopoulos, "Vehicle routing problem for urban freight transportation: A review of the recent literature," in *Operational Research in the Digital Era–ICT Challenges: 6th International Symposium and 28th National Conference on Operational Research, Thessaloniki, Greece, June 2017*. Springer, 2019, pp. 89–104.

[2] Y.-P. Wang, "Adaptive ant colony algorithm for the vrp solution of logistics distribution," *Research Journal of Applied Sciences, Engineering and Technology*, vol. 6, no. 5, pp. 807–811, 2013.

[3] G. D. Konstantakopoulos, S. P. Gayialis, and E. P. Kechagias, "Vehicle routing problem and related algorithms for logistics distribution: a literature review and classification," *Operational research*, pp. 1–30, 2020.

[4] M. Zirour, "Vehicle routing problem: models and solutions," *Journal of Quality Measurement and Analysis JQMA*, vol. 4, no. 1, pp. 205–218, 2008.

[5] J. K. Lenstra and A. R. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981.

[6] M. Gendreau and C. D. Tarantilis, *Solving large-scale vehicle routing problems with time windows: The state-of-the-art*. Cirrelt Montreal, 2010.

[7] M. Steinbrunn, G. Moerkotte, and A. Kemper, "Heuristic and randomized optimization for the join ordering problem," *The VLDB Journal*, vol. 6, no. 3, pp. 191–208, 1997.

[8] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & operations research*, vol. 13, no. 5, pp. 533–549, 1986.

[9] P. Hansen and N. Mladenović, "An introduction to variable neighborhood search," in *Meta-heuristics*. Springer, 1999, pp. 433–458.

[10] H. Jh, "Adaptation in natural and artificial systems," *Ann Arbor*, 1975.

[11] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*. Ieee, 1995, pp. 39–43.

[12] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[13] R. Qu, G. Kendall, and N. Pillay, "The general combinatorial optimization problem: Towards automated algorithm design," *IEEE Computational Intelligence Magazine*, vol. 15, no. 2, pp. 14–23, 2020.

[14] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *International conference on the practice and theory of automated timetabling*. Springer, 2000, pp. 176–190.

[15] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, B. McCollum, G. Ochoa, A. J. Parkes, and S. Petrovic, "The cross-domain heuristic search challenge–an international research competition," in *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 631–634.

[16] N. Pillay and D. Beckedahl, "EvoHyp-A Java toolkit for evolutionary algorithm hyper-heuristics," in *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2017, pp. 2706–2713.

[17] S. L. Tilahun and M. A. Tawhid, "Swarm hyperheuristic framework," *Journal of Heuristics*, vol. 25, no. 4, pp. 809–836, 2019.

[18] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín, and Y. Shi, "Hyper-heuristics to customise metaheuristics for continuous optimisation," *Swarm and Evolutionary Computation*, vol. 66, p. 100935, 2021.

[19] S. Van Rijn, C. Doerr, and T. Bäck, "Towards an adaptive cma-es configurator," in *Parallel Problem Solving from Nature–PPSN XV: 15th International Conference, Coimbra, Portugal, September 8–12, 2018, Proceedings, Part I*. Springer, 2018, pp. 54–65.

[20] R. Boks, H. Wang, and T. Bäck, "A modular hybridization of particle swarm optimization and differential evolution," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 1418–1425.

[21] A. Eiben, M. Horvath, W. Kowalczyk, and M. C. Schut, "Reinforcement learning for online control of evolutionary algorithms," in *International Workshop on Engineering Self-Organising Applications*. Springer, 2006, pp. 151–160.

[22] A. E. Gutierrez-Rodríguez, S. E. Conant-Pablos, J. C. Ortiz-Bayliss, and H. Terashima-Marín, "Selecting meta-heuristics for solving vehicle routing problems with time windows via meta-learning," *Expert Systems with Applications*, vol. 118, pp. 470–481, 2019.

[23] H. Jiang, Y. Wang, Y. Tian, X. Zhang, and J. Xiao, "Feature construction for meta-heuristic algorithm recommendation of capacitated vehicle

routing problems," *ACM Transactions on Evolutionary Learning and Optimization*, vol. 1, no. 1, pp. 1–28, 2021.

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[25] S. S. Choong, L.-P. Wong, and C. P. Lim, "Automatic design of hyper-heuristic based on reinforcement learning," *Information Sciences*, vol. 436, pp. 89–107, 2018.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[29] J. E. Pettinger and R. M. Everson, "Controlling genetic algorithms with reinforcement learning," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, 2002, pp. 692–692.

[30] F. Chen, Y. Gao, Z.-q. Chen, and S.-f. Chen, "SCGA: Controlling genetic algorithms with Sarsa (0)," in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, vol. 1. IEEE, 2005, pp. 1177–1183.

[31] W. Zhang, H. Li, W. Yang, G. Zhang, and M. Gen, "Hybrid multiobjective evolutionary algorithm considering combination timing for multitype vehicle routing problem with time windows," *Computers & Industrial Engineering*, vol. 171, p. 108435, 2022.

