Automated Algorithm Design Using Proximal Policy Optimisation with Identified Features

Wenjie Yi[a,*], Rong Qu[a], Licheng Jiao[b]

[a]*School of Computer Science, University of Nottingham, Nottingham, UK*
[b]*School of Artificial Intelligence, Xidian University, China*

Abstract

Automated algorithm design is attracting considerable recent research attention in solving complex combinatorial optimisation problems, due to that most metaheuristics may be particularly effective at certain problems or certain instances of the same problem but perform poorly at others. Within a general algorithm design framework, this study investigates reinforcement learning on the automated design of metaheuristic algorithms. Two groups of features, namely search-dependent and instance-dependent features, are firstly identified to represent the search space of the algorithm design to support effective reinforcement learning on the new task of algorithm design. With these key features, a state-of-the-art reinforcement learning technique, namely proximal policy optimisation, is used to automatically combine the basic algorithmic components within the general framework to develop effective metaheuristics. Patterns of the best designed algorithm, in particular the utilisation and transition of algorithmic components, are investigated. Experimental results on the capacitated vehicle routing problem with time windows benchmark dataset demonstrate the effectiveness of the identified features in assisting automated algorithm design with the proposed reinforcement learning model.

Keywords: automated algorithm design, feature identification, reinforcement learning, search pattern

*Corresponding author
Email addresses: wenjie.yi@nottingham.ac.uk (Wenjie Yi), rong.qu@nottingham.ac.uk (Rong Qu), lchjiao@mail.xidian.edu.cn (Licheng Jiao)

1. Introduction

To tackle complex combinatorial optimisation problems (COPs), effective metaheuristics have shown to be able to obtain acceptable solutions within a reasonable computational time in the optimisation research community. However, most metaheuristics in the literature are manually designed for a specific problem model or even to a specific problem instance. They cannot always be easily extended to solve other problems or even other instances of the same problem. Furthermore, the large amount of data generated during the search of metaheuristics, leading to either good or bad solutions, may carry useful knowledge. The knowledge retained in these data can be extracted and used for designing more general and intelligent metaheuristics; for example, with the key information on the search space to better determine the most suitable operator to be applied during different search stages. This represents one line of research, automatically designing search algorithms to solve different instances of the same COP or cross-domain COPs.

In automated algorithm design, a standard called GCOP, which models the problem of algorithm design itself as a COP, has been established to support automated design of algorithmic components and design of new metaheuristics (Qu et al., 2020). Based on the GCOP model, a general framework called AutoGCOP has been proposed to support the automated design of local search algorithms (Meng & Qu, 2021). A general search framework (GSF) has also been developed to support the automated design of both local search algorithms and population-based algorithms (Yi et al., 2022). Based on GSF, we further explore automated algorithm design using machine learning techniques in this study for solving the capacitated vehicle routing problem with time windows (CVRPTW).

Identifying effective features to characterise the search space of algorithm design and the problem instance plays an important role in the automated algorithm design process. A suitable feature set can help to distinguish different

states, which is the key to the performance improvement. For this purpose, different features have been proposed in the literature.

In supporting effective automated algorithm design, we identified and categorised existing features for designing general search algorithms into two groups. The first group, namely search-dependent features, is composed of features observing the search, such as the mean and standard deviation of the population fitness, and the average distance from the best individual (Eiben et al., 2006). The second group, namely instance-dependent features, consists of the basic characteristics of the problem instances. Taking the VRP as an example, different instance-dependent features identified in (Gutierrez-Rodríguez et al., 2019) include the vehicle capacity, the average customer demand and the average time-window size. Other types of features, such as landmarking features (Gutierrez-Rodríguez et al., 2019) and image features (Jiang et al., 2021), are not included in this study as they are specifically associated with the solution encoding scheme and therefore are not transferable for developing a general methodology; this would not serve the purpose of automated algorithm design.

The present research has been conducted with the following two motivations based on the literature review. At the application level, the CVRPTW instances share the same problem structure but differ in the data (e.g. customer locations and demand). However, existing evolutionary algorithms and metaheuristics treat each instance independently, requiring considerable human efforts in algorithm design. Furthermore, these manually designed algorithms are usually discarded after solving the specific instances. At the methodology level, utilising machine learning techniques to assist algorithm design is still at a preliminary stage albeit some successful attempts. One of the important issues is on how to identify the key features to accurately characterise the search space for building successful machine learning. Although various features have been extracted for effective algorithm design, there is a lack of a systematic investigation analysing the extracted features within a consistent and general framework. The aim of this study is therefore to identify and analyse feature sets which provide sufficient key information on the state of the evolutionary search, and to verify

the effectiveness of the identified features in assisting algorithm design with the support of machine learning techniques. More specifically, we aim to answer the following research questions (RQ):

- *RQ1* What kind of features we can identify to provide useful information for assisting algorithm design?
- *RQ2* Is machine learning effective to utilise such data and to automatically design effective search algorithms with little human intervention?
- *RQ3* Are there search patterns we can observe from the automatically designed algorithms to derive new knowledge in evolutionary computation?

The main contributions of this study are listed as follows:

- Two groups of features, search-dependent and instance-dependent features, are identified to provide the key information on the search space of algorithm design (RQ1).
- A state-of-the-art reinforcement learning technique, proximal policy optimisation, is employed to extract useful knowledge hidden in the search data, which is plugged into the algorithm design process. The influence of state representation (i.e. with different features) is investigated (RQ2).
- Search patterns of the algorithms which are automatically designed by the learning technique, consisting of the utilisation and transition of algorithmic components, are analysed to further provide insights into reusing knowledge extracted in algorithm design using machine learning (RQ3).

The remainder of this paper is organised as follows. Section 2 presents related work on the existing features and reinforcement learning methods for automated algorithm design. Section 3 describes the features identified and learning techniques proposed in this study within the general algorithm design framework. Further, in Section 4, experiments are conducted to demonstrate the effectiveness of the identified feature sets and to analyse the search patterns

of the best designed algorithms. Section 5 presents the conclusion and future work.

2. Related work

In the existing literature of automated algorithm design, there is relatively less work on systematic analysis on different types of features for developing reinforcement learning to support the design of general population-based algorithms within a coherent framework.

2.1. Existing Features for Automated Algorithm Design

To support automated design of metaheuristics, the first important step is to identify features which can comprehensively capture the characteristics of the search space of algorithms and the problem instances. Existing features used for automated algorithm design can be roughly categorised into search-dependent features and instance-dependent features.

Search-dependent features aim to describe the search process accurately. These features can be extracted during the search of metaheuristics, and roughly divided into two categories. The first category is stage-based, such as the current iteration or stage of the search process (Wauters et al., 2013). The second category is solution-based which are usually derived from the solutions obtained by the metaheuristics, such as the best fitness, the fitness growth (Gutierrez-Rodríguez et al., 2019), the number of feasible solutions, the number of feasible solutions that are better than the initial solution (Jiang et al., 2021), and the mean and the standard deviation of explored solutions (Eiben et al., 2006) (Gutierrez-Rodríguez et al., 2019).

Instance-dependent features consist of specific characteristics extracted from the problem instance definition, which can be divided into two categories. For VRP, the first category includes customer-based features, for example, the mean/maximum/median and the skewness of the customer demand and the service time of customer, etc (Jiang et al., 2021). The second category of

constraint-based features include the vehicle capacity, the average time-window size and the average time-window overlap (Gutierrez-Rodríguez et al., 2019), etc.

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

2.2. Reinforcement Learning Method for Automated Algorithm Design

Reinforcement learning (RL) techniques have been employed to repeatedly interact with an environment of algorithmic components and make decisions to maximise a reward, demonstrating a great potential to learn a policy for automatically composing new metaheuristics. In recent studies, researchers have attempted to utilise reinforcement learning techniques, such as Q-learning (QL) (Watkins & Dayan, 1992), deep Q-network (DQN) (Mnih et al., 2015) and proximal policy optimisation (PPO) (Schulman et al., 2017), in automated algorithm design. In particular, automated algorithm design is modelled as a Markov decision process (MDP), which is a typical sequential decision problem.

In the literature on automated algorithm design, QL-based automated methods showed to be effective on the unmanned aerial vehicles (Duflo et al., 2020) and different COPs (Choong et al., 2018). DQN-based automated methods also obtained promising results for the container terminal truck routing problem and online 2D strip packing problem (Zhang et al., 2022), the vehicle routing problem and the travelling salesman problem (Dantas et al., 2021). In addition, DQN-based automated methods have also been used to select evolution operators during the search process, which is a key issue in the design of search

algorithms (Yi et al., 2022) (Tian et al., 2022).

It is shown from research studies that value-based RLs are unstable and suffer from poor convergence, due to that they aim to learn a state-action value function approximation, which is then used to find a corresponding policy, rather than directly optimise the policy itself. Furthermore, some value-based RLs such as QL used in the context of automated algorithm design are based on tabular values from the discretization of the continuous state space.

Compared with the value-based RLs, there are fewer studies on policy-based RLs for the task of automated algorithm design in the literature. A possible explanation is that policy-based RLs tend to be much less sample efficient than value-based RLs. However, these policy-based RLs showed to be more stable and less prone to failure because of their stronger exploration capability. Policy-based RL has been utilised to select and combine evolution operators for solving VRP in our previous study (Yi et al., 2022).

Therefore, in this study, we apply an actor-critic RL technique, which combines the advantages of both policy-based RLs (i.e. learn the policy directly) and value-based RLs (i.e. sample efficiently) to automatically design search algorithms. More specifically, the policy function is responsible for generating actions and interacting with the environment. The state-value function is responsible for evaluating the performance of the selected actions.

