The Hybrid Metaheuristic CMSA

Christian Blum and Jaume Reixach

Contents

Abstract

In this chapter, we describe the Construct, Merge, Solve & Adapt (CMSA) algorithm, a metaheuristic framework designed to address hard combinatorial optimization problems. CMSA combines the probabilistic construction of solutions within an adaptive, iterative process with solution merging and exact optimization techniques. The algorithm probabilistically constructs solutions through heuristic methods, which are then merged into a reduced subproblem. This subproblem is solved using exact optimization approaches, mostly integer programming. Based on the feedback from this solving phase, the algorithm adapts its construction and merging strategies in subsequent iterations, progres-

C. Blum (✉) · J. Reixach
Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, Spain
e-mail: christian.blum@iiia.csic.es; jaume.reixach@iiia.csic.es

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07153-4_79-1

sively finding solutions of improving quality over time. CMSA's hybrid nature allows it to balance between the speed of heuristic construction and the accuracy of exact methods, making it particularly effective for large-scale problems. After a general description of standard CMSA, we outline a recent self-adaptive variant and a variant that uses reinforcement learning to improve the search process. All three algorithm variants are applied to the classical maximum independent set problem. Moreover, an experimental evaluation is provided to show their comparative behavior.

Keywords

Hybrid metaheuristic · Combinatorial optimization · Maximum independent set

Introduction

The "Construct, Merge, Solve & Adapt" (CMSA) algorithm [9, 15] is a metaheuristic framework designed to tackle combinatorial optimization problems, especially aiming for problem instances that are hard to solve exactly. It combines constructive heuristics, solution merging, and problem-solving techniques in an iterative process, adapting the search process based on the feedback from previous iterations. Each algorithm iteration consists of the following four steps, also shown in the general CMSA framework depicted in Fig 1. In the first step, solutions are constructed probabilistically using a heuristic approach. These solutions do not necessarily need to be of high quality. Instead, their components are expected to serve as building blocks of high-quality solutions. The heuristic approach may involve greedy algorithms, randomized methods, or other simple techniques that generate solutions quickly. After constructing solutions, the algorithm proceeds to merge their components to generate a smaller subproblem, that is, a sub-instance of the original problem instance. In other words, the algorithm employs a bottom-

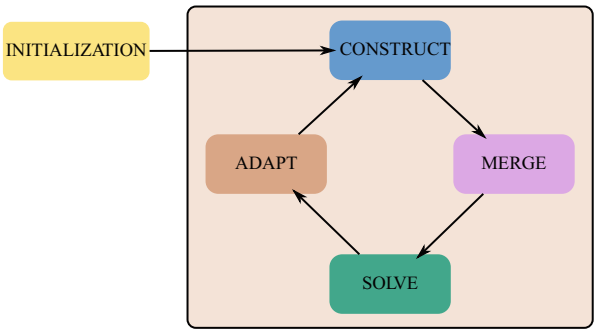

Fig. 1 General framework of CMSA

up way of generating a sub-instance by merging components originating from different solutions. The incumbent sub-instance is solved in the third step using an appropriate solver. If the combinatorial optimization problem under consideration can be expressed as an integer linear program (ILP), for example, any ILP solver may be used for this purpose. The principal idea is that the sub-instance is more manageable, making it possible to solve it much more quickly than the original problem instance. Finally, the sub-instance is adapted based on the solution to the incumbent sub-instance returned by the solver. CMSA repeats this cycle, iterating between constructing, merging, solving, and adapting, progressively improving the quality of the obtained solutions. Over time, it generally converges toward a high-quality or near-optimal solution.

The key features of CMSA are as follows. First, as a hybrid approach, it combines the strengths of constructive heuristics (speed) and exact methods (precision). CMSA is particularly suited for large-scale, complex combinatorial optimization problems where exact methods may be too slow or even inviable. It can also be seen as a methodology for using exact techniques in the context of problem instances that are too large for their standalone application.

