Expert Systems With Applications 213 (2023) 118812

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A reinforcement learning-Variable neighborhood search method for the capacitated Vehicle Routing Problem

Panagiotis Kalatzantonakis, Angelo Sifaleras [*], Nikolaos Samaras

Department of Applied Informatics, School of Information Sciences, University of Macedonia, 156 Egnatia Str., Thessaloniki 54636, Greece

ARTICLE INFO

Keywords:
Reinforcement Learning
Multi-Armed Bandits
Intelligent Optimization
Bandit Learning
Metaheuristics
Variable Neighborhood Search
Vehicle Routing Problem

ABSTRACT

Finding the best sequence of local search operators that yields the optimal performance of Variable Neighborhood Search (VNS) is an important open research question in the field of metaheuristics. This paper proposes a Reinforcement Learning method to address this question. We introduce a new hyperheuristic scheme, termed Bandit VNS, inspired by the Multi-Armed Bandit (MAB), a particular type of a single state reinforcement learning problem. In Bandit VNS, we utilize the General Variable Neighborhood Search metaheuristic and enhance it by a hyperheuristic strategy. We examine several variations of the Upper Confidence Bound algorithm to create a reliable strategy for adaptive neighborhood selection. Furthermore, we utilize Adaptive Windowing, a state of the art algorithm to estimate and detect changes in the data stream. Bandit VNS is designed for effective parallelization and encourages cooperation between agents to produce the best solution quality. We demonstrate this concept's advantages in accuracy and speed by extensive experimentation using the Capacitated Vehicle Routing Problem. We compare the novel scheme's performance against the conventional General Variable Neighborhood Search metaheuristic in terms of the CPU time and solution quality. The Bandit VNS method shows excellent results and reaches significantly higher performance metrics when applied to well-known benchmark instances. Our experiments show that, our approach achieves an improvement of more than 25% in solution quality when compared to the General Variable Neighborhood Search method using standard library instances of medium and large size.

1. Introduction

The Vehicle Routing Problem (VRP) is one of the most well-studied combinatorial optimization problems in computer science and applied mathematics. This problem's complex nature emerges when one seeks an efficient combination of routes for a fleet of vehicles used for serving clients, given a set of constraints.

The standard VRP belongs to the NP-hard class of optimization problems since it generalizes the most influential and well-known combinatorial optimization problems, the Travelling Salesman Problem (TSP) and the Bin Packing Problem (BPP), which are both prevalent NP-hard problems (Johnson and Garey, 1979). It can also be regarded as a Set Packing (SP) problem if each set's cost is viewed as the distance of the associated optimal TSP tour (Balinski and Quandt, 1964). Given the NP-hard characteristic of VRPs (Lenstra and Kan, 1981), exact methods are more suitable for small or medium-sized VRPs (Toth and Vigo, 2002a) since it is computationally challenging to solve to optimality, even with only a few hundred customer nodes.

As a result of VRP's practical significance, there is considerable interest in finding novel ways to solve this problem regardless of existing heuristic and approximate approaches. For an overview of the VRP, see Golden et al. (2008), Laporte (1992), Laporte et al. (2000) and Paolo and Daniele (2002). Traditional VRP has developed several variants. This paper focuses on a basic variant of the VRP, namely the Capacitated Vehicle Routing Problem (CVRP). Dantzig and Ramser introduced the CVRP in 1959, which is NP-hard in the strong sense because it includes the well-known Travelling Salesman and Bin Packing problems as special cases. The CVRP, which has a wide range of logistics applications, aims to find a set of routes with minimal cost to fulfill customers' demands without violating vehicle capacity constraints. Most variants of VRP are built on top of CVRP. Consequently, the research of new, efficient ways of solving the CVRP is also beneficial for other problems.

Due to the NP-hard nature of combinatorial optimization problems like the CVRP, the research community focuses on improving conventional metaheuristics by utilizing emerging techniques to find solutions over a discrete search space. To some extent, this can be explained by the remarkable adaptability of metaheuristics to solve diverse optimization problems and, at the same time, to generate

[*] Corresponding author.
E-mail addresses: pkalatzantonakis@uom.edu.gr (P. Kalatzantonakis), sifalera@uom.gr (A. Sifaleras), samaras@uom.gr (N. Samaras).

https://doi.org/10.1016/j.eswa.2022.118812
Received 24 October 2021; Received in revised form 12 August 2022; Accepted 8 September 2022
Available online 17 September 2022

quality solutions for complex optimization problems within limited computational time.

Variable Neighborhood Search (VNS) is a well-known metaheuristic method originally proposed by Mladenović and Hansen in 1997 which is constantly evolving for the solution of various optimization problems (Hansen et al., 2019a; Mladenović et al., 2021) and also machine learning tasks (Mladenović et al., 2021). VNS is a local search-based algorithm driven by systematic changes in neighborhood search structures. Systematic change of neighborhood happens both within a descent phase to find a local optimum and perturbation phase to get out of the corresponding valley (Hansen et al., 2017). VNS was chosen because it has great scalability and can extend to various optimization problems.

In view of the successful application in many fields, the recognition of reinforcement learning is growing significantly (Sultana et al., 2022; Li et al., 2021; Dulac-Arnold et al., 2021; Nazari et al., 2018; Kool et al., 2018; Russo and Van Roy, 2014; Neu et al., 2010). In the field of intelligent optimization, there has been a substantial body of work on reinforcement learning and metaheuristics, and the authors in Koulouriotis and Xanthopoulos (2008), Liu et al. (2022), Lu et al. (2020), Sun et al. (2022) and Wauters et al. (2013) have confirmed that through this hybridization, one can derive promising techniques and many prospects toward future research directions.

In contrast, only a limited number of research investigate the impact of the combination of Upper Confidence Bound (UCB) algorithms and metaheuristics. By comparison, bandit algorithms differ considerably with respect to computational complexity. The most famous UCB algorithm is UCB1 proposed by Auer et al. (2002), which enjoys low computational complexity, while the generic UCB even lower, approaching $O(1)$ for each arm in each round (Liu et al., 2018). On the other end of the spectrum lies the Kullback–Leibler algorithm (KL-UCB), which has enhanced performance on sufficiently long horizons at the price of a significantly higher computational complexity.

This paper aims at solving the CVRP using benchmark instances from CVRPLib (Uchoa et al., 2017) while benchmarking, and identifying the best adaptive method to guide the search process. Hence, our contribution is three-fold:

(i) Exploit the sequential nature of the VNS metaheuristic to obtain a reliable and fast online hyperheuristic framework with optimal behavior in terms of the CPU time and solution quality and evaluate the performance.
(ii) Conduct an experimental analysis to determine the best performing UCB algorithm and examine the behavior when applied to well-known CVRP benchmark instances.
(iii) Advance the agent's self-adaptive skills by coupling the most reliable model with the Adaptive Windowing (ADWIN), a state-of-the-art algorithm to detect abrupt changes in the operator quality distribution and restart the MAB task.

The remainder of this paper is organized as follows. In Section 2, we present a brief overview of bandit implementations in several problem domains. In Section 3 and the following subsections, the necessary background knowledge is described, including the formulation of the two-index CVRP. Section 5 describes the proposed Bandit VNS strategy.

2. Related work

Reinforcement learning is the branch of machine learning applied to problems with a sequential context. The primary objective is to design algorithms that can learn from experience and acquire optimal behavior by maximizing rewards. Reinforcement learning and deep reinforcement learning algorithms provide auspicious performance, and stable behavior (dos Santos et al., 2014; Lin et al., 2020; Delarue et al., 2020; Silva et al., 2019; Chen et al., 2020). UCB algorithms belong to the family of reinforcement learning algorithms, offering good performance with lower complexity and less complicated computations. Metaheuristics and Bandit algorithms have been successfully applied to solve various combinatorial optimization problems in various problem domains, including scheduling, bin-packing, and routing. The categories of MAB algorithms applied are presented in Table 1, with additional information on the metaheuristic and the problem domains.

Hyper-heuristics can be categorized into two main categories: heuristic selection and heuristic generation (Burke et al., 2010). Hyper-heuristics frameworks based on Multi-armed Bandits belong to the first category. Bandit algorithms are used as a learning method that can learn to select the low-level heuristics and guide the selection process.

The obtained results from the available literature prove that the proposed model can be generalized over different domains and yield satisfactory or even superior results when confronted with state-of-the-art hyper-heuristics. MOEA is the abbreviation of Multi-objective Evolutionary Algorithm, LLH stands for Low Level Heuristics. MAB categories are further discussed in Section 5.1.

Table 1
Related MAB approaches and problem domain.

MAB Method	Metaheuristic	Problem(s)	Authors
Dynamic	LLH	Exam scheduling, DynamicVRP	(Sabar et al., 2014)
Dynamic	LLH	Exam timetables, DVRP, Boolean SAT, 1D Bin-packing, Permutation Flowshop, Personnel scheduling, TSP, VRP	(Ferreira et al., 2017)
Dynamic	Evolutionary Algorithm	One-Max, Long k-paths	(Fialho et al., 2009)
Classic, Restless and Contextual	MOEA based on Dominance and Decomposition	Permutation Flowshop	(Almeida et al., 2020)
Classic	LLH	Maximum k-plex Problem	(Chen et al., 2019)
Classic, Exp. Monte Carlo	Math-heuristic	VRP with Time Windows	(Sabar et al., 2015)
Classic - Monte Carlo	LLH	Deep Belief Network tuning for Image recognition	(Sabar et al., 2017)
Fitness-Rate-Rank	LLH	Location-Routing Problem-based Low-Carbon Cold Chain	(Leng et al., 2020)
Fitness-Rate-Rank	LLH	Feature Model (FM) for Software Product Lines (SPLs)	(Strickler et al., 2016)
Fitness-Rate-Rank	LLH	One Max, NK-landscapes	(Jankee et al., 2016)

3. CVRP mathematical formulation

In this section, we briefly present the required notations, the mathematical formulations in relation to our work and the scientific background they rely on. The undirected CVRP mathematical formulation presented in this paper is originally proposed by Laporte et al. (1985) and later by Toth and Vigo (2002b) and Toth and Vigo (2014), which supports single vehicle routes. Its termed as the Two-Index Vehicle Flow Formulation or VRP1, and is defined as follows:

An undirected weighted graph $G = (V, E)$ is given where $V = \{0, 1, \dots, n\}$ is the set of $n+1$ nodes. The zero node ($node_0$) is assigned for the depot, and the rest of the nodes $V' = \{1, \dots, n\}$ are assigned to the customers. The distance between node i and node j is associated with a non-negative cost c_{ij}. A binary variable e_{ij} that takes values $\{0, 1\}$ that denotes whether that road is traversed or not. Each customer $i \in V'$ has a demand d_i, and a set of K identical vehicles, each with a capacity of C are placed at the depot (node 0) in order to satisfy the customers' needs.

Using the notation introduced above, the problem is formulated as the following optimization problem:

$$\min_e \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} e_{ij} \tag{1a}$$

subject to:

$$\sum_{i=0, i\neq j}^{n} e_{ij} = 1 \qquad \forall j \in V' \tag{2a}$$

$$\sum_{j=0, i\neq j}^{n} e_{ji} = 1 \qquad \forall i \in V' \tag{2b}$$

$$\sum_{i=0}^{n} e_{i0} = K \qquad \forall i \in V \tag{2c}$$

$$\sum_{j=0}^{n} e_{0j} = K \qquad \forall i \in V \tag{2d}$$

$$u_i - u_j + C e_{ij} \le C - d_j \quad \forall i, j \in V', i \neq j,\ s.t.\ d_i + d_j \le C \tag{2e}$$

$$d_i \le u_i \le C \qquad \forall i, j \in V' \tag{2f}$$

$$e_{i,j} \in \{0, 1\} \qquad \forall i, j \in V \tag{2g}$$

Constraints (2a) and (2b) impose the in-degree and out-degree restrictions so that precisely one vehicle enters and leaves each node because the clients' demands cannot be partially satisfied. Likewise, constraints (2c) and (2d) ensure that K cars in total depart from the depot and return to it. Constraints (2e) and (2f) are sub-tour elimination constraints that impose the capacity requirements of CVRP and reject non-feasible routes (i.e., isolated sub-tours).

4. General variable neighborhood search methodology

The essential concept of Variable Neighborhood Search is to systematically change neighborhood structures within the search procedure to extensively explore the search space. VNS is a stochastic metaheuristic method which incorporates either a simple local search part or a more powerful version denoted as Variable Neighborhood Descent (VND). The methodology of the VND alternates neighborhoods in a deterministic way and relies on three strategies to explore them: (i) at random, (ii) deterministically, or (iii) mixed (both, in a deterministic and random order) (Duarte et al., 2018).

When VND is applied as a local search procedure in VNS then a more generic VNS emerges, called General Variable Neighborhood Search (GVNS), as illustrated in Algorithm 1. Employing this General VNS approach has led to several successful applications reported in the literature (Mladenović et al., 2014; Karakostas et al., 2019).