[32] A. Narayanan, P. Misra, A. Ojha, V. Bandhu, S. Ghosh, and A. Vasan, "A reinforcement learning approach for electric vehicle routing problem with vehicle-to-grid supply," *arXiv preprint arXiv:2204.05545*, 2022.

[33] Y. Wang, L. Ran, X. Guan, J. Fan, Y. Sun, and H. Wang, "Collaborative multicenter vehicle routing problem with time windows and mixed deliveries and pickups," *Expert Systems with Applications*, vol. 197, p. 116690, 2022.

[34] P. Saksuriya and C. Likasiri, "Hybrid heuristic for vehicle routing problem with time windows and compatibility constraints in home healthcare system," *Applied Sciences*, vol. 12, no. 13, p. 6486, 2022.

[35] E. Díaz de León-Hicks, S. E. Conant-Pablos, J. C. Ortiz-Bayliss, and H. Terashima-Marín, "Addressing the algorithm selection problem through an attention-based meta-learner approach," *Applied Sciences*, vol. 13, no. 7, p. 4601, 2023.

[36] D. Wu, J. Li, J. Cui, and D. Hu, "Research on the time-dependent vehicle routing problem for fresh agricultural products based on customer value," *Agriculture*, vol. 13, no. 3, p. 681, 2023.

[37] B. Chen, R. Qu, R. Bai, and H. Ishibuchi, "An investigation on compound neighborhoods for VRPTW," in *International Conference on Operations Research and Enterprise Systems*. Springer, 2016, pp. 3–19.

[38] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *Journal of statistical physics*, vol. 34, no. 5, pp. 975–986, 1984.

[39] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.

[40] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM computing surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.

[41] R. Tavakkoli-Moghaddam, N. Safaei, and Y. Gholipour, "A hybrid simulated annealing for capacitated vehicle routing problems with the independent route length," *Applied Mathematics and Computation*, vol. 176, no. 2, pp. 445–454, 2006.

[42] F. Y. Vincent, A. P. Redi, Y. A. Hidayat, and O. J. Wibowo, "A simulated annealing heuristic for the hybrid vehicle routing problem," *Applied Soft Computing*, vol. 53, pp. 119–132, 2017.

[43] L. Wei, Z. Zhang, D. Zhang, and S. C. Leung, "A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints," *European journal of operational research*, vol. 265, no. 3, pp. 843–859, 2018.

[44] N. M. E. Normasari, V. F. Yu, C. Bachtiyar *et al.*, "A simulated annealing heuristic for the capacitated green vehicle routing problem," *Mathematical Problems in Engineering*, vol. 2019, 2019.

[45] F. Y. Vincent, H. Susanto, P. Jodiawan, T.-W. Ho, S.-W. Lin, and Y.-T. Huang, "A simulated annealing algorithm for the vehicle routing problem with parcel lockers," *IEEE Access*, vol. 10, pp. 20 764–20 782, 2022.

[46] A. S. Alfa, S. S. Heragu, and M. Chen, "A 3-opt based simulated annealing algorithm for vehicle routing problems," *Computers & Industrial Engineering*, vol. 21, no. 1-4, pp. 635–639, 1991.

[47] İ. İlhan, "An improved simulated annealing algorithm with crossover operator for capacitated vehicle routing problem," *Swarm and Evolutionary Computation*, vol. 64, p. 100911, 2021.

[48] Y. Wang, Q. Wu, and F. Glover, "Effective metaheuristic algorithms for the minimum differential dispersion problem," *European Journal of Operational Research*, vol. 258, no. 3, pp. 829–843, 2017.

[49] L. Berbotto, S. García, and F. J. Nogales, "A randomized granular tabu search heuristic for the split delivery vehicle routing problem," *Annals of Operations Research*, vol. 222, no. 1, pp. 153–173, 2014.

[50] O. Bräysy, "A reactive variable neighborhood search for the vehicle-routing problem with time windows," *INFORMS Journal on Computing*, vol. 15, no. 4, pp. 347–368, 2003.

[51] İ. Küçükoğlu and N. Öztürk, "An advanced hybrid meta-heuristic algorithm for the vehicle routing problem with backhauls and time windows," *Computers & Industrial Engineering*, vol. 86, pp. 60–68, 2015.

[52] B. Chen, R. Qu, R. Bai, and H. Ishibuchi, "A variable neighbourhood search algorithm with compound neighbourhoods for VRPTW," 2016.

[53] J. Kytöjoki, T. Nuortio, O. Bräysy, and M. Gendreau, "An efficient variable neighborhood search heuristic for very large scale vehicle routing problems," *Computers & operations research*, vol. 34, no. 9, pp. 2743–2757, 2007.

[54] M. J. Geiger and W. Wenger, "On the interactive resolution of multi-objective vehicle routing problems," in *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2007, pp. 687–699.

[55] K. Fleszar, I. H. Osman, and K. S. Hindi, "A variable neighbourhood search algorithm for the open vehicle routing problem," *European Journal of Operational Research*, vol. 195, no. 3, pp. 803–809, 2009.

