Automated Design of Search Algorithms: Learning on Algorithmic Components

Weiyao Meng[a,*], Rong Qu[a]

[a]*School of Computer Science, University of Nottingham, Nottingham, UK*

Abstract

This paper proposes AutoGCOP, a new general framework for automated design of local search algorithms. In a recently established General Combinatorial Optimisation Problem (GCOP) model, the problem of algorithm design itself is defined as a combinatorial optimisation problem. AutoGCOP defines a general framework to optimise the composition of elementary algorithmic components as decision variables in GCOP. By modelling various well-known local search meta-heuristics within a general framework, AutoGCOP supports automatic design of new novel algorithms which may be highly different from those manually designed in the literature.

Within the consistent AutoGCOP framework, various elementary algorithmic components are analysed for solving the benchmark vehicle routing problem with time window constraints and different characteristics. Furthermore, two learning models based on reinforcement learning and Markov chain are investigated to learn and enhance the compositions of algorithmic components towards the automated design of search algorithms. The Markov

*Corresponding author

Email addresses: `weiyao.meng@nottingham.ac.uk` (Weiyao Meng), `rong.qu@nottingham.ac.uk` (Rong Qu)

chain model presents superior performance learning the compositions of algorithmic components during the search, demonstrating its effectiveness for automatically designing new algorithms.

Keywords: Automated algorithm design, search algorithms, reinforcement learning, Markov chain

1. Introduction

In solving complex combinatorial optimisation problems, the demand for effective algorithms presents a challenge and burden for human experts, where a large number of decisions need to be made in the algorithm design process (Pillay et al., 2018). Performance of manually designed algorithms highly relies on the experience and effort of the human experts, who may only consider a limited number of designs, leaving a significant number of potential algorithms unexplored (Hoos, 2008). Automation in algorithm design helps to release human experts from the tedious design process and explore a larger scope of candidate algorithms, some of which may never be considered by manual designs.

Research developments in automated algorithm design are fast emerging along with the successful research findings in evolutionary computation. Based on the decisions of the algorithm design space and different aims, a new taxonomy has been defined in (Qu et al., 2020), categorising the current research in automated algorithm design into three themes, namely automated configuration, automated selection and automated composition as follows:

- Automated configuration: aims to automatically determine values for algorithmic parameters of specific target algorithm(s) to solve a collection of problem instances (Hutter et al., 2007).

- Automated selection: aims to automatically select the most appropriate algorithm from a portfolio of candidate algorithms for a set of training instances thus to solve new testing instances.

- Automated composition: aims to automatically compose or combine heuristics or components of arbitrary algorithms to solve the problem at hand online (Qu et al., 2020).

In the theme of research in automated composition, the algorithm design process is automated by composing algorithmic components. It takes a bottom-top method to work flexibly with a set of algorithmic components, thus to generate new algorithms (Qu et al., 2020). The other two themes of research in automated configuration and selection take a top-down method, to consider parameters and algorithms themselves in the decision space. In line with automated algorithm composition, this paper proposes a general framework to support automated algorithm design based on a new model established in (Qu et al., 2020), where basic elementary algorithmic components are defined as decision variables in a combinatorial optimisation problem of algorithm design.

Hyper-heuristics (Pillay & Qu, 2018) can be seen as one of the main streams in automated composition of novel algorithms. Within a two-level framework, hyper-heuristics determine "at a higher abstraction level which low-level heuristics to apply" (Cowling et al., 2000). The low-level heuristics, e.g. algorithms or operators, are called to generate heuristic algorithms on the fly. One type of hyper-heuristics, selection hyper-heuristics, automatically combines low-level heuristics by iterative selection. Various learning models have been applied at a high level to adaptively select the low-level heuristics. Reinforcement learning aims to learn the performance of individual low-level heuristics, e.g. simple reinforcement learning schemes (Nareyek, 2003), (Burke et al., 2003) and (Özcan et al., 2012) and complex reinforcement learning models that strictly follow the criteria of reinforcement learning (Khamassi et al., 2011), (Di Gaspero & Urli, 2011) and(Di Gaspero & Urli, 2012); other models observe the transition performance between each pair of low-level heuristics, e.g. MCHH (McClymont & Keedwell, 2011) with Markov chain (Kemeny & Snell, 1976) and SSHH (Kheiri & Keedwell, 2015) with hidden Markov models (Baum & Petrie, 1966). Choice function (Cowling et al., 2000) can also be seen as a learning model awarding both individual and the transition of low-level heuristics which perform well during the iterative selections. Another type of hyper-heuristics, generation hyper-heuristics, can be seen as to automate the composition process using mainly genetic programming (Banzhaf et al., 1998). Common features of a specific type of target heuristic(s) are extracted into a terminal set. The context-free grammar defines how to combine the elements in the terminal set into novel algorithm designs. The mostly studied methods in generation hyper-heuristics include dispatching rules for job scheduling problem (Pickardt et al., 2010), (Nguyen et al., 2012), local search algorithms for bin packing (Burke et al., 2011) and satisfiability testing (Fukunaga, 2008), particle swarm optimisation algorithms for continuous optimisation problems (Miranda et al., 2017), and evolutionary algorithms for function optimisation, traveling salesman problems and the Quadratic Assignment Problem (Oltean, 2005).

Another line of research in the automated composition is based on the automated combination of algorithmic components or building blocks within frameworks of specific algorithms. Various search algorithms in the literature can be instantiated within the frameworks, and novel algorithm designs can be composed automatically. The most studied algorithms include SAT solver (KhudaBukhsh et al., 2016), simulated annealing algorithms (Franzin & Stützle, 2019), iterated greedy algorithms (Mascia et al., 2013), stochastic local search algorithms (Pagnozzi & Stützle, 2019) and multi-objective evolutionary algorithms (Bezerra et al., 2015) for permutation flow shop problems, and multi-objective ant colony algorithms (Lopez-Ibanez & Stutzle, 2012) for traveling salesman problems.

In (Qu et al., 2020), a new model named General Combinatorial Optimisation Problem (GCOP) is introduced to define the problem of algorithm design itself as a combinatorial optimisation problem. The decision variables of GCOP consist of elementary algorithmic components. With the GCOP model, the design of various algorithms can be defined as flexible compositions of basic components. The objective of GCOP is to optimise the composition of these components. Solving the GCOP thus automates the design of the best algorithms for solving the problem at hand. GCOP provides a standard to support automated algorithm design (Qu et al., 2020) by formulating various search algorithms in one model.

The newly established GCOP model requires coherent frameworks to assess the performance of the elementary algorithmic components and explore the insights on designing effective algorithm compositions with these components. Existing frameworks in the automated composition concern only a subset of algorithmic components in GCOP, thus cannot provide sufficient support for automated algorithm design based on GCOP.

Based on the GCOP model, this paper presents a new general AutoGCOP framework to automatically compose elementary algorithmic components, thus to support the automated design of local search algorithms and systematic investigations on automated composition. Various algorithmic procedures in the literature can be modelled and encapsulated as general procedures within AutoGCOP. These general algorithmic procedures operate flexibly upon the elementary algorithmic components in GCOP, leading to novel local search algorithms which may not be designed manually. In other words, various local search algorithms can be automatically composed with basic components within the general AutoGCOP framework.

Within the consistent AutoGCOP framework, this paper investigates the scope of algorithm performance with these elementary algorithmic components. This is a base case to conduct further investigations on effective algorithm compositions. With the elementary components in the GCOP model, it can be observed that the performance of the composed new algorithms is satisfying, confirming the effectiveness of the most basic components in local search algorithms.

In addition, this paper investigates the learning of effective composition of the basic algorithmic components for automated algorithm design within the AutoGCOP framework. Two learning models have been studied based on probabilistic reasoning on the behaviour of algorithmic components during the search, comparing the effectiveness of two different learning perspectives.

In particular, the Markov chain based learning on the transition between pairs of components is shown to be effective composing new algorithms automatically.

The contributions of the paper are threefold. First, it presents a new general AutoGCOP framework to automatically compose elementary algorithmic components, thus to support automated design of local search algorithms. Second, within the consistent AutoGCOP framework, this paper confirms the satisfying performance of the elementary algorithmic components for the vehicle routing problems with time window constraints (VRPTW). Third, this paper evaluates reinforcement learning and Markov chain as a means of learning to compose algorithmic components, investigating the effectiveness of learning the individual performance of algorithmic components and learning the transition performance of algorithmic components. Results within the general AutoGCOP framework confirm the superior performance of the Markov chain model (which observes transition performance) to automate compositions of new local search algorithms, thus suggesting the benefits of learning the transitions between algorithmic components.

Based on the GCOP model, this study investigates automated algorithm composition within the proposed general AutoGCOP framework, using the VRPTW as the domain example. As a widely investigated optimization problem in operational research (Wong, 1983), the basic vehicle routing problem (VRP) (Fisher & Fisher, 1995) consists of ordering and assigning customer delivery demands to a set of vehicles. The objective is to minimise the total travel costs serving all the customers. Variants of VRP have been investigated with complex constraints (Braekers et al., 2016) to address different real-world scenarios. In the most widely studied VRPTW variant, customers must be served within specified time intervals (Cordeau et al., 2007). Most search algorithms in the literature adopt the weighted sum objective function (Bräysy & Gendreau, 2005b) or the hierarchical objective function, where the number of vehicles (routes) is minimised as the primary objective, followed by minimising the total travel distance or travel time as the secondary objective (Bräysy & Gendreau, 2005a).

In the rest of the paper, Section 2 presents the proposed general AutoGCOP framework for automated algorithm composition. Section 3 describes the proposed GCOP methods with learning models within AutoGCOP. Section 4 presents the experimental studies addressing the concerned research issues, followed by conclusions in Section 5.