3. Identified Features and Proposed Method

Figure 1 depicts the research framework of the proposed reinforcement learning method in the context of automated algorithm design. This study involves the use of a reinforcement learning method to automatically generate search algorithms for solving different instances of the vehicle routing problem with time windows. Specifically, the overall research framework has been defined in five steps. The first three steps are related to the definition of three key components of the RL method (i.e. state, action and reward scheme), shown in Sections 3.2.1-3.2.3. Note that one of the focuses of this study is on identifying

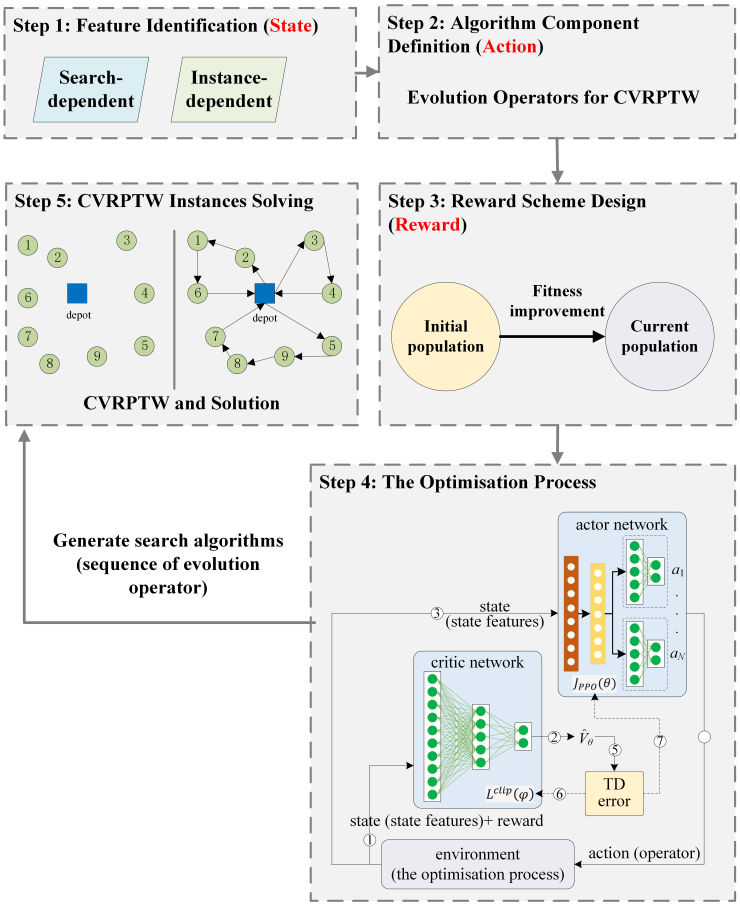


Figure 1: Research framework of the proposed reinforcement learning method in the context of automated algorithm design

key features to capture and characterise the search space of algorithm design, details of these identified features are discussed in Section 3.1. The fourth step is the main optimisation process, i.e. automatically generating search algorithms based on RL. The final step is related to solving the specific CVRPTW instance using the generated search algorithms in Step 4.

3.1. Feature Identification for Automated Algorithm Design

Identifying appropriate features is a crucial step to conduct effective reinforcement learning for automated algorithm design. Different features provide different aspects of essential information, making a significant impact on the performance of the automated algorithm design methods. In this section, two groups of features (i.e. search-dependent and instance-dependent features) are identified for developing a general methodology to solve different CVRPTW instances.

3.1.1. Search-dependent Features

To assist algorithm design, ten search-dependant key features have been extracted from the data collected during the optimisation process, as defined in Table 2.

Table 1: Symbols used for defining search-dependent features shown in Table 2

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

Features in Table 2 capture the characteristics of the search process from different aspects. With this information on the intensification and diversification of the search, the learning agent can be guided toward making better decisions

on selection of algorithm components during the search, considering the balance between searching beyond the already explored area (i.e. exploration) and focusing on the already explored area (i.e. exploitation). Such balance makes a significant impact on the performance of the designed search algorithms.

Table 2: Search-dependent features

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

3.1.2. Instance-dependent Features

Instance-dependent features assist algorithm design with the basic information on problem instances. Values of these features are determined and will not change in algorithm design once an instance is given. Related symbols are shown in Table 3 for the definitions of instance-dependent features listed.

Table 3: Symbols for instance-dependent features defined in Table 4

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

In Table 4, average time-window overlap as calculated based on Equation (1) and Equation (2) is proposed in (Gutierrez-Rodríguez et al., 2019) to measure the relationships between the time-windows of all customer in a given instance.

$$inter\left(i,j\right)=\begin{cases} I=\min\left\{d_i,d_j\right\}-\max\left\{r_i,r_j\right\},I>0 \\ 0,otherwise \end{cases} \tag{1}$$

$$union\left(i,j\right)=\begin{cases} I=\min\left\{d_i,d_j\right\}-\max\left\{r_i,r_j\right\},I>0 \\ \left(d_i-r_i\right)+\left(d_j-r_j\right),otherwise \end{cases} \tag{2}$$

Table 4: Instance-dependent features

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

3.2. Reinforcement Learning Method for Automated Algorithm Design

This study aims to address the key issue of automated algorithm design rather than considering the whole design space. Based on the general search framework (Yi et al., 2022), the key focus of algorithm design is on selecting

and composing evolution operators at different stages during the search process. An actor-critic proximal policy optimisation method is used to determine the suitable evolution operators of the search algorithm.

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

3.2.1. State Representation

The search-dependent features in Table 2 and instance-dependent features in Table 4, are used to define the state space.

3.2.2. Action Representation

Table 5: O_E: Evolutionary 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
$o_{ruin\_recreat}$	Remove m nodes from different routes in a solution and insert them back satisfying feasibility
o_{2opt}	Exchange two nodes in the same route in a solution
o_{2opt*}	Take the end sections of two routes in a solution and swap them to create two new routes

The action space is defined by the set of evolutionary operators in Table 5, which are the algorithmic components of Evolution module in the general search framework (GSF). Refer to (Yi et al., 2022) for more information of the GSF and its basic modules. The main task of the proposed PPO method is

to address the key issue of automated selection and combination of the most efficient evolutionary operators during different optimisation stages.

3.2.3. Reward Scheme

Reward scheme is used for the learning agent to determine whether the selected action is appropriate, thus, is very important for an RL method. As shown in Equation (3) and Equation (4), the reward in the proposed PPO model is calculated based on the fitness improvement of the current population over the initial population. A higher reward is given to the same fitness improvement when population fitness is optimised above a certain threshold.

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

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

3.2.4. Optimisation Process and Problem Solving

Table 6: Notations used in PPO methods shown in Figure 1 (Step 5)

Notation	Description
t	current timestep
s_t	state at timestep t
a_t	selected action at timestep t
r_t	reward value at timestep t
λ	parameter of λ-step return
V	state-action value function
θ	parameter of the actor neural network
φ	parameter of the critic neural network

The overview of the actor-critic PPO method is shown in Figure 1 (Step 5). Related notations are given in Table 6.

The reward and state (defined by the identified features) (①) obtained from the environment is taken as the input of the critic neural network, the output

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

The temporal difference error (TD-error) is calculated (⑤) by the state-action value $\hat{V}_\theta$ (②). The actor neural network is updated by $J_{PPO}(\theta)$ (⑦), as shown in Equation (5), and the critic neural network is updated based on the calculation of $L^{clip}(\varphi)$ (⑥), as shown in Equation (8).

The output of the learned policy is a probability distribution of actions (i.e. operators). The sequential sample results of one episode are the composition operators in the automated algorithm design. The action (i.e. operator) obtained (④) by the actor neural network is executed in the environment (i.e. the optimisation process).

The objective function of the actor neural network $J_{PPO}(\theta)$ is shown in Equation (5).

$$J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta\left(a_t \left| s_t\right.\right)}{\pi_{old}\left(a_t \left| s_t\right.\right)} A_t \tag{5}$$

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

$$A_t = \delta_t + (\gamma\lambda)\,\delta_t + \cdots + (\gamma\lambda)^{T-t+1}\,\delta_{T-1} \tag{6}$$

$$\delta_t = r_t + \gamma V\left(s_{t+1}\right) - V\left(s_t\right) \tag{7}$$

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

The critic neural network $L^{clip}(\varphi)$ aims at minimising the loss function, as shown in Equation (8).

$$L^{clip}(\varphi) = E_t\left\{\min\left[r_t(\varphi)\cdot A_t, clip\left[r_t(\varphi), 1-\epsilon, 1+\epsilon\right]\cdot A_t\right]\right\} \tag{8}$$

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

4. Experiments and Discussion

The experimental analysis aims to address two research issues, 1) verifying the impact of the identified features on the reinforcement learning model and 2) analysing the search patterns of the best designed algorithm compositions. In assessing the impact of the identified features, two learning models with different features are trained and tested Section 4.2. In the analysis of search patterns, the utilisation and transition of algorithmic components are included in Section 4.3.1 and Section 4.3.2, respectively. The results of the proposed method are compared with the best-known results in the literature.

4.1. Problem Definition and Dataset

To verify the performance of the learning model with different identified features, the capacitated vehicle routing problem with time windows (CVRPTW) is considered as the testing combinatorial optimisation problem. CVRPTW concerns routing for a fleet of vehicles to serve a set of customers with the known demand and time windows from and back to a depot. The vehicle capacity is limited and each customer must be visited exactly by one vehicle within the predefined time windows. To better illustrate the problem definition, Figure 2 provides an illustrative example of four routes/vehicles and a single depot to illustrate the problem.