CMSA Variants

The first and original CMSA variant [15] works as follows. First, the complete set of solution components (C) must be defined. For example, when considering the traveling salesman problem (TSP), each edge between a pair of cities may be considered a solution component. For a second example, consider the one-dimensional bin packing problem where, intuitively, each item to be packed is a solution component. For providing a problem-independent description of CMSA, we henceforth consider a generic set $C = \{c_1, \ldots, c_n\}$ of solution components. We assume that any valid solution $S \in \mathcal{S}$, where $\mathcal{S}$ is the set of all feasible solutions, can be expressed as a subset of C, that is, $S \subseteq C$. Finally, let $f : \mathcal{S} \mapsto \mathbb{N}^+$ denote the objective function we aim to minimize.

Algorithm 1 contains the pseudo-code for this first and original CMSA variant. It begins by initializing the sub-instance (C') to the empty set. Moreover, the age values of the solution components are set to zero. The best-so-far solution S^{bsf} remains uninitialized. Each iteration of CMSA consists of the following four actions.

1. The first step of each iteration consists in the probabilistic construction of n_{a} (generally valid) solutions to the given problem. This is done by calling n_{a} times function ConstructSolutionProbabilistically(C, d_{rate}) in line 6 of Algorithm 1. Note that this action is problem-dependent. A randomized greedy heuristic, whose degree of randomness is determined by parameter d_{rate}, is typically employed for solution construction tailored to the considered optimization problem.

Algorithm 1: Standard CMSA

1: **input 1:** Complete set of solution components (C)
2: **input 2:** Values for n_a, d_{rate}, age_{max}, t_{SOLVER}
3: **initialization:** $C' := \emptyset$, $age[c] := 0\ \forall c \in C$, S^{bsf} is not initialized
4: **while** CPU time limit not reached **do**
5: **for** $i := 1, \ldots, n_a$ **do**
6: $S :=$ ConstructSolutionProbabilistically(C, d_{rate})
7: **if** S^{bsf} not initialized **or** $f(S) < f(S^{bsf})$ **then** $S^{bsf} := S$ **endif**
8: **for** all $c \in S$ **and** $c \notin C'$ **do**
9: $age[c] := 0$
10: $C' := C' \cup \{c\}$
11: **end for**
12: **end for**
13: $S^{SOLVER} :=$ SolveSubinstance(C', t_{SOLVER})
14: **if** $f(S^{SOLVER}) < f(S^{bsf})$ **then** $S^{bsf} := S^{SOLVER}$ **end if**
15: AdaptSubinstance(C', S^{SOLVER}, age_{max})
16: **end while**
17: **output:** S^{bsf}

2. Next, the solution components found in these n_a solutions are added to the incumbent sub-instance C'. Their age value is set to zero if they are not already present in C'. This is shown in lines 8–11 of Algorithm 1.
3. Third, an appropriate solver is applied to the updated sub-instance C' in function SolveSubinstance(C', t_{SOLVER}) (see line 13) to derive the best solution possible within the given time limit of t_{SOLVER} CPU seconds. There are no limitations concerning the nature of this solver. It may be a problem-specific exact or heuristic solver, or it may be, for example, an ILP solver. Note that the feasible search space of C' consists of those solutions to the original (complete) problem instance that only contains components present in C'. Henceforth, the solution returned by the employed solver is called S^{SOLVER}.
4. Finally, the incumbent sub-instance C' is adapted on the basis of solution S^{SOLVER} in function Adapt(C', S^{SOLVER}, age_{max}) at line 15. This is a problem-independent step that works as follows. After incrementing the age values of all solution components in $C' \setminus S^{SOLVER}$ by one, the age values of all components present in S^{SOLVER} are set to zero. In addition, to alleviate the work of the solver, all those solution components in C' whose age value has reached the user-determined upper limit of age_{max} are removed from C'. In this context, note that the age value of a solution component, at any given time, indicates the number of consecutive CMSA iterations in which this component has been part of the sub-instance but has not been included in the solver's solution to that sub-instance.

