Annals of Operations Research
https://doi.org/10.1007/s10479-024-06243-7

ORIGINAL RESEARCH

How to improve “construct, merge, solve and adapt"? Use reinforcement learning!

Jaume Reixach[1] · **Christian Blum**[1]

Received: 21 February 2024 / Accepted: 22 August 2024

Abstract
In this work, we propose a new variant of construct, merge, solve, and adapt (CMSA), which is a recently introduced hybrid metaheuristic for combinatorial optimization. Our newly proposed variant, named reinforcement learning CMSA (RL-CMSA), makes use of a reinforcement learning (RL) mechanism trained online with data gathered during the search process. In addition to generally outperforming standard CMSA, this new variant proves to be more flexible as it does not require a greedy function for the evaluation of solution components at each solution construction step. We present RL-CMSA as a general framework for enhancing CMSA by leveraging a simple RL learning process. Moreover, we study a range of specific designs for the employed learning mechanism. The advantages of the introduced CMSA variant are demonstrated in the context of the far from most string and minimum dominating set problems, showing the improvement in performance and simplicity with respect to standard CMSA. In particular, the best performing RL-CMSA variant proposed is statistically significantly better than the standard algorithm for both problems, obtaining 1.28% and 0.69% better results on average respectively.

Keywords Combinatorial optimization · Hybrid metaheuristics · Reinforcement learning · CMSA

Acronyms

CD	Critical difference
CMSA	Construct, merge, solve, and adapt
FFMS	Far from most string
ILP	Integer linear programming
MDS	Minimum dominating set
ML	Machine learning
RL	Reinforcement learning
TSP	Travelling salesman problem
UCB	Upper-confidence-bound
VNS	Variable neighbourhood search

✉ Jaume Reixach
jaume.reixach@iiia.csic.es

Christian Blum
christian.blum@iiia.csic.es

[1] Artificial Intelligence Research Institute (IIIA-CSIC), Campus of the UAB, 08193 Bellaterra, Spain

1 Introduction

CMSA is a recently introduced hybrid metaheuristic (Blum et al., 2016; Blum, 2024). At each iteration, the algorithm deals with an initially empty sub-instance C' of the considered optimization problem. The first step of each iteration (the *construct* step) consists of the generation of several feasible solutions to the original problem instance in a probabilistic way. Subsequently, the solution components involved in these solutions are added to C' (the *merge* step) and an exact solver is applied to obtain a solution to sub-instance C' (the *solve* step). The last step (the *adapt* step) consists of removing those solution components from C' that were becoming too old with regard to an aging mechanism. In general, the CMSA metaheuristic is applicable to any problem for which (1) valid solutions can be probabilistically generated and (2) an exact solver can be devised.

Since its introduction in 2016, CMSA has been successfully applied to a range of different combinatorial optimization problems. Some of the most recent applications include the ones to the maximum disjoint dominating sets problem (Rosati et al., 2024), the electric vehicle routing problem with time windows, simultaneous pickup and deliveries, and partial vehicle charging (Akbay et al., 2022), a bus driver scheduling problem with complex break constraints (Rosati et al., 2022), and test data generation in software product lines (Ferrer et al., 2021). Moreover, existing extensions of CMSA include, for example, Adapt-CMSA (Akbay et al., 2022), which is a variant that reduces the parameter sensitivity of the original CMSA observed in some applications.

The goal of the research presented in this paper is to successfully leverage ideas from RL for improving CMSA. There are two main aspects which we aim to improve. The first concerns performance, and the second concerns obtaining a simpler algorithm by eliminating a problem-dependent component. The rest of the paper is organized as follows. The next two subsections explain the contribution of this paper and deal with related work from the literature, respectively. Next, the standard CMSA is described. Following that, the general structure of the new RL-CMSA variant is explained together with some particular implementations of the newly proposed learning mechanism. The fourth section provides the experimental study where the proposed RL-CMSA implementations are compared to the standard CMSA in the context of the FFMS and MDS problems. Finally, the last section gives some conclusions and ideas for future work.

1.1 Contribution of this paper

RL (Sutton & Barto, 2018) is an area of Machine Learning (ML) concerned with the actions of an intelligent agent in a certain environment with the goal of maximizing its cumulative reward. At each step, the agent is presented with a set of available actions and must discover which ones result in the highest reward. Generally, actions may not only produce a reward but can also influence the available actions and rewards for the following steps. In this work, we introduce a new variant of CMSA, named RL-CMSA. This new algorithm variant makes use of RL to improve the *construct* step of CMSA. As mentioned above, during the *construct* step of CMSA a certain number of solutions to the problem at hand are constructed. This is usually done by employing a problem-specific solution construction mechanism together with a greedy function. Hereby, the solution construction mechanism determines for each construction step the set of available options, while a greedy function gives a static or dynamic greedy value to each option. Exactly one of the available options is probabilistically chosen at each step—based on the greedy function values—until a complete solution is obtained.

This procedure resembles the general RL setting and motivates employing an agent for constructing solutions, which is exactly what is done in our novel RL-CMSA approach. In particular, solution constructions are performed by selecting solution components depending on associated quality measures, which are updated at each iteration through an RL strategy. For every selection of a solution component, the agent receives a reward depending on whether the selected solution component proves to be useful. Solution components' usefulness is hereby measured by their possible inclusion in solutions to sub-instances at each iteration of RL-CMSA.

The proposed RL-CMSA approach is applied to two NP-hard combinatorial optimization problems: (1) the FFMS problem and (2) the MDS problem. In both cases, the experimental results show the superiority of the new RL-CMSA variant over standard CMSA. In addition to performing strongly in comparison to standard CMSA, RL-CMSA is more general and sometimes even easier to implement. This is because, while still requiring a solution construction mechanism, it does not need a greedy function for evaluating the set of options for extending the current partial solution at each construction step.

1.2 Related work

In recent years, the field of ML has gained notorious popularity as it has brought significant advancements in different domains. This has been in part enabled by the increase in computational power and the availability of large datasets. As a consequence, more attention has been given to ML in the field of combinatorial optimization.

On the one hand, various ML end-to-end combinatorial optimization solvers have been proposed, see e.g., (Bello et al., 2016; Kool et al., 2018; Kwon et al., 2020). While this is an interesting and promising research direction, these solvers have not yet been able to compete with traditional state-of-the-art techniques and are currently limited to small problem instances. On the other hand, an alternative line of research consists of leveraging ML to enhance classic combinatorial optimization techniques. Our work belongs to this research area and more particularly consists of improving a general metaheuristic using the RL paradigm (Sutton & Barto, 2018). One early application of RL within metaheuristics is found in (Gambardella & Dorigo, 1995), where the authors make use of an RL technique known as Q-learning within the ant system metaheuristic and apply the resulting algorithm to the Travelling Salesman Problem (TSP). Some more recent work in the direction of merging RL and metaheuristics consists of a method for learning the heuristic function of beam search found in (Huber & Raidl, 2021), which is applied to the Longest Common Subsequence (LCS) and Constrained Longest Common Subsequence (CLCS) problems. Furthermore, a Variable Neighbourhood Search (VNS) based on Q-learning was devised for a machine scheduling problem in (Alicastro et al., 2021). A last algorithm proposal, applied to the TSP, that we mention here concerns the use of RL for adapting the parameters of a Biased Random Key Genetic Algorithm (BRKGA) throughout its evolutionary process, found in (Chaves & Lorena, 2021).

Two examples of recent work with a stronger relation to our contribution can be found in (Almeida et al., 2020; Kalatzantonakis et al., 2023). Both works use the setting of a classic RL problem known as the multi-armed bandit problem for selecting operators in the context of metaheuristics. In the first case, the selection concerns mutation and crossover operators in the context of a multi-objective evolutionary algorithm applied to the multi-objective permutation flow shop problem. In the second case, RL is used for learning the selection of local search operators in VNS applied to the Capacitated Vehicle Routing Problem (CVRP).

Algorithm 1 High-level pseudo-code of standard CMSA

Input 1: Set C of solution components for the problem instance to be solved.
Input 2: Values for parameters n_a, $age_{\max}$ and t_{ILP}.
$S_{\text{bsf}} = \text{NULL}$, $C' = \emptyset$
while termination conditions not met **do**
  for $j = 1, \ldots, n_a$ **do**
    $S :=$ probabilistic_solution_construction()
    for all $c_i \in S$ **and** $c_i \notin C'$ **do**
      $\text{age}_{c_i} = 0$
      $C' := C' \cup \{c_i\}$
    end for
  end for
  $S_{\text{opt}} :=$ apply_exact_solver(C', t_{ILP})
  if S_{opt} is better than S_{bsf} **then** $S_{\text{bsf}} := S_{\text{opt}}$ **end if**
  adapt($C', S_{\text{opt}}, age_{\max}$)
end while
return S_{bsf}

Our work also makes use of existing work on the multi-armed bandit problem in the context of CMSA, for selecting solution components during solution construction, instead of operators.

2 Standard CMSA

To apply CMSA to a combinatorial optimization problem, one first needs to define a set C of solution components. In this way, each valid solution to the considered problem can be represented as a subset of C. For the following description of CMSA, we assume a generic set $C = \{c_1, c_2, \ldots, c_n\}$. Moreover, note that for every valid solution S to the problem at hand, it holds that $S \subseteq C$.

Algorithm 1 illustrates the structure of standard CMSA. First, sub-instance C' is initialized as empty and the *best-so-far* solution S_{bsf} is initialized as NULL. Afterward, the main loop of the algorithm starts, in which the *construct*, *merge*, *solve*, and *adapt* steps are sequentially performed until a given time limit is reached. These four steps can be described as follows:

1. In the *construct* step, n_a solutions to the problem at hand are probabilistically constructed.
2. The *merge* step consists of extending C' with those solution components c_i that appear in at least one of the n_a constructed solutions and for which it holds that $c_i \notin C'$. Moreover, their ages are set to 0.
3. The *solve* step uses an exact solver with a time limit given by parameter t_{ILP} to solve problem instance C', obtaining a solution S_{opt} to the problem at hand.
4. Finally, the *adapt* step consists of increasing (by one) the age of solution components in $C' \setminus S_{\text{opt}}$, resetting the age of solution components in S_{opt} to 0 and erasing those solution components of C' that have an age of at least $age_{\max}$, which is a parameter of the algorithm.

The *construct* and *solve* steps are the problem-dependent parts of the algorithm. Generally, a solution construction mechanism together with a greedy function tailored to the problem at hand is used in the first, and an exact method for the tackled problem is used in the second.

Algorithm 2 High-level pseudo-code of RL-CMSA

Input 1: Set C of solution components for the problem instance to be solved.
Input 2: Values for parameters n_a, $\text{age}_{\max}$, t_{ILP}, cf_{limit} and b_{reset}.
$S_{\text{bsf}} = \text{NULL}$, $C' = \emptyset$
$q_i = 0$ for $i = 1, \ldots, n$ ▷ Initialization of the q values
while termination conditions not met **do**
  for $j = 1, \ldots, n_a$ **do**
    $S :=$ probabilistic_solution_construction(**q**)
    for all $c_i \in S$ **and** $c_i \notin C'$ **do**
      $\text{age}_{c_i} = 0$
      $C' := C' \cup \{c_i\}$
    end for
  end for
  $S_{\text{opt}} :=$ apply_exact_solver(C', t_{ILP})
  if S_{opt} is better than S_{bsf} **then** $S_{\text{bsf}} := S_{\text{opt}}$ **end if**
  adapt(C', S_{opt}, $age_{\max}$)
  update_q_values(**q**, C', S_{opt})
  $cf =$ compute_convergence_factor(**q**) ▷ Optional
  if $cf > cf_{\text{limit}}$ **then**
    $q_i = 0$ for $i = 1, \ldots, n$ ▷ Re-initialization of the q values
    if $b_{\text{reset}} = true$ **then** $C' = \emptyset$ **end if**
  end if
end while
return S_{bsf}

3 RL-CMSA

In contrast to standard CMSA, our new RL-CMSA approach keeps a quality measure q_i for every solution component $c_i \in C$, henceforth called the q-values. The set (or vector) of all q-values will be denoted by **q**. The probabilistic construction of solutions in RL-CMSA is performed depending on these q-values: solution components with higher values will have a higher chance to be selected. Moreover, in every iteration of CMSA, after the application of the exact solver, the q-values are updated. In particular, the values corresponding to those solution components in C' that form part of the solution S_{opt} found by the exact solver at the current iteration are increased. Conversely, the values of the solution components in C' that do not appear in S_{opt} are decreased.

