Automated Design of Metaheuristics Using Reinforcement Learning within a Novel General Search Framework

Wenjie Yi, Rong Qu, *Senior Member, IEEE*, Licheng Jiao, *Fellow, IEEE*, Ben Niu, *Member, IEEE*

***Abstract*—Metaheuristic algorithms have been investigated intensively to address highly complex combinatorial optimisation problems. However, most metaheuristic algorithms have been designed manually by researchers of different expertise without a consistent framework to support effective algorithm design. This paper proposes a general search framework to formulate in a unified way a range of different metaheuristics. This framework defines generic algorithmic components, including selection heuristics and evolution operators. The unified general search framework aims to serve as the basis of analysing algorithmic components for automated algorithm design. With the established new general search framework, two reinforcement learning based methods, deep Q-network based and proximal policy optimisation based methods, have been developed to automatically design a new general population-based algorithm. The proposed reinforcement learning based methods are able to intelligently select and combine appropriate algorithmic components during different stages of the optimisation process. The effectiveness and generalization of the proposed reinforcement learning based methods are validated comprehensively across different benchmark instances of the capacitated vehicle routing problem with time windows. This study contributes to making a key step towards automated algorithm design with a general framework supporting fundamental analysis by effective machine learning.**

***Index Terms*—Automated algorithm design, general search framework, metaheuristic, reinforcement learning, vehicle routing problem.**

I. Introduction

ADDRESSING highly complex Combinatorial Optimisation Problems (COPs) with various real-world constraints has proven to be one of the current research challenges in evolutionary computation. Current state-of-the-art include metaheuristic algorithms, which are successful in finding good-quality solutions within a reasonable computational time. However, most metaheuristic algorithms proposed in the literature only work for particular problem instances or at particular stages of problem-solving, and rely heavily on the experience of human experts. In addressing this issue, automated algorithm design has attracted considerable attention recently from the research community [1], [2].

W. Yi and R. Qu are with the Computational Optimisation and Learning (COL) Lab, School of Computer Science, University of Nottingham, United Kingdom (e-mail:wenjie.yi@nottingham.ac.uk; rong.qu@nottingham.ac.uk).

L. Jiao is with the Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education, School of Artificial Intelligence, Xidian University, China (e-mail:lchjiao@mail.xidian.edu.cn).

B. Niu is with the College of Management, Shenzhen University, China (e-mail:drniuben@gmail.com).

Corresponding authors: R. Qu, B. Niu.

Towards automated algorithm design, the problem of designing metaheuristics itself is defined as a combinatorial optimisation problem in [3], 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 categorized into automated algorithm configuration, algorithm selection, and algorithm composition, based on the different types of decision variables considered in the search space of algorithms [3]. 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 algorithms to solve multiple COPs, i.e. the algorithms generated do not belong to any specific search algorithms, for example genetic algorithms 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 problem. 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 study falls into the third category to investigate the elementary and basic components to automatically design search algorithms within a unified framework.

In the literature, Reinforcement Learning (RL) [4] has been used to automatically design algorithms through modelling 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 interaction 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, it learns to intelligently select the action in the current state by forming the state-action pairs through trial and error [5].

Some researchers have used the simplest tabular RL techniques, such as SARSA [4] and Q-learning (QL) [6] for evolutionary algorithm design. One research issue in applying tabular RL is concerned with the discretization of the continuous state space, leading to unreliable results [7], [8]. In

this research presented in this paper, neural network function approximation is adapted to handle continuous state to address the above issue. Besides, there are less studies on RL techniques to support effective design of evolutionary algorithms to solve constrained COPs such as the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW).