Note that CMSA is iterated until a predefined CPU time limit is reached. Upon termination, the algorithm returns the best-so-far solution S^{bsf} as output. The generic CMSA parameters include the number of solutions generated per iteration (n_a), the maximum allowable age of solution components (age_{max}), and the time limit for the

utilized solver at each iteration (t_{SOLVER}). However, additional parameters related, for example, to the sub-instance solver may be used.

Self-Adaptive CMSA

The standard CMSA algorithm was shown to be a well-working algorithm across various combinatorial optimization problems; however, certain applications have revealed its sensitivity to parameter settings. To address this issue, a self-adaptive variant of CMSA, named Adapt-CMSA, was proposed in [2]. A key feature of Adapt-CMSA is that it does not require intensive parameter tuning across subsets of the problem instances considered.

Adapt-CMSA (see Algorithm 2) differs from the standard CMSA primarily in how it manages the size of the sub-instance. In the standard CMSA, sub-instance size is controlled using age values in combination with a maximum age limit for solution components. In contrast, Adapt-CMSA does not rely on age values. Instead, after each iteration, all solution components, except those that are part of the best-so-far solution S^{bsf}, are removed from the sub-instance C' (see line 23).

Algorithm 2: Self-adaptive CMSA (Adapt-CMSA)

1: **input 1:** Complete set of solution components (C)
2: **input 2:** Values for Adapt-CMSA parameters t_{prop}, t_{SOLVER}
3: **input 3:** Values for solution construction parameters α^{LB}, α^{UB}, α_{red}
4: **initialization:** $n_{\text{a}} := 1$; $\alpha_{\text{bsf}} := \alpha^{\text{UB}}$; $C' := S^{\text{bsf}}$
5: $S^{\text{bsf}} :=$ GenerateGreedySolution(C)
6: **while** CPU time limit not reached **do**
7: **for** $i := 1, \ldots, n_{\text{a}}$ **do**
8: $S :=$ ConstructSolutionProbabilistically(C, S^{bsf}, α_{bsf})
9: $C' := C' \cup S$
10: **end for**
11: $(S^{\text{SOLVER}}, t_{\text{req}}) :=$ SolveSubinstance(C', t_{SOLVER}) ▷ This function returns two objects: (1) the obtained solution (S^{SOLVER}), (2) the required computation time (t_{req})
12: **if** $t_{\text{req}} < t_{\text{prop}} \cdot t_{\text{SOLVER}}$ and $\alpha_{\text{bsf}} > \alpha^{\text{LB}}$ **then** $\alpha_{\text{bsf}} := \alpha_{\text{bsf}} - \alpha_{\text{red}}$ **end if**
13: **if** $f(S^{\text{SOLVER}}) < f(S^{\text{bsf}})$ **then**
14: $S^{\text{bsf}} := S^{\text{SOLVER}}$
15: $n_{\text{a}} := 1$
16: **else**
17: **if** $f(S^{\text{SOLVER}}) > f(S^{\text{bsf}})$ **then**
18: **if** $n_a = 1$ **then** $\alpha_{\text{bsf}} := \min\{\alpha_{\text{bsf}} + \frac{\alpha_{\text{red}}}{10}, \alpha^{\text{UB}}\}$ **else** $n_a = 1$ **end if**
19: **else**
20: $n_{\text{a}} := n_{\text{a}} + 1$
21: **end if**
22: **end if**
23: $C' := S^{\text{bsf}}$ ▷ Only the solution components of S^{bsf} remain in C'
24: **end while**
25: **output:** S^{bsf}

Sub-instance size in Adapt-CMSA is partially controlled during the probabilistic construction of solutions at each iteration (see function ConstructSolutionProbabilistically(C, S^{bsf}, α_{bsf}) in line 8). This function receives as input not only the set of all possible solution components (C) but also the current best-so-far solution S^{bsf} and a parameter α_{bsf} (where $0 \leq \alpha_{\text{bsf}} < 1$). The parameter α_{bsf} influences the construction of new solutions by biasing them toward the best-so-far solution S^{bsf}; specifically, higher values of α_{bsf} lead to greater similarity between the generated solutions and S^{bsf}.

