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].

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 Traveling 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 problemspecific 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>
<thead>
<tr>
<th>Module</th>
<th>Different heuristics, operators or parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>random (h_r), problem-specific (h_p)</td>
</tr>
<tr>
<td>Selection for Evolution</td>
<td>probability-based operators (h_1, h_2, h_3), deterministic operators (h_4, h_5, h_6)</td>
</tr>
<tr>
<td>Evolution</td>
<td>mutation (O_mutation), crossover (O_crossover)</td>
</tr>
<tr>
<td>Selection for Replacement</td>
<td>comma-selection (h_7), plus-selection (h_8)</td>
</tr>
<tr>
<td>Termination</td>
<td>computation time (h_t), convergence (h_c)</td>
</tr>
</tbody>
</table>

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
Based on the fitness of individuals in the population archives.

**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.

Current Population archive ($A_C$) is 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 ($A^i_P$) to reserve individual trajectories.

**TABLE II**

<table>
<thead>
<tr>
<th>Parameters/components</th>
<th>Single-solution based algorithm</th>
<th>Population-based algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>$A_C = A_P =</td>
<td>A_C</td>
</tr>
<tr>
<td>Archive</td>
<td>$A_O = A^i_P$</td>
<td>$A_O \neq A^i_P$</td>
</tr>
<tr>
<td>Initialization</td>
<td>$h_i, h_p$</td>
<td>$h_i, h_p$</td>
</tr>
<tr>
<td>Selection for Evolution</td>
<td>$O_{mutation}$</td>
<td>$O_{mutation}, O_{crossover}$</td>
</tr>
<tr>
<td>Selection for Replacement</td>
<td>$b_1, b_5, h_1, h_5, h_6$</td>
<td>$b_7, h_7, b_7, h_7$</td>
</tr>
<tr>
<td>Termination</td>
<td>$h_1, h_6$</td>
<td>$h_1, h_6$</td>
</tr>
</tbody>
</table>

**TABLE III**

<table>
<thead>
<tr>
<th>Archive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_C$: Current population, $nPop =</td>
<td>A_C</td>
</tr>
<tr>
<td>$A_P$: Parent population, $\mu =</td>
<td>A_P</td>
</tr>
<tr>
<td>$A_O$: Offspring population, $\lambda =</td>
<td>A_O</td>
</tr>
<tr>
<td>$A^i_P$: Personal archive, $i \in {1, 2, \ldots, nPop}$</td>
<td>Personal archive $A^i_P$ for the individual $i$th to reserve individual trajectories.</td>
</tr>
</tbody>
</table>

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 IV**

<table>
<thead>
<tr>
<th>Heuristic/operator set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_{SE}$ for module Selection for Evolution</td>
<td>Various heuristics as defined in Table V to select the parent population $A_P$ from the current population $A_C$.</td>
</tr>
<tr>
<td>$H_{SR}$ for module Selection for Replacement</td>
<td>Various heuristics as defined in Table VI to update the current population $A_C$ based on $A_P$.</td>
</tr>
<tr>
<td>$O_E$ for module Evolution</td>
<td>Various operators as defined in Table VII to generate the offspring population $A_C$ based on $A_P$ selected by $H_{SE}$.</td>
</tr>
</tbody>
</table>

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 V**

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>$h_{1,1,1} = nPop$</td>
</tr>
<tr>
<td>$h_2$</td>
<td>Proportionate roulette wheel selection of an individual $i$ as parent from $A_P$ with a probability proportional to its fitness.</td>
</tr>
<tr>
<td>$h_3$</td>
<td>Ranking selection of an individual $i$ as parent according to the probability proportional to the ranking based on the fitness function.</td>
</tr>
<tr>
<td>$h_4$</td>
<td>Select the current individual itself as parent.</td>
</tr>
<tr>
<td>$h_5$</td>
<td>Rank selection of the best previous position as parent based on individual’s personal archive $A^i_P$.</td>
</tr>
<tr>
<td>$h_6$</td>
<td>Selection of the individual(s) with a lower fitness than the current individual as parent(s) from $A_P$.</td>
</tr>
</tbody>
</table>

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 Table VII, 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.