Algorithm 2 illustrates the general structure of RL-CMSA. First, the stored sub-instance C' is initialized as empty, the *best-so-far* solution S_{bsf} to NULL and the q-values are all initialized to zero. Inside the main loop, the usual four CMSA steps are performed in addition to an extra fifth step, which we denote by *learn* step. The four usual CMSA steps are left unchanged except for the *construct* step. Similarly to standard CMSA, the *construct* step of RL-CMSA consists of probabilistically constructing n_a solutions. In RL-CMSA this is done, however, by using the q-values. Solution components with higher values will—in probability—be chosen more often. In the new *learn* step the q-values are updated and a convergence measure can be calculated, leading to a restart of the learning procedure if deemed necessary. Such a restart consists of (1) setting the q-values back to zero and (2) emptying the subinstance C' depending on a parameter (as explained below).

Different designs were considered for the new solution construction mechanism and the update of the q-values. These designs have in part been inspired by existing work on the multi-armed bandit problem, which is a classic RL problem (Kuleshov & Precup, 2014). Multi-armed bandit problems were introduced by Robbins in 1952 (Robbins, 1952). In their simplest form, they consist of a set of k probability distributions $\{D_1, \ldots, D_k\}$. The objective

is to come up with a sampling strategy for an agent for whom the distributions are unknown. This agent iteratively samples from the distributions obtaining a reward in each iteration. In technical terms, the goal is to design a sampling strategy that maximizes the obtained sum of rewards. The distributions are generally interpreted as k arms in a slot machine and the agent is viewed as a gambler whose goal is to collect as much money as possible.

As one can notice, the RL strategy we implement into CMSA in this work results in a scenario closely resembling a multi-armed bandit problem. In the case of RL-CMSA, sampling consists of selecting one of the available solution components. However, a key difference is the following one: instead of assigning rewards after every sample, or after every solution construction, in RL-CMSA rewards are assigned after the *solve* step, depending on the result of the exact solver when solving the current sub-instance.

3.1 Update of the *q*-values

In the *learn* step, the q-values corresponding to solution components in C' are updated depending on whether they form part of the solution S_{opt} computed by the exact solver during the *solve* step. In this way, the quality of a solution component is not measured by a myopic measure of the quality of adding that solution component to a partial solution under construction. Neither is the quality related to the objective function value of the final solution to which a component was added during its construction. The quality of a component is rather measured in comparison to all other solution components in the sub-instance C' in the following way: the value q_i of a solution component c_i is increased if it forms part of S_{opt}, and decreased otherwise. This is done by giving a reward $R > 0$ in the first case, and $-R$ in the latter. The following three designs for performing the q-value update were considered:

1. The first design consists of simply summing the obtained rewards over time. At each iteration, once the reward $r_i \in \{R, -R\}$ for a solution component $c_i \in C'$ is determined, its q-value is updated as follows:
$$q_i := q_i + r_i \tag{1}$$
2. The second design is based on averaging the received rewards over time. For this purpose, a variable n_i stores the number of times the q-value q_i of a solution component c_i was updated since the start of the algorithm. At each iteration, once the reward $r_i \in \{R, -R\}$ for a solution component $c_i \in C'$ is determined, its q-value is updated as follows:
$$q_i := q_i + \frac{1}{n_i}(r_i - q_i). \tag{2}$$
3. The last design generalizes the previous one by replacing $1/n_i$ with a constant step-size parameter $\alpha > 0$. The corresponding update of the q-value q_i of a solution component $c_i \in C'$ is, therefore, as follows:
$$q_i := q_i + \alpha(r_i - q_i). \tag{3}$$

The first design option from above might introduce a bias toward solution components frequently selected. For example, with this method a solution component that gets reward R obtains the same q-value as one that was awarded rewards $R, -R$ and R. On the other hand, this does not happen for the second design. By setting the q-values to the average of the rewards, the amount of times a solution component has been selected does not produce a bias. The second and third designs are popular strategies for updating the q-values in multi-armed bandit problems (Sutton & Barto, 2018). Note that, due to having a constant step-size α, the

third design gives more weight to recent rewards than to older ones. This could be beneficial in the context of RL-CMSA as the rewards given may change over time.

3.2 Solution construction in RL-CMSA

The new *construct* step uses the q-values for probabilistically generating solutions. The construction process begins with an empty solution $S = \emptyset$, to which—at each step—one of those solution components that can be used to feasibly extend the current partial solution is added until the solution is complete. In the following, we will denote by $C_{\text{feas}} \subseteq C \setminus S$ the set of feasible solution components with respect to a partial solution S. Moreover, remember that q_i denotes the q-value associated to solution component $c_i \in C$. We propose two different designs for selecting a solution component from C_{feas}.

3.2.1 Softmax selection

The first proposed design uses a real parameter $dr \in [0, 1]$ called the determinism rate. At each step of the construction process, a solution component is selected in the following way:

1. With a probability dr, a random solution component between the ones from C_{feas} with the highest q-value is chosen.
2. Otherwise, with a probability $1 - dr$, the selection is done in a roulette-based manner with the probability p_i of selecting solution component $c_i \in C_{\text{feas}}$ given by

$$p_i = \frac{e^{\beta q_i}}{\sum_{c_k \in C_{\text{feas}}} e^{\beta q_k}} \quad . \tag{4}$$

   Hereby, $\beta \geq 0$ is a parameter that, together with dr, governs the balance between exploration and exploitation.

Note that this first selection design might lead to a convergence of the algorithm. This is because the q-values of some solution components may become considerably larger than the rest, leading to the same solution being constructed all the time, further enlarging their q-values. To remediate this issue we propose to measure the level of convergence as described below. This measurement is conducted once per iteration in the *learn* step after updating the q-values. In case high convergence is detected, the algorithm is reset by re-initializing the q-values to zero and emptying C' depending on a parameter. This mechanism depends on a convergence factor and a convergence factor limit. Whenever the convergence factor is greater than the convergence factor limit, the algorithm is re-initialized.

In the following, the calculation of the convergence factor is described. For every solution component c_i of the last constructed solution S, the probability z_i of preferring c_i to all solution components that do not form part of S is calculated. The convergence factor is then defined as the minimum of these values for all $c_i \in S$. Note that with this definition, the closer the convergence factor is to value one, the closer the algorithm is to convergence. In this context, note that the probability of choosing a particular solution component depends on the values of parameters dr and β. More specifically, the probability z_i of choosing solution component c_i can be written as follows:

$$z_i := dr \cdot \chi_i + (1 - dr) \cdot \frac{e^{\beta q_i}}{\sum_{c_k \notin S} e^{\beta q_k} + e^{\beta q_i}} \quad , \tag{5}$$

where χ_i is defined as:

$$\chi_i := \begin{cases} \frac{1}{|\{c_k \in C \setminus S \cup \{c_i\} | q_k = q_i\}|} & \text{if } q_i = \max\{q_k \mid c_k \in C \setminus S \cup \{c_i\}\} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The expression for z_i is derived from how solution components are chosen in the *construct* step. With a probability dr a solution component is chosen uniformly at random from the solution components that achieve the highest q-value, and with a probability $1 - dr$ the selection is performed in a roulette-wheel-based manner with probabilities given by the softmax expression. After having calculated probabilities z_i for every solution component $c_i \in S$, the convergence factor is computed as $cf := \min\{z_i \mid c_i \in S\}$.

Once the convergence factor cf is calculated, the algorithm checks whether it is greater than the conference factor limit defined by parameter $cf_{\text{limit}} \in [0, 1]$. If this is the case then the algorithm is re-initialized:

1. The q-values are re-initialized to zero.
2. Sub-instance C' is emptied depending on a Boolean parameter b_{reset}. If $b_{\text{reset}} = \mathit{true}$, C' is set to $\emptyset$. Otherwise, if $b_{\text{reset}} = \mathit{false}$, C' is not modified.

Emptying C' when re-initializing the algorithm completely erases the previous information gathered by the RL agent. Conversely, if C' is not emptied, some of the so-far gathered information is kept.

3.2.2 Upper-confidence-bound (UCB) selection

As an alternative to Softmax selection, we consider UCB selection (Sutton & Barto, 2018). This is a method designed for the multi-armed bandit problem which aims at sampling the distribution set according to their potential to be optimal. We consider it as an alternative way of dealing with convergence, as this selection mechanism simply assures sufficient eventual exploration. Similarly to Softmax selection, we have implemented this method in the following way:

1. With a probability dr, a random solution component between the ones with the highest q-value is chosen.
2. Otherwise, with a probability $1-dr$, UCB selection is employed, consisting of selecting randomly between the solution components whose q-values maximize the following expression:

$$q_i + \rho \cdot \sqrt{\frac{\log(n)}{n_i}} \tag{7}$$

Hereby $\rho > 0$ is a parameter, n denotes the current iteration number, $\log(n)$ denotes the natural logarithm of n, and n_i the number of times solution component c_i has been selected so far.

The square-root term in the UCB expression is a measure of the uncertainty in the estimate of q_i. Each time a solution component is selected, its corresponding square-root term decreases, hence lowering the estimated uncertainty in its q-value. Conversely, if an iteration passes and a solution component is not chosen the square-root term increases.

Note that in case the q-values are unbounded, this method is not usable as the square-root term becomes useless once the q-values become large enough. This may happen with the

first and third designs we proposed for the update of the q-values. For this reason, we will use this method together with the second design proposed above for the q-value update, which sets the q-values to the average of the rewards seen so far.

4 Experimental study

To experimentally evaluate RL-CMSA, we consider the following four algorithm variants which make use of the different designs for the update of the q-values and for the selection of solution components during solution construction. The four considered RL-CMSA variants can be described as follows:

1. RL-CMSA-1: this variant is characterized by the first design for the update of the q-values (summation of the rewards), and the use of Softmax selection. The reward (R) is set to one.
2. RL-CMSA-2: second design for the q-value update (average rewards) and Softmax selection. The value of the reward (R) is considered a parameter of the algorithm.
3. RL-CMSA-3: the same as RL-CMSA-2, just that the third design (average rewards + step-size) is used for the q-value udpate.
4. RL-CMSA-4: second design for the q-value update, in combination with the UCB selection for solution construction as an alternative way of avoiding convergence. The reward (R) is set to one.

These four RL-CMSA variants will be compared to standard CMSA. In particular, this comparison will be conducted in the context of two combinatorial optimization problems from the literature: the FFMS and the MDS problems.

All the algorithms were run in a single-threaded mode in a cluster of machines with 10-core Intel Xeon processors at 2.2 GHz with 8 GB of RAM. Moreover, they all employed the commercial solver CPLEX as the exact method used in the *solve* step.