To support automatic design of effective metaheuristic algorithms for COPs, a general search framework is firstly established, within which learning techniques can be applied to the design space of algorithms and thus support automated algorithm design. At this stage of research, instead of studying all the algorithmic components, we only focus on investigating the key issue of automatic composition of key evolutionary operators which have the biggest impact on algorithm performance. RL is used in automated algorithm composition to reward or penalize combinations of key evolutionary operators based on their performance. The research work aims to make the following contributions:

- A new general search framework (GSF) is established to formulate different single-solution based and population-based algorithms. The unified GSF serves as the basis to analyze algorithmic components, generating effective search algorithms for CVRPTW automatically.
- The automated algorithm composition process is formulated as a MDP. Two RL methods, Deep Q-Network (DQN) [9] and Proximal Policy Optimisation (PPO)[10] have been investigated within the proposed GSF to address the key issue of automatic selection and combination of the most efficient evolutionary operators during different stages of the evolution. Results on CVRPTW demonstrate the effectiveness of the trained policy compared to a search procedure without learning.
- The generalization 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.

The remainder of this paper is structured as follows. Section II presents related work on existing automated algorithm design frameworks and reinforcement learning techniques within these frameworks. Section III describes the proposed general search framework and learning techniques. In Section IV, the optimisation model of vehicle routing problem is described, and experimental results are analysed on a benchmark dataset, whereas Section V presents the conclusions and discusses future research.

II. Related Work

Most evolutionary algorithms and metaheuristics in the existing literature have been designed manually by researchers of different expertise, many with ad hoc chosen algorithms for the specific problems in hand. There is relatively less work on building general search frameworks to support effective algorithm design.

A. Existing Frameworks for Automated Algorithm Design

The algorithm design problem has been formally modelled as a COP, namely the General Combinatorial Optimisation Problem (GCOP) model in [3]. Based on the fundamental difference in the decision space, automated algorithm design can be divided into three categories: algorithm configuration, algorithm selection, and algorithm composition. A number of frameworks have been developed in the literature to support the task of automated algorithm design within these different categories.

Automated algorithm configuration aims to find a well-performing parameter setting of a target algorithm across a given set of problem instances. Frameworks built to support this task include ParamILS [11], which utilizes iterated local search, F-race [12] and irace [13], both using a racing mechanism, and the surrogate-based methods such as SPOT [14], SMAC [15], MIP-EGO [16] and Hyperopt [17].

In automated algorithm selection, a specific algorithm or a portfolio of algorithms is automatically chosen on an instance-by-instance basis. Frameworks developed include PAP [18], which integrates different evolutionary algorithms to solve numerical optimisation problems, and Hydra [19], with a configuration technique for portfolio-based algorithm selection, and machine learning based algorithm selectors [20].

In automated algorithm composition, a set of heuristics is automatically combined to generate new algorithms to solve instances across different problem domains. The most investigated technique is hyper-heuristics [21], which is broadly concerned with intelligently selecting or generating appropriate heuristics in a given situation. Frameworks developed include HyFlex [22], EvoHyp [23] and SHH [24], etc. HyFlex explores a decision space of low-level heuristics or heuristic operators (e.g., taking search operators from ten well-known techniques as building blocks [25]) while EvoHyp adapts evolutionary algorithms as high-level strategies. SHH is specifically built for automatically combining different components of swarm intelligence algorithms [24]. In addition, some composition frameworks have been built within a template of specific metaheuristics, such as CMA-ES [26] and PSO-DE [27].

The recent fast growth of automated algorithm composition is due to its greater potential to generate more general search algorithms to solve complex COPs. It is not restricted within a template of existing specific search algorithms. This study focuses on automated algorithm composition problem, drawing in advanced reinforcement learning for effective algorithm design.

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.

With the new standard in algorithm design, namely GCOP established in [3], this research systematically investigates learning techniques within the unified GSF to underpin automated algorithm design.

B. Reinforcement Learning within Automated Algorithm Composition

In recent literature on automated algorithm composition, some RLs, such as SARSA [4], QL [6] and DQN [9], have been used to support the intelligent selection of the most appropriate heuristic operators. They utilise feedback information on the performance of 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 composition defines the 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 [7]. Results on the Travelling Salesman Problem and the 0-1 Knapsack Problem have demonstrated the superiority of this automated method [7], [28], [29], [30].