Algorithm 1: General Variable Neighborhood Search (GVNS)

```
Input  : $x$ : current solution, $s_{max}$, $k_{max}$ : neighborhood count,
         $t_{max}$ : max execution time
Output: Solution $x$
1  Generate an initial solution $x$;
2  repeat
3      $step \leftarrow 1$ ;
4      // Shake: Explore random neighborhood, accept any
           solution
5      $x' \leftarrow$ Shake($x, k_{max}$); // Given x, execute $\mathcal{N}_{l=rand(0,k_{max})}$
6      // VND: Explore neighborhoods starting from $x'$
7      $x'' \leftarrow$ VND($x', k_{max}$);
8      $t \leftarrow$ CpuTime() ;
9      $step+ = 1$; // One GVNS step completed
10 until $t \ge t_{max}$ or $steps = steps_{max}$;
11 return x'';
```

Shaking is regularly employed in VNS algorithms to diversify the search and, most importantly, to escape from a local optimum. Random or semi-random execution of neighborhood operators is a commonly used approach to obtain the aforementioned goals. Each neighborhood operator is invoked once, and the solution produced is transferred to the next operator. The newly formed solution is accepted, provided that it is feasible, even if its quality is inferior to the initial solution.

By t_{max}, k_{max}, and l_{max} we denote the maximum CPU time allowed before termination, the maximum number of shaking iterations, and the number of neighborhood structures, respectively.

VND, the main ingredient of GVNS, which is mostly or entirely deterministic, is described in Algorithm 2. Standard VND variants cross the list of neighborhood structures sequentially. The neighborhood structures are placed in such a way that intensification is followed naturally by diversification (Liberti and Maculan, 2006).

Algorithm 2: Variable Neighborhood Descent (VND)

```
Input  : $x$ : current solution, $k_{max}$ : neighborhood count
Output: Solution $x$
1  $l \leftarrow 1$; // Initialize neighborhood index.
2  repeat
3      // Given $x$, search $\mathcal{N}$ with index $l$
4      Search for $x' \in \mathcal{N}_l(x)$ such that $f(x') < f(x)$;
5      if $f(x') < f(x)$ then
6          $x \leftarrow x'$; // Adopt better solution $x'$.
7          $l \leftarrow 1$; // Set neighborhood index back to 1.
8      else
9          // Increase neighborhood index $l$
10         $l+ = 1$;
11     end
12 until $l = k_{max}$;
13 // One VND step has been completed
14 return x;
```

The magnitude of the neighborhood structures' computational complexity usually dictates the order of execution, though this order does not guarantee solution quality. Solution quality relies heavily on instance characteristics (Kalatzantonakis et al., 2019; Subramanian et al., 2010). Besides the execution order, another point of interest is the mechanism the algorithm employs when a better solution occurs. Productive research on this mechanism yielded several VND variants (Hansen et al., 2019b). When an improvement is found, the algorithm shifts to that solution and the search either:

a. Basic VND: returns to the first neighborhood structure and continues.
b. Cyclic VND: moves to the next neighborhood and continues.

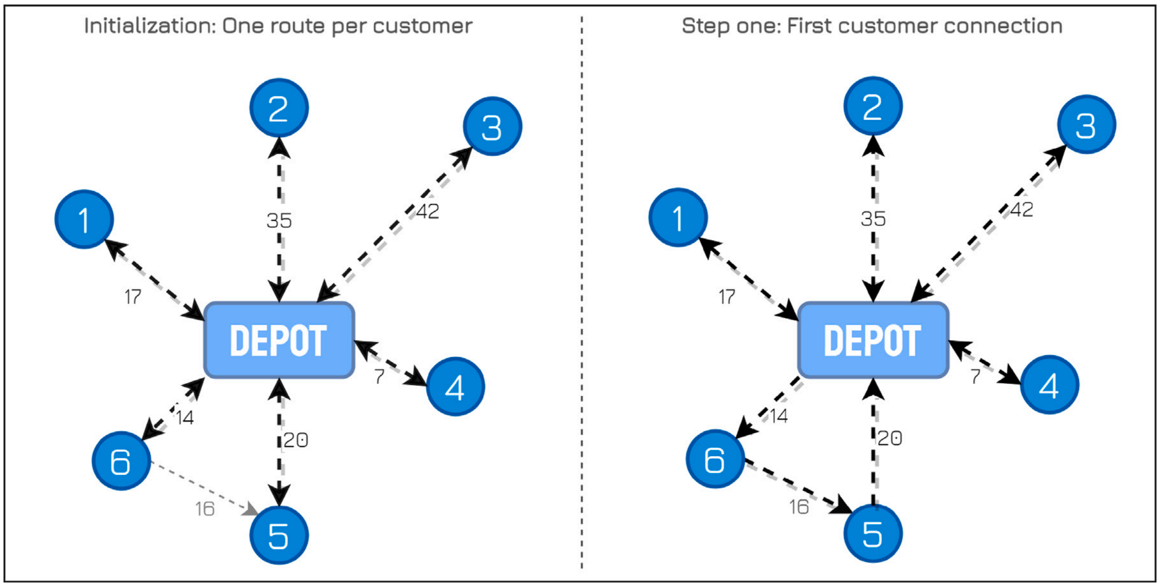
Initialization: One route per customer
Step one: First customer connection
DEPOT
1
2
3
4
5
6
17
35
42
7
14
20
16

Fig. 1. Clarke and Wright Savings algorithm applied in a CVRP instance. On the left, we illustrate the connections with the depot that are formed on initialization. On the right, the connections are dropped to form a route between two nodes in the next iteration.

c. Pipe VND: continues in the same neighborhood structure.
d. Union VND: continues in the same large neighborhood structure.

4.1. Initial solution

An initial feasible solution is generated using the Clarke and Wright Savings algorithm (CWS), one of the most well-known construction heuristic for the VRP developed by Clarke and Wright (Clarke and Wright, 1964). The CWS algorithm's central concept is based on calculating the savings for linking two customers into the same route. The obtained solution is given to Bandit VNS as a warm start. This way, we ensure that we always obtain a feasible solution. In Fig. 1, the first iteration of the CWS algorithm is displayed.

CWS always chooses the operation which causes the most extensive travel distance saving. It relies on iterative connecting routes to create a single route that maximizes cost saving. The well-known expression of CWS is depicted in Eq. (3).

$$S_{ij} = distance(Depot, i) + distance(Depot, j) - distance(i, j) \tag{3}$$

The savings from the primary route formed between nodes five, six and the depot, in the first iteration as depicted in Fig. 1 is calculated using Eq. (3).

$S_{5,6} = 20 + 14 - 16 \Rightarrow S_{5,6} = 18$

4.2. Neighborhood operators

Given a set S that contains all the possible solutions of a combinatorial problem, a neighborhood operator N is a function that maps a given solution $s \in S$ to a neighborhood of solutions $N(s) \subseteq S$. The neighborhood structure our model uses, consists of five neighborhood operators, namely: relocate(1), relocate(2), move, swap, and 2-opt. The neighborhood operators and their analytical description are shown in Table 2. Fig. 2 provides the schematic diagram of the neighborhood operators.

5. Reinforcement learning methodology

In the subsequent subsections, we introduce Bandit VNS, the implementation of several UCB algorithm variants into the GVNS metaheuristic, the reward-regret strategy, the blocking and acceptance criteria of neighborhoods, and the inclusion of ADWIN to mitigate the concept drift.

5.1. Learning with multi-armed bandits

It is well known that a suitable trade-off between exploration and exploitation is imperative for global optimization performance. This trade-off has a significant effect both on accuracy and convergence speed. The selection of the proper neighborhood structure at a certain point in the search or after an improvement relies on many factors and determines the framework's success, as described in the previous section. The main objective of this paper is to evaluate the performance of the proposed adaptive search in VNS. This decision-making process of selecting a neighborhood structure can be considered a non-stationary multi-arm bandit problem.

In the classic multi-armed bandit problem, an agent must choose an arm to pull from a set of K arms with unknown reward models. The decision is made based on the agent's knowledge from previous actions. Every action returns a random reward drawn from the probability distribution of the arm pulled. The agent's goal is to discover policies that minimize the expected regret or maximize their reward.

The Multi-arm Bandit problem was introduced by Thompson (1933) and has been used as a classic model for sequential decision making. Thomson focused on Bernoulli rewards and introduced a Bayesian algorithm now known as Thompson sampling. Later, Robbins (1952) recognized the problem's significance and constructed convergent population selection strategies giving a frequentist perspective. Lai and Robbins (1985) established a lower bound on the regret by proving that the minimum regret has a logarithmic order, and developed policies that produce the lowest regret for several well-known reward distributions (Bernoulli, Poisson, Gaussian, Laplace). Lai, (1987) introduced the renowned upper confidence-bound procedures for solving MAB. This feat is achieved by calculating a UCB for each arm's expected reward at each time and select the arm with the highest UCB.

MAB algorithms can be subdivided into categories according to the problem formulation. Some of the most important categories are:

a. Fitness Rate Rank MAB (FRRMAB). In this case, we consider that an action's reward distribution can also change by selecting other actions. FRRMAB adapts UCB1 and can include a sliding window with the selected actions' recent performance.
b. Contextual MAB. The algorithm selects an action based on a given context (side information). LinUCB algorithm (Li et al., 2010) is used to solve contextual bandit formulations.

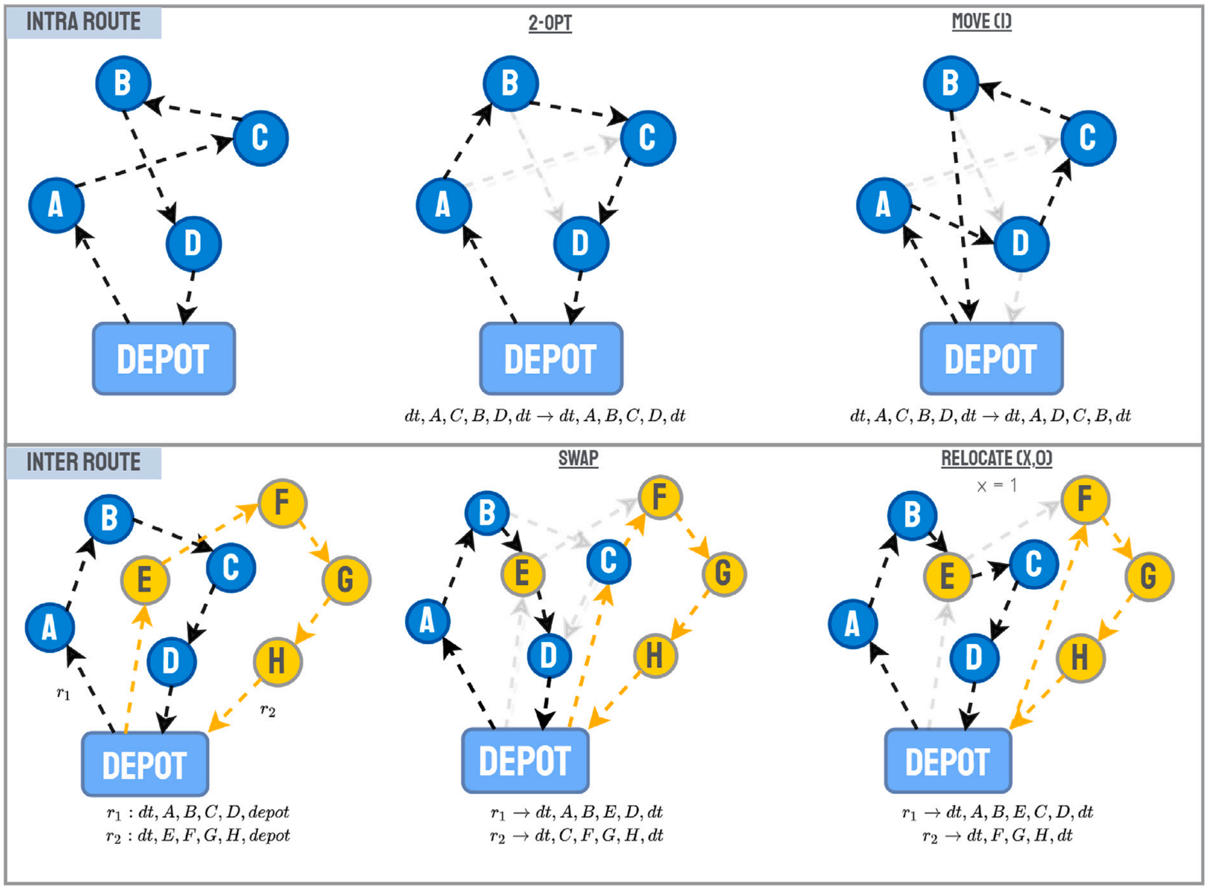
INTRA ROUTE
2-OPT
MOVE (1)
DEPOT
$dt, A, C, B, D, dt \rightarrow dt, A, B, C, D, dt$
$dt, A, C, B, D, dt \rightarrow dt, A, D, C, B, dt$
INTER ROUTE
SWAP
RELOCATE (X,0)
x = 1
r_1
r_2
$r_1 : dt, A, B, C, D, depot$
$r_2 : dt, E, F, G, H, depot$
$r_1 \rightarrow dt, A, B, E, D, dt$
$r_2 \rightarrow dt, C, F, G, H, dt$
$r_1 \rightarrow dt, A, B, E, C, D, dt$
$r_2 \rightarrow dt, F, G, H, dt$

Fig. 2. Schematic diagram of intra and inter route neighborhood operators.