[56] I. Rechenberg, "Evolution strategy: Nature's way of optimization," in *Optimization: Methods and Applications, Possibilities and Limitations: Proceedings of an International Seminar Organized by Deutsche Forschungsanstalt für Luft-und Raumfahrt (DLR), Bonn, June 1989*. Springer, 1989, pp. 106–126.

[57] V. W. Porto, T. Back, D. Fogel, and T. Michalewicz, "Evolutionary programming," *Handbook of evolutionary computation*, pp. 89–102, 2000.

[58] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, pp. 87–112, 1994.

[59] H. Awad, R. Elshaer, A. AbdElmo'ez, and G. Nawara, "An effective genetic algorithm for capacitated vehicle routing problem," in *Proceedings of the International Conference on Industrial Engineering and Operations Management*, 2018, pp. 374–384.

[60] N. Lin, Y. Shi, T. Zhang, and X. Wang, "An effective order-aware hybrid genetic algorithm for capacitated vehicle routing problems in internet of things," *IEEE Access*, vol. 7, pp. 86 102–86 114, 2019.

[61] T. Azad and M. A. A. Hasin, "Capacitated vehicle routing problem using genetic algorithm: a case of cement distribution," *International Journal of Logistics Systems and Management*, vol. 32, no. 1, pp. 132–146, 2019.

[62] J. Zhu, "Solving capacitated vehicle routing problem by an improved genetic algorithm with fuzzy c-means clustering," *Scientific Programming*, vol. 2022, 2022.

[63] H. Nazif and L. S. Lee, "Optimized crossover genetic algorithm for vehicle routing problem with time windows," *American journal of applied sciences*, vol. 7, no. 1, p. 95, 2010.

[64] S. N. Kumar, R. Panneerselvam *et al.*, "A time-dependent vehicle routing problem with time windows for e-commerce supplier site pickups using genetic algorithm," *Intelligent Information Management*, vol. 7, no. 04, p. 181, 2015.

[65] H. Abidi, K. Hassine, and F. Mguis, "Genetic algorithm for solving a dynamic vehicle routing problem with time windows," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2018, pp. 782–788.

[66] Q. Liu, P. Xu, Y. Wu, and T. Shen, "A hybrid genetic algorithm for the electric vehicle routing problem with time windows," *Control Theory and Technology*, vol. 20, no. 2, pp. 279–286, 2022.

[67] O. S. Olaniyi, A. K. James, A. A. Ibrahim, and A. F. Makanjuola, "On the application of a modified genetic algorithm for solving vehicle routing problems with time windows and split delivery," *IAENG International Journal of Applied Mathematics*, vol. 52, no. 1, pp. 1–9, 2022.

[68] C. Prins, "A simple and effective evolutionary algorithm for the vehicle routing problem," *Computers & operations research*, vol. 31, no. 12, pp. 1985–2002, 2004.

[69] T. S. Khoo and B. B. Mohammad, "The parallelization of a two-phase distributed hybrid ruin-and-recreate genetic algorithm for solving multi-objective vehicle routing problem with time windows," *Expert Systems with Applications*, vol. 168, p. 114408, 2021.

[70] A. K. Ariyani, W. F. Mahmudy, and Y. P. Anggodo, "Hybrid genetic algorithms and simulated annealing for multi-trip vehicle routing problem with time windows." *International Journal of Electrical & Computer Engineering (2088-8708)*, vol. 8, no. 6, 2018.

[71] M. Hemici, D. Zouache, B. Boualem, and K. Hemici, "An external archive guided nsga-ii algorithm for multi-depot green vehicle routing problem," in *Artificial Intelligence and Its Applications: Proceeding of the 2nd International Conference on Artificial Intelligence and Its Applications (2021)*. Springer, 2022, pp. 504–513.

[72] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[73] E. Bonabeau, M. Dorigo, G. Theraulaz, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. Oxford university press, 1999, no. 1.

[74] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied mathematics and computation*, vol. 214, no. 1, pp. 108–132, 2009.

[75] X.-l. Li, "An optimizing method based on autonomous animats: fish-swarm algorithm," *Systems Engineering-Theory & Practice*, vol. 22, no. 11, pp. 32–38, 2002.

[76] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: an overview," *Soft computing*, vol. 22, no. 2, pp. 387–408, 2018.

[77] V. Kachitvichyanukul *et al.*, "A particle swarm optimization for the capacitated vehicle routing problem," *International journal of logistics and SCM systems*, vol. 2, no. 1, pp. 50–55, 2007.

[78] T. J. Ai and V. Kachitvichyanukul, "A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery," *Computers & Operations Research*, vol. 36, no. 5, pp. 1693–1702, 2009.

[79] Y. Marinakis, M. Marinaki, and G. Dounias, "A hybrid particle swarm optimization algorithm for the vehicle routing problem," *Engineering Applications of Artificial Intelligence*, vol. 23, no. 4, pp. 463–472, 2010.

[80] H. Y. Zeng, "Improved particle swarm optimization based on Tabu search for VRP," *Journal of Applied Science and Engineering Innovation*, vol. 6, no. 2, pp. 99–103, 2019.

