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

***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 an MDP. Two RL methods, Deep Q-Networks (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

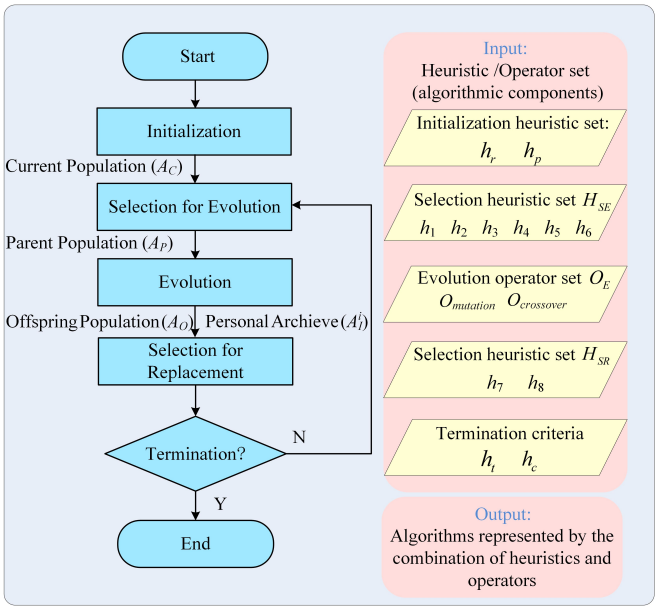


Fig. 1. General search framework

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.

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}.

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. 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.

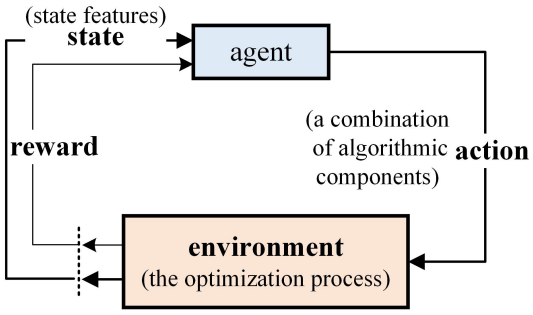


Fig. 2. Reinforcement learning in the context of automated algorithm composition in GSF

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.

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.

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

As shown in Algorithm 1, the DQN-GSF is trained on every timestep. Specifically, an action (an evolution opera-

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

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 $h_i\,(i = 1, 2, ..., 6)$ 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: $\left[r_j + \gamma \underset{a_{j+1}}{\max} \hat{Q}\left(s_{j+1}, a_{j+1}\right) - Q\left(s_j, a_j\right)\right]^2$, 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**

tor) is deterministically selected with the largest Q-value for exploitation or randomly selected for exploration (Line 8, Algorithm 1). The designed search algorithm with predefined selection heuristics is executed for one timestep (Line 9-11, Algorithm 1). The next state and reward are identified, and this experience (s_t, a_t, r_t, s_{t+1}) is stored in the memory buffer (Line 12, Algorithm 1). After that, a 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).

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 $h_i\,(i = 1, 2, ..., 6)$ from H_{SE} (fixed as h_1 in this study)
8: generate offspring population by performing the selected action a_t based on policy $\pi_k = \pi_{\theta_k}$.
9: update the population using a selection heuristic $h_i\,(i = 7, 8)$ from H_{SR} (fixed as h_8 in this study)
10: observe reward r_t based on Equation (3) and Equation (4)
11: collect experience (s_t, a_t, r_t) 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**

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 $\tau : [(s_1, a_1, r_1), (s_2, a_2, r_2), \cdots, (s_t, a_t, r_t)]$. 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.

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:

$$\text{Minimize} \quad K \tag{5}$$

$$\text{Minimize} \quad \sum_{k\in K}\sum_{v_i\in V}\sum_{v_j\in V} X_{ij}^k d_{ij} \tag{6}$$