### 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.

Table VII shows the notations used in this study. The pseudocodes of DQN-GSF and PPO-GSF are shown in Algorithm 1 and Algorithm 2, respectively.

![Fig. 2. Reinforcement learning in the context of automated algorithm composition in GSF](image)

**TABLE VII**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(o_{chq-in})</td>
<td>Exchange (m) and (n) nodes from the same route in a solution</td>
</tr>
<tr>
<td>(o_{chq-lw})</td>
<td>Exchange (m) and (n) nodes from different routes in a solution</td>
</tr>
<tr>
<td>(o_{ins-in})</td>
<td>Insert (m) nodes to other positions of the same route in a solution</td>
</tr>
<tr>
<td>(o_{ins-lw})</td>
<td>Insert (m) nodes to other positions of different routes in a solution</td>
</tr>
<tr>
<td>(o_{recom})</td>
<td>The (n) 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.</td>
</tr>
<tr>
<td>(o_{2o-opt})</td>
<td>Exchange two nodes in the same route in a solution</td>
</tr>
<tr>
<td>(o_{2o-opt_s})</td>
<td>Take the end sections of two routes in a solution and swap them to create two new routes</td>
</tr>
</tbody>
</table>

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

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**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_0)</td>
<td>The initial state</td>
</tr>
<tr>
<td>(s_t)</td>
<td>The state at timestep (t)</td>
</tr>
<tr>
<td>(a_t)</td>
<td>The selected action at timestep (t)</td>
</tr>
<tr>
<td>(r_t)</td>
<td>The reward value at timestep (t)</td>
</tr>
<tr>
<td>(NoE)</td>
<td>The number of episodes</td>
</tr>
<tr>
<td>(NoT)</td>
<td>The number of timesteps in one episode</td>
</tr>
</tbody>
</table>

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 $\varepsilon$ select a random action $a_t$, with probability $1-\varepsilon$ select an action that has a maximum Q-value: $a_t = \arg \max \limits_{a} Q(s_t, a_t)$
9: select parents using a selection heuristic $h_i$ ($i=1,2,\ldots,6$) 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 $h_i$ ($i=7,8$) 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 $(s_t, a_t, r_t, s_{t+1})$ in $D$
13: sample random minibatch of experiences $[s_j, a_j, r_j, s_{j+1}]_j$ ($J$ denotes the size of the sampled minibatch) from memory buffer $D$ and calculate the loss: $\left[r_j + \gamma \max \limits_{a_j} \hat{Q}(s_{j+1}, a_j) - Q(s_j, a_j)\right]^2$, $\gamma$ 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 $\theta_0$, 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 IX
7: select parents using a selection heuristic $h_i$ ($i=1,2,\ldots,6$) 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 (3) and Equation (4)
11: collect experience $\langle s_t, a_t, r_t \rangle$ and save it in $D$
12: end for
13: update the policy by maximizing the PPO objective $\theta_{k+1}$ based on Equation (1)
14: fit value function $\phi_{k+1}$ based on Equation (2)
15: empty memory buffer $D$
16: end for

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).

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

$$\phi_{k+1} = \arg \min \limits_{\phi} \frac{1}{|D_c| \cdot NoT} \sum \limits_{\tau \in D_c} \sum \limits_{t=0}^{NoT} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$ \tag{2}$$

$r_t(\theta)$ denotes the probability ratio $r_t(\theta) = \frac{\pi_{\theta_k}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$ is an estimator of the advantage function at timestep $t$. $\epsilon$ is a hyperparameter. $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$ denotes the modified surrogate objective by clipping the probability ratio. The rewards-to-go $\hat{R}_t$ is calculated according to trajectory $\tau = \langle (s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_t, a_t, r_t) \rangle$. 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 \((f_1, f_2)\) and four instance-dependent features \((f_3, f_4, f_5, f_6, f_7, f_8)\) are used to define the state space.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_1)</td>
<td>The total fitness improvement over the initial population fitness</td>
</tr>
<tr>
<td>(f_2)</td>
<td>The diversity of the population, measured by the standard deviation of the population fitness</td>
</tr>
<tr>
<td>(f_3)</td>
<td>The current algorithmic stage, calculated as (i \times NoT) where (i) is the index of the current timestep</td>
</tr>
<tr>
<td>(f_4)</td>
<td>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</td>
</tr>
<tr>
<td>(f_5)</td>
<td>The number of vehicles</td>
</tr>
<tr>
<td>(f_6)</td>
<td>The capacity of the vehicle</td>
</tr>
<tr>
<td>(f_7)</td>
<td>The density of the time window, i.e. the percentage of time-constrained customers</td>
</tr>
<tr>
<td>(f_8)</td>
<td>The tightness of the time window, i.e. the width of the time windows</td>
</tr>
</tbody>
</table>