[81] Y. W. Zhao, B. Wu, W. Wang, Y. L. Ma, W. Wang, and H. Sun, "Particle swarm optimization for vehicle routing problem with time windows," in *Materials Science Forum*, vol. 471. Trans Tech Publ, 2004, pp. 801–805.

[82] B. Yu, Z.-Z. Yang, and B. Yao, "An improved ant colony optimization for vehicle routing problem," *European journal of operational research*, vol. 196, no. 1, pp. 171–176, 2009.

[83] Q. Ding, X. Hu, L. Sun, and Y. Wang, "An improved ant colony optimization and its application to vehicle routing problem with time windows," *Neurocomputing*, vol. 98, pp. 101–107, 2012.

[84] B. Yu and Z. Z. Yang, "An ant colony optimization model: The period vehicle routing problem with time windows," *Transportation Research Part E: Logistics and Transportation Review*, vol. 47, no. 2, pp. 166–181, 2011.

[85] L. M. Gambardella, É. Taillard, and G. Agazzi, "Macs-vrptw: A multiple colony system for vehicle routing problems with time windows," in *New ideas in optimization*. Citeseer, 1999.

[86] B. Barán and M. Schaerer, "A multiobjective ant colony system for vehicle routing problem with time windows." in *Applied informatics*, 2003, pp. 97–102.

[87] N. Pillay and R. Qu, *Automated Design of Machine Learning and Search Algorithms*. Springer, 2021.

[88] Q. Zhao, Q. Duan, B. Yan, S. Cheng, and Y. Shi, "A survey on automated design of metaheuristic algorithms," *arXiv preprint arXiv:2303.06532*, 2023.

[89] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Automated configuration of mixed integer programming solvers," in *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer, 2010, pp. 186–202.

[90] F. Hutter, H. H. Hoos, and T. Stützle, "Automatic algorithm configuration based on local search," in *Aaai*, vol. 7, 2007, pp. 1152–1157.

[91] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-Race and iterated F-Race: An overview," *Experimental methods for the analysis of optimization algorithms*, pp. 311–336, 2010.

[92] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[93] T. Bartz-Beielstein, C. W. Lasarczyk, and M. Preuß, "Sequential parameter optimization," in *2005 IEEE congress on evolutionary computation*, vol. 1. IEEE, 2005, pp. 773–780.

[94] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*. Springer, 2011, pp. 507–523.

[95] B. van Stein, H. Wang, and T. Bäck, "Automatic configuration of deep neural networks with EGO," *arXiv preprint arXiv:1810.05526*, 2018.

[96] J. Bergstra, D. Yamins, D. D. Cox *et al.*, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in science conference*, vol. 13. Citeseer, 2013, p. 20.

[97] M. López-Ibánez and T. Stutzle, "The automatic design of multiobjective ant colony optimization algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 6, pp. 861–875, 2012.

[98] I. Araya and M.-C. Riff, "A filtering method for algorithm configuration based on consistency techniques," *Knowledge-Based Systems*, vol. 60, pp. 73–81, 2014.

[99] J. R. Rice, "The algorithm selection problem," in *Advances in computers*. Elsevier, 1976, vol. 15, pp. 65–118.

[100] M. Lindauer, "Algorithm selection, scheduling and configuration of boolean constraint solvers," Ph.D. dissertation, Universität Potsdam, Institut für Informatik, 2015.

[101] B. A. Huberman, R. M. Lukose, and T. Hogg, "An economics approach to hard computational problems," *Science*, vol. 275, no. 5296, pp. 51–54, 1997.

[102] K. Tang, F. Peng, G. Chen, and X. Yao, "Population-based algorithm portfolios with automated constituent algorithms selection," *Information Sciences*, vol. 279, pp. 94–104, 2014.

[103] L. Xu, H. Hoos, and K. Leyton-Brown, "Hydra: Automatically configuring algorithms for portfolio-based selection," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[104] J. C. Ortiz-Bayliss, I. Amaya, J. M. Cruz-Duarte, A. E. Gutierrez-Rodriguez, S. E. Conant-Pablos, and H. Terashima-Marín, "A general framework based on machine learning for algorithm selection in constraint satisfaction problems," *Applied Sciences*, vol. 11, no. 6, p. 2749, 2021.

[105] K. Sörensen, "Metaheuristics—the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.

[106] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern search technology," in *Handbook of metaheuristics*. Springer, 2003, pp. 457–474.

[107] E. K. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," *Journal of heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

[108] E. Özcan, M. Misir, G. Ochoa, and E. K. Burke, "A reinforcement learning: great-deluge hyper-heuristic for examination timetabling," in *Modeling, analysis, and applications in metaheuristic computing: advancements and trends*. IGI Global, 2012, pp. 34–55.

[109] A. Elhag and E. Özcan, "A grouping hyper-heuristic framework: Application on graph colouring," *Expert Systems with Applications*, vol. 42, no. 13, pp. 5491–5507, 2015.

[110] J. D. Walker, G. Ochoa, M. Gendreau, and E. K. Burke, "Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework," in *International conference on learning and intelligent optimization*. Springer, 2012, pp. 265–276.