2. The AutoGCOP Framework for Automated Algorithm Composition

The AutoGCOP is a new general framework to automatically compose elementary algorithmic components in the extended GCOP model, thus to support the automated design of local search algorithms. Within AutoGCOP, algorithmic procedures are encapsulated as general procedures, allowing various local search algorithms in the literature to be instantiated and novel search algorithms composed automatically. Section 2.1 describes the extended GCOP model. Section 2.2 presents the AutoGCOP framework and the instantiation of local search algorithms from AutoGCOP. The differences between AutoGCOP and existing frameworks in the automated composition are discussed in Section 2.3.

2.1. An Overview of the Extended GCOP Model

In the novel GCOP model (Qu et al., 2020), various search algorithms are broken into a finite set A of elementary algorithmic components $a \in A$. These a serve as the domain of decision variables in GCOP, defining algorithm design itself as a combinatorial optimisation problem. The solution space of GCOP consists of algorithmic composition c upon a. Each c represents a new algorithm for solving optimisation problems p, i.e. a solution s for p is obtained by a corresponding algorithmic composition c, $c \rightarrow s$. The objective of GCOP is to search for the optimal c^* which produces the optimal s^* for p, i.e. $c^* \rightarrow s^*$. With the optimisation process for solving GCOP, compositions c of a, i.e. design of various search algorithms, can be obtained automatically for solving p.

In GCOP, there are two categories of algorithmic components $a \in A_{1.0}$, i.e. operators $o_i \in A_{1.0\_o}$, and acceptance criteria $a_j \in A_{1.0\_a}$, each with their associated heuristic and parametric settings (Qu et al., 2020). The operators o_i modify values of the decision variables in s_1 to generate a new solution s_2 in the search space of p. The acceptance criteria a_j determine if s_2 is accepted in the search.

In building the AutoGCOP framework in Section 2.2, this paper extends the elementary algorithmic components $a \in A_{1.0}$ in the GCOP model with termination criteria in local search algorithms. Based on the widely investigated meta-heuristics (Blum & Roli, 2003) in the literature, various termination criteria have been modelled and added as basic procedure algorithmic components in the extended GCOP model, i.e. $t_k \in A_t$ as shown in Table 1.

With the extended algorithmic component set in the GCOP model, the

Table 1: The algorithmic components $t_k \in A_t$ in the extended GCOP model.

A_t	t_k with parameters h, n. h: measure of convergence; n: number of iterations, CPU time or threshold.
$t_{construct}$	Terminate when a complete solution is constructed
$t_{converge}(h)$	Terminate upon the convergence h
$t_{iteration}(n)$	Terminate as the number of iterations reaches n
$t_{time}(n)$	Terminate when the elapsed CPU time reaches n
$t_{threshold}(r, q, n)$	Terminate when r increased by q reaches the threshold n

AutoGCOP general framework is built in Section 2.2 to support automated algorithm composition. Note that both the acceptance criteria a_j and termination criteria t_k have been built to model elements across different search algorithms into general algorithmic components, and can be used in designing any local search algorithms for any problem p.

2.2. The AutoGCOP Framework with Extended GCOP Model

Based on the extended GCOP model, the AutoGCOP framework as shown in Algorithm 1 is proposed to automatically design local search algorithms by composing the basic algorithmic components $o_i \in A_{1.0\_o}$, $a_j \in A_{1.0\_a}$ and $t_k \in A_t$. The underlying idea in building the AutoGCOP framework is to model various local search meta-heuristics by encapsulating their common procedures (operations upon solutions s) as the most basic processes in search algorithms. In particular, the following three most basic processes have been modelled in local search algorithms.

- *Select*(A): select a basic component $o_i \in A_{1.0\_o}$, $a_j \in A_{1.0\_a}$ or $t_k \in A_t$.
- *ApplyOperator*(o_i, s): return a new solution by applying an operator o_i to solution s.

- $ApplyAcceptance(a_j, s_{new}, s)$: return solution s_{new} if it is accepted by an acceptance criterion a_j; otherwise, return solution s.

As shown in Algorithm 1, AutoGCOP consists of the Construction procedure and the Improvement procedure. These basic procedures compose the corresponding elementary algorithmic components in the extended GCOP model, i.e. decision variables in GCOP. The Construction procedure constructs a complete solution s for the optimisation problem p by composing the corresponding component sets $t_k \in A_{t_{construct}}$ and $o_i \in A_{o_{construct}}$. The Improvement procedure improves s as an initial solution searching for the best possible solution s_{best}, thus automates the composition of the corresponding component sets $t_k \in A_{t_{improve}} \cup A_{t_{inner}}$, $o_i \in A_{o_{improve}}$ and $a_j \in A_a$.

With the general AutoGCOP framework, various local search algorithms in the literature (Blum & Roli, 2003) can be defined in a unified template by composing specific algorithmic components in the Improvement procedure as shown in Table 2. In other words, these meta-heuristics can be seen as specific GCOP solutions composed *manually* by selecting the specific algorithmic components within AutoGCOP.

With the general AutoGCOP framework, a large number of new and unseen local search algorithms can be designed *automatically* by searching for solutions for GCOP, i.e. compositions c of $o_i \in A_{1.0\_o}$, $a_j \in A_{1.0\_a}$ and $t_k \in A_t$. Different techniques and algorithms can be developed within this general framework to compose algorithmic components from the respective sets in the GCOP model. This can be seen as to automate the process of human experts hand-picking algorithmic components during algorithm design.

Algorithm 1 : The general AutoGCOP framework

Input: p: an optimisation problem,

A_t: a set of termination criteria t_k, including a subset for Construction procedure $A_{t_{construct}}$, a subset for Improvement procedure $A_{t_{improve}}$ and a subset for inner loops of the Improvement procedure $A_{t_{inner}}$.

A_o: a set of operators o_i, including a subset for the Construction procedure $A_{o_{construct}}$ and a subset for the Improvement procedure $A_{o_{improve}}$,

A_a: a set of acceptance criteria a_j,

Output: s_{best}: the best-recorded solution,

```
 1: procedure CONSTRUCTION
 2:     $s \leftarrow$ An empty solution for $p$;
 3:     $t_{k_{con}} \leftarrow Select(A_{t_{construct}})$;
 4:     while $t_{k_{con}}$ is not met do
 5:         $o_i \leftarrow Select(A_{o_{construct}})$;
 6:         $s \leftarrow ApplyOperator(o_i, s)$;
 7:     end while
 8: end procedure
 9:
10: procedure IMPROVEMENT
11:     $t_{kmain} \leftarrow Select(A_{t_{improve}})$;
12:     while $t_{k_{main}}$ is not met do
13:         $t_{k_{inner}} \leftarrow Select(A_{t_{inner}})$;
14:         while $t_{k_{inner}}$ is not met do
15:             $o_i \leftarrow Select(A_{o_{improve}})$;
16:             $a_j \leftarrow Select(A_a)$;
17:             $s_{new} \leftarrow ApplyOperator(o_i, s)$;
18:             $s \leftarrow ApplyAcceptance(a_j, s_{new}, s)$;
19:             $s_{best} \leftarrow$ Update the best-recorded solution;
20:         end while
21:     end while
22: end procedure
```

Table 2: Instantiation of widely used local search metaheuristics in the literature using different elementary algorithmic components in the Improvement procedure within the unified AutoGCOP framework.

Local search algorithms	Termination criteria, operators and acceptance criteria used in Algorithm 1 ($x \leftarrow y$ denotes use y as x)
Tabu search	$t_{k_{main}} \leftarrow t_{converge}(h)$ in line 11, $t_{k_{inner}} \leftarrow t_{iteration}(1)$ in line 13, $o_i \leftarrow$Specific o_i from $A_{o_{improve}}$ in line 15, $a_j \leftarrow a_{tabu}$ in line 17.
Simulated annealing	$t_{k_{main}} \leftarrow t_{converge}(h)$ in line 11, $t_{k_{inner}} \leftarrow t_{iteration}(1)$ in line 13, $o_i \leftarrow$Specific o_i from $A_{o_{improve}}$ in line 15, $a_j \leftarrow a_{sa}$ in line 17.
Iterated local search	$t_{k_{main}} \leftarrow t_{converge}(h)$ in line 11, line 13-20 repeat with different $t_{k_{inner}}$, o_i and a_j as follows: firstly, $t_{k_{inner}} \leftarrow t_{iteration}(1)$ in line 13, $o_i \leftarrow$Specific o_i from $A_{o_{improve}}$ in line 15, $a_j \leftarrow$ *none* in line 17; secondly, $t_{k_{inner}} \leftarrow t_{converge}(h)$ in line 13, $o_i \leftarrow$Specific o_i from $A_{o_{improve}}$ in line 15, $a_j \leftarrow a_{oi}$ in line 17; thirdly, $t_{k_{inner}} \leftarrow t_{iteration}(1)$ in line 13, $o_i \leftarrow$ *none* in line 15, $a_j \leftarrow a_{oi}$ in line 17.
Variable neighborhood search	$t_{k_{main}} \leftarrow t_{converge}(h)$ in line 11, line 13-20 repeat with different $t_{k_{inner}}$, o_i and a_j as follows: firstly, $t_{k_{inner}} \leftarrow t_{iteration}(1)$ in line 13, $o_i \leftarrow$Specific o_i from $A_{o_{improve}}$ based on a certain order in line 15, $a_j \leftarrow$ *none* in line 17; secondly, $t_{k_{inner}} \leftarrow t_{converge}(h)$ in line 13, $o_i \leftarrow$Specific o_i from $A_{o_{improve}}$ based on a certain order in line 15, $a_j \leftarrow a_{oi}$ in line 17; thirdly, $t_{k_{inner}} \leftarrow t_{iteration}(1)$ in line 13, $o_i \leftarrow$ *none* in line 15, $a_j \leftarrow a_{oi}$ in line 17.