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.

\[
f_1 = \frac{f_{\text{current}}}{f_{\text{initial}}} \quad (3)
\]

\[
\text{reward} = \begin{cases} 
-f_1, & \text{if } f_1 > C \\
-f_1 - \log_{10}(f_1), & \text{if } f_1 \leq C 
\end{cases} \quad (4)
\]

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 optimized. 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 \(t_{\text{max}}\). An episode is divided into \(NoT\) timesteps, so the period of each timestep equals to \(t_{\text{max}}/NoT\).

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, CVRP, 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 CVRP 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 CVRP, a fleet of vehicles are routed to serve the customers with the minimal distance, satisfying capacity and time windows constraints. CVRP has been intensively tested as a benchmark problem in evaluating the performance of evolutionary and metaheuristic algorithms [32]. This paper will investigate the CVRP to gain a better understanding on the proposed reinforcement learning based automated algorithm design methodologies.
The CVRPTW can be mathematically formulated as follows [33]:

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 function:**

\[
\begin{align*}
\text{Minimize} & \quad K \\
\text{Minimize} & \quad \sum_{k \in K} \sum_{v_i \in V} \sum_{v_j \in V} X_{ij}^k d_{ij} \\
\end{align*}
\]

**Constraint:**

\[
\begin{align*}
\sum_{k \in K} \sum_{v_i \in V} X_{ij}^k = 1, \forall v_i \in V \setminus \{ v_o \} \\
\sum_{k \in K} \sum_{v_j \in V} X_{ij}^k = 1, \forall v_j \in V \setminus \{ v_o \} \\
\sum_{k \in K} \sum_{v_i \in V} \sum_{v_j \in V \setminus \{ v_o \}} X_{ij}^k = n \\
\sum_{v_j \in V} X_{jo}^k = 1, \forall k \in K \\
\sum_{v_i, v_j \in V} X_{ij}^k - \sum_{j \in V} X_{ji}^k = 0, \forall k \in K, v_j \in V \setminus \{ v_o \} \\
\sum_{v_i} X_{io}^k = 1, \forall k \in K \\
e_i \le b_i \le f_i, \forall v_i \in V \\
\sum_{v_i \in V} \sum_{v_j \in V} X_{ij}^k q_i \le Q, \forall k \in K \\
\sum_{v_i \in V} \sum_{v_j \in V} X_{ij}^k \in \{0,1\}, \forall v_i, v_j \in V, k \in K
\end{align*}
\]

The first objective is to minimize the number of vehicles (Equation (5)) while the second objective is to minimize the total travelled distance (Equation (6)). Constraints (7–9) limit every customer to be visited exactly once while ensuring that all customers are served. Constraints (10–12) define the route by vehicle $k$. Constraints (13) and (14) define the customer time windows constraint and vehicle capacity constraint, respectively. Constraint (15) defines the domain of the decision variable $X_{ij}^k$.

We adapt the same evaluation function in the literature, where the two objectives are transformed into a single objective with a weight [34] as shown in Equation (16).

\[
f = \sum_{k \in K} \sum_{v_i \in V} \sum_{v_j \in V} X_{ij}^k d_{ij} + 1000 \times K
\]

The Solomon benchmark dataset [35] 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, density 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 coordinates 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.