In the CVRPTW, there are two objectives, minimising the number of vehicle (NV) and minimising the total travelled distance (TD). Similar to the literature (Walker et al., 2012), the weighted sum objective function, as shown in Equation

(9), is adopted. c is set to 1000 empirically to define the higher importance of NV.

$$f = c \times NV + TD \tag{9}$$

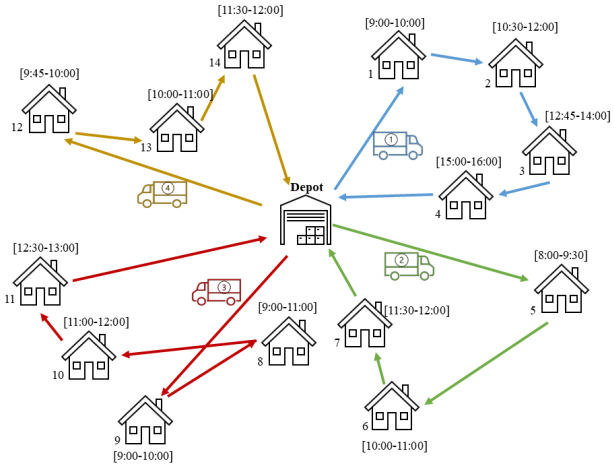


Figure 2: Example of a CVRPTW instance with four routes and a single depot

The widely studied Solomon dataset (Solomon, 1987) which consists of six sets of instances of different characteristics (C1, C2, R1, R2, RC1, RC2), is used for the investigation. Four instances are selected from each set as representatives, as shown in Table 7.

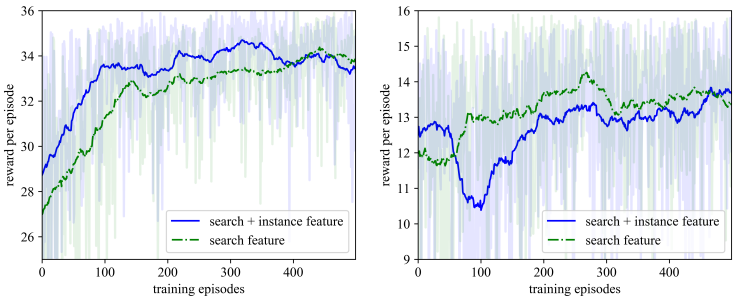
Customers in type-C instances are distributed in clusters while customers in type-R instances are distributed randomly. Type-RC instances contain a mixture of random and clustered customers. Each type of instances also differs with respect to the vehicle capacity, the density and tightness of the time windows. Note that the focus of this study is on developing a learning method to automatically design general-purpose search algorithms for solving vehicle routing problems. Therefore, the selected benchmark dataset consisting instances with different characteristics is enough to justify the performance of the developed learning method.

4.2. Effectiveness of the Identified Features

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

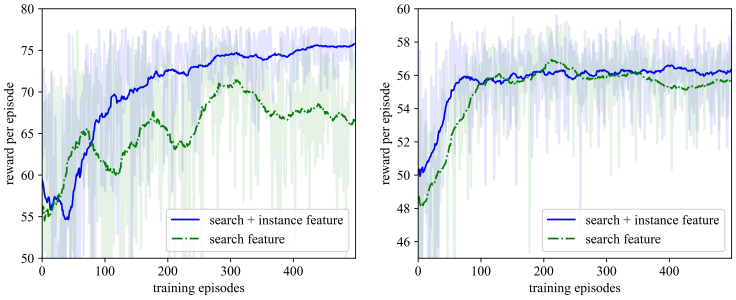
The learning of the PPO model is shown in Figures 3-5. In most instances, the performance of the learning model during the training process deteriorates without the instance-dependent feature set, except on instances R103 and RC103. This indicates that the instance-dependent features can provide useful information to the learning process, by assisting the population to accurately determine the resulting state with better action choice.

In the testing process, the same conclusion is reached that the instance-dependent feature set is effective for learning algorithm design, as shown in Table 7. When instance-dependent features are included, the four performance indicators, i.e. the average fitness value (AVG), the standard deviation of fitness value (SD), the best fitness value within 10 runs (BEST), and the gap between BEST and the best-known solution in the literature (GAP), achieve better values in most instances. Both learning models (i.e. one with both feature sets, and one with only search-dependent features) achieve quite similar BEST to the current best-known results in the literature (i.e. the GAP values are less than 5% in most instances), which verifies the effectiveness of the proposed PPO models. Noted that the aim of automated algorithm design is not trying to beat all the other manually designed metaheuristics but to develop an effective search algorithm without too much human involvement.



(a) C101

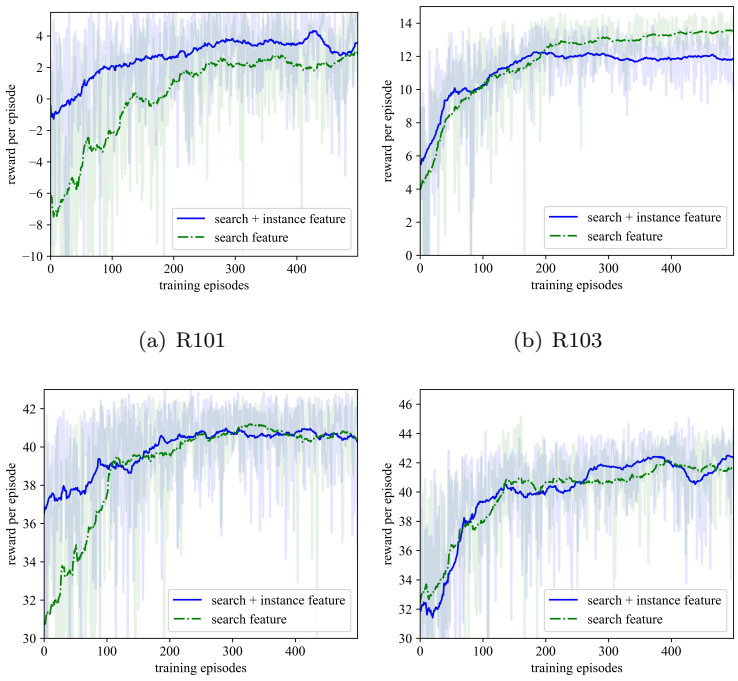
(b) C103



(c) C201

(d) C203

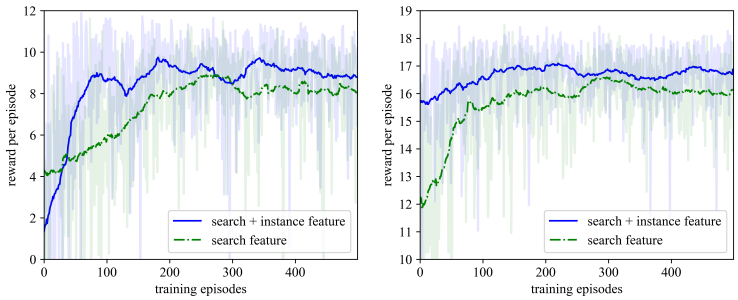
Figure 3: Influence of different feature sets on the learning model during training (type-C)



(a) R101 (b) R103

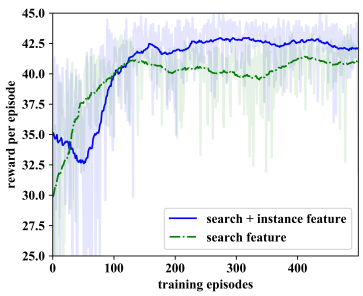
(c) R201 (d) R203

Figure 4: Influence of different feature sets on the learning model during training (type-R)

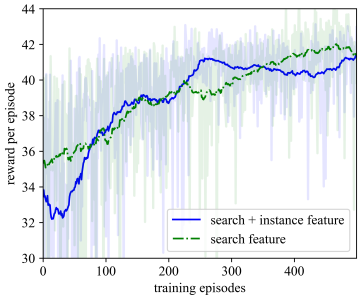


(a) RC101

(b) RC103



(c) RC201



(d) RC203

Figure 5: Influence of different feature sets on the learning model during training (type-RC)

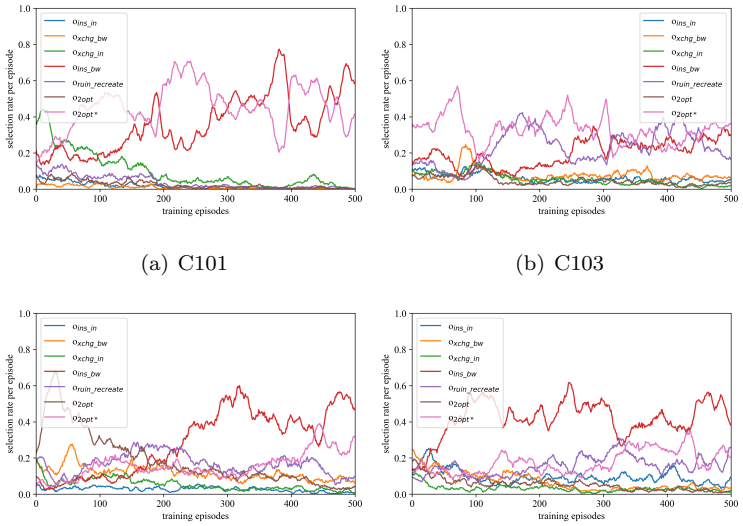
Table 7: Performance of the algorithms with different features during testing