[111] G. Koulinas, L. Kotsikas, and K. Anagnostopoulos, "A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem," *Information Sciences*, vol. 277, pp. 680–693, 2014.

[112] R. Aron and I. Chana, "QoS based resource provisioning and scheduling in grids," *The Journal of Supercomputing*, vol. 66, no. 1, pp. 262–283, 2013.

[113] I. Chana *et al.*, "Bacterial foraging based hyper-heuristic for resource scheduling in grid computing," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 751–762, 2013.

[114] B. Duhart, F. Camarena, J. C. Ortiz-Bayliss, I. Amaya, and H. Terashima-Marín, "An experimental study on ant colony optimization hyper-heuristics for solving the knapsack problem," in *Mexican Conference on Pattern Recognition*. Springer, 2018, pp. 62–71.

[115] C. B. Pop, V. R. Chifu, N. Dragoi, I. Salomie, and E. S. Chifu, "Recommending healthy personalized daily menus—a cuckoo search-based hyper-heuristic approach," in *Applied Nature-Inspired Computing: Algorithms and Case Studies*. Springer, 2020, pp. 41–70.

[116] M. A. Ahandani, M. T. V. Baghmisheh, M. A. B. Zadeh, and S. Ghaemi, "Hybrid particle swarm optimization transplanted into a hyper-heuristic structure for solving examination timetabling problem," *Swarm and Evolutionary Computation*, vol. 7, pp. 21–34, 2012.

[117] V. Pandiri and A. Singh, "A hyper-heuristic based artificial bee colony algorithm for k-interconnected multi-depot multi-traveling salesman problem," *Information Sciences*, vol. 463, pp. 261–281, 2018.

[118] S. N. Chaurasia and J. H. Kim, "An artificial bee colony based hyper-heuristic for the single machine order acceptance and scheduling problem," in *Decision science in action*. Springer, 2019, pp. 51–63.

[119] T. Wauters, K. Verbeeck, P. D. Causmaecker, and G. V. Berghe, "Boosting metaheuristic search using reinforcement learning," in *Hybrid metaheuristics*. Springer, 2013, pp. 433–452.

[120] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[121] I. C. Ramos, M. C. Goldbarg, E. G. Goldbarg, and A. D. D. Neto, "Logistic regression for parameter tuning on an evolutionary algorithm," in *2005 IEEE congress on evolutionary computation*, vol. 2. IEEE, 2005, pp. 1061–1068.

[122] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil, "Using experimental design to find effective parameter settings for heuristics," *Journal of Heuristics*, vol. 7, pp. 77–97, 2001.

[123] J. Pihera and N. Musliu, "Application of machine learning to algorithm selection for tsp," in *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*. IEEE, 2014, pp. 47–54.

[124] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla-07: The design and analysis of an algorithm portfolio for sat," in *Principles and Practice of Constraint Programming–CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings 13*. Springer, 2007, pp. 712–727.

[125] ——, "Satzilla: portfolio-based algorithm selection for sat," *Journal of artificial intelligence research*, vol. 32, pp. 565–606, 2008.

[126] J. Li, E. K. Burke, and R. Qu, "Integrating neural networks and logistic regression to underpin hyper-heuristic search," *Knowledge-Based Systems*, vol. 24, no. 2, pp. 322–330, 2011.

[127] R. Tyasnurita, E. Özcan, and R. John, "Learning heuristic selection using a time delay neural network for open vehicle routing," in *2017 IEEE Congress on Evolutionary Computation (CEC)*. Ieee, 2017, pp. 1474–1481.

[128] F. Thabtah and P. Cowling, "Mining the data from a hyperheuristic approach using associative classification," *Expert systems with applications*, vol. 34, no. 2, pp. 1093–1101, 2008.

[129] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, "Isac–instance-specific algorithm configuration," in *ECAI 2010*. IOS Press, 2010, pp. 751–756.

[130] G. Ghiani, G. Laporte, and E. Manni, "Model-based automatic neighborhood design by unsupervised learning," *Computers & Operations Research*, vol. 54, pp. 108–116, 2015.

[131] J. G. C. Costa, Y. Mei, and M. Zhang, "Cluster-based hyper-heuristic for large-scale vehicle routing problem," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–8.

[132] E. Lara-Cárdenas, A. Silva-Gálvez, J. C. Ortiz-Bayliss, I. Amaya, J. M. Cruz-Duarte, and H. Terashima-Marín, "Exploring reward-based hyper-heuristics for the job-shop scheduling problem," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 3133–3140.

[133] Y. Sakurai, K. Takada, T. Kawabe, and S. Tsuruta, "A method to control parameters of evolutionary algorithms by using reinforcement learning," in *2010 Sixth International Conference on Signal-Image Technology and Internet Based Systems*. IEEE, 2010, pp. 74–79.

[134] A. Buzdalova, V. Kononov, and M. Buzdalov, "Selecting evolutionary operators using reinforcement learning: Initial explorations," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014, pp. 1033–1036.

[135] J. Schuchardt, V. Golkov, and D. Cremers, "Learning to evolve," *arXiv preprint arXiv:1905.03389*, 2019.