Table 2
Neighborhood operators used by Bandit VNS to improve solutions.

Class	Operator	Description
Intra-route	2-Opt	Remove two edges from the route and reconnect the endpoints
	Move (1)	Move customer to a new position in the same route
Inter-route	Swap(1,1)	Exchange customers positions across two routes
	Relocate(1,0)	Relocation of a customer from one route to another one
	Relocate(2,0)	Relocation of two customers to a different route

c. Restless MAB deterministically splits the time horizon into epochs of exploration and exploitation with carefully controlled epoch lengths that increase geometrically over time.
d. Non-stationary MAB. The reward distributions are not stationary and are mainly associated with a context (side information). Hybrid UCB1 is used, coupled with statistics. In particular, the discounted UCB and the sliding window UCB.

Researchers introduced various algorithms to solve Multi-armed bandit problems. The classic MAB algorithm determines the heuristic quality as the average reward achieved by the heuristic. Classic MAB often uses the Upper Confidence Bound (UCB1) algorithm by Auer et al. (2002) after being proved optimal w.r.t. the maximization of the cumulative reward.

In the non-stationary multi-armed bandit problem, the agent must deal with the possibility of a changing environment. While an agent traverses the search space, shifts in the reward magnitude affect the reward distribution. Those changes represent an extra layer of difficulty due to the agent's need for predicting those variations in a time-efficient manner. Diverse approaches are used to obtain satisfactory results when dealing with non-stationary bandits. One approach is to assume that the distributions change infrequently or drift slowly. Alternatively, the agent can adapt to the changes by detecting variations in the environment. By using this method, the agent can add context, thus making the environment stationary. In this work we use Adaptive Windowing (AdWin). This approach is further discussed in Section 5.5.

5.2. Upper confidence bound algorithms

The Upper Confidence Bound is phrased as being optimistic in the face of uncertainty, as defined in Kuleshov and Precup (2014) and has been proved to be rate-optimal w.r.t. maximization of the cumulative reward (Auer et al., 2002) up to a constant factor. The name of this family of algorithms stems from the fact that the upper bound plays a leading role in action selection, since we are searching the action with the highest reward rate. The following algorithms were used to identify a suitable strategy.

5.2.1. UCB1

UCB1 is the first algorithm we examine from the UCB family of algorithms. The UCB1 algorithm yields an optimal asymptotic bound on the regret in $O(log(N))$ in a stationary environment. This algorithm relies on a straightforward application of Hoeffding's inequality; therefore, it is free from any prior knowledge on how the distribution resembles.

In Hoeffding's inequality, displayed in Eq. (4), we denote $Q(a)$ as the true mean, $\hat{Q}_t(a)$ as the sample mean, $N_t(a)$ as the number of times an action was taken, and $U_t(a)$ as the upper confidence bound. Picking $p = e^{-2tU_t(a)^2}$ as an upper bound, gives us high confidence that the true mean is below the sample mean. If we set $p = t^{-4}$ (heuristic approach) then we get UCB1.

$$\mathbb{P}[Q(a) > \hat{Q}_t(a) + U_t(a)] \leq e^{-2tU_t(a)^2}$$
$$\xrightarrow{p=e^{-2tU_t(a)^2}} U_t(a) = \sqrt{\frac{-logp}{2N_t(a)}} \xrightarrow{p=t^{-4}} U_t(a) = \sqrt{\frac{2\log t}{N_t(a)}} \tag{4}$$

Applied to the GVNS metaheuristic, an agent's policy is to choose an action in every iteration t that maximizes UCB1 using the following criterion:

$$A_t \doteq \underset{j=\{1...l\}}{\arg\max} \left[\hat{p}_t(a_j) + C\sqrt{\frac{2\log \sum_1^l n_l}{n_t(a_j)}} \right] \tag{5}$$

The right-hand part of Eq. (5) is composed of two distinct parts. The first, $\hat{p}_t(a_j)$, denotes the average empirical reward received, in turn, t by the neighborhood structure a with index j. Thus, it represents the expected reward in response to the selected action and is linked with the performance, thus favoring the exploitation.

The second part is associated with exploration since the agent explores the action space based on the uncertainty of an action's expected reward. The concept is that the square-root component evaluates the uncertainty or variance in assessing the reward of an action (Sutton and Barto, 2018). By $n_t(a_j)$, we denote the number of times the neighborhood operator with index j has been selected prior to time t. Consequently, when $n_t(a_j)$ approaches zero, the uncertainty of the actions that have not been selected increases. By l we denote the number of neighborhood operators, hence $\sum_1^l n_l$ is the number of iterations or more specifically the total number of operators selected up until time t. $C > 0$ is a scaling factor needed to balance the trade-off between exploitation and exploration by controlling the degree of exploration. Essentially the scaling factor C defines the confidence level. Also, C usually ranges between 1 and 2.

Additionally, we examine a modification of the UCB1 algorithm, termed UCB1Exp, by setting the exploration component equal to:

$C * \frac{\log \sum_1^l n_l}{n_t(a_j)}$, thus promoting exploration when $\log \sum_1^l n_l > \frac{n_t(a_j)}{2}$

When an operator has been chosen less than half of the total execution runs, and the operators to select are more than two, UCB1Exp promotes exploration more than the typical UCB1.

5.2.2. Bayesian UCB

Bayesian UCB algorithm (Kaufmann et al., 2012) is another popular approach for solving MAB problems. In contrast with the UCB1 approach, we can make a more reasonable bound estimation if we know the distribution upfront. By assuming the rewards of each action are normally distributed, we can replace the second term from Eq. (5) with the standard deviation of the neighborhood reward as seen in formula (6).

$$C\frac{\sigma(q_t(a_j))}{\sqrt{n_t(a_j)}} \tag{6}$$

The resulting action selection equation is in the form of Eq. (7) where with $\sigma(q_t(a_j))$ we denote the standard deviation of neighborhood j's rewards at time t and $n_t(a_j)$ indicates the number that the neighborhood with index j has been selected up until the time t. In this context, C is a hyper-parameter for defining the size of the confidence interval we want to set to a neighborhood's mean observed reward; a typical 95% confidence interval can be realized with $C = 1.96$.

$$A_t \doteq \underset{j=\{1...k\}}{\arg\max} \left[\hat{p}_t(j) + C\frac{\sigma(q_t(j))}{\sqrt{n_t(j)}} \right] \tag{7}$$

5.2.3. UCB-tuned

UCB-Tuned is another addition to the UCB-style algorithms (Auer et al., 2002) that we examine. This algorithm is a refinement of UCB1 that, besides the arithmetic mean of an action, it also considers the variance in the bias sequence. This approach's logic is that variance-aware algorithms can quickly locate sub-optimal actions, thereby diminishing the total regret. The algorithm, following the same modus operandi, selects an action by finding the argument that gives the maximum value:

$$A_t \doteq \underset{j=\{1...k\}}{\arg\max} \left[\hat{p}_t(j) + \sqrt{\frac{\log t}{n_t(j)} \min(\frac{1}{4}, V_t(j))} \right] \tag{8}$$

where

$$V_t(j) = \hat{\sigma}_t^2(j) + \sqrt{\frac{2\,\text{logt}}{n_t(j)}}$$

The factor $1/4$ signifies an upper bound on the variance of a Bernoulli random variable (Auer et al., 2002). The authors of UCB1-Tuned did not provide theoretical guarantees for the regret bounds, but later, Audibert et al., (2009) presented expected regret bounds concerning variance-aware UCB-type algorithms, comparable to UCB1-Tuned. The observation and collection of data regarding the performance of variance-aware bandits from Kuleshov and Precup (2014) indicate that variance is perhaps the only characteristic of a bandit that need to be considered.

5.2.4. Gradient UCB

Gradient UCB algorithms (Sutton and Barto, 2018) offer a different approach for selecting the best action. The UCB methods mentioned above are focusing on estimations of the action value to select actions. An entirely different strategy can be formed using Gradient Bandit Algorithms, that are based on Stochastic Gradient Ascent. The main objective of Gradient Bandit Algorithms is to learn a numerical preference for each action $a \in A = \{a_1, a_2, \dots, a_n\}$, denoted by $H_t(a)$. The optimization of $H_t(a)$ parameters aim to maximize the Expected Reward based on the following optimization function:

$$f(H_0(a), H_1(a), \dots, H_n(a)) = \sum_{a\in A} \pi(a) * \mathrm{E}[r|a] \tag{9}$$

At time $t = 0$, Gradient UCB starts with $H_0\ \forall\ a$. The action probabilities are defined under a soft-max distribution as seen in Eq. (10). With $\pi_t(a)$, we denote the probability of selecting the action a at time t based on a stochastic policy.

$$Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}} \doteq \pi_t(a) \tag{10}$$

At every step, upon performing an action A_t and obtaining a reward signal R_t, the algorithm updates the action preference values by applying the following rule:

$$\begin{aligned} H_{t+1}(A_t) &\doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(At)) \\ H_{t+1}(a) &\doteq H_t(a) + \alpha(R_t - \bar{R}_t)\pi_t(a), \qquad for\ all\ a \neq A_t \end{aligned} \tag{11}$$

To select an action, we pick a random action weighted with the action preference values H as weights. Following this weighted random selection policy, the action with a higher expectation of reward has more probabilities of being chosen. The difference between the reward R_t of the action taken and the average reward $\bar{R}_t$ up to (and including) time t is weighted by the learning rate α and the likelihood $\pi_t(At)$ of the action taken at time t. The result is the new value preference $H_t(A_t)$.

In essence, the process of learning is the update process that measures the relative value of a given action over and above the other available actions. $\bar{R}_t$ functions as a reference point in which it is determined if the likelihood of choosing action A_t in the future is increased or decreased. If R_t is greater than $\bar{R}_t$ then $H(A_t)$ increases while non-selected actions preference value decreases. All probabilities are correlated to one another; therefore, they are all updated in one turn.

We also introduce and test a hybrid UCB algorithm by adopting the Gradient UCB reward method seen in Eq. (11) and the UCB1 action selection scheme in Eq. (5), termed Hybrid Gradient UCB (HG-UCB).

5.3. Guiding local search with UCB

Bandit VNS regulates the distribution of the search effort by choosing operators. Operator selection is following the information accumulated before and during the search process. The selection process is specifically based on two sources of information: the operator execution count and the operator's objective value. This objective value is a metric identified by the operator's historical performance, and it is iteratively updated through a reward function determined by various feedback signals. The reward function address the problem of credit assignment by determining how action's outcome contribute to the overall performance of the hyperheuristic framework. Credit assignment relies on several parameters and is analyzed in Section 5.4.

The pseudocode of the bandit-based neighborhood selection approach is presented in Algorithm 3. The algorithm iterates as long as the time constraint is not violated. The first step is to perform a shake. Shaking is achieved by randomly invoking a neighborhood operator and it serves two objectives, i.e., to diversify the solution and to escape from a local optimum. The inner loop iterates provided an improvement is accomplished, and the values assigned to each operator (q_values) do not fall under the permissible lower limit (min_q). Each action is evaluated using the $credit$ function following the policy to minimize the regret.

The algorithm maintains an action-value function called $UpdateQ$ responsible for assigning an objective value to every operator. Bandit VNS core component process is responsible for operator selection, the UCB function, is presented in Algorithm 4. At any point s in time (time-step), the neighborhood gets selected using the UCB algorithm or picked randomly from the neighborhood pool. Inputs n and p are streams of data, each of them being an array containing the execution count and the rewards record, respectively. C is a scaling factor when UCB1 and Bayesian scheme is used. In Line 6 of Algorithm 4 UCB1 is depicted. Alternative selection strategies can be derived by replacing the second part of this line with any of the UCB algorithm described in Section 5.2.

5.4. Credit assignment

Bandit VNS adopts a cooperative, multi-step learning procedure where the outcome of an action referred to as reinforcement signal is infrequent and delayed in time. Identifying which actions or sequence of actions lead to better rewards can present a challenge, termed as the credit assignment problem. Credit assignment occurs online by aggregating various feedback indicators in order to transform feedback information into a suitable numerical reward based on an action's outcome.

Our hyper-heuristic framework utilizes a novel, non-binary reward scheme, adopting a different strategy for positive and negative feedback signals. Rewards, which are not directly used in the UCB procedure, are crucial for removing operators that under-perform from the operator pool that the UCB procedure explores.