2.3. Differences between AutoGCOP and existing frameworks

Generally, the AutoGCOP framework is built on the extended GCOP model, supporting flexible exploration in algorithmic compositions of elementary algorithmic components. Existing frameworks in the automated composition concern only a subset of algorithmic components in GCOP, providing limited scope for automated algorithm design.

The framework that most closely resembles the AutoGCOP framework is the selection hyper-heuristics (SHHs) (Pillay & Qu, 2018), which can be seen as automatically design of search algorithms by freely composing a set of low-level heuristics chosen by human experts. However, the generality of SHH is limited when compared to AutoGCOP in terms of two aspects as follows:

- The algorithmic components. The SHH selects pre-defined problem-specific low-level heuristics rather than elementary algorithmic components (i.e. basic operators, acceptance criteria and termination criteria). The low-level heuristics can be seen as compound components combining and accumulating the basic components in GCOP. The resulting algorithms of SHH are therefore only a subset of algorithms which can be composed of basic algorithmic components within AutoGCOP.

- The components to manage algorithmic components. The SHH framework usually applies a selection strategy to manage low-level heuristics (Özcan et al., 2008) and uses a pre-defined acceptance criteria. Therefore, the SHH framework is insufficient to manage different types of basic components in GCOP, i.e. basic operators, acceptance criteria and termination criteria.

These above issues not only limit the number of local search algorithms that can be composed with SHH but also involve more human decisions while selecting and configuring the low-level heuristics.

3. Learning in AutoGCOP

With the proposed new AutoGCOP framework, this work investigates different GCOP methods which optimise the elementary algorithmic components $o_i \in A_o$, $a_j \in A_a$ and $t_k \in A_t$ as shown in Section 3.1 to automatically design new algorithms. The investigations in particular focus on 1) the role of learning in composing basic components; and 2) the behaviour (i.e. performance) of basic components. AutoGCOP provides a unified common template to support such investigations. The widely studied VRPTW is tested to demonstrate the effectiveness of the GCOP methods.

With AutoGCOP, two learning models, namely Individual Performance (IP) learning and Transition Performance (TP) learning in Section 3.2, have been investigated. Based on probabilistic reasoning, they learn to compose $o_i \in A_{o_{improve}}$ intelligently, i.e. learning-based methods $Select(A_{o_{improve}})$ (line 15, Algorithm 1). IP learning focuses on the performance of each o_i using a simple probability matrix based on the concept of reinforcement learning. TP learning focuses on the transition between pairs of components. It is based on Markov chain which applies a transition probability matrix, where each o_i is a state.The difference between IP and TP is that TP conducts a more detailed learning, where the performance of a specific component can be seen as the sum of the performance of other possible components transferred to this component.

In comparison, two simple strategies with $Select(A_{o_{improve}})$ are tested as the baseline GCOP methods to demonstrate the effectiveness of the learning models, including a random strategy (RN) which chooses o_i with equal probability and a random gradient strategy (RG) which chooses a random o_i and continues to apply it as long as it is successful. More specifically, the simple random strategy does not attempt to learn from the behaviour of o_i, while the random gradient strategy can be considered as using a reinforcement learning mechanism with the shortest memory length possible to exploit the currently selected o_i as long as it is successful (Lissovoi et al., 2020).

3.1. Component Sets

To explore the insights on designing effective algorithm compositions with elementary algorithmic components, this research investigates a subset of the most basic components operators $o_i \in A_{1.0\_o}$ and acceptance criteria $a_j \in A_{1.0\_a}$ in the GCOP model (Qu et al., 2020) and a subset of termination criteria $t_k \in A_t$ in Table 1, as shown in Table 3, within the AutoGCOP framework. The focus is on the behaviour of operators $o_i \in A_{o_{improve}}$ (line 15, Algorithm 1), with specific components fixed in other procedures.

Table 3: The component sets considered in the AutoGCOP framework, i.e. termination criteria $t_k \in A_t$, operators $o_i \in A_o$, and acceptance criteria $a_j \in A_a$.

Component set A_t	Termination criteria t_k in A_t
$A_{t_{construct}}$	$t_{construct}$: terminate when a complete candidate solution is constructed.
$A_{t_{improve}}$	$t_{iteration}(n)$: terminate as the number of iterations reaches n.
$A_{t_{inner}}$	$t_{iteration}(1)$: terminate after conducting one iteration.
Component set A_o	Operators o_i in A_o with parameters as defined in GCOP (Qu et al., 2020) h_1: heuristics to choose the customer with the highest proximity to the most recently inserted customer based on distance and time (Walker et al., 2012). h_2: heuristics to choose the next position of the most recently inserted customer (Walker et al., 2012). h_3: random strategy.
$A_{o_{construct}}$	$o_{ins}(1, h_1, h_2)$: insert one customer chosen by h_1 to the position selected by h_2.
$A_{o_{improve}}$	$o^{in}_{xchg}(1, 1, h_3)$: swap two customers chosen by h_3. Selected customers are within one route. $o^{bw}_{xchg}(1, 1, h_3)$: swap two customers chosen by h_3. Selected customers are from different routes. $o^{in}_{ins}(1, h_3, h_3)$: insert one customer chosen by h_3 to other position selected by h_3 within the same route. $o^{bw}_{ins}(1, h_3, h_3)$: insert one customer chosen by h_3 to other position selected by h_3 in a different route. $o_{rr}(10, h_3, h_3)$: remove 10% customers chosen by h_3, and re-assign them using h_3. 2-*opt**: swap the end sections of two routes to generate two new routes (Burke et al., 2010).
Component set A_a	Acceptance criteria a_j in A_a
A_a	a_{naive}: accept all improvements; worse solutions are accepted with a probability of 0.5 (Burke et al., 2010).

Among the most basic $o_i \in A_{o_{improve}}$ adopted in the *Improvement* procedure, 2-*opt** is a problem-specific compound operator designed manually in the literature, which showed to be especially effective for VRPTW (Potvin & Rousseau, 1995). We therefore investigate the performance of o_i grouped into two sets as follows in the experiments:

- $O_{basic} = \{o^{in}_{xchg}, o^{bw}_{xchg}, o^{in}_{ins}, o^{bw}_{ins}, o_{rr}\}$;
- $O_{vrp-basic} = O_{basic} \cup \{2\text{-}opt^*\}$.

3.2. Learning models

The GCOP methods with the proposed learning models in $Select(A_{o_{improve}})$ are named as the IP-GCOP method learning individual $o_i \in A_{o_{improve}}$, and the TP-GCOP method learning the transition between o_i. The purpose of the learning models is to observe the behaviour of $o_i \in A_{o_{improve}}$ in Table 3, thus to predict their performance and choose the most appropriate without human involvement to solve the problem adaptively.

Based on the general AutoGCOP framework in Algorithm 1, the GCOP methods only adds the learning model M for selecting $o_i \in A_{o_{improve}}$ (line 15, Algorithm 1) and the method to update the learning model $Update()$ after updating s_{best} (line 19, Algorithm 1).

3.2.1. Learning model in IP-GCOP

A reinforcement learning method interacts with the environment by trial and error and takes actions given a state based on a policy, aiming to accumulate reward relating to its goal (Sutton et al., 1998). The IP-GCOP method follows a simple reinforcement learning scheme, i.e., a simple reward and penalty scheme, to learn the individual performance of elementary algorithmic components by updating the reward and penalty of each component through sequences of actions (i.e., selection of operators) to adapt to the scenarios of the search environment. A probability matrix is used as a fundamental model to record the individual performance (i.e., the reward and penalty) of elementary algorithmic components, supporting a reinforcement scheme to update the reward and penalty of each component based on its performance during the search. The promising algorithmic components can be selected and applied based on the probability matrix during the search.

The IP-GCOP method uses a simple $2 \times n$ probability matrix M_{IP}, to record the accumulated performance of each individual o_i ($i = 1, ..., n$, $n = |A_{o_{improve}}|$) when a better solution than the current best is found. The two rows record the reward and penalty of each o_i, respectively.

At each iteration of the Improvement procedure, an $o_i \in A_{o_{improve}}$ is selected using M_{IP}. With the learning models, o_i with better-accumulated performance in M are chosen for the next iteration using the roulette wheel selection in the proposed GCOP methods. The accumulated individual performance of each o_i is calculated based on the likelihood (L) of each o_i achieving improvement in the next iteration:

$$L_i = \frac{M_{IP}[1, i]}{M_{IP}[1, i] + M_{IP}[2, i]} \tag{1}$$

The IP-GCOP method uses a roulette wheel selection strategy to select the next o_l with a probability $P_{IP}(l)$ in proportion to $L(i)$:

$$P_{IP}(l) = \frac{L_l}{\sum_{k=1}^{n} L_k} \tag{2}$$

At the end of each iteration, the learning model M_{IP} is updated using

$Update()$ based on the performance of the selected $o_i \in A_{o_{improve}}$ depending on if o_i leads to a new best solution s_{best} during the search. In M_{IP}, the o_i is rewarded by increasing its corresponding value in the first row, i.e. $M_{IP}[1, i]$. Otherwise, o_i is punished by increasing its corresponding value in the second row, i.e. $M_{IP}[2, i]$.

3.2.2. Learning model in TP-GCOP

A Markov chain is a statistical model describing a sequence of states with certain probabilities to transfer between each other (Kemeny & Snell, 1976). A transition probability matrix describes the transition probabilities between states.

The TP-GCOP method is based on the Markov chain combining with a simple reinforcement scheme. The TP-GCOP method uses a Markov chain model to represent transitions between elementary algorithmic components and a transition matrix to record the transition probabilities of pairs of algorithmic components statistically. The transition matrix supports a simple reinforcement scheme to learn the transition performance between algorithmic components and update the transition probabilities in the matrix, thus support the selection of the promising algorithmic components during the search.