### B. 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.

1) **Influence of Q-value Function Approximator on the Learning Models:** 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 IX. As CVRPTW is a NP-hard problem with a finite fitness search space, QL-GSF method needs to handle a large number of states. An approximation technique based on the concept of the state aggregation [36], [37], [38] is used within the QL-GSF to aggregate 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, 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: $f_1 \in [0.0, 0.4], [0.41, 0.42], \ldots, [0.59, 0.6], [0.6, 0.7], \ldots, [0.9, 1]$; for type-C, type-R2 and type-RC2 instances, the state space is divided as: $f_1 \in [0.0, 0.3], [0.31, 0.32], \ldots, [0.49, 0.5], [0.5, 0.7], [0.7, 0.8], \ldots, [0.9, 1]$. 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. In all algorithms, the population size, the number of timesteps $NoT$ and pre-defined maximum
running time of one episode $t_{\text{max}}$ are set to 100, 50, and 600s, respectively. For training the policy, the number of episodes $NoE$ is set to 500. For testing purposes, as shown in Tables X-XII, 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 [39]. The published results (BEST) of two state-of-the-art manually designed algorithms, i.e., RT and HG, 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.

It should be noted that it is usually not possible to compare design time of automated methods and manual 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. A direct comparison on the computational expenses between the proposed automated methods and the manual 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.

On the type-C instances, the results of BEST and GAP in Table XI demonstrate that these three methods can produce the current best-known solutions [39]. 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 XI and Table XII, 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.

### Table X

<table>
<thead>
<tr>
<th>Instances</th>
<th>Type</th>
<th>RT1</th>
<th>RT2</th>
<th>RT3</th>
<th>RC1</th>
<th>RC2</th>
<th>RC3</th>
<th>BEST</th>
<th>GAP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RT1</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
<tr>
<td>RT2</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
<tr>
<td>RT3</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
<tr>
<td>RC1</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
<tr>
<td>RC2</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
<tr>
<td>RC3</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
</tbody>
</table>

### Table XI

<table>
<thead>
<tr>
<th>Instances</th>
<th>Type</th>
<th>RT1</th>
<th>RT2</th>
<th>RT3</th>
<th>RC1</th>
<th>RC2</th>
<th>RC3</th>
<th>BEST</th>
<th>GAP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RT1</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
<tr>
<td>RT2</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
<tr>
<td>RT3</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
<tr>
<td>RC1</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
<tr>
<td>RC2</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
<tr>
<td>RC3</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
<td>10023.94</td>
</tr>
</tbody>
</table>

2) 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 to avoid being trapped in a local optimum. The reward curve appears to
rise with some fluctuation 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 (17).

\[
I_{\text{smoothed}} = \frac{\text{convolve}(x_{\text{buff}}, y_{\text{buff}})}{\text{convolve}(z_{\text{buff}}, y_{\text{buff}})}
\]  

(17)

where \(x_{\text{buff}}\) is the raw reward per episode, \(y_{\text{buff}} = [1, \ldots, 1]_q\) is a vector with the length of the smooth factor \(q\), \(z_{\text{buff}} = [1, \ldots, 1]_{NoE}\) 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 Fig.3, PPO-GSF
performs better than DQN-GSF in most instances. As shown
in Table XIII, 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 [39].

On the type-R instances, as illustrated in Fig.4, PPO-GSF
outperforms DQN-GSF in terms of algorithm convergence and
solution quality. Further, Table XIV reveals that PPO-GSF
outperforms DQN-GSF 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.

On the type-RC instances as illustrated in Fig.5, PPO-GSF
clearly outperforms DQN-GSF in all instances. In Table XV,
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 [39].

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. There are mainly two reasons. First, 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.

<table>
<thead>
<tr>
<th>Instance</th>
<th>C101</th>
<th>C102</th>
<th>C105</th>
<th>C201</th>
<th>C202</th>
<th>C205</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg</td>
<td>10832.94</td>
<td>10828.94</td>
<td>10828.94</td>
<td>3591.56</td>
<td>3591.56</td>
<td>3588.88</td>
</tr>
<tr>
<td>SD</td>
<td>1148.70</td>
<td>1128.65</td>
<td>1057.97</td>
<td>1157.11</td>
<td>1157.11</td>
<td>1157.11</td>
</tr>
<tr>
<td>BEST</td>
<td>10832.94</td>
<td>10828.94</td>
<td>10828.94</td>
<td>3591.56</td>
<td>3591.56</td>
<td>3588.88</td>
</tr>
<tr>
<td>GAP</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>PPO-GSF</td>
<td>10832.94</td>
<td>10828.94</td>
<td>10828.94</td>
<td>3591.56</td>
<td>3591.56</td>
<td>3588.88</td>
</tr>
<tr>
<td>GAP</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>DQN-GSF</td>
<td>10832.94</td>
<td>10828.94</td>
<td>10828.94</td>
<td>3591.56</td>
<td>3591.56</td>
<td>3588.88</td>
</tr>
<tr>
<td>GAP</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