The self-adaptive nature of Adapt-CMSA is shown in the dynamic adjustment of α_{bsf} over time and depending on the search progress. In particular, the value of α_{bsf} can move between a lower bound α^{LB} and an upper bound α^{UB}, both algorithm parameters. At the beginning of the algorithm, α_{bsf} is initialized to its maximum α^{UB} (see line 4). In other words, at the start of the algorithm, the generated solutions will be most similar to the best-so-far solution S^{bsf}. If the incumbent sub-instance is solved within a computation time t_{req} that is less than a fraction t_{prop} of the maximum allowed computation time t_{SOLVER}, α_{bsf} is reduced by α_{red} (see line 12), where α_{red} is another algorithm parameter. The reasoning behind this adjustment is as follows: When the current sub-instance is solved easily, it suggests a relatively small search space. To expand the search space, the solutions generated in ConstructSolutionProbabilistically(C, S^{bsf}, α_{bsf}) should be more diverse relative to S^{bsf}. This is accomplished by reducing the α_{bsf} value.

Apart from α_{bsf}, the number of solution constructions per iteration (n_a) is also self-adaptive; see lines 13–22. At the start of Adapt-CMSA, n_a is set to 1 (see line 4). If the solution to the sub-instance provided by the solver (S^{SOLVER}) improves over the best-so-far solution S^{bsf}, n_a is reset to one (see line 15). Conversely, if S^{SOLVER} is strictly inferior to S^{bsf}, the corresponding sub-instance was too large (resp., difficult) for the solver to be solved within t_{SOLVER} seconds. If this case coincides with a setting of $n_a = 1$, the α_{bsf} value is marginally increased (by $\frac{\alpha_{\text{red}}}{10}$); otherwise, n_a is reset to one. In the event that $f(S^{\text{SOLVER}}) = f(S^{\text{bsf}})$, n_a is incremented by one (see line 20). This is done because solving the sub-instance failed to produce a solution better than S^{bsf}, even though the sub-instance was still solved to optimality within the allowed computation time of t_{SOLVER} seconds, suggesting that the size of the sub-instance should be increased.

CMSA Extended with Reinforcement Learning

Another potential limitation of standard CMSA is that the probabilistic construction of solutions in each iteration relies on a static, unchanging probability distribution. In other words, the probability of generating any particular valid solution remains constant throughout the algorithm's execution. In principle, this can be changed by adding a learning component to the solution construction phase. The first ones to do so were the authors of [28]. They intertwined CMSA iterations with the running of a population-based metaheuristic. At each iteration of their Learn-CMSA approach,

the sub-instance of Learn-CMSA receives input from the incumbent population of the metaheuristics and vice versa. A somewhat more straightforward approach was recently presented in [29], where the authors present a CMSA variant in which the construction of solutions is based on a simple reinforcement learning (RL) mechanism, with rewards determined on the basis of the solution returned by the solver at each iteration. This CMSA variant is called RL-CMSA and is presented in Algorithm 3.

RL-CMSA maintains a set $Q = \{q_1, \ldots, q_n\}$ of quality values which contains a q_i for each solution component $c_i \in C$. One of the main characteristics of RL-CMSA is that solutions are constructed by sampling solution components depending on these values. At the start of the algorithm, all $q_i \in Q$ are initialized to zero. Moreover, these values change during the algorithm due to cumulating rewards received after the adapt step depending on the solution S^{SOLVER} produced by the solver. In particular, those solution components from C' that form part of S^{SOLVER} receive a reward of one, while those from C' that the solver did not select receive a reward of minus one. In this way, the solver acts as a feedback mechanism for the solution construction process, and the values in Q keep track of the "quality" of the different solution components. The process of determining the rewards and adding them to the quality values is done in UpdateQ(q, C', S^{SOLVER}); see line 17.