In the case of positive feedback, Eq. (12) determines the reward. The new solution, sol_c is always smaller than sol_g for minimization problems. The right-hand part of Eq. (12) is formed from two components, r_s on the left, which is the suggested base reward, is the distance between the best global and the current solution. On the right, $N_t(u)$ is the subset of the neighborhood operators that are still in the operator pool (unblocked), and $n_t(k)$ is the execution count of the neighborhood with index k at step t. This component's role is to increase the reward

Algorithm 3: Bandit VNS

```
Input  : $x$ : current solution, $C$ : scaling factor, $t_{max}, s_{max}$ : max steps,
         $w$ : sliding window, $min_q$ : min q_value,
         $k_{max}$ : neighborhood count
Output: solution $x$
1  initialize($n, p, \hat{p}$);
2  while $t \le t_{max}$ and $step \le s_{max}$ do
3      $x' \leftarrow Shake(x, k_{max})$; // Given x, execute $\mathcal{N}_{l=rand(0,k_{max})}$
4      while true do
5          $improvement \leftarrow False$;
6          $n \leftarrow n+1$;
7          if $adaptive\_windowing(p)$ then
8              $reset(n, p)$; // Concept drift detected. Reset average.
9              $l \leftarrow rand(0, k_{max})$; // Select random neighborhood index
10         else
11             // Select neighborhood index with preferred ucb scheme
12             $l \leftarrow UCB(C, step, n, \hat{p})$;
13             $\hat{p} \leftarrow Sliding\_avg(p, w)$; // sliding window average reward
14         end
15         $x'' \leftarrow \arg\min_{y \in N_{l(x')}} f(y)$; // With $x'$ from Shake, execute
               $\mathcal{N}_{l=l_{ucb}}$
16         if $(f(x'') < f(x'))$ then
17             $x \leftarrow x''$; // Update new best solution
18             $improvement \leftarrow True$;
19         end
20         $reward \leftarrow credit(x, x', step, n)$; // action − reward
21         $p.append(reward)$;
22         // update q_values and normalize with softmax
23         $UpdateQ(q\_values, reward)$;
24         if no improvement & $max(q\_values) < min_q$ then
25             break;
26         end
27     end
28     $t \leftarrow CpuTime()$;
29     $step \leftarrow step + 1$;
30 end
31 return x;
```

Algorithm 4: UCB1 Adaptive operator selection

```
Input  : $C, s, n, \hat{p}$
Output: Selected neighborhood index
1  // $s$ : timestep, $k$ : neighborhood count
2  if operators that have not been selected exist then
3      $x \leftarrow$ uniform selection from operator pool;
4  else
5      // e.g. ucb1 selection scheme
6      $x \leftarrow \arg\max_{j=\{1...k\}}(\hat{p}_j + C\sqrt{\frac{2\log(s)}{n_j}})$;
7  end
8  $n_x \xleftarrow{x \in N_k} n_x + 1$;
```

as solution distribution gets more sparse. Taking the limit, when t tends to infinity, the reward is equal to the distance.

$$reward = r_s * N_t(u)^{\frac{-1}{n_t(k)}} \tag{12}$$

where

$$r_s = \left| sol_g - sol_c \right|$$

In contrast, when the operator yields a solution with a worse value, Eq. (13) determines the negative reward. Although similar, the difference between Eqs. (12) and (13) is that r_s is replaced with q_{t-1}, in Eq. (13). q_{t-1} represents the normalized reward that this operator

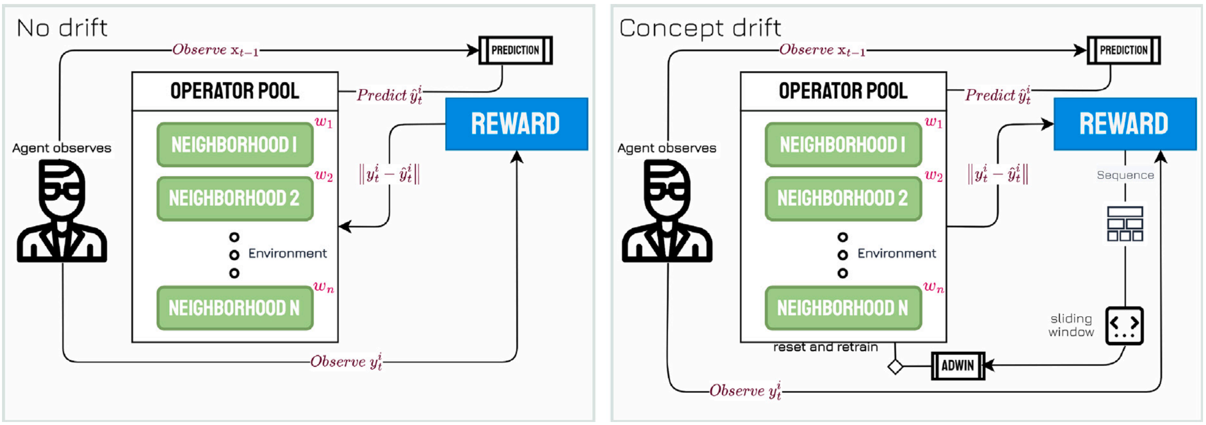

Fig. 3. Contextual bandit scheme, actions are conditioned by the concept drift of the reward sequence. Operators can be excluded in both phases and reinforced upon concept drift.

had at the previous time-step.

$$regret = -\frac{1}{2} + \left(-2q_{t-1} * N_t(u)^{\frac{-1}{n_t(k)}}\right) \tag{13}$$

The resulting reward signals from all operators per iteration ($Qvalues$) are first normalized with the softmax function setting the payoff in $[0,1]$ range. $Qvalues$ are then stacked in a sequence and further processed using ADWIN to detect concept drift, further described in the next section. Additionally, we are utilizing a sliding window with a fixed size W. Our motivation for designing a sliding window for the reward sequence was to form a short-term memory of the reward signals. Our model uses the memory of the operator's most recent performance for online filtering. If the reward signal for a specific operator drops below a threshold value b_t, filtering is used to exclude operators from the selection process. The process described above is the algorithmic basis of credit assignment, filtering, and drift detection outlined in the pseudocode of the Algorithm 3, Lines 6-28.

Fig. 3 presents the schematic diagram of the contextual bandit. By including drift detection, the agent can experiment with various actions and determine the most rewarding outcome when conditions change significantly. The agent can pick a random action or increase the second and third candidate action weights and choose the best one. In case of concept drift we restart the learning process.

If an operator's reward drops below the threshold value b_t, the operator is removed from the pool until the next iteration starts. The value of the filter parameter b_t is inspired by the Pareto Principle (Newman, 2005). When the maximum value of the q_values drops below 0.2, then execution terminates. Other b_t values that were tested to analyze the behavior of the models (0.1 and 0.05) did not produce better results.

5.5. Concept drift detection

Over time, changes to underlying relations in data occur unexpectedly. This phenomenon is called concept drift. It constitutes a problem in machine learning because models lose their predictive power.

The UCB algorithm makes a decision partly based on the rewards, assuming that an action's reward distribution stays stationary. As discussed in Section 5.1, in dynamically changing and non-static environments, the expected reward distribution may change, generating a concept drift event (Schlimmer and Granger, 1986; Widmer and Kubat, 1996), so the UCB algorithm must adjust its estimation dynamically.

Concept drift emerges in the reward stream, while the agent defines a route along which rewards are not constant as it traverses the solution domain. Concept drift events are listed in Fig. 4.

We considered concept drift to ensure that the agent will retain the accuracy to predict the neighborhood structure with the optimal reward. For CVRP, this means that the order of execution of the neighborhood structures during the search will be chosen to benefit the solution quality.

We use ADWIN to reset the learning process upon a major concept drift event. The ADWIN algorithm (Bifet and Gavalda, 2007; Bifet et al., 2018; Montiel et al., 2018), named after Adaptive sliding Window, is an efficient concept drift detector and estimation algorithm. More specifically, we use the ADWIN approach to monitor and detect abrupt variations in the standard deviation of the mean rewards. ADWIN slides a window w on the data stream and operates in $O(log(n))$ time and $O(log(n))$ space; by n we denote the window size at any given moment. If a change is detected, the size of the window will decrease. The main parameters of the algorithm are the window size W and the delta d. W parameter regulates the trade-off of space and accuracy. As W grows, accuracy increases, however, at the expense of the execution speed. Delta parameter defines the confidence (sensitivity) of the algorithm to the concept drifts. In essence, it represents an upper bound on the false-positive rate for identifying concept drifts.

Window size changes dynamically over time based on the frequency of change identified in the data. Our experiments used the Window size parameter (5, 10, and 50) to set a pre-determined period to trigger change detection. A relatively big window size ($W = 50$) was set to follow the high volatility of the reward at the start of the algorithm execution. Sensitivity d, which controls the sensitivity to change, is undoubtedly the most critical parameter. We retained the default value of 1% and also tested a value of 2%.

6. Computational experiments

In this section, we present a thorough overview of the methodology used, the UCB algorithms' hyper-parameters and the CVRP instances' characteristics. Lastly, we apply our methodology to compare six implementations, evaluate the best scheme, and compare it with a GVNS scheme.

6.1. Instances

The vehicle routing problem instances from the literature vary greatly. Some are computer-generated, while others are taken from real-life problems and have diverse warehouse locations and customer location configurations.

Our computational tests were designed to provide a well-balanced test environment and carried out using carefully selected instances from the CVRP library (CVRPLIB). Specifically, in our implementation, we picked instances from the set A, B (Augerat et al., 1995), the set X (Uchoa et al., 2017) and the set CMT(Christofides, 1979). The chosen instances have 32–120 customers and 5–25 vehicles. Set A customer's

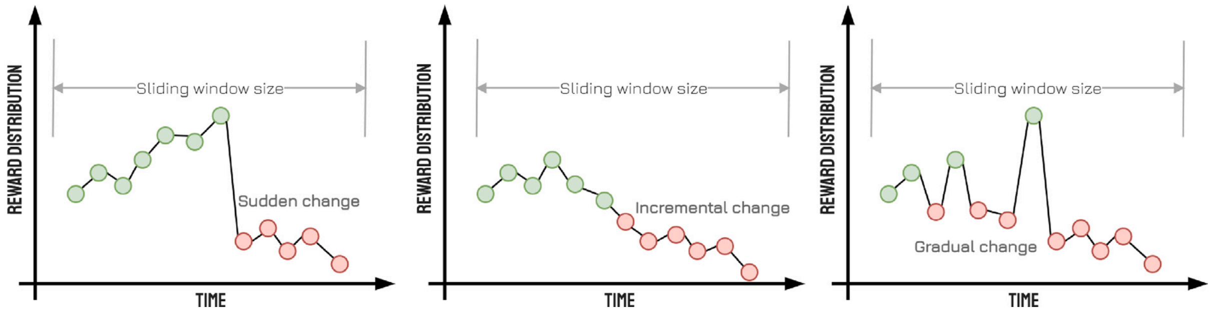

Fig. 4. We distinguish three types of Concept drift events according to the data distribution changing speed.

Table 3
Key characteristics of the selected instances from set A,B and X.

Instance	Capacity	Customers	Vehicles	Description
A-n80-k10	100	79	10	Random customer coordinates Demands uniformly distributed
B-n78-k10	100	77	10	Clustered customer coordinates Demands uniformly distributed
X-n120-k6	21	119	≥ 6	Eccentric depot Random and clustered customer coordinates Demands uniformly distributed

coordinates are generated randomly from a $100 * 100$ grid while the demands are uniformly distributed $U(1, 30)$, though $n/10$ of the demands were multiplied by 3 (Uchoa et al., 2017). The generation process of set B is the same as set A. The primary difference is that customer's coordinates in set B form clusters. The central notion is that clustered data points are visited sequentially, and transitions within clusters demand significant hops than the distances between the same cluster data points. The creation of X-Set instances targets a particular objective to provide a thorough and balanced experimental setting. In contrast to the A and B sets, the X set is heterogeneous, inspired by real-world problems, and obeys the rule of not setting a fixed number of routes; thus, i.e., the instance X-n120-k6 has a lower bound of six vehicles. The customers are grouped into clusters in the CMT set to portray practical cases. CMT1, CMT2, CMT3, and CMT11 are the same with (E-n51-k5, E-n76-k10, E-n101-k8, and M-n121-k7 without rounding the cost matrix and specifying a value for vehicles K. The key characteristics of the selected instances are summarized in Table 3.

In an attempt to achieve hyper-parameter optimization, three instances were carefully chosen because they represent their groups (sets A, B, and X) and summarize the most pertinent characteristics. Since the hyper-parameters of each model may be sensitive to the instance characteristics, this selection is essential for assessing the algorithmic models. This approach also provides an excellent opportunity to reduce the testing time for hyper-parameter tuning and selection, the latter of which is further discussed in Section 6.2. After analyzing the obtained results, the optimal hyper-parameters are utilized to compare the best model against GVNS in all known benchmark instances of sets A, B and several from X set of the CVRPLib.

6.2. Algorithmic components and hyper-parameter settings

The main three algorithmic components in the Bandit VNS approach that the researcher can specify to fine-tune the performance of the algorithm are:

(a) the type of UCB algorithm used to solve the MAB problem,
(b) the credit assignment algorithmic scheme,
(c) the concept drift detection with Adaptive Windowing.