In the TP-GCOP method, a $n \times n$ transition probability matrix M_{TP} is built based on the concept of Markov chain (Kemeny & Snell, 1976), regarding each $o_i \in A_{o_{improve}}$ as a state. The values in M_{TP} record the performance of one o_i transferring to another, learning the transition performance of pairs of o_i and o_l which contributes to a new best solution. Given the current o_i, the TP-GCOP uses a roulette wheel selection strategy to select the next o_l

with a probability $P_{TP}(l)$ in proportion to $M_{TP}\left[i,l\right]$:

$$P_{TP}(l) = \frac{M_{TP}\left[i,l\right]}{\sum_{k=1}^{n} M_{TP}\left[i,k\right]} \tag{3}$$

In M_{TP}, at the end of each iteration, a transition from o_i to o_l leading to a new better s_{best} is rewarded by increasing the corresponding value of $M_{TP}[i,\, l]$.

3.2.3. Update mechanisms for learning models

The update strategy on M is shown to be an important factor in learning algorithm design in the proposed GCOP methods. In this work, a set of $Update()$ methods as shown in Table 4 is tested to analyse the effectiveness of a set of factors that may influence the performance of o_i during the optimisation search.

Table 4: A set of update strategies to be tested in the proposed GCOP methods, i.e. $Update()$ for updating the learning models M.

$Update()$	Strategies to update the corresponding value in M
$Simple()$	By 1
$Linear()$	By the index of the current iteration
$Improve()$	By the amount of improvement / deterioration in the current iteration
$NoImprove()$	By the number of iterations since s_{best} has not been updated
$NoCall()$	For each o_i, by the number of iterations since o_i has been last called

3.2.4. An Illustrative Example

Given three operators (denoted by o_1,o_2,o_3), assume a search process with six iterations have been conducted within the AutoGCOP framework in Algorithm 1. Each o_i is determined by $Select(A_{o_{improve}})$ in Algorithm 1 with M. $Simple()$ in Table 4 is adopted in $Update()$ to update M. Initially, the values in M are all set to 1, thus each o_i is chosen with an equal probability.

With M_{IP}, assume o_2 in the first iteration generates a better solution s_1, thus $M_{IP}[1,\ 2]$ for o_2 is increased by 1. In the next iteration, o_2 is more likely to be selected applying roulette wheel on M_{IP}. Assume o_1 is selected, generating a non-improving solution s_2, so $M_{IP}[2,\ 1]$ for o_1 is increased. This is repeated in the following iterations, where o_i with better-accumulated performance as recorded in M_{IP} for finding new best solutions is more likely to be selected. Other operators, however, have the potential to be selected as well but with a smaller probability. Figure 1 presents how M_{IP} is updated during six iterations. With M_{IP}, in the seventh iteration, o_2 with a higher probability in M_{IP} is more likely to be selected.

With M_{TP}, the learning starts from the second iteration. After o_2, assume o_1 is selected using M_{TP}, generating a non-improving solution s_2, so there is no reward to update $M_{TP}[2,\ 1]$, i.e. the transition from o_2 to o_1. Using roulette wheel, each o_i has the same probability to be chosen after o_1. Assume o_3 is selected leading to a better solution s_3. $M_{TP}[1,\ 3]$ is thus increased by 1 to reward the transition from o_1 to o_3. In the following iterations, assume $M_{TP}[3,\ 2]$ and $M_{TP}[3,\ 1]$ are updated to reward the transitions from o_3 to o_2 and o_3 to o_1 after selecting o_2, o_3 and o_1. Figure 2 presents how M_{TP} is updated during six iterations. Checking the element $M_{TP}[1,\ 3]$ suggests o_3 has a higher probability to be selected in the next iteration.

Figure 1: The M_{IP} updated during six iterations

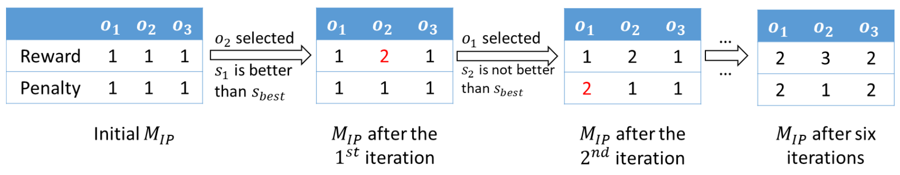
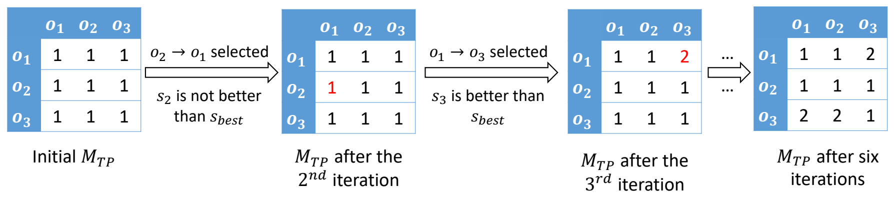

o_1 o_2 o_3
Reward 1 1 1
Penalty 1 1 1
Initial M_{IP}
o_2 selected
s_1 is better than s_{best}
1 2 1
1 1 1
M_{IP} after the 1st iteration
o_1 selected
s_2 is not better than s_{best}
1 2 1
2 1 1
M_{IP} after the 2nd iteration
...
2 3 2
2 1 2
M_{IP} after six iterations

Figure 2: The M_{TP} updated during six iterations


o_1 o_2 o_3
o_1 1 1 1
o_2 1 1 1
o_3 1 1 1
Initial M_{TP}
$o_2 \rightarrow o_1$ selected
s_2 is not better than s_{best}
o_1 1 1 1
o_2 1 1 1
o_3 1 1 1
M_{TP} after the 2nd iteration
$o_1 \rightarrow o_3$ selected
s_3 is better than s_{best}
o_1 1 1 2
o_2 1 1 1
o_3 1 1 1
M_{TP} after the 3rd iteration
...
o_1 1 1 2
o_2 1 1 1
o_3 2 2 1
M_{TP} after six iterations

4. Experimental Studies

The experimental investigations aim to address the two research issues, 1) assessing the performance of basic o_i in the AutoGCOP framework in Section 4.1, and 2) analysing the automated composition of o_i in the proposed GCOP methods using the learning models in Section 4.2. In evaluating the proposed learning models, the influence of different $Update()$ methods is also analysed. Section 4.3 compares the results of the proposed GCOP methods with the published best results by the state-of-the-art methods.

The VRPTW concerned in this work considers the dual objectives of minimising the number of vehicles (NV) and minimising the total travel distance (TD). A weighted sum objective function is adopted from the literature to evaluate VRPTW solutions s as shown in Equation (4), where c is set to 1000 empirically (Walker et al., 2012).

$$f(s) = c \times NV + TD \tag{4}$$

The investigations are conducted on two sets of the widely studied benchmark VRPTW, i.e. the Solomon 100 customers set (Solomon, 1987) and the Homberger 1000 customer set (Gehring & Homberger, 1999) as shown in Table 5, covering different instance characteristics. In particular, customers in type-R instances are randomly distributed geographically. In type-C instances, customers are distributed in clusters. RC type instances are a mix of them.

Table 5: Characteristics of the benchmark VRPTW instances.

Benchmark	Name	Size	Vehicle	Capacity	Type
Solomon	R101	100	25	200	R
Solomon	R201	100	25	1000	R
Solomon	C101	100	25	200	C
Solomon	C206	100	25	700	C
Solomon	RC103	100	25	200	RC
Solomon	RC207	100	25	1000	RC
Homberger	R1-10-1	1000	250	200	R
Homberger	R2-10-6	1000	250	1000	R
Homberger	C1-10-8	1000	250	200	C
Homberger	C2-10-1	1000	250	700	C
Homberger	RC1-10-5	1000	250	200	RC
Homberger	RC2-10-1	1000	250	1000	RC

4.1. Performance of the Basic Components

To assess the performance of composing the basic algorithmic components o_i in the AutoGCOP framework, the algorithm performance of each o_i in $O_{vrp-basic} \subseteq A_{o_{improve}}$ (as defined in Section 3.1) is compared with a random

GCOP method (i.e. RN-GCOP) with $O_{vrp-basic}$ in Section 4.1.1. The same number of evaluations is set as the stopping condition, i.e. $t_{iteration}(n)$ is adopted as t_{kmain} in Algorithm 1, for all methods.

The performance of the elementary algorithmic components $O_{basic} \subseteq A_{o_{improve}}$ is then compared against the basic problem specific compound algorithmic components $O_{vrp-basic} \subseteq A_{o_{improve}}$. The performance of RN-GCOP with O_{basic} is compared against the same method with $O_{vrp-basic}$, which includes a compound operator 2-*opt** based on the solution quality in Section 4.1.2 and the algorithm convergence in Section 4.1.3. The computation time for 100 customer instances and 1000 customer instances are set to 2 CPU hours and 4 CPU hours, respectively.

4.1.1. Performance evaluation on composing algorithmic components

Figure 3 and Figure 4 show the difference in solution value of each operator in the operator set $O_{vrp-basic}$ against the random RN-GCOP method with $O_{vrp-basic}$ for instances with 100 and 1000 customers, respectively. Comparing with the algorithm performance with each of the operators, RN-GCOP achieves better overall performance. Only for instance RC103 and C1-10-8, o^{bw}_{ins} obtains better results than other methods. The problem-specific compound operator 2-*opt** achieves better performance for instance C2-10-1. The algorithm performance of different operators varies according to problem instances. Among the operators in $O_{vrp-basic}$, the performance of o^{bw}_{ins}, o_{rr} and 2-*opt** (denoted as $o3$, $o4$ and $o5$, respectively) are relatively better than others.