However, due to the complexity of RL techniques, most studies in this field [7], [28], [29] 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 [30]. 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 [31] and different COPs within the HyFlex software framework [5] demonstrated the effectiveness of these methods.

In these studies, several search-dependent features, i.e. the observations of the search process itself, have been used to represent the state. The number of features identified, however, is limited and insufficient for learning. 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 area of research, as it is often challenging to reimplement the exact same method and subsequently replicate the experiments.

In this study, we apply two RL techniques with a neural network function approximator in the learning of automated algorithm composition. The state space with sufficient features for effective learning is carefully defined. The action space is defined as the basic algorithmic components 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. It should be noted that this study adopts an offline RL framework, in which the policy is trained offline but used in an online fashion. This is different from most of RL-based automated algorithm composition methods in the literature.

III. Learning within the General Search Framework

A. 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 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 Fig.1. The framework is composed of five modules as shown in Table I for updating the individuals and four archives as shown in Table III for storing the individuals. For each of these components, different settings, heuristics or parameters can be chosen, as shown in Tables V-VII, to automatically compose and design different general search algorithms within the GSF. Algorithms represented by the combination of heuristics and operators are set as the output.

With respect to Initialization, 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 Fig.1, Selection for Evolution, Evolution and Selection for Replacement contribute more to the search performance. Therefore, they are discussed in detail in the following section.

TABLE I
MODULES WITHIN GSF

Module	Different heuristics, operators or parameters
Initialization	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 II, e.g. different population size, the four archives and heuristic sets in the Selection for Evolution module. This paper focuses on reinforcement learning on automated design of population-based search algorithms.

Table III shows the archives defined within GSF. In the Selection for Evolution module, some individuals within the

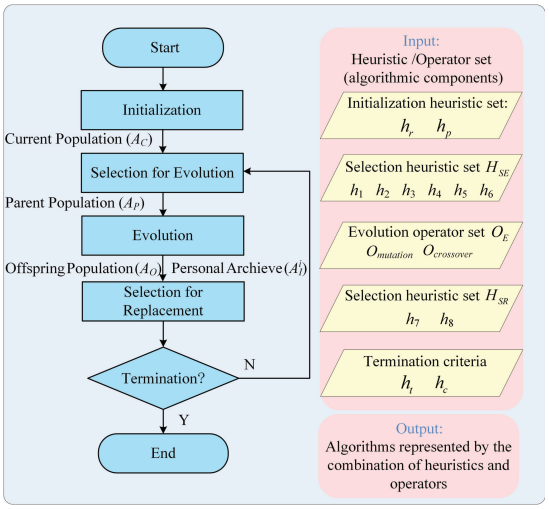

Fig. 1. General search framework

TABLE II
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	$A_C = A_P = A_O = A_I^i$	$A_C \neq A_P \neq A_O \neq A_I^i$
Initialization	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

TABLE III
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.

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.

B. Basic GSF Modules

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.

TABLE IV
THE HEURISTIC/OPERATOR SET FOR THE MODULES WITHIN GSF IN FIG.1

Heuristic/operator set	Description
H_{SE} for module Selection for Evolution	Various heuristics as defined in Table V 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 VI to update the current population A_C based on A_P.
O_E for module Evolution	Various operators as defined in Table VII to generate the offspring population A_O based on A_P selected by H_{SE}.

Without loss of generality, all selection heuristics are set for solving optimisation problems where the aim is to minimize the objective value.

1) Selection for Evolution: There are two types of heuristics in the Selection for Evolution module. As Table V 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 V
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. For each individual i, the probability to be selected 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.

2) Selection for Replacement: 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 VI.