Solutions are constructed in function ConstructSolutionProbabilistically(C, q, d_{rate}, β) in line 7 exclusively based on the quality values; that is, no heuristic

Algorithm 3: Reinforcement learning CMSA (RL-CMSA)

1: **input 1:** Complete set of solution components (C)
2: **input 2:** Values for RL-CMSA parameters n_{a}, $age_{\max}$, t_{SOLVER}, cf_{limit}, b_{reset}
3: **input 3:** Values for solution construction parameters d_{rate}, β
4: **initialization:** $C' := \emptyset$, $age[c] := 0$ and $q[c] := 0\ \forall c \in C$, S^{bsf} is not initialized
5: **while** CPU time limit not reached **do**
6: **for** $i := 1, \ldots, n_{\text{a}}$ **do**
7: $S :=$ ConstructSolutionProbabilistically($C, q, d_{\text{rate}}, \beta$)
8: **if** S^{bsf} not initialized **or** $f(S) < f(S^{\text{bsf}})$ **then** $S^{\text{bsf}} := S$ **endif**
9: **for** all $c \in S$ **and** $c \notin C'$ **do**
10: $age[c] := 0$
11: $C' := C' \cup \{c\}$
12: **end for**
13: **end for**
14: $S^{\text{SOLVER}} :=$ SolveSubinstance(C', t_{SOLVER})
15: **if** $f(S^{\text{SOLVER}}) < f(S^{\text{bsf}})$ **then** $S^{\text{bsf}} := S^{\text{SOLVER}}$ **end if**
16: AdaptSubinstance($C', S^{\text{SOLVER}}, age_{\max}$)
17: UpdateQ(q, C', S^{SOLVER})
18: $cf :=$ ComputeConvergenceFactor(q)
19: **if** $cf > cf_{\text{limit}}$ **then**
20: $q[c] := 0\ \forall c \in C$
21: **if** $b_{\text{reset}} = true$ **then** $C' := \emptyset$ **end if**
22: **end if**
23: **end while**
24: **output:** S^{bsf}

information is used to bias this solution construction. The use of parameters d_{rate} and β will be outlined later. The only problem-dependent part is to be found in determining those solution components that can be selected at each solution construction step.

As the reader might notice, if some quality values become much larger than the rest, the solution construction procedure could lead to highly similar solutions being constructed repeatedly. To avoid this, RL-CMSA implements a reset mechanism. This consists of computing a value cf denoted as the convergence factor, which gives a measure of the level of convergence, as follows. For every solution component c_i of the last solution constructed S, the probability z_i of selecting c_i over all the solution components not forming part of S is calculated. The convergence factor is then defined as $cf = \min_{c_i \in S} z_i$. Once the convergence factor is calculated, the algorithm checks if it is larger than a parameter $cf_{\text{limit}} \in [0, 1]$ known as the convergence factor limit. If this is the case, the algorithm is re-initialized, which consists of setting all quality values back to zero and emptying the sub-instance C' depending on a Boolean parameter b_{reset} (see lines 19-22). This last option allows completely erasing the information learned so far (if $b_{\text{reset}} = true$) or keeping some otherwise, in the form of the sub-instance C'.

Timeline of CMSA Applications

Table 1 provides a timeline of all applications of different CMSA variants found in the related literature to date. The first table column contains the name of the considered combinatorial optimization problem; the second column references the corresponding publication; the third column indicates the year of publication; and the last column provides information about the utilized (or introduced) CMSA variant. For easier visual differentiation between CMSA variants, colored dots are used in the last column, with a gray dot signifying a CMSA approach that does not align with any established CMSA variant. Any such exceptions are explained in the table.

Application of CMSA to Maximum Independent Set

In the final part of this chapter, we demonstrate the application of the three CMSA variants described before (standard CMSA, Adapt-CMSA, and RL-CMSA) to the NP-hard maximum independent set (MIS) problem. The MIS problem is a well-known combinatorial optimization problem in graph theory, which can be described as follows. Given is an undirected graph $G = (V, E)$. The set $N(v_i) \subset V$ denotes the set of neighbors of v_i in graph G, that is, for each $v_j \in N(v_i)$, there exists an edge $e = (v_i, v_j) \in E$. Moreover, $N[v_i]$ is the so-called closed neighborhood of v_i in G defined as follows: $N[v_i] := N(v_i) \cup \{v_i\}$. A set $S \subset V$ is a feasible solution to the MIS problem if for any pair $v_i \neq v_j \in S$ the edge set E of graph G does