4.1 Algorithm parameters

The first three RL-CMSA variants, which use Softmax selection, make use of parameters β, cf_{limit}, and b_{reset}. Hereby, β is a parameter in the Softmax equation, cf_{limit} is the limit for the convergence factor, and the Boolean parameter b_{reset} indicates whether to empty C' in the case of re-initializing the q-values. In addition, the second and third RL-CMSA variants consider the reward R as a parameter. The third RL-CMSA variant also uses parameter α, which determines the value of the step-size parameter in the q-value update expression. The fourth RL-CMSA variant makes use of parameter ρ from the UCB selection expression. On top of these, all four algorithm variants utilize parameter dr, which determines the determinism rate used when selecting solution components.

In addition to the above-mentioned parameters, all algorithms also use the standard CMSA parameters t_{ILP}, n_a, and $age_{\max}$. These determine the time limit given to the exact solver in the *solve* step, the number of solutions constructed in the *construct* step, and the age limit used in the *adapt* step respectively. In addition, they all use the CPLEX parameters $\text{cplex}_{\text{warmstart}}$, $\text{cplex}_{\text{emphasis}}$ and $\text{cplex}_{\text{abort}}$. These are Boolean parameters that modify the behavior of CPLEX. The first one controls whether the algorithm provides an initial solution to CPLEX. In case $\text{cplex}_{\text{warmstart}} = \textit{true}$, the best-so-far solution will be provided to CPLEX for warm-starting the solving process. The second parameter balances between the speed of proving optimality and the speed of improving the best solution found during a CPLEX

execution. If $\text{cplex}_{\text{emphasis}} = \textit{true}$, then CPLEX uses its highest heuristic emphasis value. Otherwise, the default setting is used. Finally, the third parameter determines whether a CPLEX execution is stopped when a solution that improves the best-so-far solution is found. This can be beneficial due to CPLEX sometimes spending a lot of resources on bound computations important for proving optimality.

4.2 Application to the far from most string (FFMS) problem

The FFMS problem is a combinatorial optimization problem arising in bioinformatics and forms part of the sequence consensus problems family. These problems find applications in different fields, such as molecular biology (Mousavi, 2010). The FFMS problem is known to be NP-hard, meaning that it can not be solved in polynomial time unless P = NP (Lanctot et al., 2003). Given a set of equal-length input strings over an alphabet Σ and a threshold $t > 0$, the problem aims at finding a string of the same length that maximizes the number of input strings with which its *Hamming* distance is at least t. Hereby, given two strings s and s' of length m, their *Hamming* distance $d_H(s, s')$ is defined as the number of positions where their corresponding characters differ. That is:

$$d_H(s, s') := \left|\{k \in \{1, \ldots, m\} \mid s[k] \neq s'[k]\}\right| \tag{8}$$

An instance of the FFMS problem is denoted by $(\mathcal{S}, \Sigma, t)$, where $\mathcal{S}$ is a set of n input strings $\{s_i\}_{i=1}^{n}$ of length m over alphabet Σ and $0 < t \leq m$ is the threshold. Every string of length m over alphabet Σ is then a feasible solution to the problem. The goal is to find a feasible string s that maximizes the following objective function:

$$f_1(s) := \left|\{s' \in \mathcal{S} \mid d_H(s, s') \geq t\}\right| \tag{9}$$

In practice, our CMSA and RL-CMSA implementations make additional use of a secondary objective function to differentiate between two solutions having the same primary objective function value (Eq. 9). This secondary objective function can be stated as follows:

$$f_2(s) := \sum_{s' \in \{t \in \mathcal{S} \mid d_H(t,s) \geq t\}} d_H(s, s') + \max_{s' \in \{t \in \mathcal{S} \mid d_H(t,s) < t\}} \{d_H(s, s')\} \tag{10}$$

As the reader may notice, a higher value for $f_2(s)$ makes a small change in s less probable to lead to a decrease in the main objective function $f(s)$. Therefore, a solution s is deemed better than a solution s' (that is, $f(s) > f(s')$) if and only if (1) $f_1(s) > f_1(s')$ or (2) ($f_1(s) = f_1(s')$ and $f_2(s) > f_2(s')$). This lexicographic function was proposed in (Blum & Pinacho-Davidson, 2023) to remedy the negative effect of large plateaus in the search space of the FFMS problem.

In the following, we explain the definition of solution components for the application of CMSA and the RL-CMSA variants. In addition, it will be described how sub-instances are solved by making use of CPLEX. Finally, we will introduce the method for generating solutions used in the *construct* step of standard CMSA.

4.2.1 Solution components

A natural way of defining the set C of solution components in the case of the FFMS problem is the following one. For every combination of a position $k = 1, \ldots, m$ of a solution string and a character $a \in \Sigma$ set C contains the corresponding solution component $c_{k,\mathtt{a}}$, that is

$$C := \{c_{k,\mathtt{a}} \mid k = 1, \ldots, m, \text{ and } \mathtt{a} \in \Sigma\} \quad . \tag{11}$$

Therefore, at every step $j = 1, \ldots, m$ of the solution construction process the set of feasible solution components is $C_{\text{feas}} := \{c_{j,\mathtt{a}} \mid \mathtt{a} \in \Sigma\}$.

4.2.2 Probabilistic solution construction

In the following, we describe how solutions are generated in standard CMSA and the RL-CMSA variants. All algorithms make use of the same solution construction mechanism in which a letter for each position $j \in \{1, \ldots, m\}$ is determined sequentially from $j = 1$ to $j = m$. In other words, at the j-th construction step exactly one solution component from $C_{\text{feas}} = \{c_{j,\mathtt{a}} \mid \mathtt{a} \in \Sigma\}$ is chosen. How this is done is different in CMSA and the RL-CMSA variants. Standard CMSA makes a probabilistic use of the following greedy function for this purpose. Given a position $1 \leq j \leq m$ and a character $\mathtt{a} \in \Sigma$, the corresponding frequency $f_{j,\mathtt{a}}$ is defined by:

$$f_{j,\mathtt{a}} := \frac{\left|\{s \in \mathcal{S} \mid s[j] = \mathtt{a}\}\right|}{|\mathcal{S}|}$$

For choosing a letter for position j the following is done:

1. With a probability $0 \leq dr_{\text{CMSA}} \leq 1$, the solution component (letter-position assignment) with the lowest frequency value is selected, breaking ties randomly.
2. Otherwise, with probability $1 - dr_{\text{CMSA}}$, a solution component is chosen from C_{feas} utilizing letter probabilities proportional to the inverse of their frequencies. That is, the probability for choosing solution component $c_{j,\mathtt{a}}$ is set to:

$$\frac{1/f_{j\mathtt{a}}}{\sum_{\alpha \in \Sigma} 1/f_{j\alpha}} \tag{12}$$

Hereby, $dr_{\text{CMSA}} \in [0, 1]$ is a parameter called the determinism rate of CMSA.

In contrast, the RL-CMSA variants choose a letter for each position j of a solution s by means of the q-values. In particular, in the case of the FFMS, we make use of a q-value $q_{j,\mathtt{a}}$ for every solution component $c_{j,\mathtt{a}} \in C$. At the j-th solution construction step, one of the solution components is chosen from $C_{\text{feas}} = \{c_{j,\mathtt{a}} \mid \mathtt{a} \in \Sigma\}$ by Softmax, respectively UCB, selection.

4.2.3 Integer linear programming (ILP) model and sub-instance solving

In CMSA algorithms, sub-instances are—if possible—modeled in terms of ILP models which are then solved, at each iteration, using an ILP solver. As mentioned before, in this work we use the commercial solver CPLEX for this purpose. The standard ILP model of the FFMS problem uses two sets of binary variables. The first contains a variable $x_{j,\mathtt{a}}$ for every $j = 1, \ldots, m$ and $\mathtt{a} \in \Sigma$, while the second contains a variable y_j for every $j = 1, \ldots, m$.

$$\max \sum_{i=1}^{n} y_i \tag{13}$$

$$\text{subject to} \sum_{\mathtt{a} \in \Sigma} x_{j,\mathtt{a}} = 1, \qquad \text{for } j = 1, \ldots, m \tag{14}$$

$$\sum_{j=1}^{m} x_{j,s_i[j]} \leq m - t \cdot y_i, \qquad \text{for } i = 1, \ldots, n$$

$$x_{j,\mathtt{a}},\, y_i \in \{0, 1\} \tag{15}$$

Notably, variable $x_{j,\mathtt{a}}$ takes value one if character $\mathtt{a}$ is chosen for position j of the solution string and takes value zero otherwise. Constraint (14) makes sure that only one character is chosen for each position. Additionally, constraint (15) together with the maximization goal make y_i take value 1 if and only if the Hamming distance between the solution string and input string s_i is at least t.

To solve a sub-instance $C' \subseteq C$, for all $c_{j,\mathtt{a}} \in C \backslash C'$ the constraint $x_{j,\mathtt{a}} = 0$ is added to this ILP model. In other words, the values of those variables that correspond to solution components not forming part of sub-instance C' are fixed to zero.

4.2.4 Experimental evaluation

For evaluating RL-CMSA in the context of the FFMS problem, we generated the following set of benchmark instances. Each instance is a collection of n strings of size m with characters from an alphabet Σ of size $|\Sigma|$. Additionally, every instance has a threshold associated, denoted as t, indicated in terms of a proportion of m. The benchmark set contains 720 instances for every value of $|\Sigma| \in \{4, 12, 20\}$. These are further divided into 30 instances for every combination of $n \in \{100, 200, 300, 400\}$ and $m \in \{100, 500, 1000\}$. Moreover, two threshold values depending on $|\Sigma|$ are considered for all instances: $(0.8m, 0.85m)$ for instances with $|\Sigma| = 4$, $(0.97m, 1.0m)$ for $|\Sigma| = 12$, and $(0.99m, 1.0m)$ for $|\Sigma| = 20$.
Parameter tuning. In addition to the instances described above, the benchmark set contains 72 tuning instances, one for every combination of n, m, $|\Sigma|$, and t. We tuned all five algorithms twice, once for all instances when considering the lower threshold t of each threshold pair, and once concerning the higher threshold. This was done because from earlier work it is known that the change of the threshold value changes the nature of the problem more than a change in n or m. However, in an attempt to reduce the number of tuning instances, instances with $m = 500$ were excluded. Hence, both tuning runs used 24 tuning instances. Moreover, a budget of 3000 algorithm runs was given to both tuning runs, and every algorithm execution was allowed a time limit of 600 CPU seconds, for both the tuning and evaluation runs.

Table 1 provides the parameter values obtained after conducting the two tuning runs for every algorithm. There are two columns per algorithm, which contain the parameter values obtained for the lower (left column) and higher thresholds (right column) respectively.
Results. Each algorithm variant was applied exactly once to each problem instance, with a computation time limit of 600 CPU seconds. The results obtained by standard CMSA and the four different RL-CMSA versions are reported in Tables 2, 3, 4. For each combination of m, $|\Sigma|$ and t, and each algorithm, we present the average length of the best solutions found and the average execution time that was needed for obtaining these best solutions. Columns $\overline{|s|}$ and $\bar{t}_{best}[s]$ contain the two respective values. The three tables offer the results for the instances with $|\Sigma| = 4$, $|\Sigma| = 12$ and $|\Sigma| = 20$, respectively. There are 30 different benchmark instances for every combination of n, m, and t. Hence, the presented values are averages over these 30 instances. The obtained results show the following:

- The four RL-CMSA implementations obtain, on average, better results than CMSA (see last table rows).
- The only exception are the instances of alphabet size $|\Sigma| = 20$, for which RL-CMSA-2 and RL-CMSA-3 obtain slightly worse average results than CMSA.