In addition, there are four control parameters in the Bandit VNS framework depending on the UCB algorithm chosen that need to be tuned:

(i) the scaling factor C,
(ii) the Confidence Interval $C.I$,
(iii) the learning rate α,
(iv) the upper bound d, and the window size W.

The scaling factor C is the property that regulates the trade-off between exploration and exploitation and affects the UCB1, as discussed in Section 5.2. Confidence Interval concerns the Bayesian algorithm, and the learning rate (step-size parameter) controls the Gradient bandit behavior. ADWIN is capable of regulating itself to the data stream without requiring hard-coded parameters, nevertheless as discussed in Section 4, W affects accuracy and speed, and d affects the sensitivity.

An analysis is carried over the hyper-parameter configurations to compare the UCB schemes' performance fairly and reveal what settings will yield better solutions. The explored hyper-parameter settings are drawn based on reason and published literature (Sutton and Barto, 2018, Auer et al.; 2002; Kaufmann et al., 2012) and range near the default values. Table 4 presents the chosen hyper-parameter values.

All six algorithms follow the traditional ϵ-greedy policy to alleviate the exploration/exploitation dilemma. ϵ is a noise-related parameter, which usually takes values close to zero. In every iteration, the algorithm will execute as expected; it will select an action α given a state s_t; with a probability of $1-\epsilon$; otherwise, it will select an operator uniformly at random (Eq. (14)) where d is a uniformly-distributed random variable pulled from the set of valid actions A. The noise threshold ϵ was set to 0.1 across all experiments, which is generally considered the default value in literature. This strategy ensures that all the action space is explored.

$$Pr\{A_t = a\} \doteq \begin{Bmatrix} \arg\max\ Q(s_t, \alpha)_{\alpha\in A} & \mathit{with\ probability}\ 1-\epsilon, \\ d \sim u(A) & \mathit{with\ probability}\ \epsilon, \end{Bmatrix} \tag{14}$$

If the value of ϵ were zero, the algorithm would never explore but consistently exploit the knowledge it already has.

In the light of a fair comparison, the general full-factorial Design Of Experiments (DOE) methodology is applied to explore the effects of hyper-parameters and determine the optimal parameters. One hundred executions for each hyper-parameter setting combination at five, ten, and twenty GVNS iterations were launched and generated 33,000 observations for each iteration set.

Table 4
The examined hyper-parameter values for the UCB algorithms.

	Scaling factor (C)	Confidence Interval (CI)	Learning rate (a)	Window size (W)	Delta (d)	Epsilon (ϵ)
UCB1	0.5, 1.0, 1.5, 2.0	–	–	5, 10, 50	1%, 2%	10%
UCB Tuned	–	–	–	5, 10, 50	1%, 2%	10%
Bayesian UCB	–	90%, 95%, 99%, 99.9%	–	5, 10, 50	1%, 2%	10%
Gradient UCB	–	–	0.1, 0.5, 0.9, 1.5, 2.5, 3.0, 4.0, 5.0	5, 10, 50	1%, 2%	10%
UCB1Exp	0.5, 1.0, 1.5, 2.0	–	–	5, 10, 50	1%, 2%	10%
HG-UCB	0.5, 1.0, 1.5, 2.0	–	0.1, 0.5, 0.9, 1.5, 2.5, 3.0, 4.0, 5.0	5, 10, 50	1%, 2%	10%

Table 5
Normality test for the algorithmic models. With red and pink, we denote the rejection of the null hypothesis for a significance level of $\alpha = 0.01$ and $\alpha = 0.05$, respectively. With green color, we denote algorithms that exhibit a normal distribution.

Model	Statistic	Normal Distribution	Fitting of normal distribution	
		p-value	Information Criterion	Loglikelihood
UCB1	Kolmogorov–Smirnov	0.0003	AIC: 6272.9150 BIC: 6283.9710	−3134.4570
	Cramer–von Mises	0.0010		
	Anderson–Darling	0.0006		
	Shapiro–Francia	0.0007		
UCB Tuned	Kolmogorov–Smirnov	**0.6645**	AIC: 2526.5370 BIC: 2535.7500	-1261.2680
	Cramer–von Mises	**0.7762**		
	Anderson–Darling	**0.5306**		
	Shapiro–Francia	**0.3157**		
Bayesian UCB	Kolmogorov–Smirnov	0.0178	AIC: 13693.7500 BIC: 13706.3800	−6844.8770
	Cramer–von Mises	0.0333		
	Anderson–Darling	0.0130		
	Shapiro–Francia	0.0015		
Gradient UCB	Kolmogorov–Smirnov	0.0000	AIC: 12252.2400 BIC: 12264.6900	-6124.1220
	Cramer–von Mises	0.0001		
	Anderson–Darling	0.0000		
	Shapiro–Francia	0.0009		
UCB1Exp	Kolmogorov–Smirnov	0.0468	AIC: 6202.4490 BIC: 6213.5050	−3099.2240
	Cramer–von Mises	**0.1113**		
	Anderson–Darling	**0.0584**		
	Shapiro–Francia	0.0432		
HG-UCB	Kolmogorov–Smirnov	0.0039	AIC: 13409.0200 BIC: 13421.5900	-6702.5100
	Cramer–von Mises	0.0158		
	Anderson–Darling	**0.0107**		
	Shapiro–Francia	0.0279		
Classic VNS	Kolmogorov–Smirnov	0.0106	AIC: 2341.0900 BIC: 2350.3030	−1168.5450
	Cramer–von Mises	0.0004		
	Anderson–Darling	**0.0052**		
	Shapiro–Francia	**0.0119**		

7. Computational results

In this section we report and analyze the findings of the computational experiments performed to assess and test the effectiveness of the Bandit VNS framework on the selected capacitated vehicle routing problem instances.

The Bandit algorithms were implemented using Python 3.9 compiled in C using Cython 0.29. The experiments were performed using an Intel© Core^{TM} i9 7940X CPU central processing unit clocked at 3.50 GHz with 32 GB of memory and 28 parallel threads under the Microsoft Windows 10 operating system. The parallel GVNS implementation scheme was based on the parallel cooperative model presented in Kalatzantonakis et al. (2019). All statistical analyses were performed in R statistics, version 4.2.1 (R. Core Team, 2020).

Before advancing to the comparative performance evaluation, it is essential to verify that the difference in the mean accuracy between the models is statistically significant. In order to do that, we tested whether parametric or non-parametric tests can be used. Four different goodness-of-fit tests were utilized, including Anderson–Darling, Cramer–von Mises, Lilliefors Kolmogorov–Smirnov, Shapiro–Francia. Furthermore, standard Goodness-Of-Fit (GOF) criteria were employed using Fitting distributions (Myung, 2003; Delignette-Muller and Dutang, 2015) such as the AIC (Akaike Information Criteria), the BIC (Bayesian Information Criterion), and the Log-likelihood. The level of significance for all tests was set at $p < 0.05$.

To assess the performance of each algorithm, we computed the difference between the instances' declared upper bound or the optimum solution provided by CVRPLib, using equation (15). When error grows,

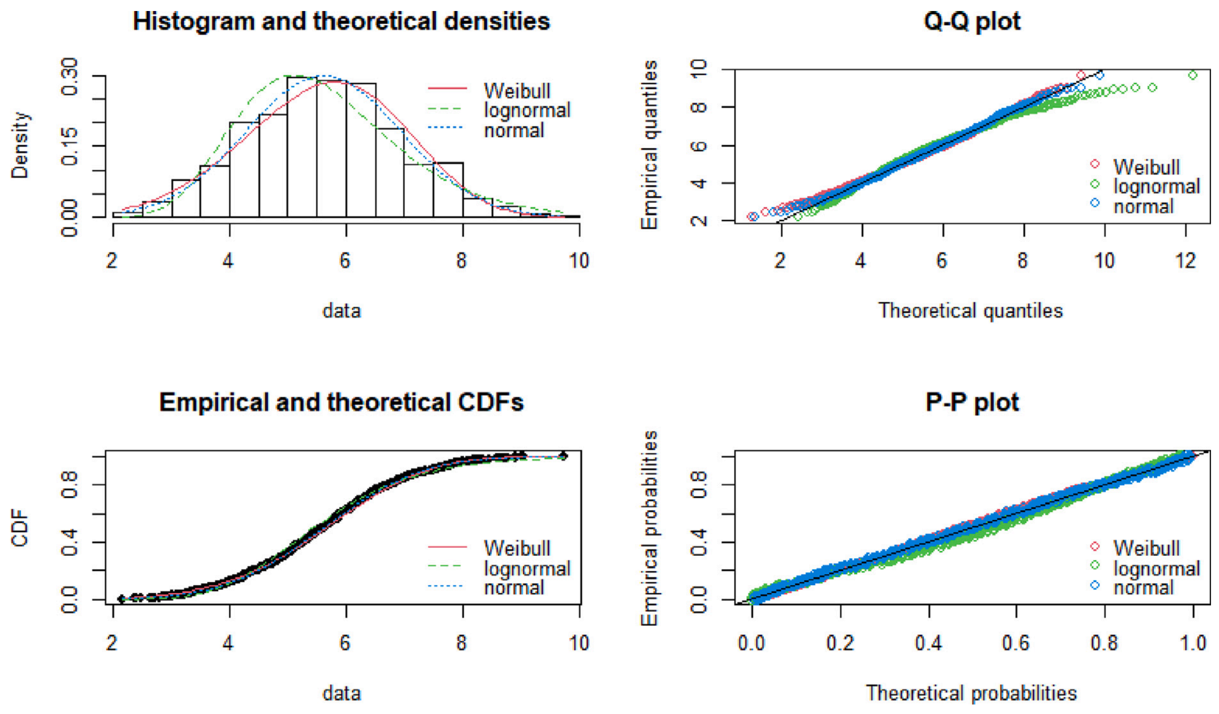

Fig. 5. Visual inspection of the normality plots for UCB1EXP.

Table 6
Best performing model (averaging all hyper parameter settings).

Rank #	Algorithmic model	Average mean error	Average median error
1	HG-UCB	5.3443	5.3965
2	UCB1EXP	5.4131	5.4760
3	UCB1	5.4523	5.5376
4	Bayesian UCB	5.5352	5.5933
5	UCB Tuned	5.5966	5.6363
6	Gradient UCB	5.8355	5.9201
7	CLASSIC GVNS	5.9372	5.9961

solution quality degrades. The performance constitutes the contributing data points we employ to test for normality of each algorithm.

$$error = \frac{(Optimum - Objective\ value)}{Optimum} \tag{15}$$

In all cases except UCB Tuned, the null hypothesis H_0 of normality of the error structure is rejected. Table 5 summarizes the goodness-of-fit statistics applied on the data set. In line with this, the alternative hypothesis was accepted, that the data exhibited non-normal distribution when the data under analysis failed normality tests.

For UCB1EXP, a visual investigation of the normality plots in Fig. 5 showed a slight deviation from normality and as expected, few positive outliers and a cluster around a central value. Therefore non-parametric tests (distribution-free) were used to analyze all models except UCB Tuned and UCB1EXP that conformed to normality.

The algorithms' performance metrics results, grouped by model, are summarized in Table 6. The results display the performance for each model, on average, including all the hyper-parameter setting tests.

The most beneficial hyper-parameter settings for each model that lead to enhanced performance in terms of action accuracy, error performance measurements (mean and median error), and total execution time grouped by GVNS step are presented in Table 7. The action accuracy (AAC) performance is measured by counting the average neighborhood executions within a GVNS iteration per second per thread (the higher, the better).

For all models the best Adwin parameters were $W = 5$ and $d = 0.02$, thus, Adwin parameters are omitted from Table 7. In the order of significance of the main performance factors, the first position is allocated to the mean error, the second to the median error, and the action accuracy.

Focusing only on the hyper-parameters mentioned above that consistently gave the best outcomes, we ranked the performance of all seven models (1: best and 7: worst). Table 8 shows the final ranking.

Wilcoxon signed-rank test and Student's t-Test were used for non-parametric and parametric paired samples, respectively, to evaluate the pairwise statistical performance of the algorithms. In both statistical experiments, Bonferroni corrections were applied with a level of significance of $\alpha < 0.05$. We can reject the null hypothesis (H_0), that the median difference of the algorithmic pair is zero, when $\alpha \leq 0.05$. Table 9 displays the results.

The location shifts from H_0 that the Wilcoxon and Student's test attempts to identify are not significant enough to reject the null hypothesis for UCB1, UCB1Exp, and Bayesian UCB. Those three algorithms failed to differentiate on pairwise comparison. Gradient UCB also failed to differentiate from UCB Tuned, and misses statistical significance by 1.24%. In the rest of this analysis, we focus on HG-UCB and UCB1 as those are the models that produce satisfactory results, and there is a statistically significant difference between them.