Table 1 Applications of CMSA until October 2024 (in alphabetical order)

Optimization problem	Publications	Year (first)	CMSA variant
Minimum covering arborescence	[15]	2016	Standard CMSA
Minimum common string partition	[8, 13, 15]	2016	Standard CMSA
Multi-dimensional knapsack	[13, 14, 24]	2016	Standard CMSA
Repetition-free longest common subsequence	[10, 11]	2016	Standard CMSA
Binary optimization	[12]	2019	Problem specific
Project scheduling	[35]	2019	Standard CMSA with ACO for solution construction
Minimum capacitated dominating set	[27]	2019	Standard CMSA
Routing of air-ground robots	[6]	2019	Standard CMSA
Maximum happy vertices problem	[21, 23, 34]	2019	Standard CMSA
Score-constrained packing	[22]	2020	Standard CMSA
Maintenance of nuclear power plants	[18]	2021	Standard CMSA
Test data generation in software product lines	[20]	2021	Standard CMSA
Minimum positive influence dominating set	[2]	2022	Adapt-CMSA
Bus driver scheduling	[30]	2022	Standard CMSA
Electric vehicle routing with time windows, simultaneous pickup and deliveries, and partial vehicle charging	[1]	2022	Adapt-CMSA
Unit disk cover	[5]	2022	Standard CMSA
Closest string problem	[16]	2023	Standard CMSA
Two-echelon electric vehicle routing with simultaneous pickup and deliveries	[3]	2023	Adapt-CMSA
Maximum disjoint dominating set	[31, 32]	2023	Standard CMSA with multiple solution constructors
Multi-way multi-dimensional number partitioning	[17]	2023	Adapt-CMSA
Rooted max tree coverage	[37]	2023	Standard CMSA
Diversity and equity optimization	[26]	2024	Standard CMSA
Minimum dominating set and far from most string problems	[29]	2024	RL-CMSA
Electric vehicle routing with time windows and simultaneous pickups and deliveries	[4]	2024	Standard CMSA
Variable-sized bin packing	[4]	2024	Adapt-CMSA
Far from most string problem	[28]	2024	Learn-CMSA

not contain the edge $e = (v_i, v_j)$. In other words, S is a feasible solution if no two nodes from S are neighbors in G. The optimization goal is to find a feasible solution that is as large as possible.

A typical ILP model for the MIS problem uses a binary variable x_i for each node $v_i \in V$. The formulation of the model is as follows:

$$\max \quad \sum_{v_i \in V} x_i \tag{1}$$

$$\text{subject to} \quad x_i + x_j \leq 1 \qquad \forall\, e = (v_i, v_j) \in V \tag{2}$$

$$x_i \in \{0, 1\} \qquad \forall\, v_i \in V$$

Constraints (2) make sure that any solution contains at most one endpoint of an edge in E.

Definition of the Solution Components

A key step in designing any CMSA algorithm is defining the set of solution components (C). For the MIS problem, we adopt a straightforward approach: The set C includes a component c_i for each node $v_i \in V$ of the input graph $G = (V, E)$. Therefore, it is important to keep in mind that, throughout the following discussion, both v_i and c_i refer to the same node v_i of the input graph G.

Probabilistic Construction of MIS Solutions

The process for constructing a valid MIS solution in standard CMSA is described in Algorithm 4. It begins with an empty solution, $V_{\text{sol}} = \emptyset$. At each step of the construction, $\tilde{V} \subseteq V$ is defined as the set of nodes that do not have a single neighbor included in V_{sol}. Exactly one node is selected from $\tilde{V}$ at each step using the function ChooseFrom($\tilde{V}$, d_{rate}). As mentioned before, the value of d_{rate} determines the randomness of the solution construction process. The selection process for a node is outlined as follows.