Instance	Vehicle	Density of the	Best-known solutions	Both feature sets				Search-dependent features only			
	capacity	time windows	in the literature	AVG	SD	BEST	GAP	AVG	SD	BEST	GAP
C101	200	100%	10828.94 (Rochat & Taillard, 1995)	**10828.94**	**0**	**10828.94**	**0**	**10828.94**	1.8E-12	**10828.94**	**0**
C102	200	75%	10828.94 (Rochat & Taillard, 1995)	**10829.40**	**0.93**	**10828.94**	**0**	10829.93	2.011	**10828.94**	**0**
C103	200	50%	10828.06 (Rochat & Taillard, 1995)	**10856.97**	**12.07**	**10837.15**	**0.08%**	10858.79	17.80	10839.29	0.1%
C104	200	25%	10824.78 (Rochat & Taillard, 1995)	**10916.01**	20.64	**10887.99**	**0.58%**	10921.54	16.28	10896.9	0.67%
C201	700	100%	3591.56 (Rochat & Taillard, 1995)	**3591.56**	**4.55E-13**	**3591.56**	**0**	**3591.56**	**4.55E-13**	**3591.56**	**0**
C202	700	75%	3591.56 (Rochat & Taillard, 1995)	**3591.56**	4.55E-13	**3591.56**	**0**	**3591.56**	**2.4E-10**	**3591.56**	**0**
C203	700	50%	3591.17 (Rochat & Taillard, 1995)	**3592.67**	**4.48**	**3591.17**	**0**	3592.90	3.66	**3591.17**	**0**
C204	700	25%	3590.6 (Rochat & Taillard, 1995)	3613.71	9.35	3599.58	0.25%	**3613.05**	**8.95**	**3598.93**	**0.23%**
R101	200	100%	20645.79 (Homberger, 2000)	**20658.64**	**3.60**	**20653.64**	**0.04%**	20659.79	3.85	20654.3	0.04%
R102	200	75%	18486.12 (Rochat & Taillard, 1995)	**18519.28**	**18.02**	**18499.30**	**0.07%**	18621.61	293.55	18511.18	0.14%
R103	200	50%	14292.68 (Li & Lim, 2003)	**15048.02**	416.20	**14365.01**	**0.51%**	15256.21	**11.75**	15228.24	6.5%
R104	200	25%	10007.24 (Mester et al., 2007)	**11090.22**	23.09	**11051.22**	**10.43%**	11097.74	**15.93**	11063.67	10.56%
R201	1000	100%	5252.37 (Homberger & Gehring, 1999)	**5320.51**	**13.56**	5290.00	0.72%	5320.84	23.52	**5288.58**	**0.69%**
R202	1000	75%	4191.7 (Rousseau et al., 2002)	**5003.77**	**341.68**	**4307.68**	**2.77%**	5163.65	11.86	5141.74	22.66%
R203	1000	50%	3939.54 (Woch & Łebkowski, 2009)	4036.05	**10.23**	4016.02	1.94%	**4035.76**	16.50	**4010.36**	**1.8%**
R204	1000	25%	2825.52 (Bent & Van Hentenryck, 2004)	**3732.08**	270.78	**2920.27**	**3.35%**	3828.59	**5.09**	3822.51	35.29%
RC101	200	100%	15696.94 (Taillard et al., 1997)	**16839.69**	**300.13**	16703.71	6.41%	16896.18	396.64	**16677.99**	**6.25%**
RC102	200	75%	13554.75 (Taillard et al., 1997)	**15431.22**	**294.17**	**14550.96**	**7.35%**	15545.00	13.85	15528.14	14.56%
RC103	200	50%	12261.67 (Shaw, 1998)	**13152.01**	294.17	**12356.41**	**0.77%**	**13359.17**	15.04	13329.82	8.71%
RC104	200	25%	12135.487 (Cordeau et al., 2001)	12233.65	388.17	**11248.41**	**1.01%**	12131.09	281.68	11287.67	1.37%
RC201	1000	100%	5406.91 (Mester et al., 2007)	**5492.96**	18.48	**5455.76**	**0.90%**	5505.30	12.14	5482.79	1.40%
RC202	1000	75%	4367.09 (Czech & Czarnas, 2002)	5287.89	29.74	**5226.54**	**19.68%**	**5261.77**	**12.14**	5242.94	20.06%
RC203	1000	50%	4049.62 (Czech & Czarnas, 2002)	4191.46	22.89	4165.96	2.87%	4180.07	18.61	**4137.23**	**2.16%**
RC204	1000	25%	3798.41 (Mester et al., 2007)	**3877.75**	**17.78**	**3851.04**	**1.39%**	3879.58	**9.79**	3863.21	1.71%

4.3. Search Pattern Analysis of the Best Designed Algorithm

4.3.1. Utilisation of Algorithm Components

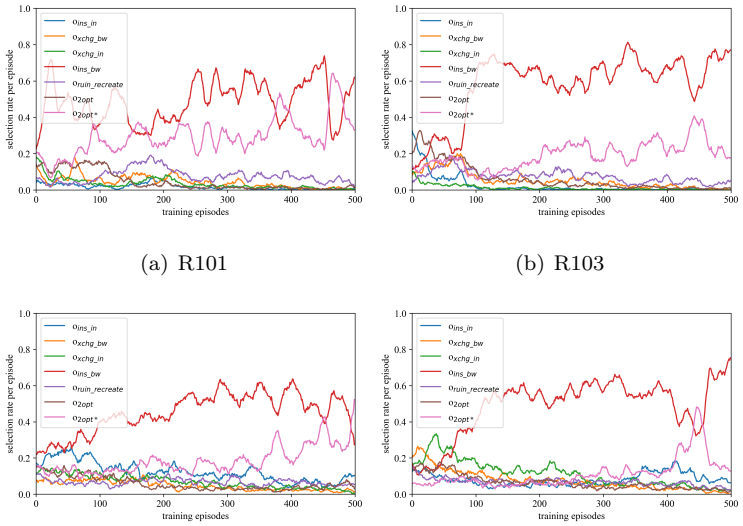
Figures 6-8 show the proportion of each operator called in the PPO model with search-dependent and instance-dependent features during the training process, while Figures 9-11 show the utilisation of operators in the PPO models with only search-dependent features. Both PPO models identify *ins_bw* and 2*opt*∗ as the most frequently selected operators in the best designed algorithms. *ins_bw* is selected most often by both PPO models, although this phenomenon is more obvious in the type-R and type-RC instances. Although the operators with a high frequency of combination are quite similar in both PPO models, the specific utilisation rates of each operator during each episode are different, indicating that the algorithm compositions obtained by these two PPO models (i.e, one with only search-dependent features and one with both search-dependent and instance-dependent features) are different.



(a) C101 (b) C103

(c) C201 (d) C203

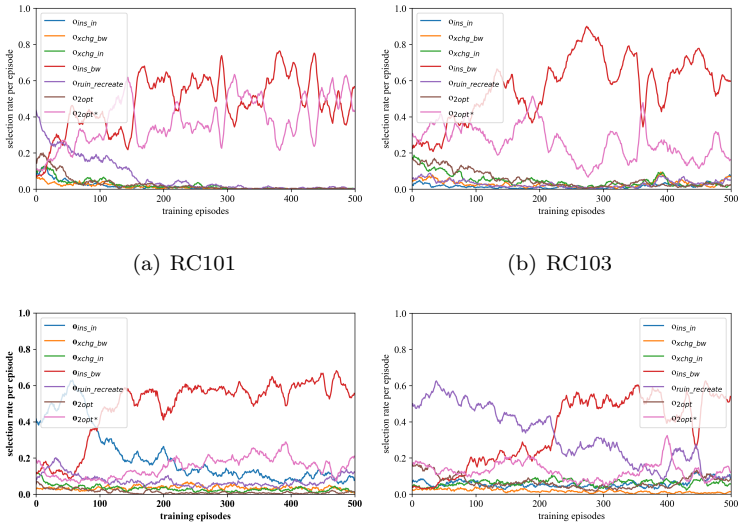
Figure 6: Utilisation of operators during training (type-C, with search-dependent and instance-dependent features)



(a) R101 (b) R103

(c) R201 (d) R203

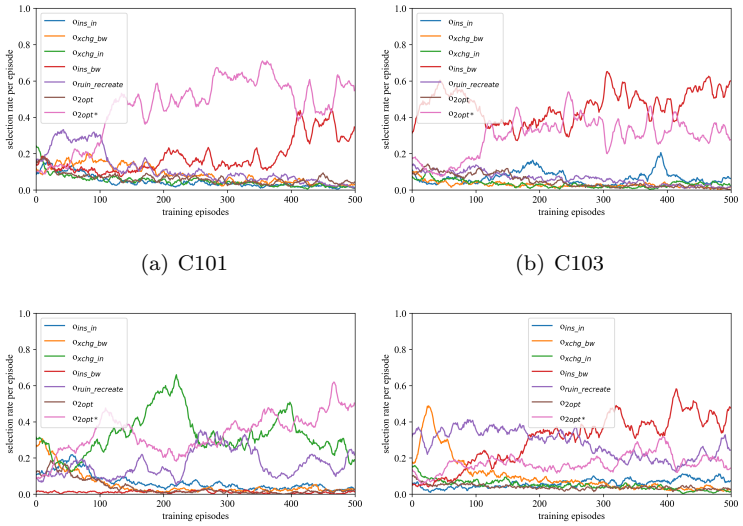
Figure 7: Utilisation of operators during training (type-R, with search-dependent and instance-dependent features)



(a) RC101 (b) RC103

(c) RC201 (d) RC203

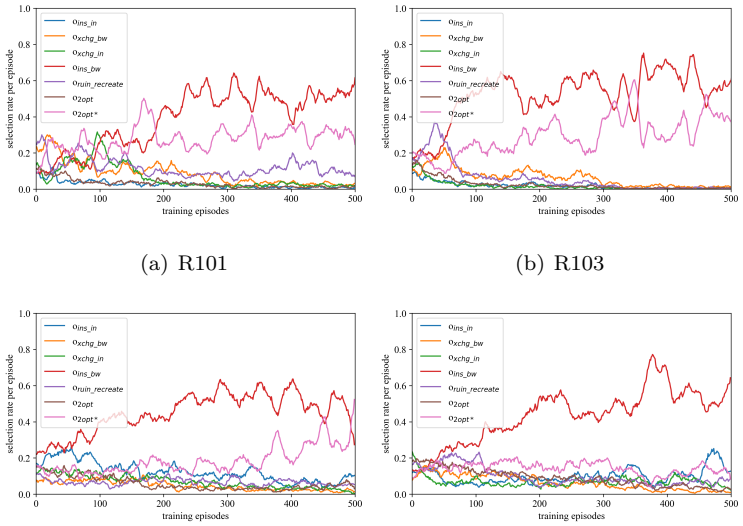
Figure 8: Utilisation of operators during training (type-RC, with search-dependent and instance-dependent features)