[136] G. Duflo, G. Danoy, E.-G. Talbi, and P. Bouvry, "Automated design of efficient swarming behaviours: a Q-learning hyper-heuristic approach," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 227–228.

[137] S. Wagner and M. Affenzeller, "Heuristiclab: A generic and extensible optimization environment," in *Adaptive and Natural Computing Algorithms: Proceedings of the International Conference in Coimbra, Portugal, 2005*. Springer, 2005, pp. 538–541.

[138] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4j: a modular framework for meta-heuristic optimization," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pp. 1723–1730.

[139] J. J. Durillo and A. J. Nebro, "jmetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, 2011.

[140] J. J. Merelo and J.-M. García-Valdez, "Mapping evolutionary algorithms to a reactive, stateless architecture: using a modern concurrent language," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2018, pp. 1870–1877.

[141] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "Pisa—a platform and programming language independent interface for search algorithms," in *Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003, Faro, Portugal, April 8–11, 2003. Proceedings 2*. Springer, 2003, pp. 494–508.

[142] W. Yi, R. Qu, L. Jiao, and B. Niu, "Automated design of metaheuristics using reinforcement learning within a novel general search framework," *IEEE Transactions on Evolutionary Computation*, 2022, doi: 10.1109/TEVC.2022.3197298.

[143] SINTEF, "VRPTW benchmark problems, on the sintef transport optimization portal," *https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/100-customers/*, 2008.

[144] Y. Rochat and É. D. Taillard, "Probabilistic diversification and intensification in local search for vehicle routing," *Journal of heuristics*, vol. 1, no. 1, pp. 147–167, 1995.

[145] J. Homberger, "Eine verteilt-parallele metaheuristik," in *Verteilt-parallele Metaheuristiken zur Tourenplanung*. Springer, 2000, pp. 139–165.

[146] H. Li and A. Lim, "Local search with annealing-like restarts to solve the VRPTW," *European journal of operational research*, vol. 150, no. 1, pp. 115–127, 2003.

[147] D. Mester, O. Bräysy, and W. Dullaert, "A multi-parametric evolution strategies algorithm for vehicle routing problems," *Expert Systems with Applications*, vol. 32, no. 2, pp. 508–517, 2007.

[148] J. Homberger and H. Gehring, "Two evolutionary metaheuristics for the vehicle routing problem with time windows," *INFOR: Information Systems and Operational Research*, vol. 37, no. 3, pp. 297–318, 1999.

[149] L.-M. Rousseau, M. Gendreau, and G. Pesant, "Using constraint-based operators to solve the vehicle routing problem with time windows," *Journal of heuristics*, vol. 8, no. 1, pp. 43–58, 2002.

[150] M. Woch and P. Łebkowski, "Sequential simulated annealing for the vehicle routing problem with time windows," *Decision Making in Manufacturing and Services*, vol. 3, pp. 87–100, 2009.

[151] R. Bent and P. Van Hentenryck, "A two-stage hybrid local search for the vehicle routing problem with time windows," *Transportation Science*, vol. 38, no. 4, pp. 515–530, 2004.

[152] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin, "A tabu search heuristic for the vehicle routing problem with soft time windows," *Transportation science*, vol. 31, no. 2, pp. 170–186, 1997.

[153] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *International conference on principles and practice of constraint programming*. Springer, 1998, pp. 417–431.

[154] J.-F. Cordeau, G. Laporte, and A. Mercier, "A unified tabu search heuristic for vehicle routing problems with time windows," *Journal of the Operational research society*, vol. 52, no. 8, pp. 928–936, 2001.

[155] Z. J. Czech and P. Czarnas, "Parallel simulated annealing for the vehicle routing problem with time windows," in *Proceedings 10th Euromicro workshop on parallel, distributed and network-based processing*. IEEE, 2002, pp. 376–383.

[156] J. Berger and M. Barkaoui, "A parallel hybrid genetic algorithm for the vehicle routing problem with time windows," *Computers & operations research*, vol. 31, no. 12, pp. 2037–2053, 2004.

[157] J. Baras and V. Borkar, "A learning algorithm for markov decision processes with adaptive state aggregation," in *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*, vol. 4. IEEE, 2000, pp. 3351–3356.

[158] T. Mori and S. Ishii, "Incremental state aggregation for value function estimation in reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 41, no. 5, pp. 1407–1416, 2011.

[159] S. P. Singh, T. Jaakkola, and M. I. Jordan, "Reinforcement learning with soft state aggregation," *Advances in neural information processing systems*, pp. 361–368, 1995.

[160] W. Meng and R. Qu, "Automated design of search algorithms: Learning on algorithmic components," *Expert Systems with Applications*, vol. 185, p. 115493, 2021.

[161] B. Crawford, R. Soto, J. Lemus-Romani, M. Becerra-Rozas, J. M. Lanza-Gutiérrez, N. Caballé, M. Castillo, D. Tapia, F. Cisternas-Caneo, J. García *et al.*, "Q-learnheuristics: towards data-driven balanced metaheuristics," *Mathematics*, vol. 9, no. 16, p. 1839, 2021.