Constraint:

$$\sum_{k\in K}\sum_{v_i\in V} X_{ij}^k = 1, \forall v_i \in V\backslash\{v_o\} \tag{7}$$

$$\sum_{k\in K}\sum_{v_j\in V} X_{ij}^k = 1, \forall v_j \in V\backslash\{v_o\} \tag{8}$$

$$\sum_{k\in K}\sum_{v_i\in V}\sum_{v_j\in V\backslash\{v_o\}} X_{ij}^k = n \tag{9}$$

$$\sum_{v_j\in V} X_{oj}^k = 1, \forall k \in K \tag{10}$$

$$\sum_{v_i\in V} X_{ij}^k - \sum_{v_j\in V} X_{ji}^k = 0, \forall k \in K, v_j \in V\backslash\{v_o\} \tag{11}$$

$$\sum_{v_i} X_{io}^k = 1, \forall k \in K \tag{12}$$

$$e_i \leqslant b_i \leqslant f_i, \forall v_i \in V \tag{13}$$

$$\sum_{v_i\in V}\sum_{v_j\in V} X_{ij}^k q_i \leqslant Q, \forall k \in K \tag{14}$$

$$X_{ij}^k \in \{0,1\}, \forall v_i, v_j \in V, k \in K \tag{15}$$

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).

$$f = \sum_{k\in K}\sum_{v_i\in V}^{K}\sum_{v_j\in V} X_{ij}^k d_{ij} + 1000 \times K \tag{16}$$

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

Apart from the Q-value function approximator, the experimental environment and parameters settings are identical for these three algorithms. In all algorithms, the population size, the number of timesteps NoT and pre-defined maximum

running time of one episode $t_{\max}$ are set to 100, 50, and 600s, respectively. For training the policy, the number of episodes NoE is set to 500. 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].

It should be noted that a direct comparison on the computational expenses between the proposed methods and the state-of-the-art algorithms which produce the best-known solutions is difficult due to the different computer platforms and/or 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 certain 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.

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.

TABLE X
COMPARISONS ON SELECTED TYPE-C INSTANCES (INFLUENCE OF Q-VALUE FUNCTION APPROXIMATOR). $\sharp$ AND $*$ INDICATE DQN-GSF IS SIGNIFICANTLY DIFFERENT FROM RANDOM-GSF AND QL-GSF, RESPECTIVELY, I.E., $p < 0.05$

Instance		C101	C102	C105	C201	C202	C205
Best-known solutions		10828.94 [40]	10828.94 [40]	10828.94 [40]	3591.56[40]	3591.56[40]	3588.88[40]
Random-GSF	AVG	11061.1	10928.73	10832.97	3604.44	3616.24	3599.05
	SD	483.51	110.36	12.75	16.64	24.9	15.82
	BEST	**10828.94**	**10828.94**	**10828.94**	**3591.56**	**3591.56**	**3588.88**
	GAP	**0**	**0**	**0**	**0**	**0**	**0**
QL-GSF	AVG	**10828.94**	10860.36	**10828.94**	**3591.56**	3618.87	3591.75
	SD	**0**	45.11	**0**	**0**	**17.32**	8.62
	BEST	**10828.94**	**10828.94**	**10828.94**	**3591.56**	**3591.56**	**3588.88**
	GAP	**0**	**0**	**0**	**0**	**0**	**0**
DQN-GSF	AVG	10831.47 $\sharp$	**10840.84** $\sharp$ $*$	10833.93	**3591.56**	**3616.15**	**3588.88** blue$\sharp$
	SD	7.61	**24.65**	14.99	**0**	17.44	**0**
	BEST	**10828.94**	**10828.94**	**10828.94**	**3591.56**	**3591.56**	**3588.88**
	GAP	**0**	**0**	**0**	**0**	**0**	**0**

TABLE XI
COMPARISONS ON SELECTED TYPE-R INSTANCES (INFLUENCE OF Q-VALUE FUNCTION APPROXIMATOR). $\sharp$ AND $*$ INDICATE DQN-GSF IS SIGNIFICANTLY DIFFERENT FROM RANDOM-GSF AND QL-GSF, RESPECTIVELY, I.E., $p < 0.05$

Instance		R101	R102	R105	R201	R202	R205
Best-known solutions		20645.79 [41]	18486.12 [40]	15377.11[40]	5252.37[42]	4191.7[43]	3994.42[43]
Random-GSF	AVG	21213.58	19541.53	16777.28	5445.83	5285.64	4193.57
	SD	525.00	25.30	503.17	75.07	50.04	37.23
	BEST	21637.7	19510.7	16421.4	**5318.79**	5183.26	4139.7
	GAP	4.8%	5.5%	6.8%	**1.3%**	23.66%	3.6%
QL-GSF	AVG	21893.56	**19505.09**	16938.17	5384.10	5213.21	4179.57
	SD	**397.57**	**19.62**	500.57	40.39	**22.08**	**27.64**
	BEST	21665.47	19486.77	16438.95	5336.10	5183.86	4131.79
	GAP	4.94%	5.41%	6.91%	1.59%	23.67%	3.44%
DQN-GSF	AVG	**21171.67** $\sharp$ $*$	19516.86 $\sharp$	**16736.59** $*$	**5366.27** $\sharp$ $*$	**5191.93** $\sharp$	**4168.28** $\sharp$ $*$
	SD	500.27	20.15	**459.63**	**27.17**	31.23	36.63
	BEST	**20655.81**	**19481.96**	**16414.47**	5325.66	**5137.16**	**4086.29**
	GAP	**0.05%**	**5.39%**	**6.75%**	1.40%	**22.56%**	**2.30%**