(a) C101 (b) C103

(c) C201 (d) C203

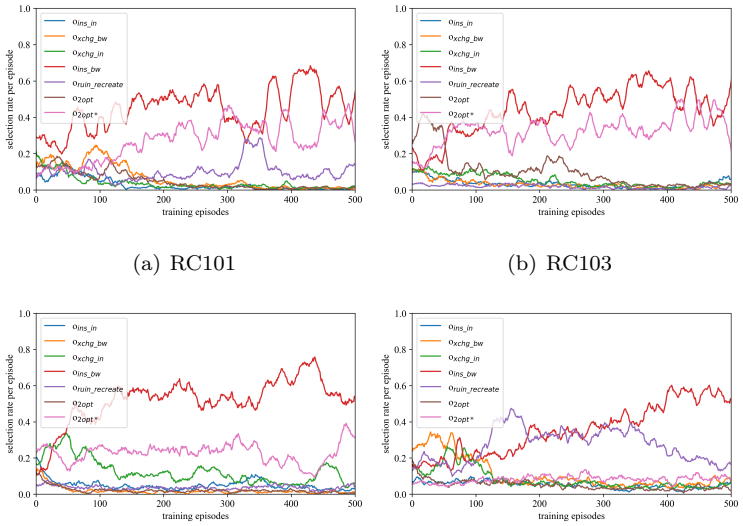
Figure 9: Utilisation of operators during training (type-C, with only search-dependent features)



(a) R101 (b) R103

(c) R201 (d) R203

Figure 10: Utilisation of operators during training (type-R,with only search-dependent features)



(a) RC101 (b) RC103

(c) RC201 (d) RC203

Figure 11: Utilisation of operators during training (type-RC, with only search-dependent features)

4.3.2. Transition of Algorithmic Components

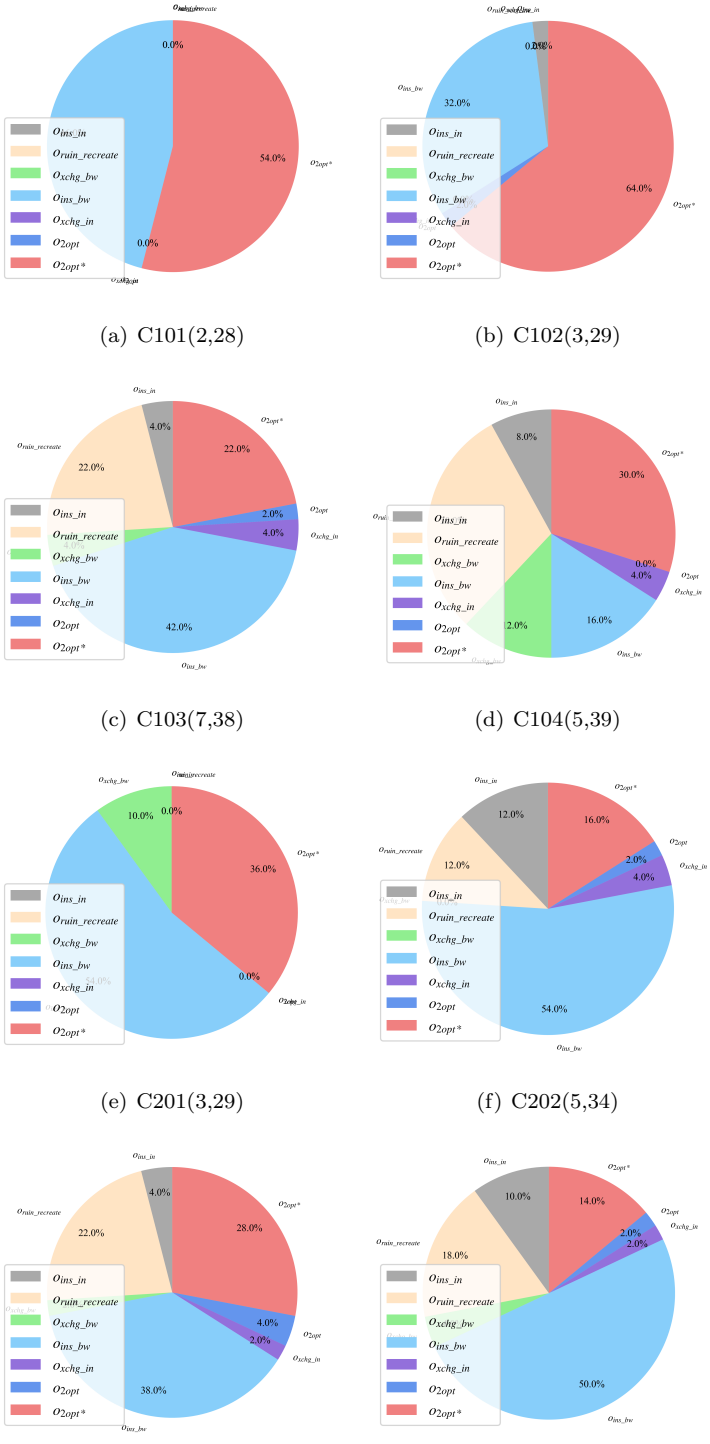
An analysis of the best designed algorithms that are automatically designed by the PPO models with different feature sets is analysed in this section. Figures 12-14 show the transition pattern of operators in the best designed algorithm compositions obtained by the PPO model with both feature sets, including the number of operators, the proportion of each operator and the number of transitions between operators during 50 timesteps. For example, for C101 instance in Figure 12, only two out of seven operators are called, namely 54.0% for the $2opt*$ operator and 46% for the ins_bw operator with the corresponding number of transitions between operators (28). The transition patterns of the best designed algorithm obtained by the PPO model with only search-dependent features are shown in Appendix.

As can be seen from all the figures, the diversity of the operators in the best designed algorithm compositions (i.e. the number of operators) increases when the capacity of the vehicle increases. Taking type-C instances as examples, Figure 12 shows that the number of called operators in the best designed algorithm compositions (obtained by the PPO model with two feature sets) for solving the type-C2 instances with a larger vehicle capacity (i.e. 700), is larger than that of the type-C1 instances with a smaller vehicle capacity (i.e. 200). The same phenomenon can be observed in type-R and type-RC instances, details of which are shown in Figure 13 and Figure 14. Similar conclusions can be reached regarding the diversity of operators in the best designed algorithm compositions obtained by the PPO model with only search-dependent features, as shown in figures in Appendix.

There is a negative correlation between the diversity of operators and the density of the time-window (i.e. the percentage of time-constrained customers), as shown in Figures 12-14. Taking type-RC instances as examples, Figure 14 shows that with the decrease of time window density, dropping from 100% in RC101 to 25% in RC104, the diversity of operators increases, rising from 3 to 6. The same phenomenon can be observed in the other types of instances as well.

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

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



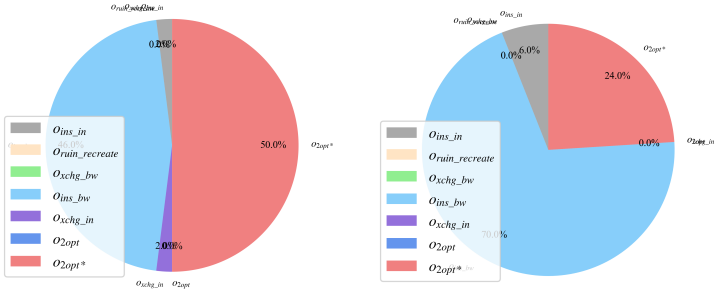
(a) C101(2,28) (b) C102(3,29)

(c) C103(7,38) (d) C104(5,39)

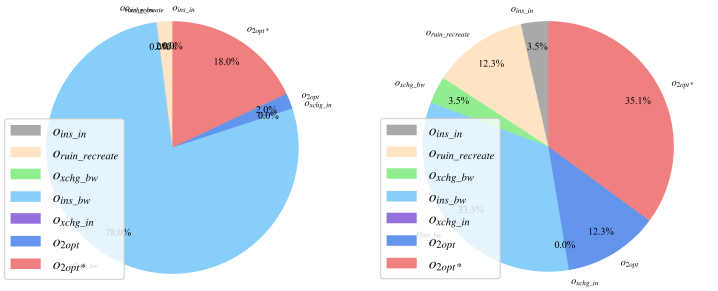
(e) C201(3,29) (f) C202(5,34)

(g) C203(7,38) (h) C204(6,37)

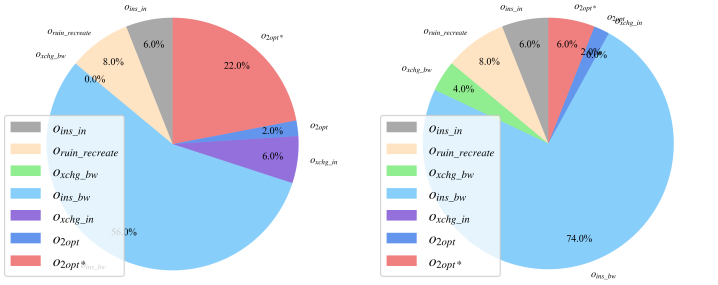
Figure 12: Transition of operators in the best designed algorithm (type-C, with search-dependent and instance-dependent features)



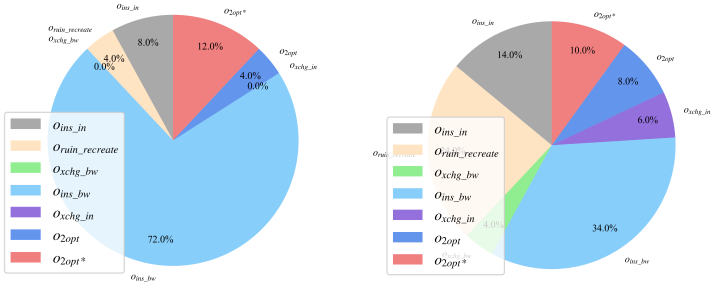
(a) R101(4,28) (b) R102(3,25)