Table 1 Parameter values obtained after tuning for the FFMS problem. Two tuning runs are performed for every algorithm. One for the lower thresholds ($0.8m$, $0.97m$, $0.99m$) and one for the higher threshold ($0.85m$, $1.00m$, $1.00m$). A dash (–) denotes that the algorithm does not use the corresponding parameter

	Allowed r ange	CMSA		RL-CMSA-1		RL-CMSA-2		RL-CMSA-3		RL-CMSA-4	
t_{ILP}	$\{1, 2, \ldots, 50\}$	48	32	36	28	40	45	44	34	40	10
dr	$\{0.0, 0.01, \ldots, 0.99\}$	–	–	0.36	0.95	0.85	0.40	0.21	0.58	0.53	0.02
n_a	$\{1, 2, \ldots, 50\}$	17	35	14	46	44	42	31	30	5	33
age_{max}	$\{1, 2, \ldots, 10\}$	7	10	9	7	9	4	10	10	10	4
$\text{cplex}_{\text{warmstart}}$	$\{0, 1\}$	1	1	1	1	1	1	1	1	1	0
$\text{cplex}_{\text{emphasis}}$	$\{0, 1\}$	1	1	1	1	1	1	1	1	1	1
$\text{cplex}_{\text{abort}}$	$\{0, 1\}$	0	1	0	0	0	1	0	0	0	0
β	$\{0.0, 0.01, \ldots, 2.0\}$	–	–	0.70	1.12	0.70	0.16	1.13	0.79	–	–
b_{reset}	$\{0, 1\}$	–	–	1	0	0	1	1	0	–	–
cf_{limit}	$\{0.90, 0.91, \ldots, 1.0\}$	–	–	0.95	0.95	0.97	0.97	0.95	0.91	–	–
r	$\{1, 2, \ldots, 10\}$	–	–	–	–	4.68	1.82	6.99	5.22	–	–
α	$\{0.1, 0.2, \ldots, 1.0\}$	–	–	–	–	–	–	0.23	0.82	–	–
ucb_c	$\{0.0, 0.1, \ldots, 5.0\}$	–	–	–	–	–	–	–	–	0.44	3.87
dr_{CMSA}	$\{0.0, 0.01, \ldots, 0.99\}$	0.35	0.48	–	–	–	–	–	–	–	–

Table 2 Comparison of CMSA and the four proposed RL-CMSA variants for the FFMS problem instances with $|\Sigma| = 4$.

n	m	th	CMSA		RL-CMSA- 1		RL-CMSA- 2		RL-CMSA- 3		RL-CMSA- 4	
			$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$
100	100	0.8	**76.07**	129.23	75.97	134.0	75.6	75.36	75.87	89.16	76.03	219.39
100	500	0.8	79.37	193.15	81.07	258.66	79.13	249.58	79.1	244.99	**82.33**	461.66
100	1000	0.8	78.97	249.11	83.27	380.96	80.4	364.75	79.97	405.81	**84.4**	492.42
200	100	0.8	98.57	106.89	**101.43**	216.27	100.87	165.68	99.73	242.11	100.2	319.19
200	500	0.8	94.2	234.71	94.07	277.99	96.17	395.88	95.97	438.18	**98.27**	484.96
200	1000	0.8	86.7	417.1	92.97	532.67	90.07	509.05	89.47	580.37	**97.47**	572.5
300	100	0.8	118.5	114.87	119.63	362.91	118.7	456.82	119.03	484.3	**126.17**	406.98
300	500	0.8	96.37	295.47	99.03	368.7	99.6	463.53	101.77	524.2	**104.63**	487.1
300	1000	0.8	88.33	472.47	94.63	565.84	91.17	533.49	91.8	591.0	**98.9**	589.98
400	100	0.8	141.8	170.77	144.97	328.59	143.43	452.89	143.87	479.35	**149.33**	401.88
400	500	0.8	95.27	384.24	101.1	453.05	100.63	490.36	100.47	550.75	**106.23**	533.85
400	1000	0.8	86.4	544.58	95.23	577.3	91.2	541.89	94.67	590.19	**97.4**	594.31
100	100	0.85	38.4	106.95	38.37	40.91	38.47	66.58	**38.57**	116.43	38.47	275.95
100	500	0.85	27.97	225.42	27.93	55.87	**28.0**	196.97	27.97	72.45	27.63	478.08
100	1000	0.85	**25.2**	229.84	24.5	62.49	25.1	218.46	24.77	122.4	22.0	50.72
200	100	0.85	46.53	120.81	46.07	48.35	46.6	100.58	46.33	167.45	**46.97**	360.35
200	500	0.85	**27.93**	217.18	27.17	115.07	27.77	224.2	27.43	146.11	24.77	567.31
200	1000	0.85	**24.93**	283.5	24.77	92.43	**24.93**	255.17	24.7	269.05	16.73	591.99
300	100	0.85	49.8	95.61	50.4	40.95	50.87	132.01	50.57	186.87	**51.87**	421.13
300	500	0.85	27.6	220.7	27.17	104.47	**27.8**	227.17	27.63	292.93	22.0	572.63
300	1000	0.85	**25.13**	310.81	23.9	183.55	24.9	292.12	24.6	344.27	14.8	561.65
400	100	0.85	**53.57**	106.36	51.0	76.36	53.37	136.34	51.57	147.01	53.53	468.29
400	500	0.85	27.5	289.57	27.07	165.47	27.23	291.06	**28.53**	374.47	19.93	580.17
400	1000	0.85	24.03	336.02	23.1	228.0	**24.1**	361.28	23.87	418.25	12.93	577.56
	Averages		64.13	243.97	**65.62**	236.29	65.25	300.05	65.34	328.25	65.54	461.25

Bold value in a row indicates a best value for the problem instance corresponding to that row

Table 3 Comparison of CMSA and the four proposed RL-CMSA variants for the FFMS problem instances with $|\Sigma| = 12$.

n	m	th	CMSA		RL-CMSA- 1		RL-CMSA- 2		RL-CMSA- 3		RL-CMSA- 4	
			$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$
100	100	0.97	74.1	212.25	**74.13**	262.15	74.1	203.66	74.03	228.75	74.03	228.75
100	500	0.97	66.63	277.79	67.43	390.97	67.4	308.74	**68.23**	447.03	**68.23**	447.03
100	1000	0.97	64.0	293.85	64.4	292.47	63.83	247.68	**65.33**	332.88	**65.33**	332.88
200	100	0.97	**98.0**	274.53	97.87	280.4	97.43	199.52	97.27	240.74	97.27	240.74
200	500	0.97	74.07	316.19	75.43	419.16	74.43	348.78	**76.07**	548.6	**76.07**	548.6
200	1000	0.97	67.17	316.43	68.23	297.67	68.0	257.22	**69.1**	411.72	**69.1**	411.72
300	100	0.97	111.5	276.12	**113.23**	249.19	112.07	203.91	111.73	251.09	111.73	251.09
300	500	0.97	76.1	327.43	78.07	438.66	77.2	370.17	**78.53**	532.56	**78.53**	532.56
300	1000	0.97	67.9	305.39	**69.73**	385.95	68.57	272.06	69.7	451.66	69.7	451.66
400	100	0.97	123.2	284.09	**124.0**	282.25	122.83	182.89	122.37	255.53	122.37	255.53
400	500	0.97	77.57	334.7	80.53	467.82	78.53	367.64	**80.77**	574.63	**80.77**	574.63
400	1000	0.97	68.23	343.07	69.97	368.69	69.17	294.91	**70.13**	476.1	**70.13**	476.1
100	100	1.0	31.43	120.36	**31.53**	198.29	**31.53**	149.39	31.4	87.54	31.4	87.54
100	500	1.0	**19.33**	98.86	19.3	332.61	**19.33**	112.73	19.2	187.29	19.2	187.29
100	1000	1.0	17.0	46.18	**17.07**	297.36	**17.07**	153.82	17.0	190.55	17.0	190.55
200	100	1.0	34.6	153.45	**34.73**	204.35	34.47	87.88	34.6	130.07	34.6	130.07
200	500	1.0	19.23	76.03	19.37	256.75	19.3	159.1	**19.4**	269.48	**19.4**	269.48
200	1000	1.0	17.0	51.3	**17.03**	267.31	**17.03**	202.05	17.0	233.01	17.0	233.01
300	100	1.0	36.23	129.72	**36.33**	238.39	36.17	182.63	36.3	155.93	36.3	155.93
300	500	1.0	19.2	97.21	**19.53**	211.86	19.13	74.39	19.17	251.94	19.17	251.94
300	1000	1.0	16.97	154.56	**17.0**	257.58	16.63	180.09	**17.0**	267.89	**17.0**	267.89
400	100	1.0	**37.23**	127.64	**37.23**	215.16	37.03	139.55	37.1	171.43	37.1	171.43
400	500	1.0	19.07	81.34	**19.4**	295.79	19.2	82.58	19.07	291.77	19.07	291.77
400	1000	1.0	**16.97**	106.38	**16.97**	300.63	16.73	165.56	16.83	282.69	16.83	282.69
Averages			52.20	200.20	**52.85**	300.48	52.38	206.12	52.81	302.95	52.81	302.95

Bold value in a row indicates a best value for the problem instance corresponding to that row

Table 4 Comparison of CMSA and the four proposed RL-CMSA variants for the FFMS problem instances with $|\Sigma| = 20$.

n	m	th	CMSA		RL-CMSA- 1		RL-CMSA- 2		RL-CMSA- 3		RL-CMSA- 4	
			$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s\rvert}$	$\bar{t}_{best}[s]$
100	100	0.99	**86.93**	264.18	86.8	340.17	86.77	354.57	86.7	289.77	**86.93**	343.13
100	500	0.99	84.33	289.07	84.23	263.97	84.3	238.22	**84.47**	274.28	84.33	371.33
100	1000	0.99	81.13	238.01	82.2	363.23	81.47	369.57	81.77	329.39	**82.37**	482.04
200	100	0.99	**117.93**	288.9	117.53	290.23	117.73	218.22	117.37	223.52	117.83	369.71
200	500	0.99	93.6	383.37	93.13	222.34	**93.73**	254.29	93.27	222.68	93.27	390.0
200	1000	0.99	86.53	403.65	87.97	392.85	87.3	305.23	**88.17**	258.16	87.53	463.65
300	100	0.99	136.23	287.32	**136.43**	293.3	135.77	229.36	135.63	221.25	136.0	323.21
300	500	0.99	**97.87**	423.38	97.47	264.78	97.47	243.77	96.93	207.97	97.0	375.41
300	1000	0.99	88.4	397.48	**89.57**	366.82	89.33	330.86	89.5	265.75	89.17	457.22
400	100	0.99	**151.33**	317.64	151.1	297.16	150.4	226.14	149.97	235.68	151.2	356.39
400	500	0.99	**99.53**	381.12	98.93	242.14	99.43	250.43	98.1	188.94	99.0	363.46
400	1000	0.99	89.03	387.21	**90.93**	383.92	90.17	303.07	90.13	298.03	89.9	442.7
100	100	1.0	**62.23**	65.82	62.07	279.75	62.13	58.51	62.1	215.89	62.17	45.58
100	500	1.0	43.63	273.42	43.8	234.2	43.33	195.72	43.43	246.82	**44.3**	306.74
100	1000	1.0	37.83	187.77	38.03	326.79	37.5	209.96	37.77	209.79	**39.5**	362.67
200	100	1.0	77.1	212.56	**77.2**	245.91	76.93	127.28	76.6	282.61	76.73	143.46
200	500	1.0	43.77	237.56	44.13	286.73	43.97	288.44	43.87	252.45	**46.07**	511.86
200	1000	1.0	37.47	172.53	38.2	256.08	37.73	238.04	37.7	220.21	**40.2**	430.01
300	100	1.0	**84.63**	170.86	84.27	243.04	83.5	110.76	83.77	253.05	84.5	369.9
300	500	1.0	**44.57**	278.63	44.53	257.92	43.93	239.32	44.47	288.59	43.9	392.82
300	1000	1.0	38.03	244.75	38.2	281.44	37.4	219.12	37.87	220.14	**40.2**	511.33
400	100	1.0	**90.0**	194.44	89.6	256.06	88.6	185.81	88.93	223.78	88.67	480.85
400	500	1.0	44.37	282.37	**44.87**	275.58	44.13	284.43	44.63	283.48	42.83	206.38
400	1000	1.0	37.87	193.87	38.07	260.2	37.53	251.84	38.13	258.32	**38.37**	471.83
Averages			77.26	274.00	77.47	288.53	77.11	238.87	77.14	248.77	**77.58**	373.82