To further investigate whether the worse-performing operators o^{in}_{xchg}, o^{bw}_{xchg} and o^{in}_{ins} (denoted by $o0$, $o1$ and $o2$) are useful for all problem instances,

Figure 3: Comparison in the average solution objective value (out of ten runs) between each operator in $O_{vrp-basic}$ against RN-GCOP with $O_{vrp-basic}$ on the 100-customer instances. $o0$: o^{in}_{xchg}. $o1$: o^{bw}_{xchg}. $o2$: o^{in}_{ins}. $o3$: o^{bw}_{ins}. $o4$: o_{rr}. $o5$: 2-*opt**.

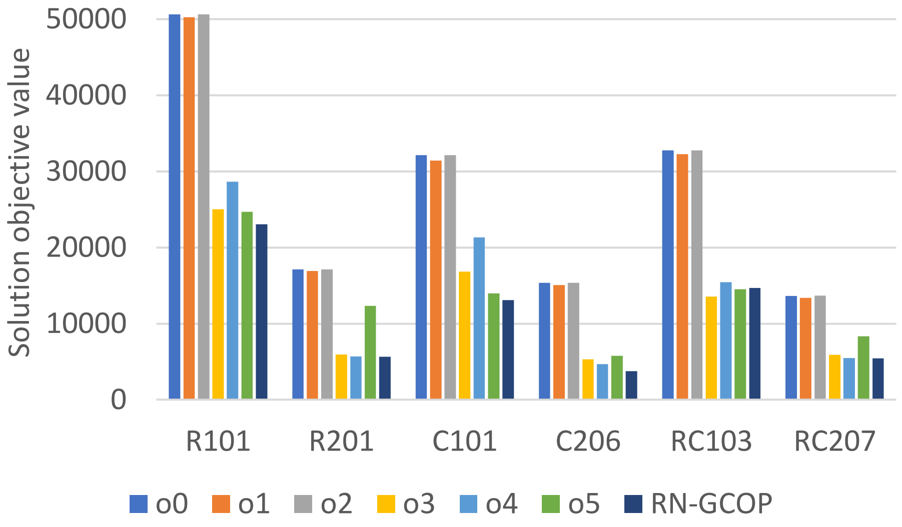
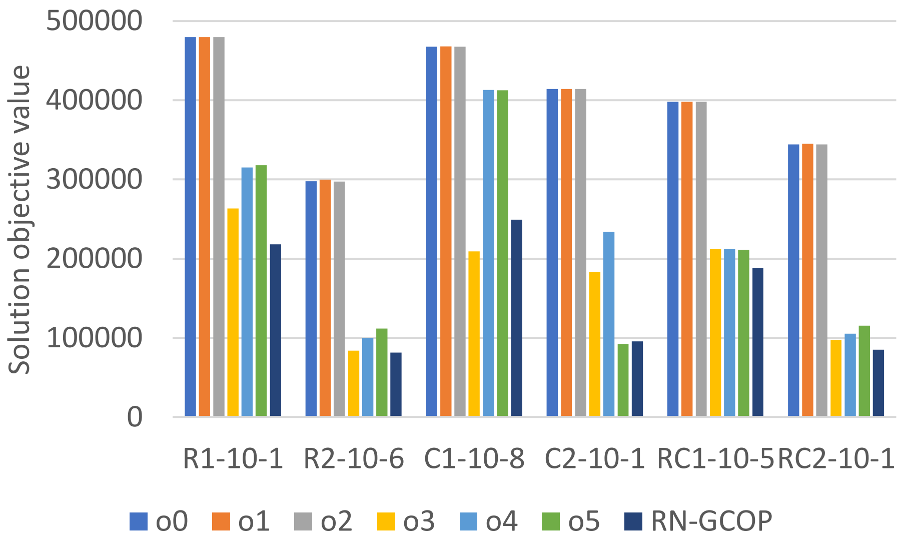


Figure 4: Comparison in the average solution objective value (out of ten runs) between each operator in $O_{vrp-basic}$ on the 1000-customer instances. $o0$: o^{in}_{xchg}. $o1$: o^{bw}_{xchg}. $o2$: o^{in}_{ins}. $o3$: o^{bw}_{ins}. $o4$: o_{rr}. $o5$: 2-*opt**.



the performance of RN-GCOP with $O_{vrp-basic}$ is compared against the same method with a subset of $O_{vrp-basic}$ which excludes the worse-performing operators. The same number of evaluations is set as the stopping condition for each method.

Table 6 shows the comparison in solution objective value between RN-GCOP with different operator sets. The results support the observations in Figure 3 and Figure 4 that the performance of different operators can be relatively different according to problem instances. RN-GCOP with the six operators $O_{vrp-basic}$ (denoted as RN6) achieves better results for the instance R2-10-6, C2-10-1 and RC2-10-1. This supports that the three worse-performing operators are useful in some cases.

Table 6: Solution quality of RN-GCOP with different operator sets. RN3: RN-GCOP with a subset of $O_{vrp-basic}$ (which excludes three worse-performing operators). RN6: RN-GCOP with six operators in $O_{vrp-basic}$. Average of objective function values out of 10 runs are presented.

Instance	R101	R201	C101	C206	RC103	RC207
RN3	**21750.22**	**5533.27**	**11978.00**	**3690.10**	**14192.18**	**5300.13**
RN6	23047.82	5631.21	13103.39	3752.15	14675.39	5432.32
Instance	R1-10-1	R2-10-6	C1-10-8	C2-10-1	RC1-10-5	RC2-10-1
RN3	**214630.66**	85006.18	**245291.17**	106239.80	**185248.76**	90826.98
RN6	218268.34	**81627.15**	249231.61	**95457.94**	188173.73	**85106.16**

In general, it is better to combine operators during the search than to use one operator in all cases. This confirms the effectiveness of the idea of GCOP which utilises algorithmic components with complementary strengths. The difference in the performance of operators requires effective GCOP methods, such as learning, to adapt to different problem instances in the automated composition process.

4.1.2. *Performance evaluation on solution quality*

Table 7 presents the results of the random RN-GCOP method with different operator sets $O_{vrp-basic}$ and O_{basic}. It is obvious that RN-GCOP with $O_{vrp-basic}$ outperforms that with O_{basic} in all instances.

Table 7: Solution quality of RN-GCOP with different operator sets $O_{vrp-basic}$ and O_{basic} using the same computational time. RN_basic: RN-GCOP with O_{basic}; RN_vrp: RN-GCOP with $O_{vrp-basic}$. Average of objective function values out of 10 runs are presented.

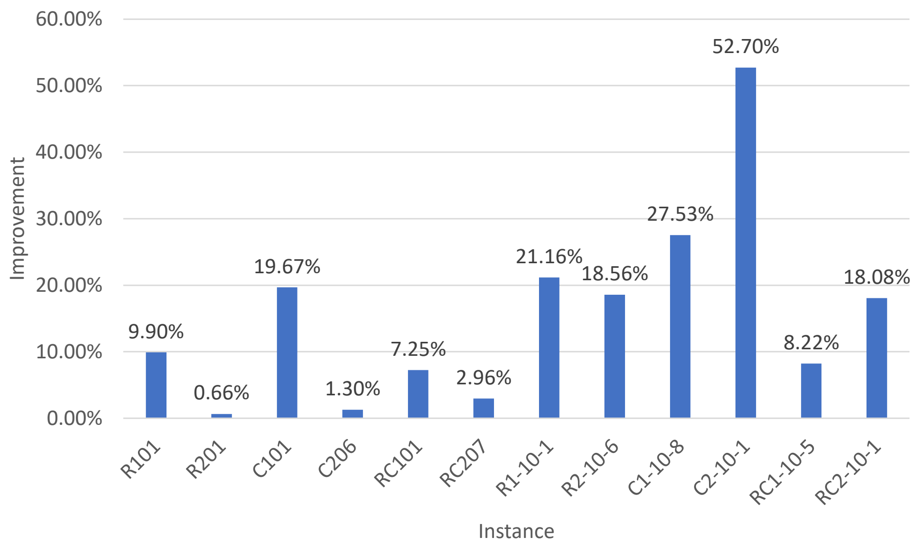
Instance	R101	R201	C101	C206	RC103	RC207
RN_basic	24163.07	5551.13	14842.15	3713.68	14592.45	5392.30
RN_vrp	21770.72	5514.45	11922.23	3665.37	13534.16	5232.89
Instance	R1-10-1	R2-10-6	C1-10-8	C2-10-1	RC1-10-5	RC2-10-1
RN_basic	270650.70	97718.23	330129.70	191309.90	201882.50	101099.60
RN_vrp	213375.90	79577.04	239243.90	90495.82	185281.10	82816.58

Figure 5 shows the improvement from $O_{vrp-basic}$ in RN-GCOP against O_{basic}. Although improvements vary among instances, RN-GCOP with $O_{vrp-basic}$ is better on solving larger instances of 1000 customers. Among the different types of customer distributions, the performance of $O_{vrp-basic}$ is relatively better for solving instances of type-C compared to O_{basic}, although the improvements vary between type-C and type-R. Improvements on type-RC of mixed customer distributions are smaller with $O_{vrp-basic}$, i.e. with the 2-*opt** operator.

4.1.3. *Performance evaluation on algorithm convergence*

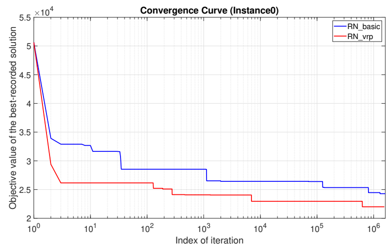
Figure 6 and Figure 7 present further detailed convergence of RN-GCOP with different operator sets for different types of instances, mapping the final results of type-C, type-R and type-RC instances in Figure 5.

Figure 5: The improvement of the random GCOP method with $O_{vrp-basic}$ (denoted as RN_vrp) compared to the same method with O_{basic} (denoted as RN_basic). The amount of Improvements = $(RN\_basic - RN\_vrp)/RN\_basic$.