(c) R103(3,21) (d) R104(5,31)

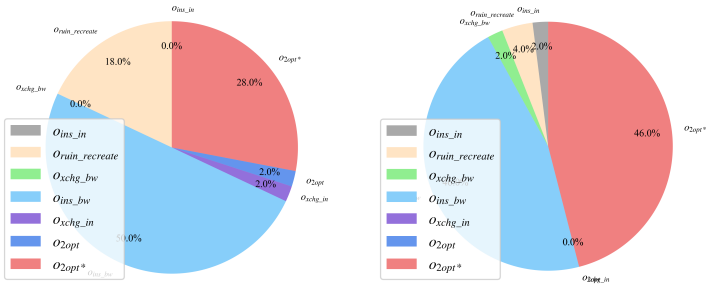


(e) R201(7,35) (f) R202(6,22)

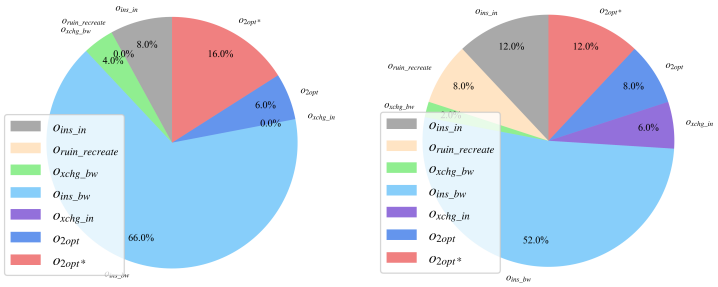


(g) R203(5,18) (h) R204(7,36)

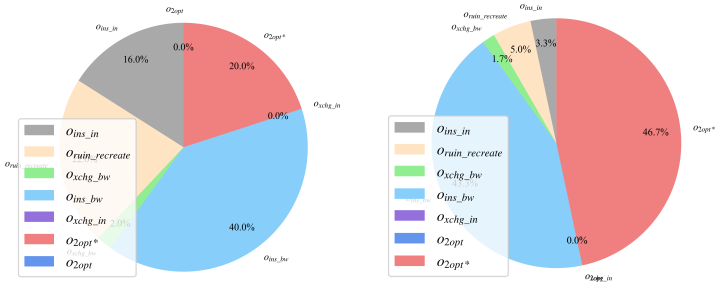
Figure 13: Transition of operators in the best designed algorithm (type-R, with search-dependent and instance-dependent features)



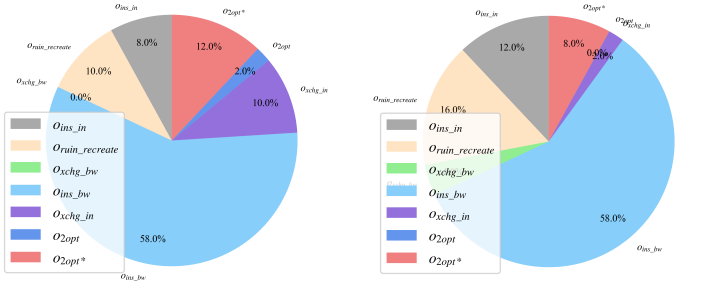
(a) RC101(3,21) (b) RC102(4,26)



(c) RC103(5,24) (d) RC104(6,37)



(e) RC201(5,33) (f) RC202(5,29)



(g) RC203(6,32) (h) RC204(6,35)

Figure 14: Transition of operators in the best designed algorithm (type-RC, with search-dependent and instance-dependent features)

5. Conclusion

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

With controlled experiments on the state representation, the impact of the identified features on the reinforcement learning model is verified on the benchmark instances of the capacitated vehicle routing problem with time windows. The results show that both search-dependent and instance-dependent features can provide useful information to the learning process by assisting the population to accurately detect the resulting state with better action choice.

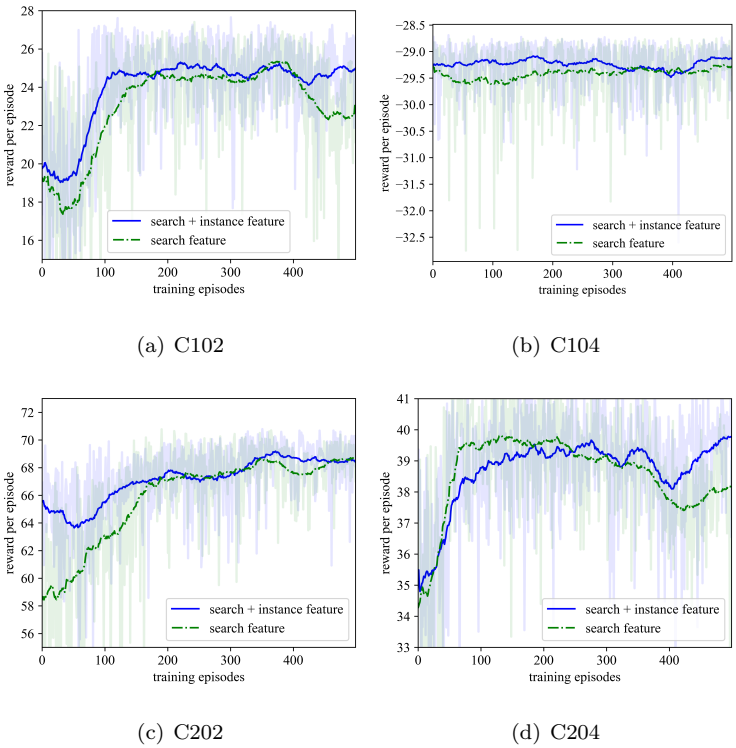
Regarding the search pattern of the best designed algorithms, utilisation and transition of evolution operators are analysed. The analysis shows that two reinforcement learning models with different features identify ins_bw and $2opt*$ as the most frequently selected components. Different operators are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

For future work, the proposed feature sets and reinforcement learning models can be applied to automate the process of algorithm design by making other decisions, including selecting evolution heuristics and replacement heuristics. It would be interesting to further improve the reinforcement learning models on the extended whole algorithm design space with enhanced scheme to the loss function to encourage exploration of the learning agent.

6. Acknowledgement

This research has been funded by the School of Computer Science, University of Nottingham, UK.

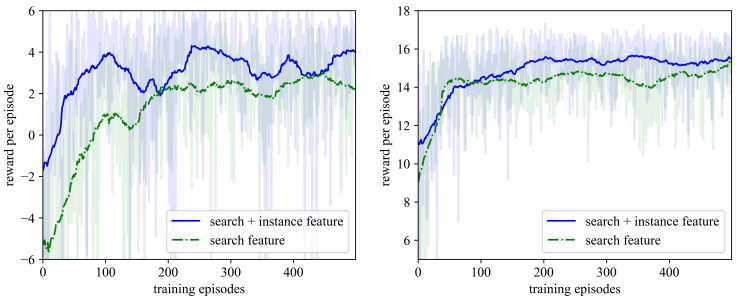
7. Appendix



(a) C102 (b) C104

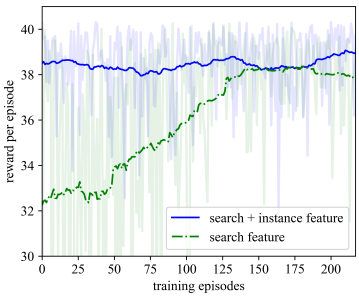
(c) C202 (d) C204

Figure 15: Influence of different feature sets on the learning model during training (type-C)

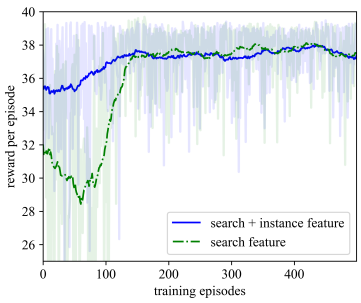


(a) R102

(b) R104

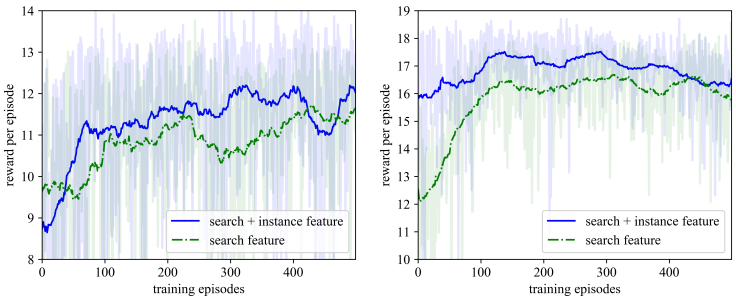


(c) R202

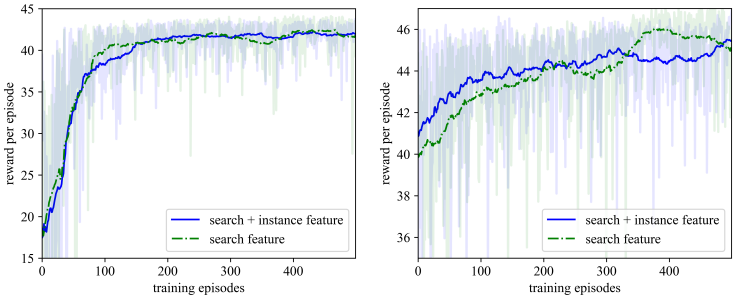


(d) R204

Figure 16: Influence of different feature sets on the learning model during training (type-R)