TABLE VI
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) Evolution Operators: 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 study, we focus on investigating various mutation operators, which are defined in TableVII, for solving CVRPTW. Note that these general basic operators (exchange, insert and remove, etc.) can be adapted accordingly to automatically design algorithms for different COPs.

TABLE VII
O_E: EVOLUTION OPERATORS FOR CVRPTW

Operator	Description
o_{chg_in}	Exchange m and n nodes from 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 other positions of 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

C. Reinforcement Learning for Automated Algorithm Composition in GSF

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 a MDP, which is composed of a set of possible states and a set of selectable actions. Each state-action pair is given a total reward value (Q-value).

With the established GSF, RL is used for automated algorithm composition as shown in Fig.2. The actions are the selectable combinations of algorithmic components (i.e. evolution operators). The states are defined by different features of the search process, the solution and the instance, as shown in Table IX. The automated algorithm composition 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 [4] and QL [6], have been used to select heuristic operators in the literature. However, a Q-table cannot handle continuous state space, leading to unreliable results. To address this issue, in this study, 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-based methods based on their policy update mechanism [4]. 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 research.

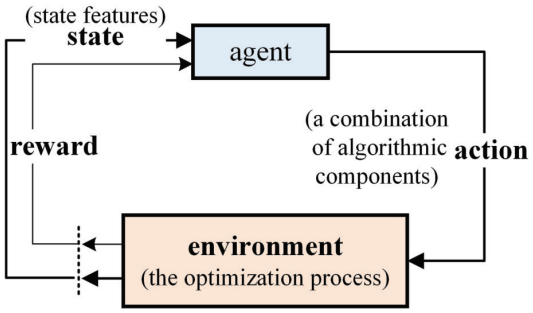

Fig. 2. Reinforcement learning in the context of automated algorithm composition in GSF

In value-based RL, DQN [9], 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 [10], which outperforms other policy gradient approaches, is selected, named PPO-GSF in this study.

Table VIII shows the notations used in this study. 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 evolutionary operators which have the most impact on evolutionary algorithms. With the newly established GSF, at this stage of research, 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 sub-modules, we can focus on examining the results only due to different settings in the Evolution module. From the preliminary experimental analysis, compared with the Evolution module, it is observed that the choice of components in the Selection for Evolution and Replacement modules has a smaller impact on the algorithm performance. Therefore, 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 VIII
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

In DQN-GSF and PPO-GSF, the two RL techniques, DQN and PPO, are firstly applied in multiple episodes to train the

Algorithm 1 Pseudocode of DQN-GSF

1: Initialize memory buffer D
2: Initialize evaluation action-value function Q network and target action-value function $\hat{Q}$ network
3: Generate initial population, record the initial state s_0
4: **for** episode k = 1 to NoE **do**
5: initialize the state s_0
6: **for** timestep t = 1 to NoT **do**
7: observe the current state s_t by calculating values of different state features in Table IX
8: with probability ε select a random action a_t, with probability $1-\varepsilon$ select an action that has a maximum Q-value: $a_t = arg\max Q\left(s_t, a_t\right)$
9: select parents using a selection heuristic $h_i\left(i=1,2,...,6\right)$ from H_{SE} (fixed as h_1 in this study)
10: generate offspring population by performing the selected action a_t to state s_t
11: update the population using a selection heuristic $h_i\left(i=7,8\right)$ from H_{SR} (fixed as h_8 in this study)
12: observe reward r_t based on Equation (3) and Equation (4), and next state s_{t+1}, store experience $\left(s_t, a_t, r_t, s_{t+1}\right)$ in D
13: sample random minibatch of experiences $\left[s_j, a_j, r_j, s_{j+1}\right]_J$ (J denotes the size of the sampled minibatch) from memory buffer D and calculate the loss: $\left[r_j + \gamma \underset{a_{j+1}}{\max} \hat{Q}\left(s_{j+1}, a_{j+1}\right) - Q\left(s_j, a_j\right)\right]^2$, γ denotes the discount factor.
14: perform gradient descent with respect to Q network in order to minimize the loss
15: every N timesteps reset $\hat{Q} = Q$
16: **end for**
17: **end for**