Algorithm 4: Function ConstructSolutionProbabilistically(C, d_{rate}) of standard CMSA

1: **input:** a graph $G = (V, E)$
2: $V_{\text{sol}} := \emptyset$
3: $\tilde{V} := V \setminus \bigcup_{v_i \in V_{\text{sol}}} N[v_i]$
4: **while** $\tilde{V} \neq \emptyset$ **do**
5: $\quad v_j := \text{ChooseFrom}(\tilde{V}, d_{\text{rate}})$
6: $\quad V_{\text{sol}} := V_{\text{sol}} \cup \{v_j\}$
7: $\quad \tilde{V} := V \setminus \bigcup_{v_i \in V_{\text{sol}}} N[v_i]$
8: **end while**
9: **output:** a CMSA-solution S, corresponding to solution V_{sol}

Function ChooseFrom($\tilde{V}, d_{\text{rate}}$): A node $v_j \in \tilde{V}$ is chosen according to the greedy function $\mu(v_j) := |N(v_j)|$, which evaluates the degree of v_j. In this selection process, nodes with lower degrees are preferred. To choose a node v_j from $\tilde{V}$, a random number r is drawn uniformly from [0, 1]. If $r \leq d_{\text{rate}}$, node $v_j \in \tilde{V}$ is chosen such that

$$v_j := \text{argmin}\{\mu(v_i) \mid v_i \in \tilde{V}\} \ . \tag{3}$$

Alternatively, when $r > d_{\text{rate}}$, a set of $\min\{l_{\text{size}}, |\tilde{V}|\}$ nodes from $\tilde{V}$ is selected and added to a candidate set $L \subseteq \tilde{V}$ such that

$$\mu(v_i) \leq \mu(v_k) \quad \text{for all } v_i \in L, v_k \in \tilde{V} \setminus L \ . \tag{4}$$

A vertex $v_j \in L$ is then selected uniformly at random. It is important to highlight that once the construction of a solution V_{sol} is complete, it is converted into a corresponding CMSA-solution S. This conversion entails replacing each node $v_i \in V_{\text{sol}}$ with its corresponding solution component c_i. The obtained CMSA-solution S is then returned by Algorithm 4.

In Adapt-CMSA, solutions are constructed according to the same principle. Function GenerateGreedySolution(C) of line 4 of Algorithm 2, for example, is obtained from Algorithm 4 described above by using a setting of $d_{\text{rate}} = 1.0$. In other words, this function generates a solution deterministically. Next, ConstructSolutionProbabilistically(C, S^{bsf}, α_{bsf}) in line 8 of Algorithm 2 executes the solution construction from Algorithm 4 with the following change in function ChooseFrom($\tilde{V}, d_{\text{rate}}$). For being able to select a node from $\tilde{V}$, a probability $\mathbf{p}(v_k)$ is calculated for all $v_k \in \tilde{V}$ in the following way. For this purpose, first, a value $h(v_k)$ is determined for each vertex $v_k \in \tilde{V}$:

$$h(v_k) := \begin{cases} \frac{\alpha_{\text{bsf}}}{\mu(v_k)} & \text{if } c_k \in S^{\text{bsf}} \\ \frac{(1-\alpha_{\text{bsf}})}{\mu(v_k)} & \text{otherwise} \end{cases} \tag{5}$$

Second, the probabilities are defined as follows:

$$\mathbf{p}(v_k) := \frac{h(v_k)}{\sum_{v_l \in \tilde{V}} h(v_l)} \quad \forall\, v_k \in \tilde{V} \tag{6}$$

Subsequently, a node v_j is selected from $\tilde{V}$ based on these probabilities. Recall that the bias toward the best-so-far solution S^{bsf} is controlled by the parameter $\alpha_{\text{bsf}} \in [0, 1]$, where a higher α_{bsf} value increases this bias. Notably, this bias does not exist in standard CMSA.