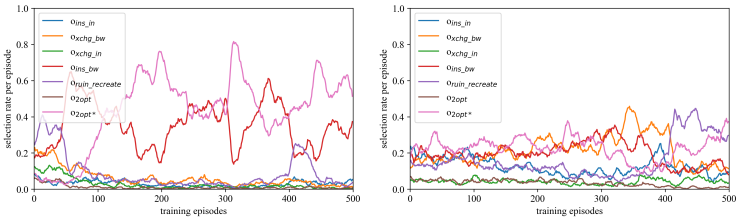


(a) RC102 (b) RC104

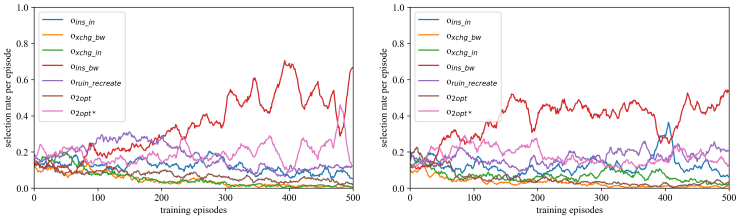


(c) RC202 (d) RC204

Figure 17: Influence of different feature sets on the learning model during training (type-RC)

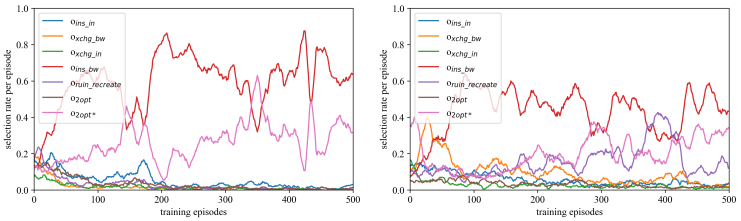


(a) C102 (b) C104

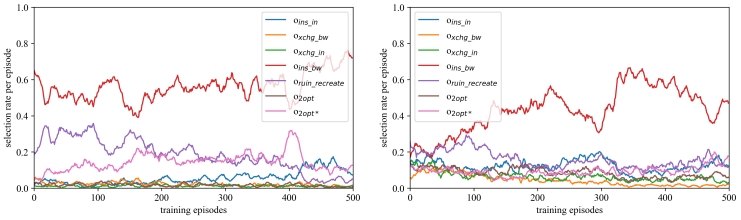


(c) C202 (d) C204

Figure 18: Utilisation of operators during training (type-C, with search-dependent and instance-dependent features)

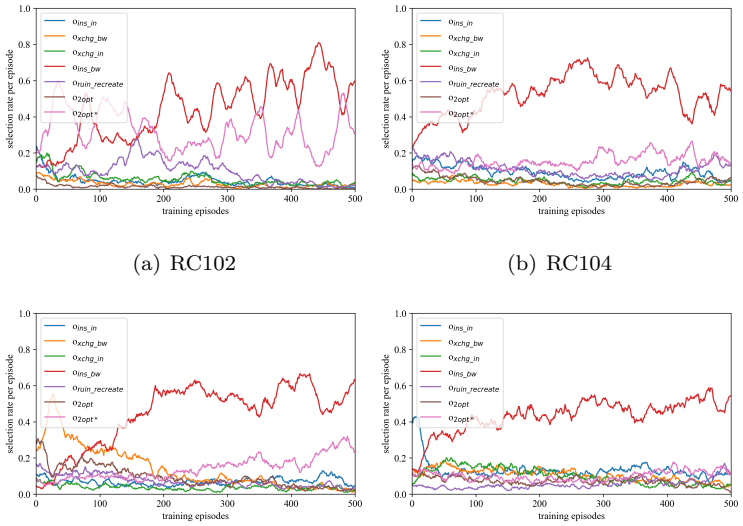


(a) R102 (b) R104



(c) R202 (d) R204

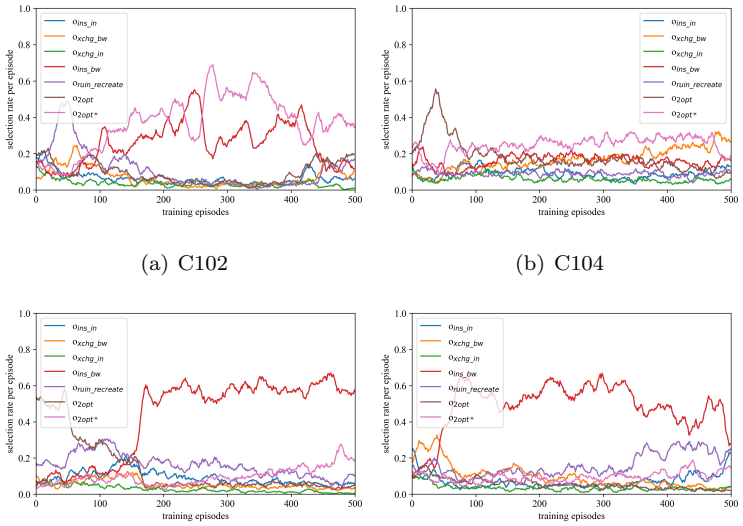
Figure 19: Utilisation of operators during training (type-R, with search-dependent and instance-dependent features)



(a) RC102 (b) RC104

(c) RC202 (d) RC204

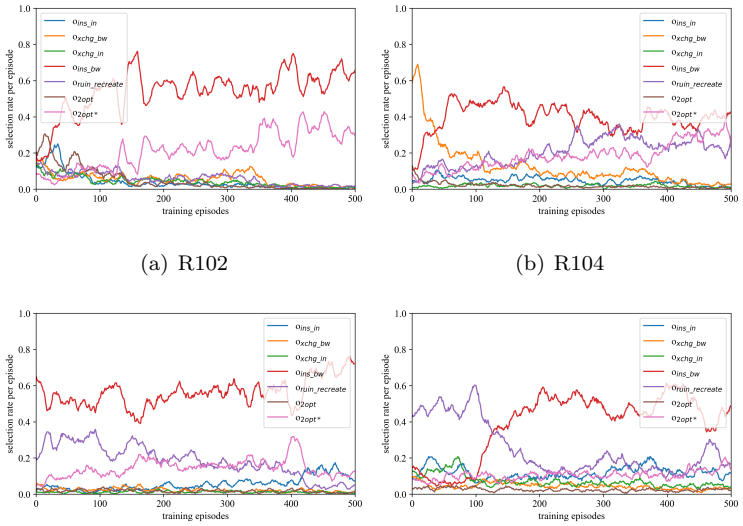
Figure 20: Utilisation of operators during training (type-RC, with search-dependent and instance-dependent features)



(a) C102 (b) C104

(c) C202 (d) C204

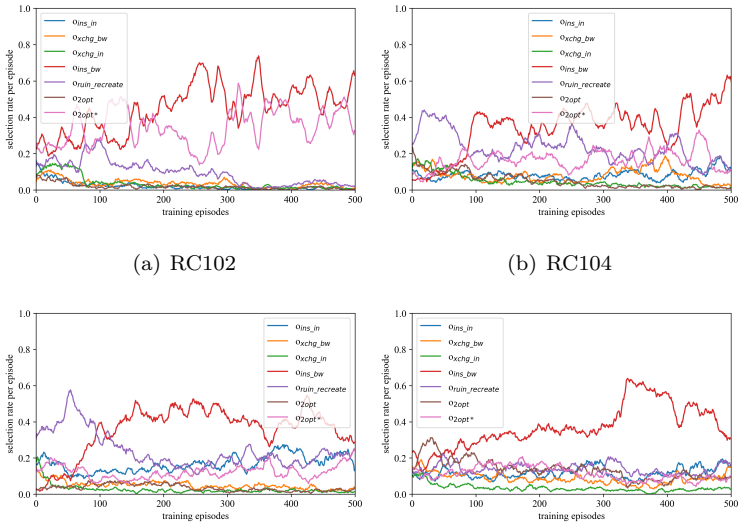
Figure 21: Utilisation of operators during training (type-C, with only search-dependent features)



(a) R102 (b) R104

(c) R202 (d) R204

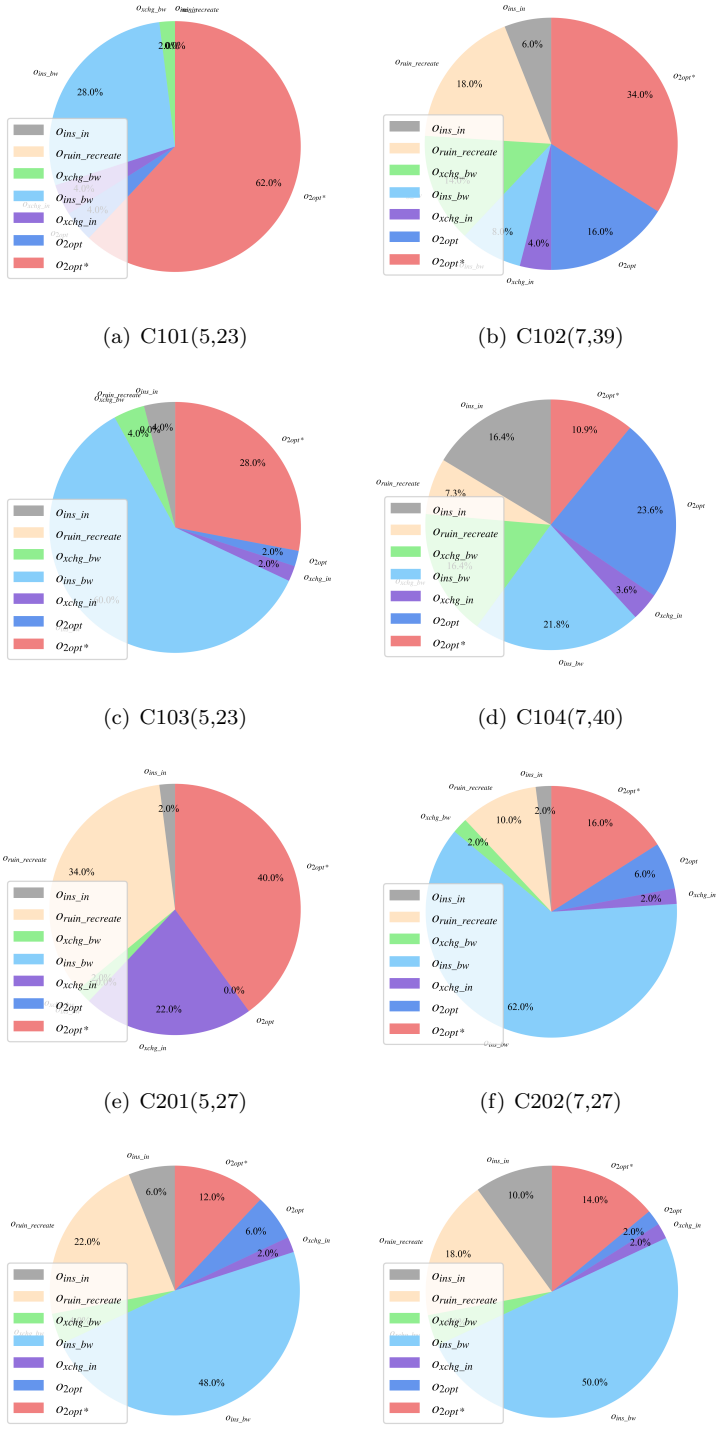
Figure 22: Utilisation of operators during training (type-R,with only search-dependent features)