Algorithm 2 Pseudocode of PPO-GSF

1: Initialize memory buffer D
2: Initialize policy parameters θ_0, value function parameters Φ_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 IX
7: select parents using a selection heuristic $h_i\left(i=1,2,...,6\right)$ 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 $h_i\left(i=7,8\right)$ from H_{SR} (fixed as h_8 in this study)
10: observe reward r_t based on Equation (3) and Equation (4)
11: collect experience $\left(s_t, a_t, r_t\right)$ and save it in D
12: **end for**
13: update the policy by maximizing the PPO objective θ_{k+1} based on Equation (1)
14: fit value function Φ_{k+1} based on Equation (2)
15: empty memory buffer D
16: **end for**

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 $\left(s_t, a_t, r_t, s_{t+1}\right)$ is stored in the memory buffer (Line 12, Algorithm 1). After that, a minibatch 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 synchronized 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 maximizing the PPO objective based on Equation (1) (Line 13, Algorithm 2) and the value function is fitted by time differential error based on Equation (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).

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

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

$r_t\left(\theta\right)$ 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. ϵ 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 $\tau : \left[\left(s_1, a_1, r_1\right), \left(s_2, a_2, r_2\right), \cdots, \left(s_t, a_t, r_t\right)\right]$. Please refer to [10] for more detail about these two equations.

1) State Representation: Different state features, including search-dependent, solution-dependent and instance-dependent features, are first distinguished in this section.

Search-dependent features observe the search process, such as the total improvement over the initial solution. 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. 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 many cases, the solution-dependent features cannot be used to develop a general methodology since they are problem-specific. Therefore, in this study, as shown in Table IX, four search-dependent features (f_1-f_4) and four instance-dependent features (f_5-f_8) are used to define the state space.

TABLE IX
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 window, i.e. the percentage of time-constrained customers
f_8	The tightness of the time window, i.e. the width of the time windows

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) in Table VII. Once an action is selected, it is applied to the whole population.

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, as shown in Equation (3) and Equation (4). When population fitness is optimised above a certain threshold, a larger reward is given for the same fitness improvement.

$$f_1 = \frac{f_{current}}{f_{initial}} \tag{3}$$

$$reward = \begin{cases} -f_1, & if\ f_1 > C \\ -f_1 - \log_{10}(f_1), & if\ f_1 \leqslant C \end{cases} \tag{4}$$

Two methods are used in setting the reward: normalize f_1 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 maximize the total fitness improvement itself, which is what really needs to be optimised. Compared to the simple positive/negative reward 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) Episode Setting: An episode is defined as the whole optimisation process. Since the time-based stopping criteria is used in this study, 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 DQN-GSF and PPO-GSF are executed for NoE episodes. For testing purposes, the designed DQN-GSF and PPO-GSF are executed for one episode.

IV. EXPERIMENTS AND DISCUSSION

The proposed RL-GSF methods within the novel GSF are investigated and evaluated on one of the mostly studied COPs, CVRPTW, in this research. 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 RL-GSF methods are implemented in Java environment with IntelliJ IDEA 2020.3.3 as the development tool.

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 generalization 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 in section IV-B1. To analyse the influence of the policy update mechanism on learning models, DQN-GSF and PPO-GSF, are assessed in section IV-B2. The generalization 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 in section IV-C1 and section IV-C2, respectively.

A. Problem Definition and Dataset

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. CVRPTW has been intensively tested as a benchmark problem in evaluating the performance of evolutionary and metaheuristic algorithms [32]. This paper will investigate the CVRPTW to gain a better understanding on the proposed reinforcement learning based automated algorithm design methodologies.