TABLE XII
COMPARISONS ON SELECTED TYPE-RC INSTANCES (INFLUENCE OF Q-VALUE FUNCTION APPROXIMATOR). $\sharp$ AND $*$ INDICATE DQN-GSF IS SIGNIFICANTLY DIFFERENT FROM RANDOM-GSF AND QL-GSF, RESPECTIVELY, I.E., $p < 0.05$

Instance		RC101	RC102	RC105	RC201	RC202	RC205
Best-known solutions		15696.94 [44]	13554.72 [44]	14628.44[45]	5406.91[46]	4367.09[47]	5297.19[46]
Random-GSF	AVG	**17340.05**	16196.35	16971.02	5631.78	5439.67	5533.55
	SD	**537.58**	543.03	508.91	73.60	53.03	62.02
	BEST	16687.5	**14527.04**	16627.61	5561.31	5315.21	5435.81
	GAP	6.3%	7.2%	13.7%	2.9%	21.7%	2.6%
QL-GSF	AVG	17840.39	15967.39	**16932.14**	**5595.50**	5464.09	**5430.87**
	SD	716.74	499.91	**463.88**	**30.34**	**31.41**	**33.28**
	BEST	16686.71	15502.68	16613.85	5553.28	5446.44	5396.74
	GAP	6.31%	14.37%	13.56%	2.71%	24.72%	**1.87%**
DQN-GSF	AVG	17523.59 $\sharp$ $*$	**15653.58** $\sharp$ $*$	16936.21 $\sharp$	5659.06 $*$	**5338.71** $\sharp$ $*$	5547.81 $*$
	SD	621.65	**294.00**	468.76	296.92	33.30	305.41
	BEST	**16662.50**	15490.86	**16589.13**	**5473.00**	**5251.44**	**5396.45**
	GAP	**6.15%**	**14.28%**	**13.4%**	**1.22%**	**20.25%**	1.88%

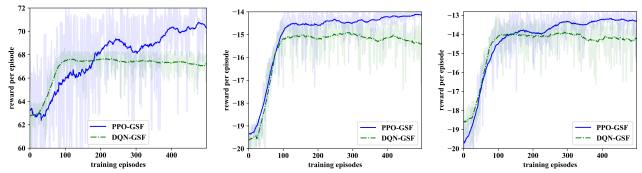
The policies of PPO-GSF and DQN-GSF are gradually improved during the training process. However, a certain degree of randomness must be maintained to avoid being trapped in a local optimum. The reward curve appears to rise with some fluctuation as a result of this. To present the training effects more clearly, the reward curve is smoothed by using a sliding window filter method (moving average), as shown in Equation (17).

$$l^{smoothed} = \frac{convolve\left(x_{buff}, y_{buff}\right)}{convolve\left(z_{buff}, y_{buff}\right)} \tag{17}$$

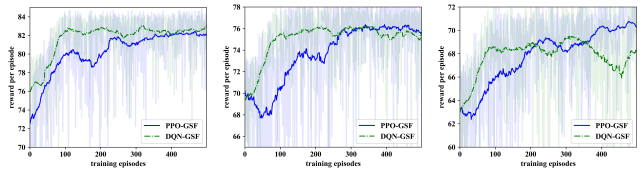
where x_{buff} is the raw reward per episode, $y_{buff} = [1, \cdots, 1]_q$ is a vector with the length of the smooth factor q, $z_{buff} = [1, \cdots, 1]_{NoE}$ is a vector with the length of the whole training data. q is set to 5 in the experiment.

On the type-C instances, as illustrated in 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].