[162] S. N. Chaurasia and J. H. Kim, "An evolutionary algorithm based hyperheuristic framework for the set packing problem," *Information Sciences*, vol. 505, pp. 1–31, 2019.

[163] S. N. Richter and D. R. Tauritz, "The automated design of probabilistic selection methods for evolutionary algorithms," in *Proceedings of the genetic and evolutionary computation conference companion*, 2018, pp. 1545–1552.

[164] N. Lourenço, F. Pereira, and E. Costa, "Learning selection strategies for evolutionary algorithms," in *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer, 2013, pp. 197–208.

[165] C. Huang, Y. Li, and X. Yao, "A survey of automatic parameter tuning methods for metaheuristics," *IEEE transactions on evolutionary computation*, vol. 24, no. 2, pp. 201–216, 2019.

[166] E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, and K. Tierney, "A survey of methods for automated algorithm configuration," *Journal of Artificial Intelligence Research*, vol. 75, pp. 425–487, 2022.

[167] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary computation*, vol. 27, no. 1, pp. 3–45, 2019.

[168] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[169] W. Yi, R. Qu, and L. Jiao, "Automated algorithm design using proximal policy optimisation with identified features," *Expert Systems with Applications*, vol. 216, p. 119461, 2023.

[170] F. Neri, C. Cotta, and P. Moscato, *Handbook of memetic algorithms*. Springer, 2011, vol. 379.

[171] S.-J. Jian and S.-Y. Hsieh, "A niching regression adaptive memetic algorithm for multimodal optimization of the euclidean traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, 2022.

[172] M. Germanos, D. Aza, A. B. Karami, and J. Possik, "A distributed memetic algorithm with a semi-greedy operator for the traveling salesman problem," in *2022 IEEE/ACM 26th International Symposium on*

*Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 2022, pp. 143–150.

[173] A. Berahhou, Y. Benadada, and K. Bouanane, "Memetic algorithm for the dynamic vehicle routing problem with simultaneous delivery and pickup," *International Journal of Industrial Engineering Computations*, vol. 13, no. 4, pp. 587–600, 2022.

[174] J. Xiao, J. Du, Z. Cao, X. Zhang, and Y. Niu, "A diversity-enhanced memetic algorithm for solving electric vehicle routing problems with time windows and mixed backhauls," *Applied Soft Computing*, p. 110025, 2023.

[175] S. Afsar, J. J. Palacios, J. Puente, C. R. Vela, and I. González-Rodríguez, "Multi-objective enhanced memetic algorithm for green job shop scheduling with uncertain times," *Swarm and Evolutionary Computation*, vol. 68, p. 101016, 2022.

[176] R. Li, W. Gong, C. Lu, and L. Wang, "A learning-based memetic algorithm for energy-efficient flexible job shop scheduling with type-2 fuzzy processing time," *IEEE Transactions on Evolutionary Computation*, 2022.

[177] B. Crevier, J.-F. Cordeau, and G. Laporte, "The multi-depot vehicle routing problem with inter-depot routes," *European journal of operational research*, vol. 176, no. 2, pp. 756–773, 2007.

[178] C. Lin, "A vehicle routing problem with pickup and delivery time windows, and coordination of transportable resources," *Computers & Operations Research*, vol. 38, no. 11, pp. 1596–1609, 2011.

[179] M. Gendreau, P. Dejax, D. Feillet, and C. Gueguen, "Vehicle routing problem with time windows and split deliveries," Technical report, 2006-851, Laboratoire d'Informatique d'Avignon, Tech. Rep., 2006.

[180] J. M. Framinan, J. N. Gupta, and R. Leisten, "A review and classification of heuristics for permutation flow-shop scheduling with makespan objective," *Journal of the Operational Research Society*, vol. 55, no. 12, pp. 1243–1255, 2004.

[181] J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, and L. De Boeck, "Personnel scheduling: A literature review," *European journal of operational research*, vol. 226, no. 3, pp. 367–385, 2013.

# Appendix A

# The Neural Networks in DQN-GSF and PPO-GSF

The neural networks in DQN-GSF and PPO-GSF are shown in Figure A.1 and Figure A.2, respectively.