A statistical solution quality analysis based on the average error (according to Eq. (15)) was generated by analyzing and averaging the data output of the best hyper-parameters for each GVNS step. A box-plot, visualization of all UCB models, and the GVNS metaheuristic framework is displayed in Fig. 6. As shown in Fig. 7, execution time in HG-UCB is significantly better than all others UCB models except Gradient and Bayesian UCB. In the rest of this analysis, we focus on HG-UCB and UCB1 as those are the models that produce satisfactory results, and there is a statistically significant difference between them. Also, in Figs. 8 and 9, the impact of instance characteristics on the performance of all algorithms is visualized.

After calibrating a range of hyper-parameters for all models, the next step was to run similar experiments utilizing the best model with the most promising parameters against several CVRP instances. The most accurate model was HG-UCB based on the model assessment scores, such as statistical potentials of mean error, robustness against time and GVNS steps.

Table 7
Hyper parameter settings that yielded the best objective value.

Model	GVNS Steps	Hyper parameters	GVNS Steps	Hyper parameters	GVNS Steps	Hyper parameters
UCB1	5	**C**: 0.5	10	**C**: 0.5	20	**C**: 1.0
Bayesian UCB	5	**C.I**: 90%	10	**C.I**: 95%	20	**C.I**: 99%
Gradient UCB	5	**C**: 0.5 **a**: 2.0	10	**C**: 1.0 **a**: 2.0	20	**C**: 2.0 **a**: 2.0
UCB1EXP	5	**C**: 2.0	10	**C**: 0.5	20	**C**: 0.5
HG-UCB	5	**C**: 1.0 **a**: 5.0	10	**C**: 2.0 **a**: 5.0	20	**C**: 2.0 **a**: 5.0

Table 8
Model ranking based on best hyper parameter settings.

		GVNS steps : 5			
Rank #	Model	Mean	Median	AAC	Mean CPU time
1	HG-UCB	6.1832	6.1972	0.2450	27.1341
2	UCB1	6.2734	6.3728	0.2590	25.4154
3	Bayesian UCB	6.2946	6.4721	0.2471	26.0476
4	UCB1EXP	6.2995	6.4225	0.2584	25.0518
5	UCB Tuned	6.4237	6.4367	0.2119	47.5727
6	Gradient UCB	6.5720	6.5799	0.3662	16.8914
7	Classic GVNS	6.7467	6.7689	0.2313	36.1216
		GVNS steps : 10			
Rank #	Model	mean	median	AAC	Mean CPU time
1	HG-UCB	5.2497	5.3582	0.2735	47.4241
2	UCB1EXP	5.3355	5.3488	0.2679	50.3411
3	UCB1	5.3744	5.4822	0.2749	47.8597
4	Bayesian UCB	5.4643	5.4456	0.2730	48.5111
5	UCB Tuned	5.6513	5.7142	0.2322	89.1099
6	Gradient UCB	5.7498	5.8226	0.3708	32.7915
7	Classic GVNS	5.9587	6.0004	0.2357	69.9459
		GVNS steps : 20			
Rank #	Model	mean	median	AAC	Mean CPU time
1	HG-UCB	4.1632	4.0848	0.2972	67.9731
2	UCB1EXP	4.4477	4.5347	0.2882	69.9257
3	UCB1	4.4550	4.5154	0.3328	61.0046
4	Bayesian UCB	4.4685	4.5413	0.2925	69.0110
5	UCB Tuned	4.7150	4.7581	0.2489	153.7122
6	Gradient UCB	4.8675	4.8446	0.3871	52.1994
7	Classic GVNS	5.1062	5.2150	0.2378	128.2903

Table 9
Results of Wilcoxon and t-Test for algorithmic model comparison.

Compared methods		Evaluation	
Algorithm 1	Algorithm 2	p Value	Result
UCB1Exp	UCB1	0.9410	Fails to reject H_0
UCB1Exp	UCB Tuned	0.0076	H_0 rejected
UCB1Exp	Classic VNS	0.0000	H_0 rejected
UCB1Exp	HG-UCB	0.0050	H_0 rejected
UCB1Exp	Bayesian UCB	0.4111	Fails to reject H_0
UCB1Exp	Gradient UCB	0.0000	H_0 rejected
UCB1	UCB Tuned	0.0111	H_0 rejected
UCB1	Classic VNS	0.0000	H_0 rejected
UCB1	HG-UCB	0.0030	H_0 rejected
UCB1	Bayesian UCB	0.4596	Fails to reject H_0
UCB1	Gradient UCB	0.0000	H_0 rejected
UCB Tuned	Classic VNS	0.0000	H_0 rejected
UCB Tuned	HG_UCB	0.0000	H_0 rejected
UCB Tuned	Bayesian UCB	0.0269	H_0 rejected
UCB Tuned	Gradient UCB	0.0624	Fails to reject H_0
Classic VNS	HG-UCB	0.0000	H_0 rejected
Classic VNS	Bayesian UCB	0.0000	H_0 rejected
Classic VNS	Gradient UCB	0.0263	H_0 rejected
HG-UCB	Bayesian UCB	0.0000	H_0 rejected
HG-UCB	Gradient UCB	0.0000	H_0 rejected
Bayesian UCB	Gradient UCB	0.0000	H_0 rejected

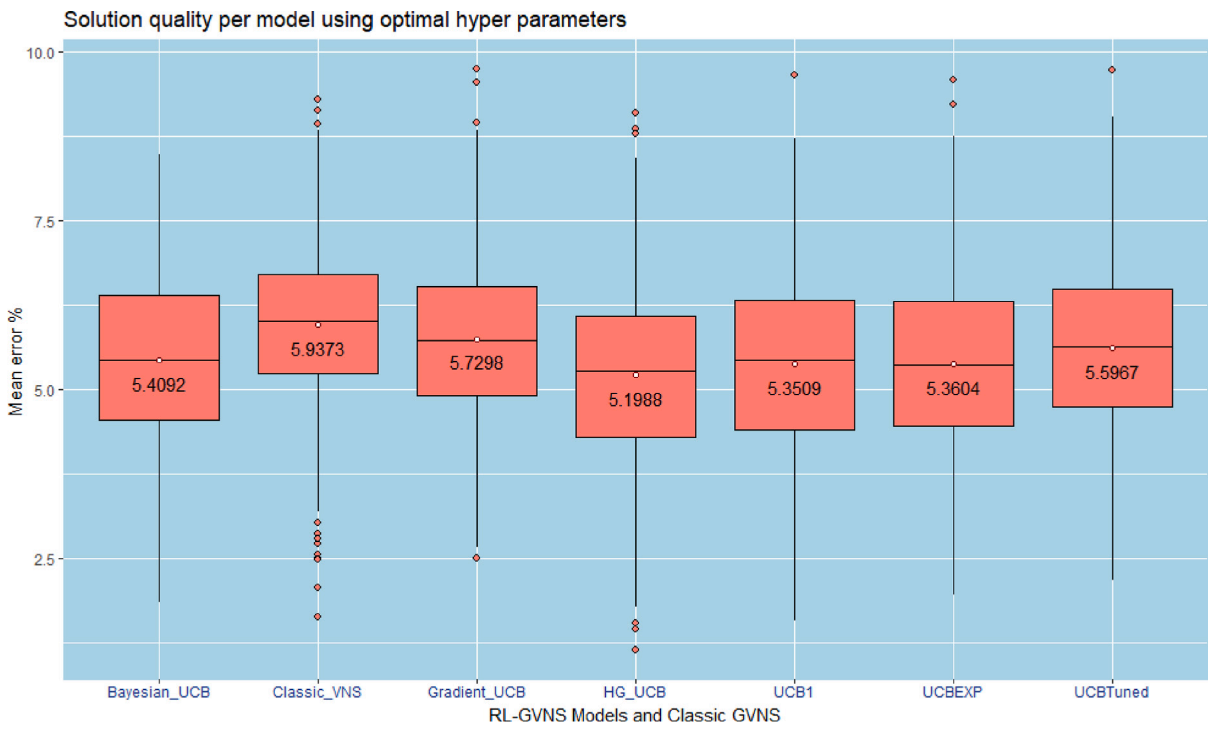

Fig. 6. Optimum performance using the most efficient hyper-parameters.

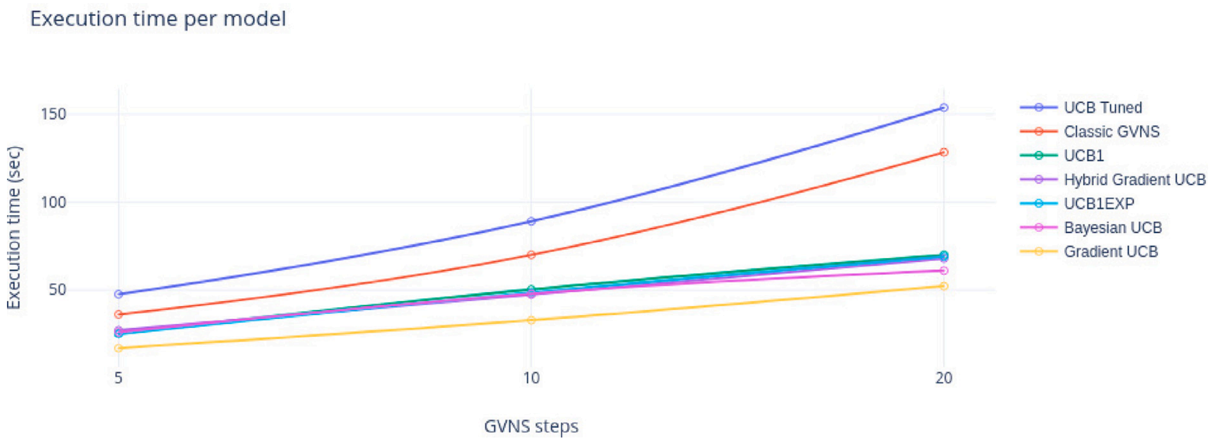

Fig. 7. The mean execution time per model per GVNS step.

We, therefore, focus our analysis on the following parameters: the scaling factor (C) set to 2 and the learning rate (α) set to 5, The AdWin window size W and the upper bound d are set to 5 and 0.02, respectively. Finally, the noise threshold ϵ was set to 0.1.

7.1. Performance comparison

A comparison with state-of-the-art vehicle routing algorithms, using benchmark instances from set CMT, indicates that the proposed algorithm is both efficient and lightweight and provides results with good solution quality. In Table 13, we establish the high potential of our approach through a comparison with six different algorithms. Those algorithms belong to three of the most competitive approaches in the literature: exact solvers, heuristics, and reinforcement learning methods. Concentrating on methods using heuristics and strictly based on solution quality, the LKH3 heuristic solver (Lin and Kernighan, 1973; Helsgaun, 2017) achieves state-of-the-art status. Two more heuristic algorithms are examined: OR-Tools (Google) and Clarke and Wright (1964) saving algorithm. The RL group consists of Nazari et al. (2018) and Kool et al. (2018). The table presents results, per instance, for the average distance. The second column displays the solution's average deviation from the optimum as a percentage (Gap %), and the third is the total execution time.

The experiment is designed to give a thorough evaluation, with each algorithm constrained within a time limit of 1 min for each benchmark instance of CMT. The parameters for the LKH3 algorithm were: $max\_trials = 10, runs = 10, time\_limit = 60, stop\_at\_optimum = yes$. Results of average distance are obtained by executing ten iterations. Data for the RL group and Gurobi solver were obtained from Ardon (2022).

Although the comparison is essential to illustrate the potential value of the model, however, comparing different implementations can be challenging because they can vary according to language performance (Python, Cython vs. C), hardware (GPU vs. CPU), and architecture (parallelization vs. single-threaded implementations).