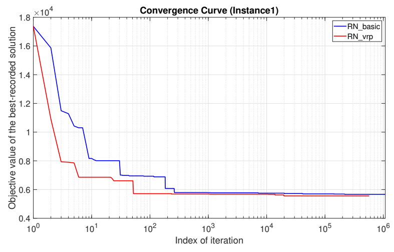


It is shown in Figure 7 that in general, for type-C instances, the gaps are bigger, followed by type-R instances as reflected in both Figure 5 and Figure 7. For type-RC instances, the gaps are much smaller. RN-GCOP with $O_{vrp-basic}$ converges faster and outperforms RN-GCOP with O_{basic} throughout the search for all types of instances.

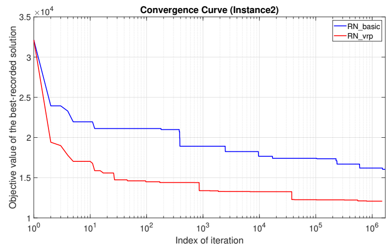
However, RN_vrp is not always better than RN_basic, e.g. Figure 6 (b) and (f) suggest RN_basic may reach satisfactory performance with the elementary algorithmic components o_i. As reflecting in both Figure 5 and Figure 6 (b) and (f), the gaps when the algorithms converge are relatively small.



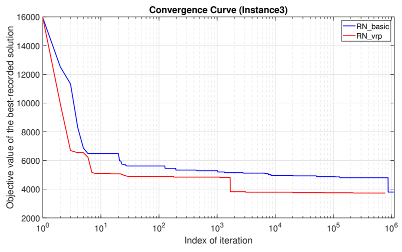
(a) Instance R101



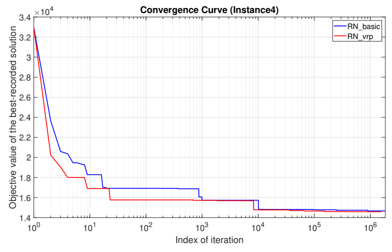
(b) Instance R201



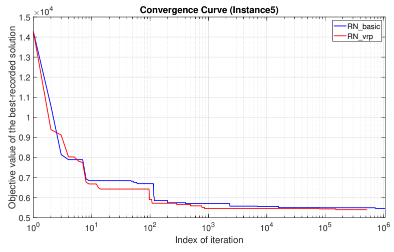
(c) Instance C101



(d) Instance C206



(e) Instance RC101

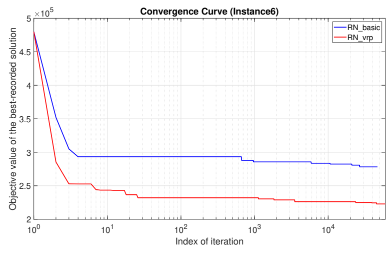


(f) Instance RC207

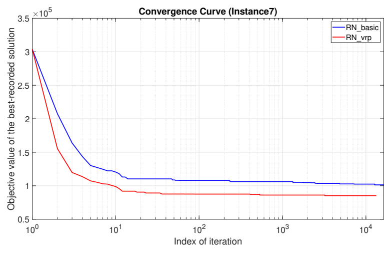
Figure 6: Convergence curves of RN-GCOP with different operator sets on the 100 customer instances (computation time: 2 CPU hours). RN_basic: RN-GCOP with O_{basic}; RN_vrp: RN-GCOP with $O_{vrp-basic}$.

4.1.4. Discussions

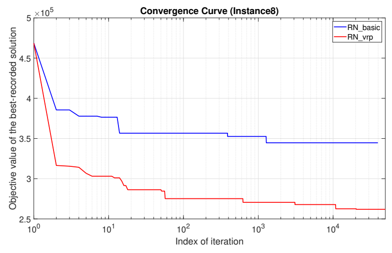
In summary, with the most basic algorithmic components, GCOP methods can obtain satisfying performance, reaching performance as good as using problem-specific compound components on some benchmark instances.



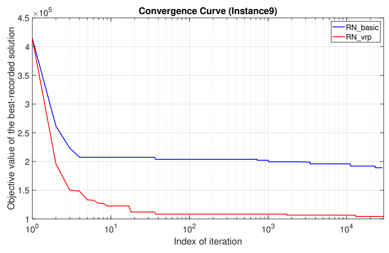
(a) Instance R1-10-1



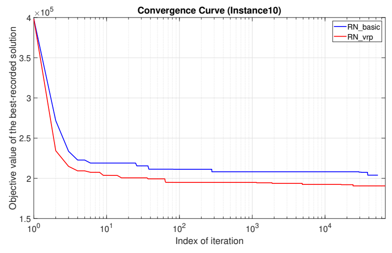
(b) Instance R2-10-6



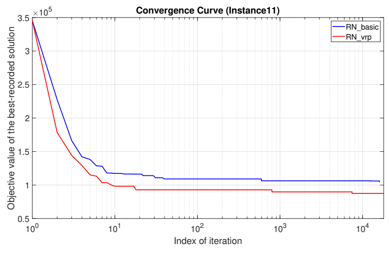
(c) Instance C1-10-8



(d) Instance C2-10-1



(e) Instance RC1-10-5



(f) Instance RC2-10-1

Figure 7: Convergence curves of RN-GCOP with different operator sets on the 1000 customer instances (computation time: 4 CPU hours). RN_basic: RN-GCOP with O_{basic}; RN_vrp: RN-GCOP with $O_{vrp-basic}$.

Larger instances may require a longer time for the most basic components to reach better results compared to those obtained using specifically designed components.

The algorithm performance of each operator is different according to problem instances. For type-RC instances, the improvements on solution quality from problem-specific compound components are relatively small. The most basic components should be given a longer computation time to reach comparable solution quality for type-R instances. For type-C instances, particularly for larger instances, the improvement in solution quality and computation time from problem-specific components are relatively significant.

The problem-specific compound component 2-opt^* swaps the end sections of two routes thus can more likely retain better sections in the solution, moving the search towards promising areas in the solution space. The improvements are due to the domain knowledge used to devise 2-opt^*. The most elementary components make only the most basic moves in the search space, and are applicable to different problems. Their generality needs to be compensated by more computational time to reach solutions obtained by 2-opt^*. In the following performance analysis, $O_{vrp-basic}$ is employed to assess the learning models.

4.2. Effectiveness of the Learning Models

To assess how well the proposed learning models in the proposed GCOP methods, IP-GCOP and TP-GCOP are compared with RN-GCOP and a random gradient GCOP method (RG-GCOP) with the same component set first. A set of *Update*() methods to update the learning model is then tested to identify their influence on learning effectiveness.

4.2.1. Comparison with Random GCOP Methods

The best and average results obtained from IP-GCOP and TP-GCOP are compared with those from RN-GCOP and RG-GCOP in Table 8. Both learning models are embedded with $Simple()$ update methods. The same number of evaluations is set as the stopping condition, i.e. $t_{iteration}(n)$ is adopted as t_{kmain} in Algorithm 1, for all GCOP methods. Similar computational time is observed for these approaches. Overall, TP-GCOP performs better than IP-GCOP, which is better than RN-GCOP and RG-GCOP in most instances. Only for one small instance, RG-GCOP obtains better results than other methods.

To analyse whether the differences observed between IP-GCOP and TP-GCOP are statistically significant, the Lilliefors test is used, shown that they do not always follow a normal distribution. The Mann–Whitney–Wilcoxon test is therefore performed with a 95% confidence level to conduct the pair-wise comparisons between the two GCOP methods. Table 9 shows that TP-GCOP has a better overall performance compared to IP-GCOP, especially for solving large instances.

The proportion each operator o_i is called in the best algorithm compositions by different GCOP methods are compared on two example instances C101 and C206, in Figure 8 and Figure 9, respectively. The o_i selected in IP-GCOP and TP-GCOP are highly different, indicating the algorithm compositions, i.e. new algorithms automatically designed with the two learning models, are highly different. Both learning models identify 2-*opt** (denoted as o_5) as the most selected component in the best algorithms, although it is automatically selected more often by TP-GCOP compared to IP-GCOP.

4.2.2. Influence of Different Update Methods to the Learning Models

The influence of different $Update()$ methods as specified in Section 3.2 is examined to identify the best intra-domain general method (of the per-

Table 8: Comparison between GCOP methods with different learning (IP-GCOP and TP-GCOP) against the random RN-GCOP and RG-GCOP method. Best and average of objective function values out of 31 runs are presented.