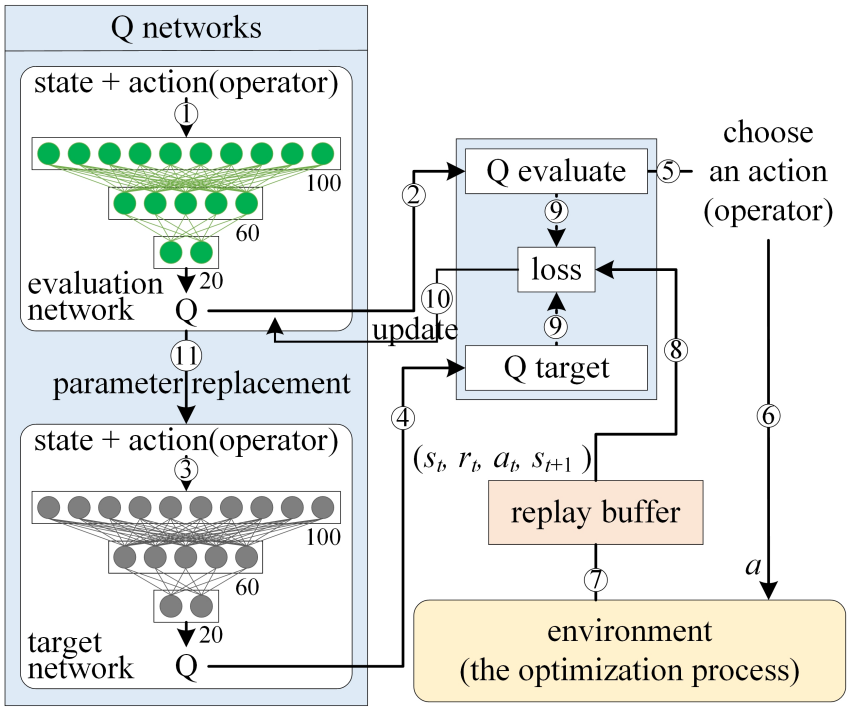



**Figure A.1:** *Details of the neural network of DQN-GSF*

In Figure A.1, the state and action are taken as the input (①, ③) for the Q networks, evaluation network and target network, respectively. The parameters of the target network are replaced (⑪) by the evaluation network every $n$ episodes. The output of the Q networks is a set of Q values of all actions (②, ④). The action is decided by the maximal Q values (⑤). The selected action $a$ is executed (⑥) in the environment, called one step. The $(s_t, r_t, a_t, s_{t+1})$ generated at each step is stored in the replay buffer (⑦). The loss of the parameters of the evaluation network is calculated(⑧). After the update of the Q networks (⑩, ⑪), the data flows back to ①.
In Figure A.2, the state is taken as the input (①) of the actor neural network, the output of which is a probability distribution of all actions (②). The action

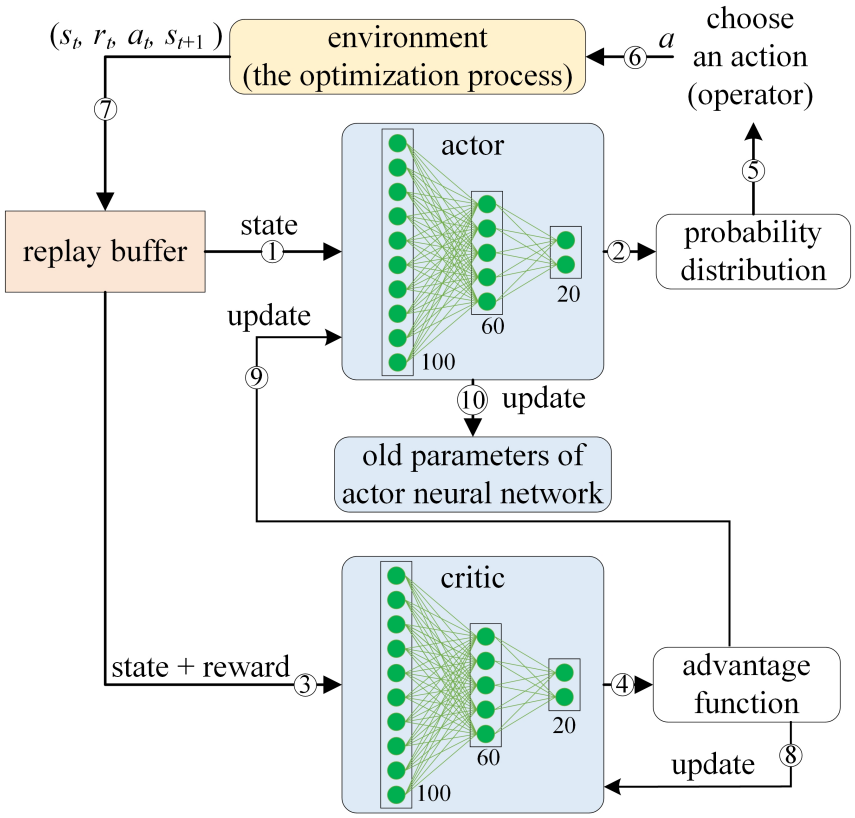



**Figure A.2:** *Details of the neural network of PPO-GSF*

is decided by the obtained probability distribution (⑤). The selected action $a$ is executed (⑥) in the environment, called one step. The $(s_t, r_t, a_t, s_{t+1})$ generated at each step is stored in the replay buffer (⑦). The parameters of the critic neural network are updated (⑧) by minimising the advantage function value. On the other hand, the state and reward are taken as the input (③) of the critic neural network, and the output, advantage function value of current state (④), guides the update direction of the actor neural network (⑨). Then, the data flows back to ①.

In reinforcement learning, the learning rate, discount rate, and size of the neural network are the key hyper parameters in the algorithms. Specifically, the learning rate is adjusted adaptively, set as 0.002 at the beginning and halved when the output is stable. The discount rate is set to 0.99 thus the learned policy focuses more on sequential decisions. The topology of the network is set based on the complexity of the problem, and the number of layers and neurons has been shown in Figure A.1 and Figure A.2.