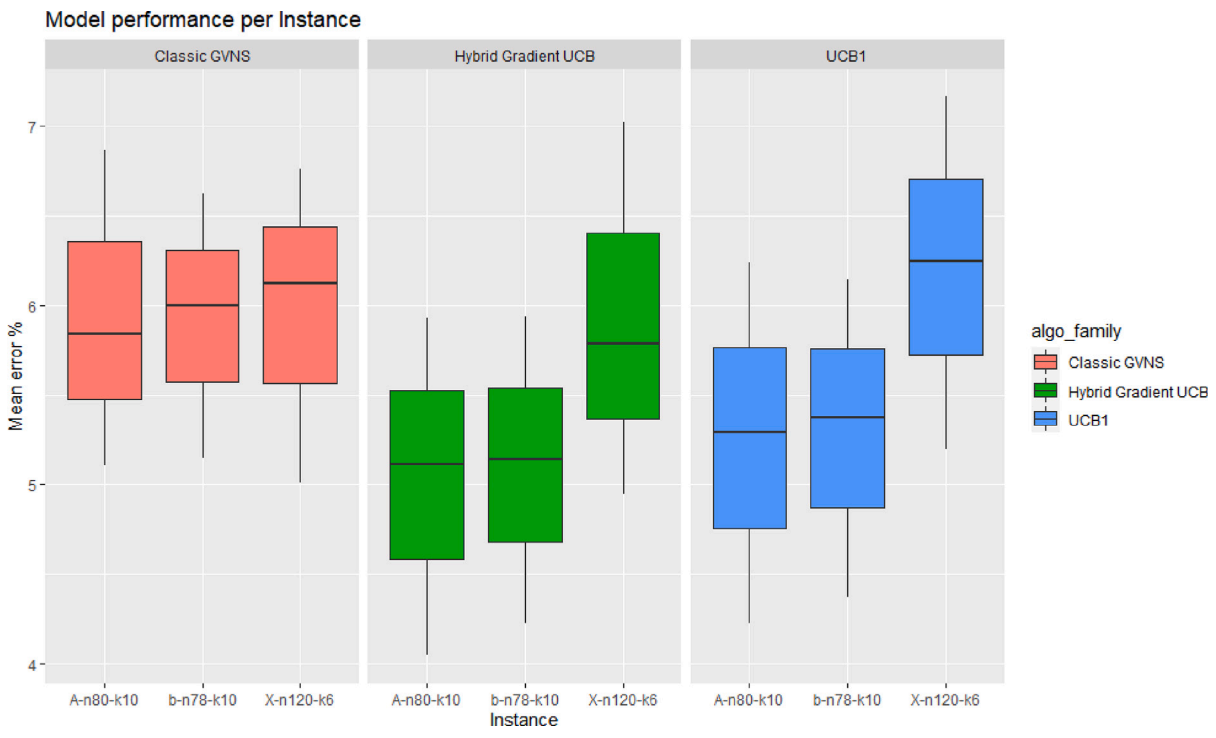

Fig. 8. Overall per-instance performance for UCB1 and HG-UCB.

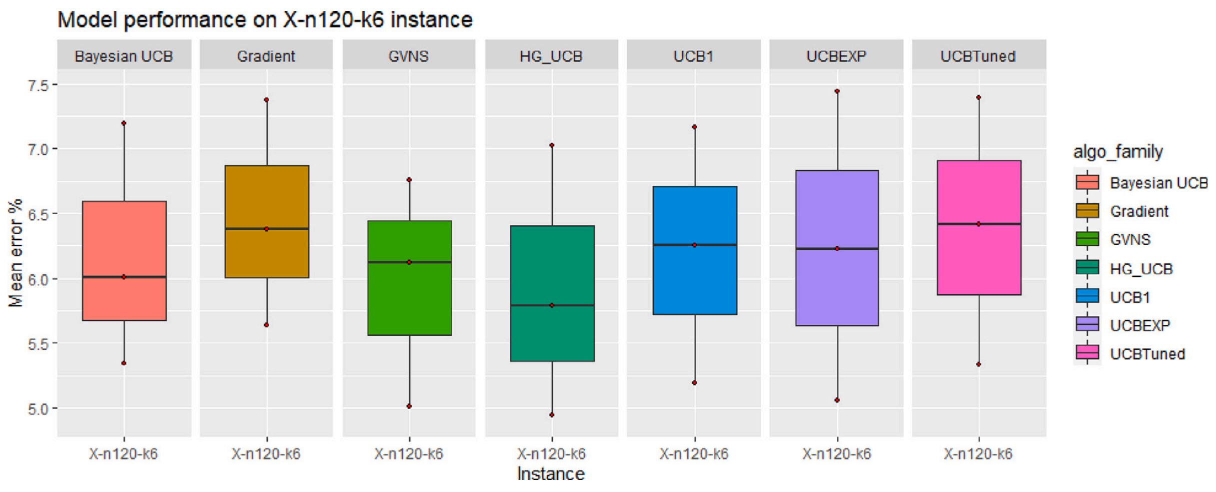

Fig. 9. Performance on X-n120-k6 instance for all models.

7.2. Discussion

The Hybrid Gradient algorithm is the model that performed the best across all test cases in the initial experiments, as shown in Table 8. Additionally, the performance ranking remains almost unaffected by the GVNS steps; thus, the performance remains robust over time. As execution time grows, HG-UCB outperforms the traditional GVNS framework by 18.5% in terms of solution quality (20 GVNS steps).

The Hybrid Gradient UCB can outperform VNS across almost all instances of the A, B, and X Set of CVRP in terms of solution quality, computation time, and convergence rate. Improvement in solution quality ranges from 4.5% percent to 26.5% percent. The UCB1 scheme, the second-best model, produces, on average, 9.9% percent better than the GVNS framework.

Despite the heavier computational load caused by the increased computational complexity of the UCB methods, the filtering process suspended execution when the reward dropped below the threshold set by b_t. The suspension of inefficient neighborhoods leads to decreased total execution time. A balance between execution time and solution quality can be achieved by fine-tuning the b_t parameter.

It can be observed in Table 8 that UCB-Tuned requires a long execution time. This effect can be observed when a method selects the neighborhoods that yielded the best results, especially at the beginning of the exploration when a large margin of correction was possible. Relocation fits the pattern since it is the most computationally expensive neighborhood and produces significant improvement when applied initially, but after a relatively small number of iterations, it significantly slows down. UCB-Tuned empirically exceeds UCB1 in terms of frequency of choosing the best action (Auer et al., 2002; Burtini et al., 2015), but in optimization, exploration and exploitation must be balanced. On the other hand, the Gradient model produces the best results in terms of execution speed. It promotes fast and efficient neighborhoods, but it reaches a plateau faster and stagnates, triggering the filtering process since the reward rate drops below the b_t threshold.

Table 10
VNS against HG-UCB in Set A, based on the values of error.

		CVRPLib Set A	
	Instance name	HG-UCB mean error	VNS mean error
1	A-n32-k5	0.1892%	0.1071%
2	A-n33-k5	0.0151%	0.0675%
3	A-n33-k6	0.0403%	0.1007%
4	A-n34-k5	0.1404%	0.1655%
5	A-n36-k5	1.6628%	2.0617%
6	A-n37-k5	0.0818%	0.0744%
7	A-n37-k6	0.5833%	0.9223%
8	A-n38-k5	0.6835%	0.8252%
9	A-n39-k5	0.6090%	0.9778%
10	A-n39-k6	0.8525%	0.7480%
11	A-n44-k6	1.5653%	1.8253%
12	A-n45-k6	2.3560%	2.4023%
13	A-n45-k7	1.3565%	1.8697%
14	A-n46-k7	0.8496%	1.2697%
15	A-n48-k7	2.5673%	2.5219%
16	A-n53-k7	2.2642%	2.9009%
17	A-n54-k7	1.4533%	2.6567%
18	A-n55-k9	1.2720%	1.7928%
19	A-n60-k9	2.5366%	3.6952%
20	A-n61-k9	2.2237%	2.5337%
21	A-n62-k8	3.2572%	4.0926%
22	A-n63-k10	2.6193%	3.3815%
23	A-n63-k9	2.4863%	3.8137%
24	A-n64-k9	4.0859%	5.5944%
25	A-n65-k9	2.8123%	2.9921%
26	A-n69-k9	3.4536%	3.6510%
	Average Gap	1.6103%	2.0401%
	Improvement (%)	26.2434%	

Table 11
VNS against HG-UCB in Set B, based on the values of error.

		CVRPLib Set B	
	Instance name	HG-UCB mean error	VNS mean error
1	b-n31-k5	0.2592%	0.3993%
2	b-n34-k5	0.1390%	0.1580%
3	b-n35-k5	0.4572%	0.5250%
4	b-n38-k6	0.2970%	0.2228%
5	b-n39-k5	0.6574%	0.5582%
6	b-n41-k6	0.9000%	0.9601%
7	b-n43-k6	0.7881%	0.7940%
8	b-n44-k7	1.2748%	1.5918%
9	b-n45-k5	0.7189%	0.9329%
10	b-n45-k6	3.9066%	3.9450%
11	b-n50-k7	0.1745%	0.3481%
12	b-n50-k8	1.1188%	1.2379%
13	b-n51-k7	0.000%	0.000%
14	b-n52-k7	1.3117%	1.4600%
15	b-n56-k7	1.0898%	1.3441%
16	b-n57-k7	0.000%	0.000%
17	b-n57-k9	1.5955%	2.0528%
18	b-n63-k10	4.0804%	4.4417%
19	b-n64-k9	3.7605%	4.4837%
20	b-n66-k9	2.3453%	2.7553%
21	b-n67-k10	4.1199%	4.5410%
22	b-n68-k9	3.0771%	3.2446%
23	b-n78-k10	6.0025%	6.1491%
	Average Gap	1.6554%	1.8324%
	Improvement	10.6928%	

In contrast with the relocation neighborhood, other neighborhoods that require less computational effort like $2 - opt$ and $move(1)$ yield smaller but frequent rewards.

Remarkably, the fact that the null hypothesis for the Wilcoxon test was not rejected suggests that even though the two methods follow a different trajectory path, they produce similar results. Additional analysis revealed that in the X-n120-k6 instance, Gradient UCB and UCB Tuned produced almost the same solution quality exhibiting the worst performance (Fig. 9).

Table 12
VNS against HG-UCB in Set X, based on the values of error.

		CVRPLib Set X	
	Instance name	HG-UCB mean error	VNS mean error
1	X-n101-k25	3.5419%	4.169%
2	X-n106-k14	2.5488%	2.772%
3	X-n110-k13	4.4541%	4.691%
4	X-n115-k10	6.5690%	7.032%
5	X-n120-k6	6.1007%	6.6973%
6	X-n125-k30	3.4693%	4.1681%
7	X-n129-k18	7.2141%	8.1560%
8	X-n134-k13	7.3767%	7.1911%
9	X-n139-k10	9.1838%	7.9316%
10	X-n143-k7	11.3161%	9.9392%
11	X-n148-k46	3.5120%	4.2880%
12	X-n153-k22	4.1616%	4.9308%
13	X-n157-k13	3.6606%	3.9700%
14	X-n162-k11	9.1788%	7.3511%
15	X-n167-k10	10.6379%	9.9837%
16	X-n172-k51	3.8806%	5.4719%
17	X-n176-k26	5.2610%	6.3334%
18	X-n181-k23	3.3869%	3.4311%
19	X-n186-k15	9.5608%	10.3193%
20	X-n190-k8	6.5027%	6.6798%
21	X-n195-k51	4.9670%	6.5967%
	Average Gap	6.0230%	6.2906%
	Improvement	4.4422%	

The results in the Tables 10, 11, 12 demonstrate that the RL component approach introduced in this study reduces the computation time and improves the quality of the solution in 75%–90% of instances. Similar observations supporting our findings were also made by Chen et al. (2020). The authors utilized a reinforcement learning scheme with VNS to solve real-instances of Periodic Vehicle Routing Problem with Time Windows and Open Routes (PVRPTW-O) and reported a better-balanced exploration–exploitation trade-off in the search process. Additionally, Silva et al. (2019) applied reinforcement learning, more specifically the Q-Learning algorithm in combination with a metaheuristic framework, to solve vehicle routing and scheduling problems. Their framework, like Bandit-VNS, utilized a multi-agent structure. They conclude that the cooperation among the agents affects the quality of the solutions.

8. Conclusions and future work

This paper proposes a framework that combines reinforcement learning and metaheuristics to solve the capacitated vehicle routing problem. We presented five reinforcement learning algorithms in detail and introduced a novel hybrid online RL scheme. This scheme enhanced the results of the metaheuristic GVNS framework both in terms of solution quality and execution speed. The extra overhead generated by the RL components was mitigated by filtering out inefficient actions. Additionally, a concept drift detection algorithm was attached to all models to adjust learner estimations dynamically and reinitialize when necessary. The proposed Bandit learning algorithms have a small computation footprint and are easy to apply.

This paper combined reinforcement learning and metaheuristics, two well-studied research topics that reveal promising methods and several future research directions. Opportunities for future work include further research on other variants of VRP, in addition to other combinatorial problems. Other suggestions of future research directions and specific challenges that need further exploration include delayed reward mechanisms to thoroughly address the exploration–exploitation dilemma, online optimization of all GVNS parameters, and transfer learning methods that allow knowledge to pass to new domains.

Table 13
Comparison of HG-UCB against six different algorithms using CMT set.