Bold value in a row indicates a best value for the problem instance corresponding to that row

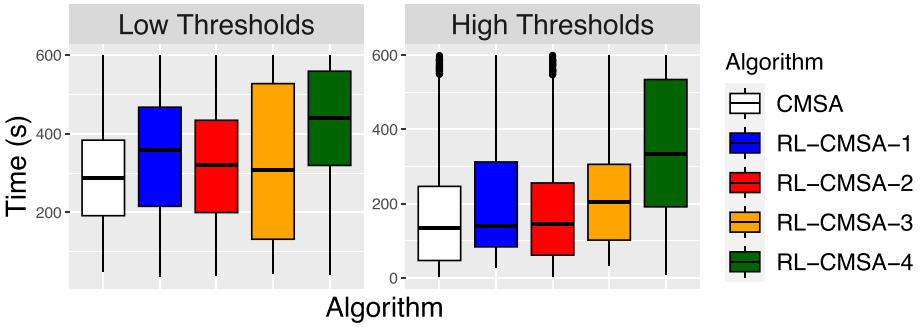


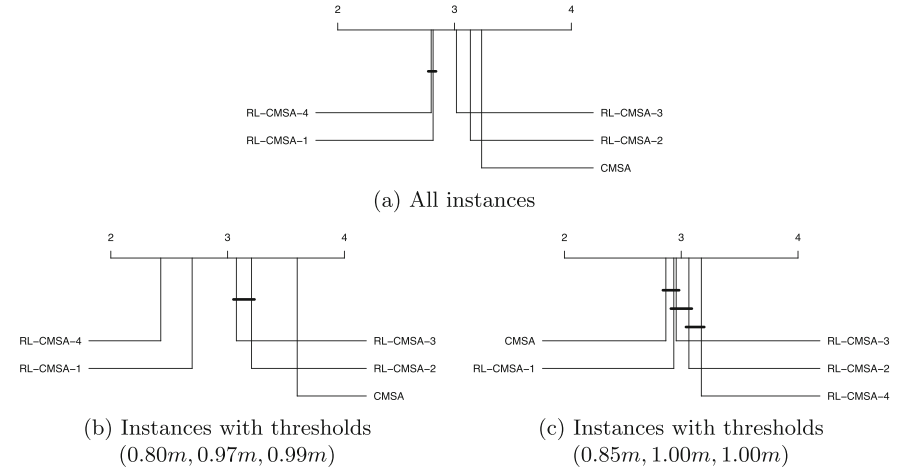
Fig. 1 Time spent by the algorithms for obtaining their best solutions to the FFMS problem instances

- RL-CMSA-1 obtains the best results except for the instances with $|\Sigma| = 20$. For these, RL-CMSA-4 is the algorithm that obtains the best average results.
- Concerning computation time, the first three RL-CMSA variants seem to require a similar amount of time to the one required by the standard CMSA.
- In contrast, RL-CMSA-4 finds its best solutions later, employing a larger part of the 600-second time limit.

Figure 1 illustrates the differences in computation time further. It contains five box plots for the two threshold groups, representing the time required by each algorithm variant. All algorithms use more time to find their best solutions for the low threshold instances in comparison to the high threshold ones. This is because the latter instances are much harder, which causes the algorithms sometimes to get stuck in local optima.

Figure 2 contains Critical Difference (CD) plots for the FFMS problem results, generated using the *R* package `scmamp` (Calvo & Santafé Rodrigo, 2016). Each plot shows the average rank of every algorithm on the x-axis, with a horizontal bar between algorithms denoting non-significant differences. The Friedman rank-sum test indicated, with high significance, that at least one algorithm performs differently than the rest. Thus, we employed Finner's procedure (García et al., 2010) as the *post-hoc* method for pairwise comparison. The CD plots show the results obtained regarding this method, using a significance level of 0.05. The one from Fig. 2a considers all instances together. We can observe that standard CMSA obtains the worst average rank and that the differences between CMSA and the RL-CMSA variants are statistically significant. Moreover, RL-CMSA-1 and RL-CMSA-4 are the best-performing algorithms, being better than the rest with statistical significance. Figures 2b and c show CD plots for the lower and higher threshold instances respectively. For the lower threshold instances, CMSA also obtains the worst average rank and is the worst algorithm with statistical significance. In the case of the higher threshold instances, the differences between the algorithms in terms of average rank are much smaller. In this case, CMSA obtains the best average rank but the differences with RL-CMSA-1 and RL-CMSA-3 are non-statistically significant. Interestingly, the best-performing algorithm for the lower threshold instances, RL-CMSA-4, obtains the worst average rank for the higher threshold ones.

With the intention of better understanding the behavioral differences between CMSA and the proposed RL-CMSA variants we generated the graphics in Figs. 3 and 4. They contain a plot for 12 exemplary FFMS problem instances with $m = 500$ for every combination of $n \in \{100, 400\}$ and $|\Sigma| \in \{4, 12, 20\}$. In particular, the plots in Fig. 3 show the fraction of solution constructions (y-axis) for which each solution component (x-axis) was selected. Note that in all these plots the solution components are ordered from the most selected one



(a) All instances

(b) Instances with thresholds $(0.80m, 0.97m, 0.99m)$

(c) Instances with thresholds $(0.85m, 1.00m, 1.00m)$

Fig. 2 CD plots concerning the FFMS results

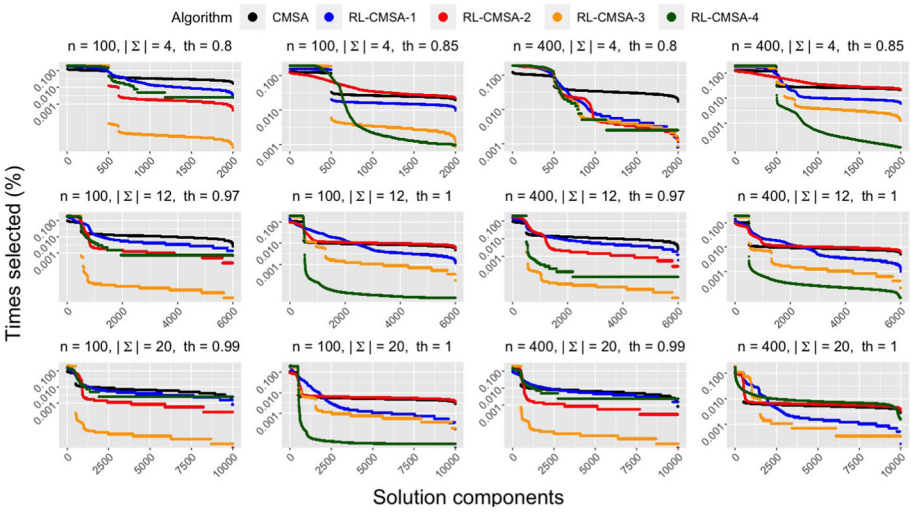


Fig. 3 Exploration plots for the FFMS problem. The x-axis represents the solution components and the y-axis the percentage of times each one was chosen

(left) to the least selected one (right). Hereby, the y-axis is plotted in a logarithmic scale, to be able to see differences in a better way.

The most important aspect shown in these graphics is that CMSA—in comparison to the RL-CMSA variants—generally selects to a lesser extent the highly chosen solution components, while it generally shows a higher fraction of selection for less chosen solution components. This can simply be explained by the presence of RL in the RL-CMSA variants, which leads to higher exploitation of seemingly good solution components. However, there are also differences between the RL-CMSA variants. Algorithm variants RL-CMSA-3 and RL-CMSA-4, for example, generally show a lesser degree of exploration than the other RL-CMSA variants.

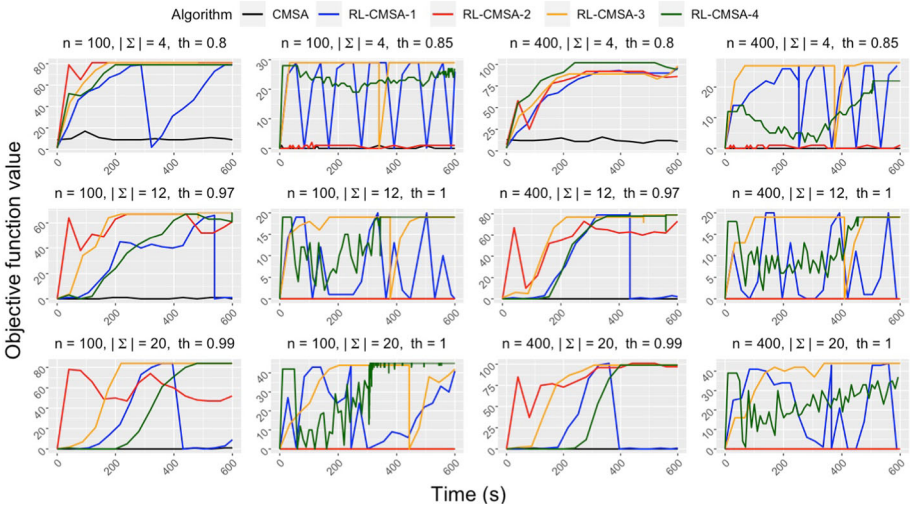


Fig. 4 Evolution of the objective function values of the constructed solutions over time for the FFMS problem

Figure 4 contains 12 plots, concerning the problem instances already considered in Fig. 3. These graphics show, for all five algorithm variants, the quality of the solutions constructed over time. Run time is represented on the x-axis in terms of seconds. The y-axis shows the objective function values of the constructed solutions. To improve the visualization, only the value of the best solution constructed at each iteration is used for plotting.

These graphics clearly show one of the benefits of implementing RL into CMSA. Due to employing learning, the RL-CMSA variants construct solutions of higher quality than the latter. Observe, for example, that the quality of the solutions constructed by the RL-CMSA variants grows over time. Conversely, in the case of CMSA, the quality of the solutions constructed stays more or less constant as there is no form of learning involved. Notably, the solutions constructed by CMSA are of really low quality as their objective function values often are close to 0.