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



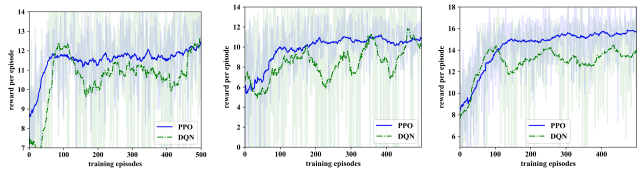
(d) C201 (e) C202 (f) C205

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

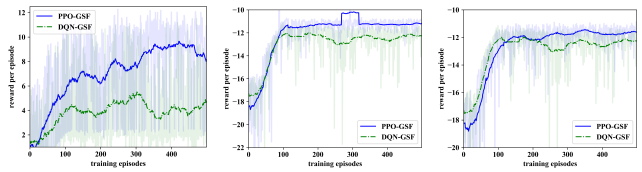
TABLE XIII
COMPARISONS ON SELECTED TYPE-C INSTANCES (INFLUENCE OF POLICY UPDATE MECHANISMS). ∗ INDICATES PPO-GSF IS SIGNIFICANTLY DIFFERENT FROM DQN-GSF, I.E., $p < 0.05$

Instance		C101	C102	C105	C201	C202	C205
Best-known solutions		10828.94 [40]	10828.94 [40]	10828.94[40]	3591.56[40]	3591.56[40]	3588.88[40]
DQN-GSF	AVG	11148.36	10900.96	10832.46	**3591.56**	3612.52	**3588.88**
	SD	487.93	112.83	10.57	**0**	13.95	**0**
	BEST	**10828.94**	**10828.94**	**10828.94**	**3591.56**	**3591.56**	**3588.88**
	GAP	**0**	**0**	**0**	**0**	**0**	**0**
PPO-GSF	AVG	**10834.01** ∗	**10855.22** ∗	**10831.72**	**3591.56**	**3595.29** ∗	**3588.88**
	SD	**10.15**	**50.24**	**8.36**	**0**	**11.19**	**0**
	BEST	**10828.94**	**10828.94**	**10828.94**	**3591.56**	**3591.56**	**3588.88**
	GAP	**0**	**0**	**0**	**0**	**0**	**0**

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 achieves better results in terms of all four indicators AVG, SD, BEST and GAP in most instances. The GAPs of PPO-GSF are less than 3% in most instances except on the R202 instance.



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



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

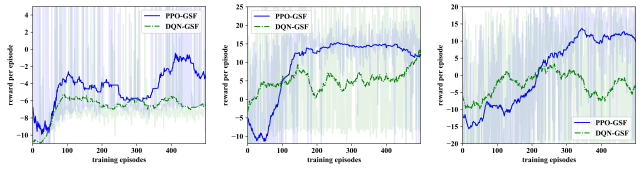
Fig. 4. Influence of Policy Update Mechanisms on the Learning Models (type-R problem instances)

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

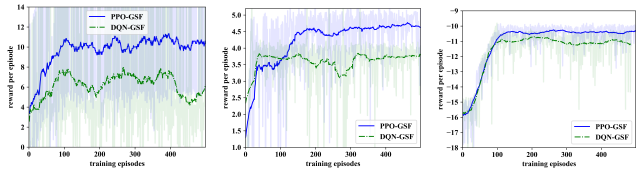
TABLE XIV
COMPARISONS ON SELECTED TYPE-R INSTANCES (INFLUENCE OF POLICY UPDATE MECHANISMS). ∗ INDICATES PPO-GSF IS SIGNIFICANTLY DIFFERENT FROM DQN-GSF, I.E., $p < 0.05$

Instance		R101	R102	R105	R201	R202	R205
Best-known solutions		20645.79 [41]	18486.12 [40]	15377.11 [40]	5252.37 [42]	4191.7[43]	3994.42[43]
DQN-GSF	AVG	20981.19	19910.38	16434.78	**5378.77**	5201.13	4280.21
	SD	454.50	452.07	489.72	44.67	34.34	308.16
	BEST	20656.49	19495.43	**15410.25**	**5304.48**	5159.01	4131.92
	GAP	**0.05%**	5.5%	**0.02%**	**0.99%**	23.1%	3.4%
PPO-GSF	AVG	**20665.46** ∗	**18708.26** ∗	**16329.39** ∗	5382.98	**5200.721**	**4145.18** ∗
	SD	**10.16**	**413.91**	**321.27**	**39.15**	**29.83**	**29.52**
	BEST	**20655.81**	**18493.03**	15418.00	5318.35	**5144.32**	**4094.81**
	GAP	**0.05%**	**0.04%**	0.03%	1.3%	**22.7%**	**2.5%**

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].