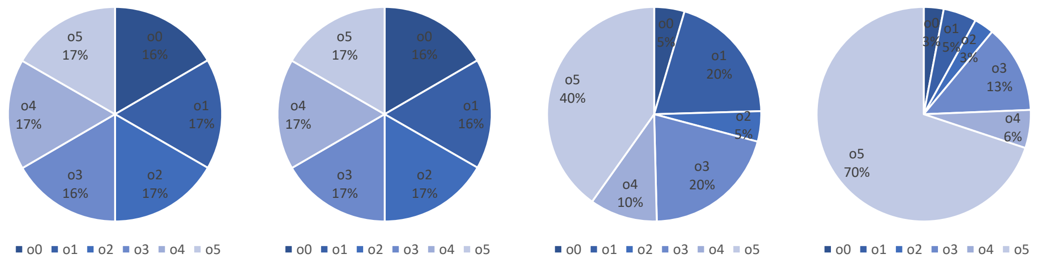
Instance		R101	R201	C101	C206	RC103	RC207
RN	Best	22941.88	5569.55	12027.70	3709.01	14609.87	5355.37
	AVG	23055.69	5637.15	13030.92	3752.32	14684.06	5437.76
	AVG Time(s)	176	436	175	316	169	452
RG	Best	21914.76	5584.46	12158.35	**3704.71**	14622.91	5385.61
	AVG	22946.41	5630.69	12802.19	3756.83	14687.69	5444.48
	AVG Time(s)	175	436	176	317	169	452
IP	Best	21792.20	5481.94	**10828.94**	3707.82	14545.99	5325.75
	AVG	22027.43	**5592.05**	11823.04	**3737.37**	14618.26	5384.51
	AVG Time(s)	225	818	261	579	223	809
TP	Best	**20683.49**	**5476.38**	**10828.94**	3708.99	**13523.36**	**5302.97**
	AVG	**21599.58**	5600.57	**11072.53**	3738.01	**14538.34**	**5368.54**
	AVG Time(s)	234	1069	284	598	234	970
Instance		R1-10-1	R2-10-6	C1-10-8	C2-10-1	RC1-10-5	RC2-10-1
RN	Best	214494.95	80507.45	238474.41	92016.80	185565.28	81506.01
	AVG	218091.44	81341.35	248982.83	94935.60	188031.34	84957.79
	AVG Time(s)	225	1044	280	509	2142	799
RG	Best	192876.50	72191.55	216253.93	79846.80	176046.18	73084.74
	AVG	206357.04	78072.89	235120.04	89442.99	183384.09	79385.30
	AVG Time(s)	224	1026	281	500	210	802
IP	Best	187732.06	71466.97	184622.87	63594.81	175301.01	71337.83
	AVG	198588.32	74280.80	213053.01	70601.03	179682.39	74856.27
	AVG Time(s)	257	1578	277	610	235	918
TP	Best	**160065.59**	**62520.70**	**155129.10**	**50841.53**	**152887.15**	**61935.10**
	AVG	**164202.91**	**63939.58**	**173015.06**	**52943.00**	**162072.30**	**63673.53**
	AVG Time(s)	221	1785	231	794	225	1133

formance across multiple instances from the same domain) for updating the learning model in IP-GCOP and TP-GCOP, respectively. All methods are

Table 9: Performance comparison between IP-GCOP and TP-GCOP using the Mann–Whitney–Wilcoxon test. The comparison between TP ↔ IP is shown as +, -, or ∼ when TP-GCOP is significantly better than, worse than, or statistically equivalent to IP-GCOP, respectively.

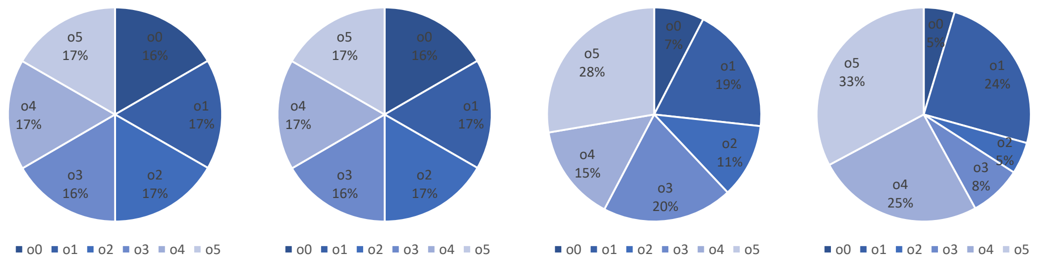
Instance	R101	R201	C101	C206	RC103	RC207
TP ↔ IP	∼	+	∼	+	+	+
Instance	R1-10-1	R2-10-6	C1-10-8	C2-10-1	RC1-10-5	RC2-10-1
TP ↔ IP	+	+	+	+	+	+

Figure 8: Proportion of each operator called in the best algorithm compositions obtained by IP-GCOP and TP-GCOP, compared with RN-GCOP and RG-GCOP, for solving instance C101.



(a) RN (b) RG (c) IP (d) TP

Figure 9: Proportion of each operator called in the best algorithm compositions obtained by IP-GCOP and TP-GCOP, compared with RN-GCOP and RG-GCOP, for solving instance C206.



(a) RN (b) RG (c) IP (d) TP

evaluated for the same number of times, to compare the results out of ten runs.

The results across different instances differ by a large scale, are therefore normalised into a range [0, 1]. The normalisation scheme in (Di Gaspero & Urli, 2012), as shown in Equation (5), is used, where $x(i)$ represent the objective function values calculated using Equation (4), and x_{best} and x_{worst} is the best and worst results obtained, respectively. An intra-domain performance score is then calculated as the sum of normalised results over all instances. The normalised scores of the learning models with different update strategies in Table 10 show that TP-GCOP with *Simple*() obtained the best intra-domain performance. For IP-GCOP, the most suitable update method is *Linear*().

$$x_{norm(i)} = \frac{x(i) - x_{best}}{x_{worst} - x_{best}}. \tag{5}$$

Table 10: The intra-domain scores for IP-GCOP and TP-GCOP with different update methods. The best results (smallest values) for each method are in bold.

Update()	IP-GCOP	TP-GCOP
Simple()	28.34	**15.52**
Improve()	20.98	52.31
NoCall()	23.79	23.31
Linear()	**16.14**	43.64
NoImprove()	21.68	21.51

Further analysis on the intra-domain performance scores of the update strategies using the Lilliefors test showed that they do not always follow a normal distribution. The Mann–Whitney–Wilcoxon test is therefore performed with a 95% confidence level to conduct the pairwise comparisons between the two GCOP methods with different update strategies statistically.

Table 11 shows that IP-GCOP with *Linear*() has a better overall performance compared to other update methods, especially for solving large instances. For TP-GCOP in Table 12, *Simple*() led to better overall performance, especially for solving small instances.

Table 11: Pairwise performance comparison between different *Update*() methods with IP-GCOP using the Mann–Whitney–Wilcoxon test. The comparison between A ↔ B is shown as +, -, or ∼ when A is significantly better than, worse than, or statistically equivalent to B, respectively.

Update()	Instance					
A ↔ B	R101	R201	C101	C206	RC103	RC207
Linear ↔ *Simple*	∼	∼	+	∼	+	∼
Linear ↔ *Improve*	∼	∼	∼	−	∼	∼
Linear ↔ *NoCall*	∼	∼	+	∼	+	∼
Linear ↔ *NoImprove*	∼	∼	∼	∼	∼	∼
Update()	Instance					
A ↔ B	R1-10-1	R2-10-6	C1-10-8	C2-10-1	RC1-10-5	RC2-10-1
Linear ↔ *Simple*	+	+	+	+	+	+
Linear ↔ *Improve*	+	+	+	+	+	+
Linear ↔ *NoCall*	+	+	+	+	+	+
Linear ↔ *NoImprove*	∼	∼	∼	+	∼	∼

4.3. Comparison with the Best-known Approaches

The best solutions from TP-GCOP with *Simple*() and IP-GCOP with *Linear*() are compared with the published best results produced by various

Table 12: Pairwise performance comparison for the TP-GCOP with different update methods using the Mann–Whitney–Wilcoxon test.

Update()	Instance					
A ↔ B	R101	R201	C101	C206	RC103	RC207
Linear ↔ *Simple*	+	+	∼	∼	∼	∼
Linear ↔ *Improve*	+	∼	+	∼	∼	∼
Linear ↔ *NoCall*	+	+	∼	∼	∼	∼
Linear ↔ *NoImprove*	∼	∼	∼	∼	∼	+
Update()	Instance					
A ↔ B	R1-10-1	R2-10-6	C1-10-8	C2-10-1	RC1-10-5	RC2-10-1
Linear ↔ *Simple*	+	−	∼	+	∼	+
Linear ↔ *Improve*	∼	∼	∼	∼	∼	∼
Linear ↔ *NoCall*	+	∼	+	+	∼	−
Linear ↔ *NoImprove*	∼	−	∼	−	−	−

state-of-the-art methods. In the VRP literature, the results of the best-known solutions for VRPTW are usually ranked using a hierarchical objective function, considering the number of vehicles NV as the primary objective and the total travel distance TD as the second objective (Bräysy & Gendreau, 2005a). In this study, a solution with lower NV is considered better than the others with higher NV. For those solutions with the same NV, the lower TD the better.

Table 13 shows the results of the proposed GCOP methods compared to the best results reported in the literature by different approaches. It can be seen that the proposed GCOP methods can obtain competitive results in the number of vehicles (i.e. NV) in most small instances, especially on instance C101, where both of the proposed methods obtained the best solution in the literature.

It should be noted that the results from the proposed GCOP methods are obtained by automatically designed new algorithms without any human involvement, while the best results in the literature are obtained by different methods specifically designed for VRPTW. The main research objective in this study is not to beat the tailor-made state-of-the-art approaches by yet another algorithm, but to investigate the learning in automatic design of new algorithms by composing only the most basic algorithmic components within the general AutoGCOP framework.

Table 13: Comparison of solution quality between the published best-known results and the best solutions from IP-GCOP and TP-GCOP out of ten runs. NV denotes the number of vehicles. TD denotes the total travel distance. Results that are better than or the same as the best known are in bold.

Instance	Best-known results			IP-GCOP		TP-GCOP	
	NV	TD	Ref.	NV	TD	NV	TD
R101	19	1650.80	(SINTEF, a)	**19**	1653.76	**19**	1654.07
R201	4	1252.37	(SINTEF, a)	**4**	1624.12	**4**	1443.91
C101	10	828.94	(SINTEF, a)	**10**	**828.94**	**10**	**828.94**
C206	3	588.49	(SINTEF, a)	**3**	754.75	**3**	671.54
RC103	11	1261.67	(SINTEF, a)	12	1401.44	12	1399.13
RC206	3	1061.14	(SINTEF, a)	4	1297.91	4	1258.75
R1-10-1	100	53412.11	(SINTEF, b)	101	58815.26	**100**	62024.51
R2-10-6	19	29978.02	(SINTEF, b)	21	41170.48	21	40851.73
C1-10-8	92	42629.91	(SINTEF, b)	103	44867.96	106	47013.85
C2-10-1	30	16879.24	(SINTEF, b)	32	18141.82	33	18149.23
RC1-10-5	90	45069.37	(SINTEF, b)	95	53594.62	97	54482.19
RC2-10-1	20	30276.27	(SINTEF, b)	29	33446.19	28	32627.48

5. Conclusions

Based on the General Combinatorial Optimisation Problem (GCOP) model which defines the problem of algorithm design as a combinatorial optimisation problem, new algorithms can be designed automatically by searching in a space of compositions of elementary algorithmic components. In this paper, a general AutoGCOP framework is built to support the automatic composition of elementary algorithmic components based on the general GCOP model, thus to design local search algorithms automatically.