(a) RC102 (b) RC104

(c) RC202 (d) RC204

Figure 23: Utilisation of operators during training (type-RC, with only search-dependent features)



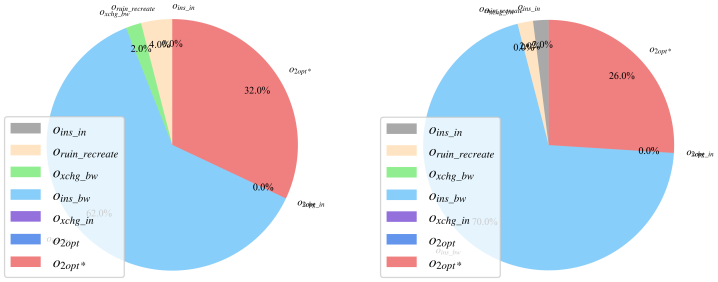
(a) C101(5,23) (b) C102(7,39)

(c) C103(5,23) (d) C104(7,40)

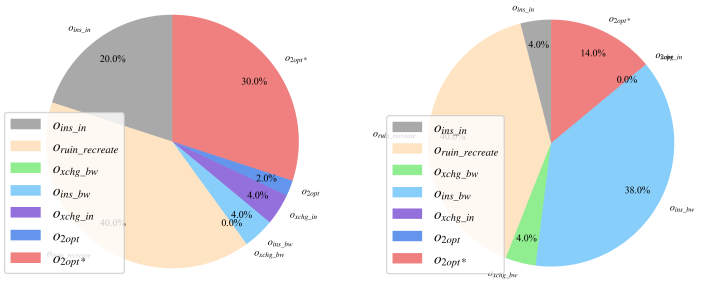
(e) C201(5,27) (f) C202(7,27)

(g) C203(7,39) (h) C204(6,35)

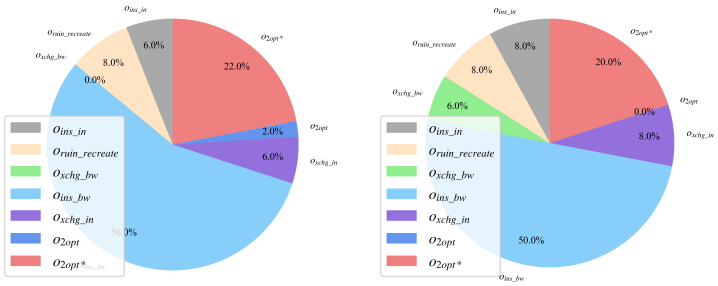
Figure 24: Transition of operators in the best designed algorithm (type-C, with only search-dependent features)



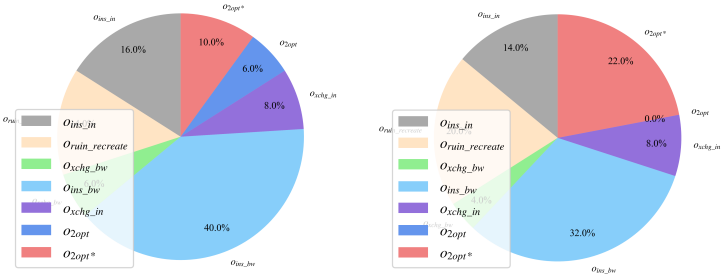
(a) R101(4,24) (b) R102(3,20)



(c) R103(6,33) (d) R104(5,32)

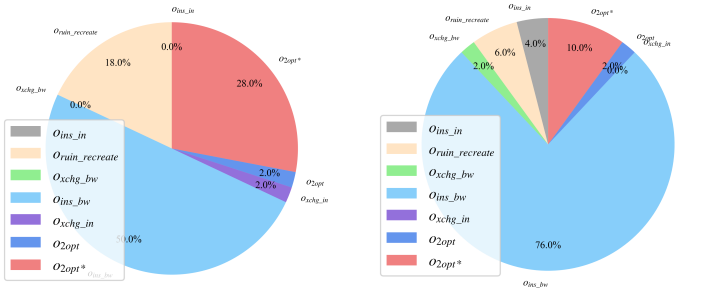


(e) R201(5,29) (f) R202(6,32)



(g) R203(7,38) (h) R204(6,35)

Figure 25: Transition of operators in the best designed algorithm (type-R, with only search-dependent features)



(a) RC101(4,31) (b) RC102(5,20)

(c) RC103(7,25) (d) RC104(5,25)

(e) RC201(4,28) (f) RC202(6,36)

(g) RC203(6,36) (h) RC204(6,44)

Figure 26: Transition of operators in the best designed algorithm (type-RC, with only search-dependent features)

References

Bent, R., & Van Hentenryck, P. (2004). A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, *38*, 515–530.

Choong, S. S., Wong, L.-P., & Lim, C. P. (2018). Automatic design of hyper-heuristic based on reinforcement learning. *Information Sciences*, *436*, 89–107.

Cordeau, J.-F., Laporte, G., & Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, *52*, 928–936.

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

Dantas, A., Rego, A. F. d., & Pozo, A. (2021). Using deep q-network for selection hyper-heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (pp. 1488–1492).

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

Eiben, A., Horvath, M., Kowalczyk, W., & Schut, M. C. (2006). Reinforcement learning for online control of evolutionary algorithms. In *International Workshop on Engineering Self-Organising Applications* (pp. 151–160). Springer.

Gutierrez-Rodríguez, A. E., Conant-Pablos, S. E., Ortiz-Bayliss, J. C., & Terashima-Marín, H. (2019). Selecting meta-heuristics for solving vehicle routing problems with time windows via meta-learning. *Expert Systems with Applications*, *118*, 470–481.

Homberger, J. (2000). Eine verteilt-parallele metaheuristik. In *Verteilt-parallele Metaheuristiken zur Tourenplanung* (pp. 139–165). Springer.

Homberger, J., & Gehring, H. (1999). Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR: Information Systems and Operational Research*, *37*, 297–318.

Jiang, H., Wang, Y., Tian, Y., Zhang, X., & Xiao, J. (2021). Feature construction for meta-heuristic algorithm recommendation of capacitated vehicle routing problems. *ACM Transactions on Evolutionary Learning and Optimization*, *1*, 1–28.

Li, H., & Lim, A. (2003). Local search with annealing-like restarts to solve the vrptw. *European journal of operational research*, *150*, 115–127.

Meng, W., & Qu, R. (2021). Automated design of search algorithms: Learning on algorithmic components. *Expert Systems with Applications*, *185*, 115493.

Mester, D., Bräysy, O., & Dullaert, W. (2007). A multi-parametric evolution strategies algorithm for vehicle routing problems. *Expert Systems with Applications*, *32*, 508–517.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al. (2015). Human-level control through deep reinforcement learning. *nature*, *518*, 529–533.

Qu, R., Kendall, G., & Pillay, N. (2020). The general combinatorial optimization problem: Towards automated algorithm design. *IEEE Computational Intelligence Magazine*, *15*, 14–23.

Rochat, Y., & Taillard, É. D. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics*, *1*, 147–167.

Rousseau, L.-M., Gendreau, M., & Pesant, G. (2002). Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of heuristics*, *8*, 43–58.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, .

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming* (pp. 417–431). Springer.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, *35*, 254–265.

Taillard, É., Badeau, P., Gendreau, M., Guertin, F., & Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation science*, *31*, 170–186.

Tian, Y., Li, X., Ma, H., Zhang, X., Tan, K. C., & Jin, Y. (2022). Deep reinforcement learning based adaptive operator selection for evolutionary multi-objective optimization. *IEEE Transactions on Emerging Topics in Computational Intelligence*, .

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

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*, 279–292.

Wauters, T., Verbeeck, K., Causmaecker, P. D., & Berghe, G. V. (2013). Boosting metaheuristic search using reinforcement learning. In *Hybrid metaheuristics* (pp. 433–452). Springer.

Woch, M., & Łebkowski, P. (2009). Sequential simulated annealing for the vehicle routing problem with time windows. *Decision Making in Manufacturing and Services*, *3*, 87–100.

Yi, W., Qu, R., Jiao, L., & Niu, B. (2022). Automated design of metaheuristics using reinforcement learning within a novel general search framework. *IEEE*

Transactions on Evolutionary Computation, (pp. 1–1). doi:10.1109/TEVC.2022.3197298.

Zhang, Y., Bai, R., Qu, R., Tu, C., & Jin, J. (2022). A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties. *European Journal of Operational Research*, *300*, 418–427.