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

Fig. 3. Influence of Policy Update Mechanisms on the Learning Models (type-C problem instances)

Fig. 4. Influence of Policy Update Mechanisms on the Learning Models (type-R problem instances)

Fig. 5. Influence of Policy Update Mechanisms on the Learning Models (type-RC problem instances)
Secondly, PPO-GSF can ensure that the learned policy is monotonically increasing due to its effective value function optimisation method, leading to better exploitation.

C. Generalization 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: generalization across the same-type instances and generalization across different-type instances.

1) Generalization 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 XVI of applying these policies to other five instances demonstrate a good degree of generalization. NV denotes the number of vehicles and TD denotes the total distance. 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 [39].

2) Generalization 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 Table XVII to other twelve instances again demonstrate the generalization 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 instance, in most instances, the trained policies can obtain non-dominated solutions to the best-known solution. 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.

V. Conclusion

In this study, a general search framework (GSF) is firstly established to formulate different metaheuristics, including single-solution based algorithms and population-based algorithms. 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 window 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.

For future work, the proposed general search framework can be extended to formulate multi-objective evolutionary algorithms to support the automated design of multi-objective algorithms. Precise measure of population diversity in both the solution space and the objective space, as well as fitness landscape analysis on the search space of algorithm compositions may further identify search-dependent features to better represent the state, enhancing the reinforcement learning based methods towards effective learning on algorithm design.

APPENDIX

The detail of the neural network used in the DQN-GSF and the PPO-GSF are shown in Fig.6 and Fig.7, respectively.

In Fig.6, 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 \((13)\) by the evaluation network every \(n\) episodes. The output of the Q networks is a set of Q values of all actions \((2, 3)\). 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 \((1), (3)\), the data flows back to \(1\).

In Fig.7, 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 is decided by the obtained probability distribution \((5)\). The selected action \(a\) is executed
TABLE XVI
GENERALIZATION ACROSS THE SAME-TYPE OF INSTANCES. * INDICATES PPO-GSF IS SIGNIFICANTLY DIFFERENT FROM DQN-GSF, I.E., p < 0.05

<table>
<thead>
<tr>
<th>Instance</th>
<th>NV</th>
<th>TD</th>
<th>AVGF</th>
<th>SD</th>
<th>BEST</th>
<th>GAP</th>
<th>AVGF</th>
<th>SD</th>
<th>BEST</th>
<th>GAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>R102</td>
<td>17</td>
<td>1486.12 [40]</td>
<td>18409.39</td>
<td>19198.08</td>
<td>455.54</td>
<td>18499.58</td>
<td>0.07%</td>
<td>18499.58</td>
<td>0.07%</td>
<td>18918.03</td>
</tr>
<tr>
<td>R105</td>
<td>14</td>
<td>1377.11 [40]</td>
<td>16466.11</td>
<td>15475.15</td>
<td>438.62</td>
<td>16343.12</td>
<td>0.08%</td>
<td>16343.12</td>
<td>0.08%</td>
<td>15451.87</td>
</tr>
<tr>
<td>R201</td>
<td>4</td>
<td>1252.37 [41]</td>
<td>4088.07</td>
<td>4088.07</td>
<td>4165.78</td>
<td>58.63</td>
<td>4069.41</td>
<td>1.88%</td>
<td>4069.41</td>
<td>1.88%</td>
</tr>
<tr>
<td>R202</td>
<td>3</td>
<td>1591.70 [43]</td>
<td>4098.07</td>
<td>4062.06</td>
<td>4165.78</td>
<td>58.63</td>
<td>4069.41</td>
<td>1.88%</td>
<td>4069.41</td>
<td>1.88%</td>
</tr>
<tr>
<td>R205</td>
<td>3</td>
<td>994.42 [43]</td>
<td>4062.06</td>
<td>4062.06</td>
<td>4165.78</td>
<td>58.63</td>
<td>4069.41</td>
<td>1.88%</td>
<td>4069.41</td>
<td>1.88%</td>
</tr>
</tbody>
</table>