With the encapsulated common processes in local search algorithms, AutoGCOP allows instantiations of existing algorithms designed by manually determining algorithmic components. That is, a large number of existing local search algorithms can be seen as specific solutions of GCOP implemented in AutoGCOP. Furthermore, the AutoGCOP framework underpins the automated design of new and unseen algorithms by using different GCOP methods which compose the algorithmic components automatically.

Based on the AutoGCOP framework, this study focuses on addressing two important and fundamental issues. The performance of the most basic algorithmic components is analysed to justify their effectiveness in designing search algorithms capable of solving complex combinatorial optimisation problems. Two GCOP methods have also been proposed based on different learning models to investigate learning within the unified template of AutoGCOP framework in designing new algorithms automatically.

The basic elementary algorithmic components present a satisfying performance given enough computational time, which confirms their effectiveness in automatically designing search algorithms to solve hard VRPTW problems online. In addition, including problem-specific algorithmic components in the basic component set can greatly improve the efficiency of search, reaching a similar solution quality with less computation time especially for solving larger instances with specific problem structure. This efficiency is gained by the domain knowledge used to devise problem-specific algorithmic components, which may not be available or consistent in practice. The general AutoGCOP with elementary algorithmic components presents a promising framework across different problems and may be employed by developers of different expertise.

The proposed GCOP methods based on the AutoGCOP framework have been investigated with effective learning ability to observe the behaviour of algorithmic components. Particularly, compared to the learning model which records and learns from the performance of individual components, the Markov chain based learning model which adaptively records the transition performance between pairs of basic components shows superior overall performance for problems of different sizes and structures.

The research presented in this paper can be extended into various interesting research directions. Future work will consider the general AutoGCOP framework in designing multi-objective algorithms. Analysis of the performance of basic components for addressing different problem domains within the common AutoGCOP framework may also reveal their effectiveness on different problem features and structures. Last but not least, with the GCOP model which defines a vast design space of algorithms, effective patterns from different algorithmic compositions may be discovered within AutoGCOP using machine learning. Such patterns or new knowledge extracted from the design space of algorithms can be applied offline to design new effective algorithms, which may be difficult by human experts, releasing them from computational burdens to focus on applying the new knowledge to solve more problems effectively.

Acknowledgements

This work was supported by the University of Nottingham.

References

Banzhaf, W., Nordin, P., Keller, R. E., & Francone, F. D. (1998). *Genetic programming*. Springer.

Baum, L. E., & Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, *37*, 1554–1563.

Bezerra, L. C., López-Ibánez, M., & Stützle, T. (2015). Automatic component-wise design of multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *20*, 403–417.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, *35*, 268–308.

Braekers, K., Ramaekers, K., & Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, *99*, 300–313.

Bräysy, O., & Gendreau, M. (2005a). Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation science*, *39*, 104–118.

Bräysy, O., & Gendreau, M. (2005b). Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation science*, *39*, 119–139.

Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vázquez-Rodríguez, J. A., & Gendreau, M. (2010). Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In *IEEE congress on evolutionary computation* (pp. 1–8). IEEE.

Burke, E. K., Hyde, M. R., & Kendall, G. (2011). Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation*, *16*, 406–417.

Burke, E. K., Kendall, G., & Soubeiga, E. (2003). A tabu-search hyper-heuristic for timetabling and rostering. *Journal of heuristics*, *9*, 451–470.

Cordeau, J.-F., Laporte, G., Savelsbergh, M. W., & Vigo, D. (2007). Vehicle routing. *Handbooks in operations research and management science*, *14*, 367–428.

Cowling, P., Kendall, G., & Soubeiga, E. (2000). A hyperheuristic approach to scheduling a sales summit. In *International Conference on the Practice and Theory of Automated Timetabling* (pp. 176–190). Springer.

Di Gaspero, L., & Urli, T. (2011). A reinforcement learning approach for the cross-domain heuristic search challenge. In *Proceedings of the 9th Metaheuristics International Conference (MIC 2011), Udine, Italy*.

Di Gaspero, L., & Urli, T. (2012). Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In *International Conference on Learning and Intelligent Optimization* (pp. 384–389). Springer.

Fisher, M., & Fisher, M. (1995). Chapter 1 vehicle routing. *Handbooks in Operations Research and Management Science*, *8*, 1–33.

Franzin, A., & Stützle, T. (2019). Revisiting simulated annealing: A component-based analysis. *Computers & Operations Research*, *104*, 191–206.

Fukunaga, A. S. (2008). Automated discovery of local search heuristics for satisfiability testing. *Evolutionary computation*, *16*, 31–61.

Gehring, H., & Homberger, J. (1999). A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99* (pp. 57–64). Citeseer volume 2.

Hoos, H. H. (2008). *Computer-aided design of high-performance algorithms*. Technical Report Technical Report TR-2008-16, University of British Columbia, Department of Computer Science.

Hutter, F., Hoos, H. H., & Stützle, T. (2007). Automatic algorithm configuration based on local search. In *Proceedings of the 22nd national conference on Artificial intelligence-Volume 2* (pp. 1152–1157).

Kemeny, J. G., & Snell, J. L. (1976). *Markov chains*. Springer-Verlag, New York.

Khamassi, I., Hammami, M., & Ghédira, K. (2011). Ant-q hyper-heuristic approach for solving 2-dimensional cutting stock problem. In *2011 IEEE Symposium on Swarm Intelligence* (pp. 1–7). IEEE.

Kheiri, A., & Keedwell, E. (2015). A sequence-based selection hyper-heuristic utilising a hidden Markov model. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (pp. 417–424). ACM.

KhudaBukhsh, A. R., Xu, L., Hoos, H. H., & Leyton-Brown, K. (2016). SATenstein: Automatically building local search SAT solvers from components. *Artificial Intelligence*, *232*, 20–42.

Lissovoi, A., Oliveto, P. S., & Warwicker, J. A. (2020). Simple hyper-heuristics control the neighbourhood size of randomised local search optimally for leadingones. *Evolutionary computation*, *28*, 437–461.

Lopez-Ibanez, M., & Stutzle, T. (2012). The automatic design of multiobjective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, *16*, 861–875.

Mascia, F., López-Ibánez, M., Dubois-Lacoste, J., & Stützle, T. (2013). From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness. In *International Conference on Learning and Intelligent Optimization* (pp. 321–334). Springer.

McClymont, K., & Keedwell, E. C. (2011). Markov chain hyper-heuristic (MCHH): an online selective hyper-heuristic for multi-objective continuous problems. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation* (pp. 2003–2010). ACM.

Miranda, P. B., Prudêncio, R. B., & Pappa, G. L. (2017). H3AD: A hybrid hyper-heuristic for algorithm design. *Information Sciences*, *414*, 340–354.

Nareyek, A. (2003). Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer decision-making* (pp. 523–544). Springer.

Nguyen, S., Zhang, M., Johnston, M., & Tan, K. C. (2012). A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, *17*, 621–639.

Oltean, M. (2005). Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, *13*, 387–410.

Özcan, E., Bilgin, B., & Korkmaz, E. E. (2008). A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, *12*, 3–23.

Özcan, E., Misir, M., Ochoa, G., & Burke, E. K. (2012). A reinforcement learning: great-deluge hyper-heuristic for examination timetabling. In *Modeling, Analysis, and Applications in Metaheuristic Computing: Advancements and Trends* (pp. 34–55). IGI Global.

Pagnozzi, F., & Stützle, T. (2019). Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems. *European journal of operational research*, *276*, 409–421.

Pickardt, C., Branke, J., Hildebrandt, T., Heger, J., & Scholz-Reiter, B. (2010). Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness. In *Proceedings of the 2010 Winter Simulation Conference* (pp. 2504–2515). IEEE.

Pillay, N., & Qu, R. (2018). *Hyper-Heuristics: Theory and Applications*. Springer.

Pillay, N., Qu, R., Srinivasan, D., Hammer, B., & Sorensen, K. (2018). Automated design of machine learning and search algorithms [Guest Editorial]. *IEEE Computational Intelligence Magazine*, *13*, 16–17.

Potvin, J.-Y., & Rousseau, J.-M. (1995). An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, *46*, 1433–1446.

Qu, R., Kendall, G., & Pillay, N. (2020). The General Combinatorial Optimization Problem: Towards Automated Algorithm Design. *IEEE Computational Intelligence Magazine*, *15*, 14–23.

SINTEF (a). VRPTW benchmark problems, on the sintef transport optimisation portal. https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/100-customers/. Published April 18, 2008.

SINTEF (b). VRPTW benchmark problems, on the sintef transport optimisation portal. https://www.sintef.no/projectweb/top/vrptw/homberger-benchmark/1000-customers/. Published April 18, 2008.

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

Sutton, R. S., Barto, A. G. et al. (1998). *Introduction to reinforcement learning* volume 135. MIT press Cambridge.

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.

Wong, R. T. (1983). Combinatorial optimization: Algorithms and complexity (Christos H. Papadimitriou and Kenneth Steiglitz). *SIAM Review*, *25*, 424.