In addition, these graphics show the differences in the behavior of the learning processes of the different RL-CMSA variants. In the context of RL-CMSA-1 and RL-CMSA-3, for example, important drops in solution quality can be noted over time. These correspond to algorithm restarts. Hereby, RL-CMSA-1 conducts most restarts, most notably for the high threshold instances for which it restarts multiple times. On the other hand, RL-CMSA-3 restarts once for every high threshold instance, except for the instance with $n = 400$, $|\Sigma| = 20$ and $t = 1.00m$.

4.3 Application to the minimum dominating set (MDS) problem

The MDS problem is another well-known NP-hard combinatorial optimization problem from the literature. Given a graph, the MDS problem aims at finding a smallest subset of nodes such that every node of the graph is either part of this subset or has at least one neighbor in it. More formally, let $G = (V, E)$ be an undirected graph. The MDS problem aims at finding a smallest $\tilde{V} \subseteq V$ such that for every $v \in V$ at least one of the following two conditions holds:

1. $v \in \tilde{V}$

2. $v' \in \tilde{V}$ for some $v' \in N(v)$

Hereby, $N(v)$ denotes the set of neighbors of v in G. That is, $N(v) := \{v' \in V \mid (v', v) \in E\}$. A subset of nodes that fulfills the previous two conditions is called a dominating set of G. Hence, the MDS problem aims at finding a minimum dominating set, as the name of the problem suggests. The MDS problem has applications in different fields, such as in wireless sensor networks (Pino et al., 2018) and natural language processing (Shen & Li, 2010).

The following subsections introduce the definition of solution components, the way of solving sub-instances, and the way of constructing valid solutions.

4.3.1 Solution components

A natural way of defining the solution components in the case of the MDS problem consists of introducing a solution component for every node of the input graph. The set of solution components is then $C = V$. Therefore, we henceforth employ the v-notation instead of the c-notation for solution components, that is, $C := \{v_1, \ldots, v_n\}$, where each solution component v_i is a node of the input graph G. At each step of the construction process of a solution $S \subseteq C$, the set of available solution components consists of all the nodes except for those that are already covered by a node in S and have no uncovered neighbors.

4.3.2 Probabilistic solution construction

Both CMSA and the RL-CMSA variants utilize the following solution construction mechanism. It starts with an empty solution $S := \emptyset$. At each step of the process exactly one node (that is, a solution component) is added until a valid solution—being a dominating set—is obtained. Hereby, let $C_{\text{feas}} \subseteq C$ denotes—as before—the set of feasible solution components at the current step, which—in the case of the MDS problem—is defined as the set of nodes that can cover one or more nodes not already covered by the current partial solution S.

CMSA makes use of the following greedy function for choosing, at each construction step, a node from C_{feas}. For the introduction of this greedy function, let $N[v] := N(v) \cup \{v\}$ denote the closed neighborhood of v and $N[v \mid S] \subseteq N[v]$ denote the set of uncovered neighbors of v concerning partial solution S. For the choice of a node to be added to S, the following is done in CMSA:

1. With a probability $0 \leq dr_{\text{CMSA}} \leq 1$, a node $v \in C_{\text{feas}}$ is chosen as follows:

$$v := \underset{v' \in C_{\text{feas}}}{\arg\max} \left\{ \left| N[v' \mid S] \right| \right\} \tag{16}$$

2. Otherwise, with a probability $1 - dr_{\text{CMSA}}$, a number of $\min\{l^{\text{size}}_{\text{CMSA}}, \left|C_{\text{feas}}\right|\}$ nodes from C_{feas} are stored in $L \subseteq C_{\text{feas}}$ such that:

$$\left| N[v \mid S] \right| \leq \left| N[v' \mid S] \right| \quad \text{for all } v \in L, v' \in C_{\text{feas}} \setminus L \tag{17}$$

   A node $v \in L$ is then chosen uniformly at random and added to S.

Hereby, dr_{CMSA} and $l^{\text{size}}_{\text{CMSA}}$ are parameters of the CMSA algorithm.

The RL-CMSA variants avoid using this greedy function. They make use of a set of q-values containing a value q_i for each node (solution component) $v_i \in C$. The choice of a node at each construction step is made via Softmax selection, UCB selection respectively.

4.3.3 ILP model and sub-instance solving

Similarly to the FFMS problem, our algorithms utilize the commercial solver CPLEX in their *solve* step. The following ILP model for the MDS problem is employed by CPLEX.

$$\min \sum_{v_i \in V} x_i \tag{18}$$

$$\text{subject to} \sum_{v_j \in N(v_i)} x_j + x_i \geq 1, \qquad \text{for } v_i \in V$$

$$x_i \in \{0, 1\}, \qquad \text{for } v_i \in V \tag{19}$$

As one can see, binary variable x_i takes value one if solution component $v_i \in V$ forms part of the solution and value zero otherwise. Constraints (19) cause solutions to be dominating sets as, for every node, it is required that either the node itself and/or one of its neighbors belong to the solution. Finally, the minimization goal causes the size of the final solution to be minimum.

To solve a sub-instance $C' \subseteq C$, for all $v_j \in C \backslash C'$ the constraint $x_j = 0$ is added to this ILP model. In other words, the values of those variables that correspond to solution components (nodes) not forming part of sub-instance C' are fixed to zero.

4.3.4 Experimental evaluation

For evaluating standard CMSA and the RL-CMSA variants for the MDS problem, we used a benchmark set consisting of graphs of different sizes and densities generated by using the following three graph models: Erdös-Rényi (Erdös & Rényi, 1959), Watts-Strogatz (Watts & Strogatz, 1998) and Barabási-Albert (Barabási & Albert, 1999). The first is one of the best-known random graph models for generating graphs using two parameters: the number of nodes and the probability of the existence of an edge between any pair of nodes. The second is used for generating small-world networks, which have a short average shortest path length between nodes and maintain a high level of local clustering. Finally, the latter produces graphs with a majority of low-degree nodes and a few significantly higher-degree ones.

We generated 30 graphs of every graph type and for every combination of $|V| \in \{500, 1000, 1500, 2000\}$ and four different graph densities. Densities are controlled by parameters p, k, and m for the three graph models, respectively. The four densities considered are $p \in \{0.00416381, 0.0062414, 0.0103881, 0.020705\}$ and $k, m \in \{2, 3, 5, 10\}$.[1] Henceforth, these will be called 1st, 2nd, 3rd, and 4th density level respectively. The benchmark set therefore consists of 480 graphs for every model, totalling 1440 instances. Additionally, it contains one tuning instance for every graph type, density level, and size.

Parameter tuning. The five CMSA variants were tuned using the tuning instances of the lowest and highest density levels, that is, the instances concerning the 2nd and 3rd density levels were disregarded to speed up the procedure. This amounts to 24 tuning instances in total. As in the case of the FFMS problem, tuning was conducted using the *R* tool `irace` (López-Ibáñez et al., 2016) with a budget of 3000 experiments per tuning run. For both tuning and evaluation, every algorithm execution was given a time limit of 150, 300, 450, and 600 CPU seconds for instances of sizes $|V| \in \{500, 1000, 1500, 2000\}$ respectively.

[1] Note that the edge probabilities (p) in the Erdös-Rényi model were selected such that the densities of the produced graphs matches the ones of the other models.

Table 5 presents the parameter values obtained after tuning, together with their allowed ranges. The only change in comparison to the FFMS problem is the additional CMSA-parameter $l^{\text{size}}_{\text{CMSA}}$ and the allowance of a smaller range of values for t_{ILP}. Parameter $l^{\text{size}}_{\text{CMSA}}$ is used together with dr_{CMSA} in the standard CMSA solution construction procedure and the allowed range for parameter t_{ILP} was shortened due to the ILP solver requiring less time for solving MDS sub-instances, compared to FFMS sub-instances.

Results. Tables 6, 7, 8, 9 present the obtained results for the MDS problem instances. The same structure is used as in the case of the FFMS problem, just that each table in the case of the MDS problem presents the results for problem instances of a specific graph size. Each result is an average over the 30 problem instances for a specific combination of graph type, $|V|$, and density level. The results allow us to observe the following:

- RL-CMSA-1 generally performs best, followed by the standard CMSA.
- The differences between the algorithms grow with growing graph size.
- RL-CMSA-1 improves over CMSA to a larger extent in the context of Erdös-Rényi and Watts-Strogatz graphs than for Barabási-Albert graphs.
- Interestingly, algorithm variants RL-CMSA-2, RL-CMSA-3, and RL-CMSA-4 generally perform slightly worse than standard CMSA, except RL-CMSA-3 for smaller problem instances.
- These results together with the ones of the FFMS problem show that RL-CMSA-1 seems to be the best RL-CMSA variant.

Figure 6 provides four box plot graphics, one for each value of $|V|$, showing the time taken by each algorithm to encounter the best solution in each run. These graphics show that the four RL-CMSA variants spent a similar amount of time in finding the best solutions in their respective runs. Remember that for the MDS problem, the time limit was set to 150, 300, 450, and 600 CPU seconds for the four considered graph sizes, respectively. RL-CMSA-1 is the algorithm that employs the most time for instances with $|V| = 1500$ and $|V| = 2000$ which coincides with it being the best-performing algorithm for these large instances.

To check for statistical significance, the CD plots shown in Fig. 5 were produced. These were again generated using the *R* package `scmamp` utilizing the same statistical testing procedure as in the case of the FFMS problem. The CD plot in Fig. 5a considers all problem instances together, while the CD plots in Figs. 5b–d are restricted to problem instances of a specific graph type. It can be observed that RL-CMSA-1 is the algorithm that obtains the best average rank and that the difference with the rest of the algorithms is statistically significant. Studying the CD plots for instance subsets, it can be seen that for Barabási-Albert instances, the resulting average ranks are similar, with RL-CMSA-1 and RL-CMSA-2 being slightly better than standard CMSA, without statistical significance. However, for Erdös-Rényi and Watts-Strogatz instances, RL-CMSA-1 is the best-performing algorithm and the differences are, this time, statistically significant.

To gain a deeper understanding of the algorithm behavior, Figs. 7 and 8 show—as in the case of the FFMS problem—the exploration behavior of the algorithms, respectively the evolution of the quality of the constructed solutions over time. For this purpose, one instance was considered for every graph type and size ($|V|$), totaling 12 instances. All of the selected instances are from the 3rd density level. As in the case of the FFMS problem, Fig. 7 plots for every algorithm and problem instance the fraction of solution constructions in which solution components were selected. The x-axis represents the solution components, ordered from the most to the least selected one for each algorithm.