TABLE XVII
GENERALIZATION ACROSS DIFFERENT-TYPE OF INSTANCES. * INDICATES PPO-GSF IS SIGNIFICANTLY DIFFERENT FROM DQN-GSF, I.E., p < 0.05

<table>
<thead>
<tr>
<th>Instance</th>
<th>NV</th>
<th>TD</th>
<th>AVGF</th>
<th>SD</th>
<th>BEST</th>
<th>GAP</th>
<th>AVGF</th>
<th>SD</th>
<th>BEST</th>
<th>GAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>C101</td>
<td>10</td>
<td>828.94 [40]</td>
<td>10828.94</td>
<td>10828.94</td>
<td>10828.94</td>
<td>0</td>
<td>10828.94</td>
<td>0</td>
<td>10828.94</td>
<td>0</td>
</tr>
<tr>
<td>C102</td>
<td>10</td>
<td>828.94 [40]</td>
<td>10828.94</td>
<td>10828.94</td>
<td>10828.94</td>
<td>0</td>
<td>10828.94</td>
<td>0</td>
<td>10828.94</td>
<td>0</td>
</tr>
<tr>
<td>C105</td>
<td>10</td>
<td>828.94 [40]</td>
<td>10828.94</td>
<td>10828.94</td>
<td>10828.94</td>
<td>0</td>
<td>10828.94</td>
<td>0</td>
<td>10828.94</td>
<td>0</td>
</tr>
<tr>
<td>C201</td>
<td>3</td>
<td>591.56 [40]</td>
<td>3591.56</td>
<td>3591.56</td>
<td>3591.56</td>
<td>0</td>
<td>3591.56</td>
<td>0</td>
<td>3591.56</td>
<td>0</td>
</tr>
<tr>
<td>C202</td>
<td>3</td>
<td>591.56 [40]</td>
<td>3591.56</td>
<td>3591.56</td>
<td>3591.56</td>
<td>0</td>
<td>3591.56</td>
<td>0</td>
<td>3591.56</td>
<td>0</td>
</tr>
<tr>
<td>C205</td>
<td>3</td>
<td>588.88 [40]</td>
<td>3588.88</td>
<td>3588.88</td>
<td>3588.88</td>
<td>0</td>
<td>3588.88</td>
<td>0</td>
<td>3588.88</td>
<td>0</td>
</tr>
<tr>
<td>RC101</td>
<td>14</td>
<td>1696.94 [44]</td>
<td>16623.58</td>
<td>1569.74</td>
<td>17402.55</td>
<td>662.31</td>
<td>16650.43</td>
<td>6.07%</td>
<td>17221.51</td>
<td>6.07%</td>
</tr>
<tr>
<td>RC102</td>
<td>12</td>
<td>1554.75 [44]</td>
<td>1477.54</td>
<td>1358.07</td>
<td>15552.85</td>
<td>22.76</td>
<td>15524.71</td>
<td>14.53%</td>
<td>15562.96</td>
<td>42.14%</td>
</tr>
<tr>
<td>RC105</td>
<td>13</td>
<td>1629.44 [45]</td>
<td>1473.36</td>
<td>1463.17</td>
<td>16933.63</td>
<td>47.03</td>
<td>16936.77</td>
<td>13.45%</td>
<td>16633.88</td>
<td>28.28%</td>
</tr>
<tr>
<td>RC201</td>
<td>4</td>
<td>1406.91 [46]</td>
<td>5439.89</td>
<td>5418.86</td>
<td>5561.91</td>
<td>38.81</td>
<td>5530.18</td>
<td>1.82%</td>
<td>5552.57</td>
<td>42.40%</td>
</tr>
<tr>
<td>RC202</td>
<td>3</td>
<td>1297.19 [47]</td>
<td>5333.71</td>
<td>5302.42</td>
<td>5333.71</td>
<td>41.46</td>
<td>5381.83</td>
<td>1.60%</td>
<td>5440.90</td>
<td>42.06%</td>
</tr>
<tr>
<td>RC205</td>
<td>4</td>
<td>1287.19 [46]</td>
<td>5333.71</td>
<td>5302.42</td>
<td>5333.71</td>
<td>41.46</td>
<td>5381.83</td>
<td>1.60%</td>
<td>5440.90</td>
<td>42.06%</td>
</tr>
</tbody>
</table>