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



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

Fig. 5. Influence of Policy Update Mechanisms on the Learning Models (type-RC problem instances)

TABLE XV
COMPARISONS ON SELECTED TYPE-RC INSTANCES (INFLUENCE OF POLICY UPDATE MECHANISMS). ∗ INDICATES PPO-GSF IS SIGNIFICANTLY DIFFERENT FROM DQN-GSF, I.E., $p < 0.05$

Instance		RC101	RC102	RC105	RC201	RC202	RC205
Best-known solutions		15696.94 [44]	13554.72 [44]	14628.44[45]	5406.91[46]	4367.09[47]	5297.19[46]
DQN-GSF	AVG	17586.44	16146.86	17041.4	5661.65	**5351.60**	5535.85
	SD	728.65	**486.43**	499.10	269.55	**45.33**	284.21
	BEST	**16570.05**	15526.71	16603.84	5507.77	5293.64	5361.69
	GAP	**5.56%**	14.55%	13.5%	1.87%	21.2%	**1.2%**
PPO-GSF	AVG	**17521.35** ∗	**16005.04** ∗	**16543.15** ∗	**5567.00** ∗	5359.51	**5450.50** ∗
	SD	**410.16**	525.97	**309.87**	**78.01**	57.68	**36.60**
	BEST	16679.80	**15511.07**	**15617.44**	**5467.35**	**5246.94**	**5359.96**
	GAP	6.3%	**14.4%**	**6.8%**	**1.12%**	**20.1%**	**1.2%**

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

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

	Best-known solution in the literature		DQN-GSF (trained policy)				PPO-GSF (trained policy)			
Instance	NV	TD	AVG	SD	BEST	GAP	AVG	SD	BEST	GAP
R102	17	1486.12 [40]	19198.08	**455.54**	**18499.58**	**0.07%**	**18918.03** $*$	486.78	18507.80	0.12%
R105	14	1377.11 [40]	16446.11	438.62	15475.15	0.64%	**16343.12** $*$	**297.87**	**15451.57**	**0.48%**
R201	4	1252.37 [42]	5369.89	39.99	5325.66	1.40%	**5344.44**	**24.74**	**5311.65**	**1.13%**
R202	3	1191.70 [43]	5180.19	**31.71**	5131.13	22.41%	**5103.13** $*$	273.80	**4286.01**	**2.25%**
R205	3	994.42 [43]	4165.78	58.63	**4069.41**	**1.88%**	**4135.23**	**26.08**	4096.00	2.54%

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

	Best-known solution in the literature		DQN-GSF (trained policy)				PPO-GSF (trained policy)			
Instance	NV	TD	AVG	SD	BEST	GAP	AVG	SD	BEST	GAP
C101	10	828.94 [40]	**10828.94**	**0**	**10828.94**	**0**	**10828.94**	**0**	**10828.94**	**0**
C102	10	828.94 [40]	**10856.95**	**52.99**	**10828.94**	**0**	10867.18	57.35	**10828.94**	**0**
C105	10	828.94 [40]	**10828.94**	**0**	**10828.94**	**0**	10836.85	15.84	**10828.94**	**0**
C201	3	591.56 [40]	**3591.56**	**0**	**3591.56**	**0**	**3591.56**	**0**	**3591.56**	**0**
C202	3	591.56 [40]	**3608.60**	25.82	**3591.56**	**0**	3615.97 $*$	**12.53**	**3591.56**	**0**
C205	3	588.88 [40]	**3588.88**	**0**	**3588.88**	**0**	**3588.88**	**0**	**3588.88**	**0**
RC101	14	1696.94 [44]	17402.55	662.31	**16650.43**	**6.07%**	**17221.51** $*$	**501.87**	16671.46	6.20%
RC102	12	1554.75 [44]	**15552.85**	**22.76**	15524.71	14.53%	15562.96	425.14	**14678.64**	**8.29%**
RC105	13	1629.44 [45]	16933.63	473.08	16596.77	13.45%	**16633.88** $*$	**28.28**	**16592.69**	**13.42%**
RC201	4	1406.91 [46]	5561.91	**38.81**	5505.18	1.82%	**5552.57**	42.40	**5478.44**	**1.3%**
RC202	3	1367.09 [47]	5332.53	45.89	**5243.20**	**20.06%**	**5323.74**	**44.29**	5251.08	20.24%
RC205	4	1297.19 [46]	5441.44	**41.46**	**5381.83**	**1.60%**	**5440.90**	42.06	5386.30	1.68%

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.