Method		CMT1			CMT2			CMT3			CMT11		
		Avg. error	Gap (%)	Time (sec)	Avg. error	Gap (%)	Time (sec)	Avg. error	Gap (%)	Time (sec)	Avg. error	Gap (%)	Time (sec)
Exactsolver	Gurobi(AMPL)	548.40	11.40	60	888.70	6.40	60	924.90	12.00	60	1108.10	6.30	60
Heuristic	CW	1408.00	62.74	0.02	1944.00	57.03	0.06	2957.00	72.06	0.13	2210.00	52.84	0.27
	OR-Tools	556.50	6.10	60	890.80	6.60	60	875.10	5.90	60	1178.10	13.00	60
	LKH3	524.61	0.00	25.73	849.70	1.72	60.05	835.26	1.10	60.58	1107.48	6.27	61.45
RL	Kool	531.90	1.40	1.50	867.20	3.80	1.70	882.50	6.80	1.90	1207.70	15.90	3.00
	Nazari	562.10	7.10	7.20	907.50	8.60	11.80	915.30	10.80	17.50	1221.80	17.20	21.80
Hybrid RLHeuristic	HG-UCB	524.61	0.00	25.42	855.26	2.33	60	851.42	2.96	60	1181.34	11.72	60
Optimum CVRPLib		524.61	0.00	-	835.26	0.00	-	826.14.	0.00	-	1042.11	0.00	–

CRediT authorship contribution statement

Panagiotis Kalatzantonakis: Conceptualization, Software, Writing – original draft. **Angelo Sifaleras:** Methodology, Writing – reviewing and editing, Supervision. **Nikolaos Samaras:** Writing – reviewing and editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Almeida, C. P., Gonçalves, R. A., Venske, S., Lüders, R., & Delgado, M. (2020). Hyper-heuristics using multi-armed bandit models for multi-objective optimization. *Applied Soft Computing*, *95*, Article 106520.

Ardon, L. (2022). Reinforcement learning to solve NP-hard problems: An application to the CVRP. arXiv preprint arXiv:2201.05393.

Audibert, J.-Y., Munos, R., & Szepesvári, C. (2009). Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, *410*(19), 1876–1902.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*(2–3), 235–256.

Augerat, P., Naddef, D., Belenguer, J., Benavent, E., Corberan, A., & Rinaldi, G. (1995). Computational results with a branch and cut code for the capacitated vehicle routing problem. *Institut IMAG, University Joseph Fourier Grenoble I, Technical Report INPG-RR-949-M*.

Balinski, M. L., & Quandt, R. E. (1964). On an integer program for a delivery problem. *Operations Research*, *12*(2), 300–304.

Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining* (pp. 443–448). SIAM.

Bifet, A., Gavaldà, R., Holmes, G., & Pfahringer, B. (2018). *Machine learning for data streams: with practical examples in MOA*. MIT Press.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In *Handbook of metaheuristics* (pp. 449–468). Springer.

Burtini, G., Loeppky, J., & Lawrence, R. (2015). A survey of online experiment design with the stochastic multi-armed bandit. arXiv preprint arXiv:1510.00757.

Chen, B., Qu, R., Bai, R., & Laesanklang, W. (2020). A variable neighborhood search algorithm with reinforcement learning for a real-life periodic vehicle routing problem with time windows and open routes. *RAIRO-Operations Research*, *54*(5), 1467–1494.

Chen, P., Wan, H., Cai, S., Luo, W., & Li, J. (2019). Combining reinforcement learning and configuration checking for maximum k-plex problem. arXiv preprint arXiv:1906.02578.

Christofides, N. (1979). Combinatorial optimization. *A Wiley-Interscience Publication.*

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, *12*(4), 568–581.

CVRPLIB (2021). CVRPLIB - All instances. http://vrp.atd-lab.inf.puc-rio.br/index.php/en. (Accessed 15 June 2021).

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, *6*(1), 80–91.

Delarue, A., Anderson, R., & Tjandraatmadja, C. (2020). Reinforcement learning with combinatorial actions: An application to vehicle routing. *Advances in Neural Information Processing Systems*, *33*.

Delignette-Muller, M. L., & Dutang, C. (2015). Fitdistrplus: An R package for fitting distributions. *Journal of Statistical Software*, *64*(4), 1–34.

dos Santos, J. P. Q., de Melo, J. D., Neto, A. D. D., & Aloise, D. (2014). Reactive search strategies using reinforcement learning, local search algorithms and variable neighborhood search. *Expert Systems with Applications*, *41*(10), 4939–4949.

Duarte, A., Sánchez-Oro, J., Mladenović, N., & Todosijević, R. (2018). Variable neighborhood descent. In R. Martí, P. M. Pardalos, & M. G. C. Resende (Eds.), *Handbook of heuristics* (pp. 341–367). Cham: Springer International Publishing.

Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., & Hester, T. (2021). Challenges of real-world reinforcement learning: Definitions, benchmarks and analysis. *Machine Learning*, *110*, 2419–2468.

Ferreira, A. S., Gonçalves, R. A., & Pozo, A. (2017). A multi-armed bandit selection strategy for hyper-heuristics. In *2017 IEEE Congress on Evolutionary Computation* (pp. 525–532). IEEE.

Fialho, Á., Da Costa, L., Schoenauer, M., & Sebag, M. (2009). Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In *International Conference on Learning and Intelligent Optimization* (pp. 176–190). Springer.

Golden, B. L., Raghavan, S., & Wasil, E. A. (2008). *The vehicle routing problem: Latest advances and new challenges, vol. 43*. Springer Science & Business Media.

Google (2022). Google's Operations Research tools, URL https://developers.google.com/optimization.

Hansen, P., Mladenović, N., Brimberg, J., & Pérez, J. A. M. (2019a). Variable neighborhood search. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (pp. 57–97). Cham: Springer International Publishing.

Hansen, P., Mladenović, N., Brimberg, J., & Pérez, J. A. M. (2019b). Variable neighborhood search. In *Handbook of metaheuristics* (pp. 57–97). Springer.

Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2017). Variable neighborhood search: Basics and variants. *EURO Journal on Computational Optimization*, *5*(3), 423–454.

Helsgaun, K. (2017). An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde University*, 24–50.

Jankee, C., Verel, S., Derbel, B., & Fonlupt, C. (2016). Distributed adaptive meta-heuristic selection: Comparisons of selection strategies. In S. Bonnevay, P. Legrand, N. Monmarché, E. Lutton, & M. Schoenauer (Eds.), *Artificial evolution* (pp. 83–96). Cham: Springer International Publishing.

Johnson, D. S., & Garey, M. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co..

Kalatzantonakis, P., Sifaleras, A., & Samaras, N. (2019). On a cooperative VNS parallelization strategy for the capacitated vehicle routing problem. In N. Matsatsinis, Y. Marinakis, & P. Pardalos (Eds.), *11968, Learning and Intelligent Optimization (LION 13). Lecture Notes in Computer Science* (pp. 231–239). Springer Cham.

Karakostas, P., Sifaleras, A., & Georgiadis, M. C. (2019). A general variable neighborhood search-based solution approach for the location-inventory-routing problem with distribution outsourcing. *Computers & Chemical Engineering*, *126*, 263–279.

Kaufmann, E., Cappé, O., & Garivier, A. (2012). On Bayesian upper confidence bounds for bandit problems. In N. D. Lawrence, & M. Girolami (Eds.), *Proceedings of machine learning research: vol. 22, Proceedings of the fifteenth international conference on artificial intelligence and statistics* (pp. 592–600). La Palma, Canary Islands: PMLR.

Kool, W., Van Hoof, H., & Welling, M. (2018). Attention, learn to solve routing problems!. arXiv preprint arXiv:1803.08475.

Koulouriotis, D. E., & Xanthopoulos, A. (2008). Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems. *Applied Mathematics and Computation*, *196*(2), 913–922.

Kuleshov, V., & Precup, D. (2014). Algorithms for multi-armed bandit problems. arXiv preprint arXiv:1402.6028.

Lai, T. L. (1987). Adaptive treatment allocation and the multi-armed bandit problem. *The Annals of Statistics*, *15*(3), 1091–1114.

Lai, T. L., & Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, *6*(1), 4–22.

Laporte, G. (1992). The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, *59*(3), 345–358.

Laporte, G., Gendreau, M., Potvin, J.-Y., & Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, *7*(4–5), 285–300.

Laporte, G., Nobert, Y., & Desrochers, M. (1985). Optimal routing under capacity and distance restrictions. *Operations Research*, *33*(5), 1050–1073.

Leng, L., Zhang, J., Zhang, C., Zhao, Y., Wang, W., & Li, G. (2020). Decomposition-based hyperheuristic approaches for the bi-objective cold chain considering environmental effects. *Computers & Operations Research*, *123*, Article 105043.

Lenstra, J. K., & Kan, A. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, *11*(2), 221–227.

Li, L., Chu, W., Langford, J., & Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on world wide Web* (pp. 661–670).

Li, J., Xin, L., Cao, Z., Lim, A., Song, W., & Zhang, J. (2021). Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, *23*(3), 2306–2315.

Liberti, L., & Maculan, N. (2006). *Global optimization: From theory to implementation, vol. 84*. Springer Science & Business Media.

Lin, B., Ghaddar, B., & Nathwani, J. (2020). Deep reinforcement learning for electric vehicle routing problem with time windows. arXiv preprint arXiv:2010.02068.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, *21*(2), 498–516.

Liu, R., Piplani, R., & Toro, C. (2022). Deep reinforcement learning for dynamic scheduling of a flexible job shop. *International Journal of Productions Research*, *60*(13), 4049–4069.

Liu, F., Wang, S., Buccapatnam, S., & Shroff, N. B. (2018). UCBoost: A boosting approach to tame complexity and optimality for stochastic bandits. In J. Lang (Ed.), *Proceedings of the twenty-seventh international joint conference on artificial intelligence* (pp. 2440–2446).

Lu, H., Zhang, X., & Yang, S. (2020). A learning-based iterative method for solving vehicle routing problems. In *8th International conference on learning representations* (pp. 1–12).

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100.

Mladenović, N., Jarboui, B., Elleuch, S., Mussabayev, R., & Rusetskaya, O. (2021). Variable neighborhood programming as a tool of machine learning. In P. M. Pardalos, V. Rasskazova, & M. N. Vrahatis (Eds.), *Black box optimization, machine learning, and no-free lunch theorems* (pp. 221–271). Cham: Springer International Publishing.

Mladenović, N., Sleptchenko, A., Sifaleras, A., & Omar, M. (Eds.), (2021). Variable neighborhood search. In *LNCS*: *vol. 12559*, *8th International conference, ICVNS 2021, Abu Dhabi, United Arab Emirates, October 21-25, 2021, Revised Selected Papers*. Springer, Cham.

Mladenović, N., Todosijević, R., & Urošević, D. (2014). Two level general variable neighborhood search for attractive traveling salesman problem. *Computers & Operations Research*, *52*, 341–348.

Montiel, J., Read, J., Bifet, A., & Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, *19*(72), 1–5.

Myung, I. J. (2003). Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, *47*(1), 90–100.

Nazari, M., Oroojlooy, A., Snyder, L., & Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. *Advances in Neural Information Processing Systems*, *31*.

Neu, G., Antos, A., György, A., & Szepesvári, C. (2010). Online Markov decision processes under bandit feedback. In *Advances in neural information processing systems* (pp. 1804–1812).

Newman, M. E. (2005). Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, *46*(5), 323–351.

Paolo, T., & Daniele, V. (2002). The vehicle routing problem. In *SIAM Monographs on Discrete Mathematics and Applications*, Philadelphia, PA: Society for Industrial and Applied Mathematics.

R. Core Team (2020). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing, URL https://www.R-project.org.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *American Mathematical Society. Bulletin*, *58*(5), 527–535.

Russo, D., & Van Roy, B. (2014). Learning to optimize via posterior sampling. *Mathematics of Operations Research*, *39*(4), 1221–1243.

Sabar, N. R., Ayob, M., Kendall, G., & Qu, R. (2014). A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics*, *45*(2), 217–228.

Sabar, N. R., Turky, A., Song, A., & Sattar, A. (2017). Optimising deep belief networks by hyper-heuristic approach. In *2017 IEEE congress on evolutionary computation* (pp. 2738–2745).

Sabar, N. R., Zhang, X. J., & Song, A. (2015). A math-hyper-heuristic approach for large-scale vehicle routing problems with time windows. In *2015 IEEE congress on evolutionary computation* (pp. 830–837).

Schlimmer, J. C., & Granger, R. H. (1986). Incremental learning from noisy data. *Machine Learning*, *1*(3), 317–354.

Silva, M. A. L., de Souza, S. R., Souza, M. J. F., & Bazzan, A. L. C. (2019). A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. *Expert Systems with Applications*, *131*, 148–171.

Strickler, A., Lima, J. A. P., Vergilio, S. R., & Pozo, A. T. (2016). Deriving products for variability test of feature models with a hyper-heuristic approach. *Applied Soft Computing*, *49*, 1232–1242.

Subramanian, A., Drummond, L. M. d. A., Bentes, C., Ochi, L. S., & Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, *37*(11), 1899–1911.

Sultana, N., Chan, J., Sarwar, T., & Qin, A. (2022). Sample-efficient, exploration-based policy optimisation for routing problems. arXiv preprint arXiv:2205.15656.

Sun, Z., Benlic, U., Li, M., & Wu, Q. (2022). Reinforcement learning based tabu search for the minimum load coloring problem. *Computers & Operations Research*, *143*, Article 105745.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, *25*(3/4), 285–294.

Toth, P., & Vigo, D. (2002a). *The vehicle routing problem*. SIAM.

Toth, P., & Vigo, D. (2002b). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, *123*(1–3), 487–512.

Toth, P., & Vigo, D. (2014). *Vehicle routing: Problems, methods, and applications*. SIAM.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., & Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, *257*(3), 845–858.

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

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*(1), 69–101.