Fig. 6. Details of the neural network of DQN-GSF method

Fig. 7. Details of the neural network of PPO-GSF method

Q networks
state + action(operator)

choose an action (operator)

Q evaluate
loss
Q target

environment (the optimization process)

replay buffer

actor

enviroment (the optimization process)

update

old parameters of actor neural network

update

advantage function

update

probability distribution

state + reward

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITY OF NOTTINGHAM. Downloaded on January 03,2023 at 09:51:08 UTC from IEEE Xplore. Restrictions apply.
of layers and neurons has been shown in Fig.6 and Fig.7.

REFERENCES


Wenjie Yi received the B.Sc. and M.Sc. degrees from Shenzhen University, Shenzhen, China, in 2016 and 2019, respectively. She is currently pursuing the Ph.D. degree in computer science with the School of Computer Science, University of Nottingham, Nottingham, U.K.

Her current research interests include evolutionary computation, machine learning, automated algorithm design and vehicle routing problem.

Rong Qu (Senior Member, IEEE) received the B.Sc. degree in computer science and its applications from Xidian University, Xi’an, China, in 1996, and the Ph.D. degree in computer science from the University of Nottingham, Nottingham, U.K. in 2003. She is an Associate Professor with the School of Computer Science, University of Nottingham. Her research interests include the modeling and optimization for logistics transport scheduling, personnel scheduling, network routing, portfolio optimization, and timetabling problems by using EAs, mathematical programming, constraint programming in operational research and artificial intelligence. These computational techniques are integrated with knowledge discovery, machine learning, and data mining to provide intelligent decision support on logistic fleet operations at SMEs, workforce scheduling at hospitals, policy making in education, and cyber security for connected and autonomous vehicles.

Dr. Qu is an Associated Editor at the IEEE Computational Intelligence Magazine, the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the Journal of Operational Research Society, and PeerJ Computer Science. She has been the Vice-Chair of the Evolutionary Computation Task Committee since 2019. She is the Vice-Chair Technical Committee on Intelligent Systems Applications from 2015 to 2018 at IEEE Computational Intelligence Society.

Licheng Jiao (Fellow, IEEE) received the B.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 1982, and the M.S. and Ph.D. degrees from Xi’an Jiaotong University, Xi’an, China, in 1984 and 1990, respectively.

Since 1992, he has been a Professor with the School of Electronic Engineering, Xidian University, Xi’an, where he is currently the Director of the Key Laboratory of Intelligent Perception and Image Understanding of the Ministry of Education of China. His research interests include image processing, natural computation, machine learning, and intelligent information processing.

Dr. Jiao is a Councilor of the Chinese Institute of Electronics; a Committee Member of the Chinese Committee of Neural Networks; an Expert of the Academic Degrees Committee of the State Council, the Chairman of the Awards and Recognition Committee; the Vice Board Chairperson of the Chinese Association of Artificial Intelligence; a Foreign Member of the Academia European and the Russian Academy of Natural Sciences; and a Fellow of IET, CAAI, CIE, CCF, and CAA.

Ben Niu (Member, IEEE) received his Ph.D. degree from Shenyang Institute of Automation of the Chinese Academy of Sciences, Shenyang, China, in 2008.

He is presently serving as a Guangdong Zhuhai Scholar Professor in Department of Management Science, Shenzhen University. He has published more than 150 papers in the international Journals and international conferences. His main fields of research are evolutionary computation and operation research.