The exploration plots show some differences to the FFMS case. First, the behavior of standard CMSA is considerably different from the one displayed for the FFMS problem. In

Table 5 Parameter values obtained after tuning for the MDS problem. Every algorithm is tuned exactly once. A dash (–) denotes that the algorithm does not use the corresponding parameter

	Allowed range	CMSA	RL-CMSA- 1	RL-CMSA- 2	RL-CMSA- 3	RL-CMSA- 4
t_{ILP}	$\{1, 2, \ldots, 20\}$	13	6	14	5	17
dr	$\{0.0, 0.01, \ldots, 0.99\}$	–	0.44	0.68	0.51	0.43
n_a	$\{1, 2, \ldots, 50\}$	4	10	8	2	1
age_{max}	$\{1, 2, \ldots, 10\}$	3	1	7	6	6
$\text{cplex}_{\text{warmstart}}$	$\{0, 1\}$	0	0	0	0	0
$\text{cplex}_{\text{emphasis}}$	$\{0, 1\}$	1	1	1	1	1
$\text{cplex}_{\text{abort}}$	$\{0, 1\}$	0	0	0	0	1
β	$\{0.0, 0.01, \ldots, 2.0\}$	–	0.28	0.94	0.28	–
b_{reset}	$\{0, 1\}$	–	1	1	0	–
cf_{limit}	$\{0.90, 0.91, \ldots, 1.0\}$	–	0.98	0.97	1.00	–
r	$\{1, 2, \ldots, 10\}$	–	–	1.78	1.73	–
α	$\{0.1, 0.2, \ldots, 1.0\}$	–	–	–	0.56	–
ucb_c	$\{0.0, 0.1, \ldots, 5.0\}$	–	–	–	–	3.61
dr_{CMSA}	$\{0.0, 0.01, \ldots, 0.99\}$	0.29	–	–	–	–
$l^{\text{size}}_{\text{CMSA}}$	$\{3, 4, \ldots, 50\}$	35	–	–	–	–

Table 6 Comparison of CMSA with the four RL-CMSA variants for the MDS problem instances with $|V| = 500$

Graph type	Density level	CMSA		RL-CMSA- 1		RL-CMSA- 2		RL-CMSA- 3		RL-CMSA- 4	
		$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$
Barabási-Albert	1st	**101.23**	0.04	**101.23**	0.13	**101.23**	0.13	**101.23**	0.21	**101.23**	0.10
Barabási-Albert	2nd	**71.97**	0.96	**71.97**	3.11	**71.97**	3.11	**71.97**	5.99	**71.97**	0.62
Barabási-Albert	3rd	47.90	18.01	**47.87**	20.58	**47.87**	20.58	47.90	45.57	47.97	2.01
Barabási-Albert	4th	27.13	18.94	**27.10**	28.65	**27.10**	28.65	27.33	61.15	27.23	12.69
Erdös-Rényi	1st	**209.97**	0.20	**209.97**	0.08	**209.97**	0.08	**209.97**	0.15	**209.97**	0.03
Erdös-Rényi	2nd	**153.37**	0.66	**153.37**	0.17	**153.37**	0.17	**153.37**	0.30	**153.37**	0.10
Erdös-Rvnyi	3rd	101.90	42.66	**101.67**	28.90	**101.67**	28.90	**101.67**	41.95	101.77	15.33
Erdös-Rényi	4th	60.37	55.80	60.60	41.67	60.60	41.67	**60.30**	83.29	60.47	53.35
Watts-Strogatz	1st	110.33	42.43	**110.13**	39.12	**110.13**	39.12	**110.13**	58.31	110.27	13.13
Watts-Strogatz	2nd	82.40	70.20	**82.23**	57.67	**82.23**	57.67	82.40	67.55	82.30	36.43
Watts-Strogatz	3rd	**57.20**	72.67	57.67	38.12	57.67	38.12	57.23	90.98	57.40	65.09
Watts-Strogatz	4th	**34.87**	59.06	35.13	51.79	35.13	51.79	34.93	89.77	35.13	51.11
Averages		88.22	31.80	88.25	25.83	88.25	25.83	**88.20**	45.44	88.26	20.83

Bold value in a row indicates a best value for the problem instance corresponding to that row

Table 7 Comparison of CMSA with the four RL-CMSA variants for the MDS problem instances with $|V| = 1000$

Graph type	Density level	CMSA		RL-CMSA- 1		RL-CMSA- 2		RL-CMSA- 3		RL-CMSA- 4	
		$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$
Barabási-Albert	1st	**202.07**	0.37	**202.07**	0.43	**202.07**	0.52	**202.07**	0.92	**202.07**	0.26
Barabási-Albert	2nd	145.07	15.69	**145.00**	20.90	**145.00**	31.27	**145.00**	45.32	145.13	9.49
Barabási-Albert	3rd	92.27	64.60	**92.20**	41.20	**92.20**	53.41	92.43	92.13	92.40	37.44
Barabási-Albert	4th	50.40	72.50	50.67	56.36	**50.33**	54.26	50.60	130.27	50.63	60.22
Erdös Rényi	1st	241.43	90.70	**241.17**	20.15	**241.17**	49.75	**241.17**	47.19	241.23	16.50
Erdös Rényi	2nd	175.30	145.99	174.37	114.85	174.83	144.09	**174.27**	144.09	174.73	116.80
Erdös Rényi	3rd	122.07	126.97	**120.67**	176.69	122.47	168.58	122.10	178.12	122.80	149.31
Erdös Rényi	4th	75.73	114.87	**75.23**	220.15	76.50	183.62	76.40	169.53	77.10	108.30
Watts-Strogatz	1st	220.90	163.33	220.10	83.28	**220.07**	135.09	220.17	95.54	**220.07**	75.81
Watts-Strogatz	2nd	166.33	151.48	165.03	132.51	165.80	191.26	**164.57**	173.63	165.80	168.26
Watts-Strogatz	3rd	117.37	156.41	**115.33**	167.53	117.90	175.81	117.23	198.06	118.80	137.70
Watts-Strogatz	4th	72.10	126.95	**70.63**	245.70	73.30	137.09	73.73	148.71	73.77	112.12
Averages		140.09	102.49	**139.37**	106.65	140.14	110.40	139.98	118.63	140.38	82.68

Bold value in a row indicates a best value for the problem instance corresponding to that row

Table 8 Comparison of CMSA with the four RL-CMSA variants for the MDS problem instances with $|V| = 1500$

Graph type	Density level	CMSA		RL-CMSA- 1		RL-CMSA- 2		RL-CMSA- 3		RL-CMSA- 4	
		$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$
Barabási-Albert	1st	303.97	18.43	**303.83**	1.52	**303.83**	1.02	**303.83**	2.10	**303.83**	0.52
Barabási-Albert	2nd	213.73	48.63	**213.67**	61.82	**213.67**	94.31	213.73	69.76	213.83	20.63
Barabási-Albert	3rd	136.83	146.81	**136.77**	98.99	136.80	131.22	136.83	140.56	137.13	103.54
Barabási-Albert	4th	**73.30**	107.81	73.60	128.50	73.33	114.41	73.80	154.06	73.83	141.01
Erdös Rényi	1st	263.17	261.17	261.30	172.09	262.63	277.38	**261.00**	303.51	262.00	311.76
Erdös Rényi	2nd	198.47	221.23	**194.80**	296.68	199.47	249.92	199.07	250.18	199.83	243.84
Erdös Rényi	3rd	140.50	194.25	**138.17**	271.25	142.67	218.57	141.97	266.84	141.90	193.44
Erdös Rényi	4th	87.40	135.35	**86.13**	335.87	87.57	88.53	88.47	260.33	88.57	205.78
Watts-Strogatz	1st	331.10	284.44	**329.60**	160.07	**329.60**	222.72	329.80	134.79	329.63	125.67
Watts-Strogatz	2nd	251.77	239.13	**248.30**	225.37	251.30	267.37	248.73	323.75	251.87	330.84
Watts-Strogatz	3rd	178.60	216.38	**174.57**	233.85	180.13	205.79	184.77	271.20	181.40	278.46
Watts-Strogatz	4th	110.97	143.81	**106.73**	328.33	113.73	166.40	114.47	259.05	113.73	182.92
Averages		190.82	168.12	**188.96**	192.86	191.23	169.80	191.37	203.01	191.46	178.20

Bold value in a row indicates a best value for the problem instance corresponding to that row

Table 9 Comparison of CMSA with the four RL-CMSA variants for the MDS problem instances with $|V| = 2000$

Graph type	Density level	CMSA		RL-CMSA- 1		RL-CMSA- 2		RL-CMSA- 3		RL-CMSA- 4	
		$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$	$\overline{\lvert s \rvert}$	$\bar{t}_{best}[s]$
Barabási-Albert	1st	403.70	23.70	**403.57**	1.73	**403.57**	1.86	**403.57**	3.06	**403.57**	0.75
Barabási-Albert	2nd	285.67	81.66	**285.40**	121.12	285.50	107.11	285.57	103.34	285.63	56.80
Barabási-Albert	3rd	180.93	177.78	**180.87**	149.31	**180.87**	210.69	180.93	166.92	180.93	184.27
Barabási-Albert	4th	95.57	152.56	95.70	215.08	**95.50**	246.96	95.70	192.39	96.03	180.58
Erdös Rényi	1st	289.10	255.75	**284.17**	290.83	289.90	292.71	294.13	320.99	291.20	395.06
Erdös Rényi	2nd	218.77	283.50	**214.07**	401.48	223.87	176.65	222.63	353.27	222.93	309.83
Erdös Rényi	3rd	156.30	208.89	**153.50**	369.36	161.40	110.74	159.30	352.17	159.43	231.19
Erdös Rényi	4th	96.17	181.82	**95.07**	436.07	98.30	175.25	99.23	429.18	97.20	253.25
Watts-Strogatz	1st	442.63	367.59	440.70	200.91	440.33	337.78	**440.27**	262.85	440.73	184.14
Watts-Strogatz	2nd	337.77	328.00	**331.67**	292.25	338.77	306.87	335.67	413.62	339.87	427.45
Watts-Strogatz	3rd	241.97	243.74	**233.13**	378.62	246.40	137.86	249.87	335.55	249.80	251.58
Watts-Strogatz	4th	152.57	58.85	**145.80**	421.32	158.70	187.22	164.10	101.52	156.27	249.17
Averages		241.76	196.99	**238.64**	273.17	243.59	190.98	244.25	252.91	243.63	227.01

Bold value in a row indicates a best value for the problem instance corresponding to that row

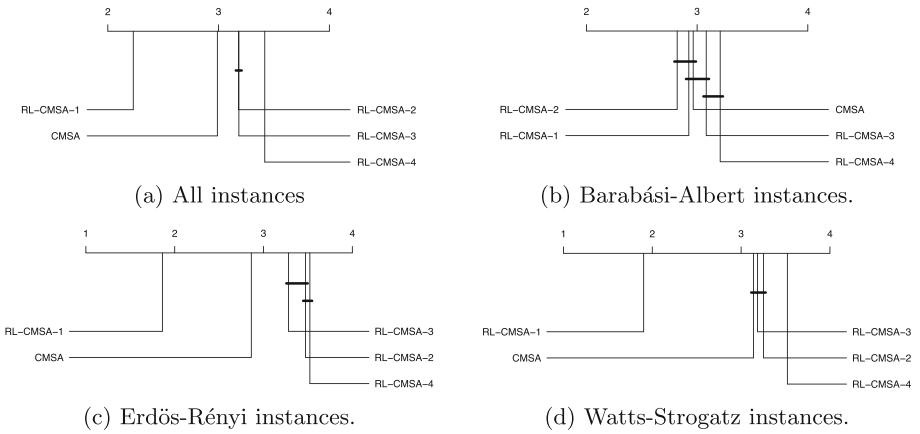


Fig. 5 CD plots concerning the MDS results

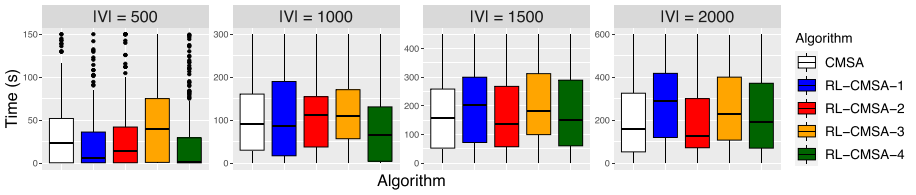


Fig. 6 Time spent by the algorithms for obtaining their best solutions to the MDS problem instances