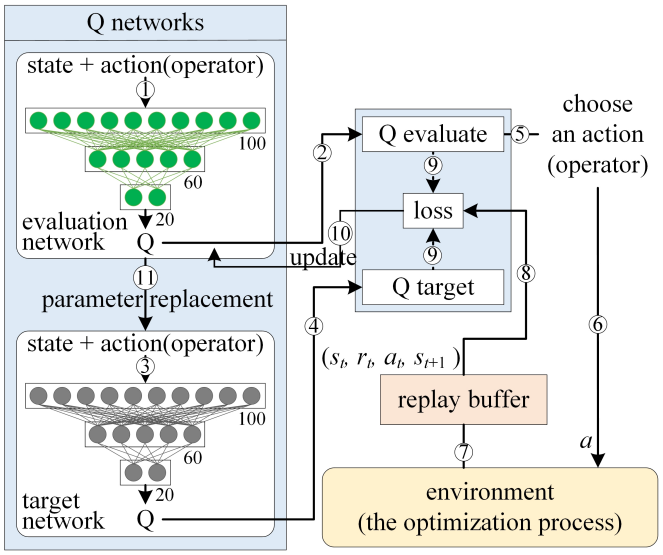


Fig. 6. Details of the neural network of DQN-GSF method

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 (⑪) by the evaluation network every n episodes. The output of the Q networks is a set of Q values of all actions (②, ④). The action is decided by the maximal Q values (⑤). The selected action a is executed (⑥) in the environment, called one step. The (s_t, r_t, a_t, s_{t+1}) generated at each step is stored in the replay buffer (⑦). The loss of the parameters of the evaluation network is calculated(⑧). After the update of the Q networks (⑩, ⑪), the data flows back to ①.

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

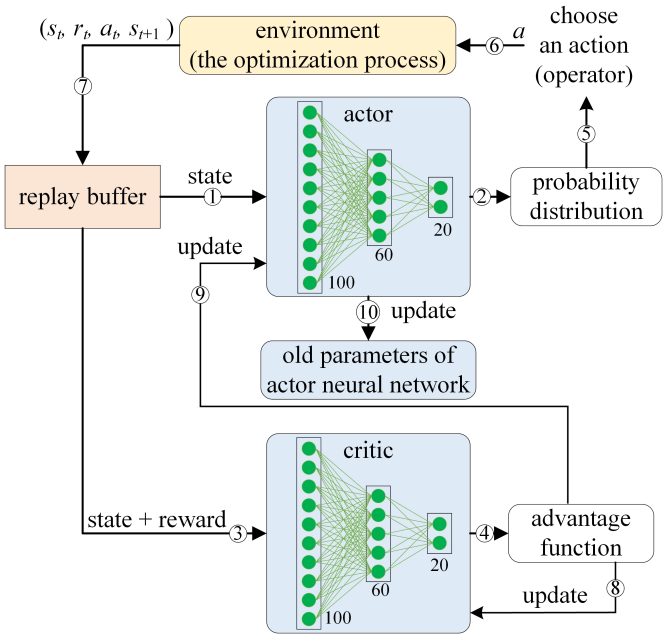


Fig. 7. Details of the neural network of PPO-GSF method

In RL, the learning rate, discount rate, and size of the neural network are the key hyper parameters in the algorithms. Specifically, the learning rate is adjusted adaptively, set as 0.002 at the beginning and halved when the output is stable. The discount rate is set to 0.99 thus the learned policy focuses more on sequential decisions. The topology of the network is set based on the complexity of the problem, and the number of layers and neurons has been shown in Fig.6 and Fig.7.

REFERENCES

[1] N. Pillay, R. Qu, D. Srinivasan, B. Hammer, and K. Sorensen, "Automated design of machine learning and search algorithms [guest editorial]," *IEEE Computational intelligence magazine*, vol. 13, no. 2, pp. 16–17, 2018.

[2] H. J. Escalante, Q. Yao, W.-W. Tu, N. Pillay, R. Qu, Y. Yu, and N. Houlsby, "Guest editorial: Automated machine learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 2887–2890, 2021.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[26] S. v. Rijn, C. Doerr, and T. Bäck, "Towards an adaptive cma-es configurator," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2018, pp. 54–65.

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

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

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

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

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

[32] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, part ii: Metaheuristics," *Transportation science*, vol. 39, no. 1, pp. 119–139, 2005.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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