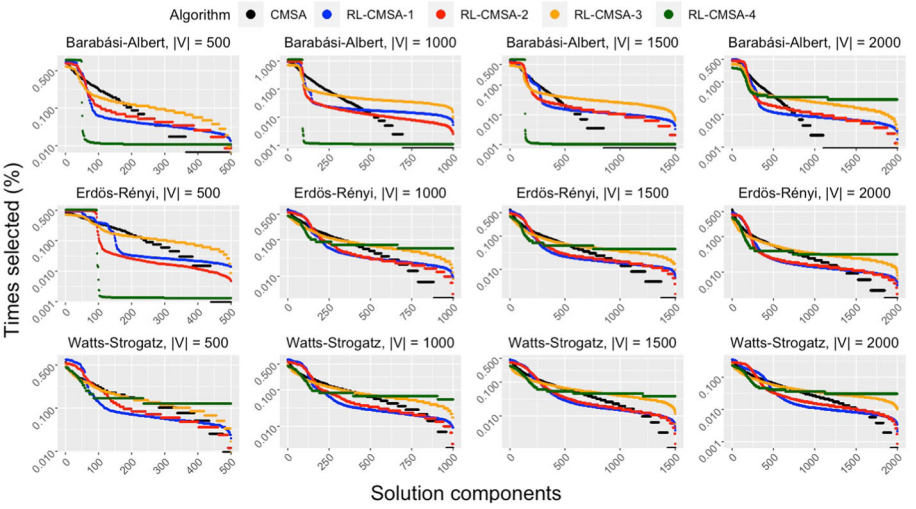


Fig. 7 Exploration plots for the MDS problem. The x-axis represents the solution components and the y-axis the fraction of solution constructions in which each one was chosen

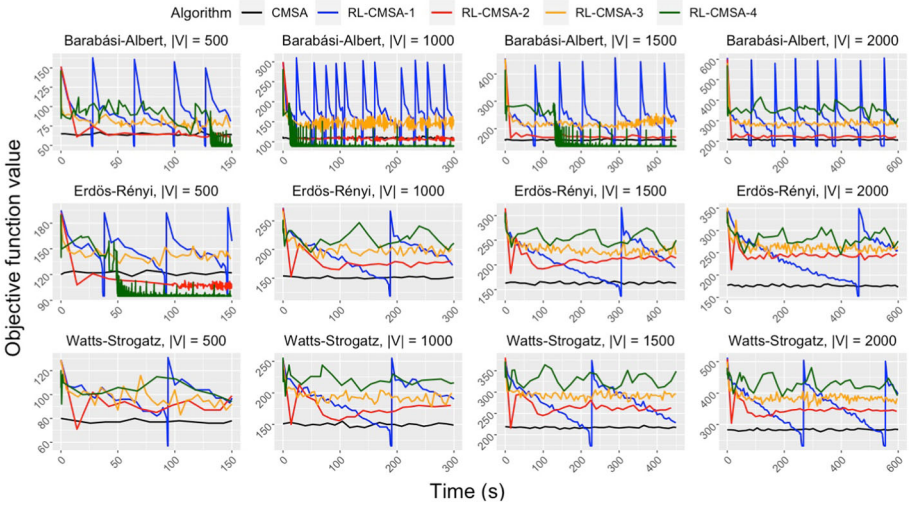


Fig. 8 Evolution of the objective function values of the constructed solutions over time for the MDS problem

particular, the exploration of few-chosen solution components drops drastically (in steps) at some point. We conjecture that this is because of the use of the $l^{\text{size}}_{\text{CMSA}}$ parameter that limits the number of selectable solution components (nodes) at each construction step. While its use is beneficial for the global performance of CMSA, apparently it reduces the exploration capability of the algorithm. In addition to this observation, we can also detect differences in the relative behavior of the RL-CMSA variants. While, in the context of the FFMS problem, RL-CMSA-1 and RL-CMSA-2 showed a higher exploration of few-chosen solution components than RL-CMSA-3 and RL-CMSA-4, this is generally the other way around for the MDS problem. This is except for smaller Barabási-Albert and Erdös-Rényi graphs.

Finally, Fig. 8 plots the quality (in terms of the objective function values) of the solution constructions performed over the run-time of the algorithms. Note that, in the case of the MDS problem, higher-quality solutions correspond to lower objective function values (which was the opposite for the FFMS problem).

Again, these graphics show the learning process of the RL-CMSA versions. In this case, CMSA performs better solution constructions compared to the RL-CMSA versions than it did for the FFMS problem. For most instances, RL-CMSA-1 constructs the best quality solutions, which coincides with the fact that this is the best-performing algorithm for this problem. The other three RL-CMSA versions perform worse solution constructions in general, except for RL-CMSA-4 which is the best at constructing solutions for the Barabasi instances.

A last interesting observation is that RL-CMSA-1 is the only algorithm that performs restarts for the MDS problem. As seen before, every restart produces a spike in the graphs showing the evolution of solution quality. This is because a restart causes the information gathered so far to be erased.

5 Conclusions and future work

The use of ML techniques for supporting and improving metaheuristics is a successful current trend in the literature. Following this trend, this work has introduced a new version of the

hybrid metaheuristic CMSA by adding a RL component for constructing solutions at each iteration. This new CMSA variant, called RL-CMSA, improves over the standard CMSA in two aspects. First of all, its application is not dependent on a tailored greedy function for evaluating solution components at each solution construction step. Therefore, RL-CMSA can be seen as a more general algorithm than standard CMSA, which often is also easier to implement. The goodness/usefulness of solution components is learned online in RL-CMSA by means of the q-value sampling and update. Moreover, RL-CMSA was shown to improve over standard CMSA in terms of empirical performance both in the context of the FFMS and the MDS problem. The main conclusion of this research is that equipping CMSA with the proposed simple learning mechanism is highly successful. Therefore, this new variant should be tested for problems in which CMSA excels as it could potentially perform even better. Specifically, the best-performing variant is RL-CMSA-1, which is statistically significantly better than the standard algorithm for both problems. As our experimental evaluation shows, this variant successfully learns to construct solutions in the construct step, learning to generate better solutions than the ones constructed by the greedy probabilistic method employed by CMSA for both problems.

We introduced RL-CMSA as a general framework, leaving room for alternative particular implementations of the q-value update and solution component sampling. While we proposed four different designs, we believe that an avenue for future work could consist of devising even better designs, further improving the performance obtained by our proposal. It would also be interesting to compare both standard CMSA and RL-CMSA variants for other combinatorial optimization problems, with the goal of obtaining further confirmation of the improvements brought by the RL component. Moreover, it would also be of great interest to explore if the proposed learning mechanism can harm performance for some problems. This might happen for so-called deceptive problems, for which the learning process might introduce a bias towards areas of the search space that do not contain the best solutions that can be found.

Finally, one limitation of the proposed learning mechanism is that the partial solution under construction is not taken into consideration for evaluating the goodness of a solution component. In general, the partial solution under construction plays a role in deciding whether a solution component is suitable for extension. Another path for future work could consist of extending the learning mechanism so that this contextual information is taken into account for deciding the quality of solution components. Doing this would change the employed RL process from being state-less to having a concept of state, which would be the partial solution under construction.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. The research presented in this paper was supported by grants TED2021-129319B-I00 and PID2022-136787NB-I00 funded by MCIN/AEI/10.13039/501100011033.

Declarations

Conflict of interest The authors declare no Conflict of interest

Ethical approval This article does not contain any studies with human participants performed by any of the authors.

References

Akbay, M. A., Kalayci, C. B., & Blum, C. (2022). Application of CMSA to the electric vehicle routing problem with time windows, simultaneous pickup and deliveries, and partial vehicle charging. In: Metaheuristics international conference (pp. 1–16). Springer.

Akbay, M. A., López Serrano, A., & Blum, C. (2022). A self-adaptive variant of CMSA: Application to the minimum positive influence dominating set problem. *International Journal of Computational Intelligence Systems, 15*(1), 44.

Alicastro, M., Ferone, D., Festa, P., Fugaro, S., & Pastore, T. (2021). A reinforcement learning iterated local search for makespan minimization in additive manufacturing machine scheduling problems. *Computers and Operations Research, 131*, 105272.

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*, 106520.

Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *Science, 286*(5439), 509–512.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940

Blum, C. (2024) Construct, merge, solve and adapt: A hybrid metaheuristic for combinatorial optimization. Springer (in press). https://doi.org/10.1007/978-3-031-60103-3

Blum, C., & Pinacho-Davidson, P. (2023). Application of negative learning ant colony optimization to the far from most string problem. In L. Pérez Cáceres & T. Stützle (Eds.), *Evolutionary computation in combinatorial optimization* (pp. 82–97). Springer.

Blum, C., Pinacho, P., López-Ibáñez, M., & Lozano, J. A. (2016). Construct, merge, solve and adapt a new general algorithm for combinatorial optimization. *Computers and Operations Research, 68*, 75–88.

Calvo, B., & Santafé Rodrigo, G. (2016). scmamp: Statistical comparison of multiple algorithms in multiple problems. The R Journal, 8(1). https://doi.org/10.32614/RJ-2016-017

Chaves, A. A., & Lorena, L. H. N. (2021). An adaptive and near parameter-free BRKGA using q-learning method. In: 2021 IEEE congress on evolutionary computation (CEC) (pp. 2331–2338). IEEE.

Erdös, P., & Rényi, A. (1959). On random graphs I. *Publicationes Mathematicae Debrecen, 6*(290–297), 18.

Ferrer, J., Chicano, F., & Ortega-Toro, J. A. (2021). CMSA algorithm for solving the prioritized pairwise test data generation problem in software product lines. *Journal of Heuristics, 27*, 229–249.

Gambardella, L. M., & Dorigo, M. (1995). Ant-q: A reinforcement learning approach to the traveling salesman problem. In: Machine learning proceedings 1995 (pp. 252–260). Elsevier.

García, S., Fernández, A., Luengo, J., & Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information sciences, 180*(10), 2044–2064.

Huber, M., & Raidl, G. R.(2021). Learning beam search: Utilizing machine learning to guide beam search for solving combinatorial optimization problems. In: International conference on machine learning, optimization, and data science (pp. 283–298). Springer.

Kalatzantonakis, P., Sifaleras, A., & Samaras, N. (2023). A reinforcement learning-variable neighborhood search method for the capacitated vehicle routing problem. *Expert Systems with Applications, 213*, 118812.

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

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

Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., & Min, S. (2020). Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems, 33*, 21188–21198.

Lanctot, J. K., Li, M., Ma, B., Wang, S., & Zhang, L. (2003). Distinguishing string selection problems. *Information and Computation, 185*(1), 41–55.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58.

Mousavi, S. R. (2010). A hybridization of constructive beam search with local search for far from most strings problem. *International Journal of Computer and Information Engineering, 4*(8), 1200–1208.

Pino, T., Choudhury, S., & Al-Turjman, F. (2018). Dominating set algorithms for wireless sensor networks survivability. *IEEE Access, 6*, 17527–17532.

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

Rosati, R. M., Kletzander, L., Blum, C., Musliu, N., & Schaerf, A. (2022). Construct, merge, solve and adapt applied to a bus driver scheduling problem with complex break constraints. In: International conference of the Italian association for artificial intelligence (pp. 254–267). Springer.

Rosati, R. M., Bouamama, S., & Blum, C. (2024). Multi-constructor CMSA for the maximum disjoint dominating sets problem. *Computers and Operations Research, 161*, 106450.

Shen, C., & Li, T. (2010). Multi-document summarization via the minimum dominating set. In: Proceedings of the 23rd international conference on computational linguistics (Coling 2010) (pp. 984–992).

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

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature, 393*(6684), 440–442.