Finally, in RL-CMSA, solutions are also constructed as shown in Algorithm 4. However, the choice of a node at each construction step (as done by function ChooseFrom($\tilde{V}, d_{\text{rate}}$)) is performed solely based on the quality values in Q (see the

description of RL-CMSA), in the following way. First, with a probability d_{rate}, this selection is done randomly among those nodes (resp., solution components) from $\tilde{V}$ with the highest quality values; otherwise, this is done probabilistically based on the following probabilities p_i for each $v_i \in \tilde{V}$:

$$p_i = \frac{e^{\beta q_i}}{\sum_{v_k \in \tilde{V}} e^{\beta q_k}} \quad . \tag{7}$$

Hereby, $d_{\text{rate}} \in [0, 1]$ and $\beta \geq 0$ are parameters of the solution construction procedure, which balance between the exploration and exploitation of solution components.

Sub-instance Solving

Sub-instances C' are solved in the same way in all three CMSA variants considered in this chapter. To do so, the MIS ILP model is extended by adding the following set of constraints:

$$x_i = 0 \quad \text{for all } c_i \in C \setminus C' \tag{8}$$

Thus, only solution components (or nodes) that are part of the sub-instance C' can be chosen to appear in solutions. Note that the corresponding functions for sub-instance solving in all three CMSA variants return the ILP solver solution in terms of a set S^{SOLVER} of solution components. Moreover, the application of the ILP solver is subject to a time limit of t_{SOLVER} CPU seconds, which means that the returned solution (S^{SOLVER}) is not necessarily an optimal solution to sub-instance C'.

Experimental Evaluation

The experimental evaluation presented in this section includes the following algorithms:

1. Greedy: A greedy heuristic derived by applying the heuristic from Algorithm 4 in a deterministic manner.
2. CPLEX: The application of CPLEX 22.1 to each problem instance using the default parameter settings of CPLEX.
3. CMSA: The application of standard CMSA
4. Adapt-CMSA: The application of the self-adaptive CMSA variant described before.
5. RL-CMSA: The application of the CMSA variant supported by reinforcement learning.

CPLEX 22.1 is executed sequentially (one thread) in standalone mode and within the CMSA variants. For conducting the experiments, we used the IIIA-CSIC in-house high-performance computing cluster of machines equipped with Intel® Xeon® 5670 CPUs having 12 cores of 2.933 GHz and at least 32 GB of RAM.

Benchmark Set

To ensure a controlled experimentation environment, we chose to create a benchmark set featuring graphs of different sizes and densities, utilizing the following three network models:

1. **The Erdös-Rényi model [19]**. Named after mathematicians Paul Erdös and Alfréd Rényi, this model is one of the earliest and simplest models for generating random graphs. Introduced in 1959, this model is defined by two parameters: the number of nodes (n) and the probability (p) that an edge exists between any pair of nodes.
2. **The Watts-Strogatz model [36]**. This model generates small-world networks, that is, networks that feature a small average shortest path length between vertices while maintaining a high degree of local clustering. This model was introduced by Duncan J. Watts and Steven Strogatz in 1998.
3. **The Barabási-Albert model [7]**. Developed by Albert-László Barabási and Réka Albert in 1999, the main characteristic of this preferential attachment model is its ability to produce networks with a scale-free degree distribution. In such networks, the degrees of nodes follow a power-law distribution, where a few nodes have a very high degree while most nodes have much lower degrees.

Thirty graphs were generated using each of the 3 models for each combination of $|V| \in \{500, 1000, 1500, 2000\}$ and 4 distinct graph densities. In Erdős-Rényi graphs, density is controlled by the edge probability (p), in Watts-Strogatz graphs by the parameter (k) and in Barabási-Albert graphs by the parameter (m). Altogether, this benchmark set includes 480 graphs for each network model. We used the `igraph` library's implementations of these graph models.

ILP Solver Parameters

In addition to the CMSA-specific parameters of the three CMSA variants that will be applied to the MIS problem in this chapter, the following parameters related to the utilized ILP solver are used. Note that these parameters are explained in the context of CPLEX version 22.1 which was used for the implementations.

1. $\text{ilp}_{\text{emphasis}} \in \{\texttt{true}, \texttt{false}\}$ is a Boolean parameter indicating whether or not to utilize "heuristic emphasis." Hereby, $\text{ilp}_{\text{emphasis}} = \texttt{false}$ denotes CPLEX's default setting, while $\text{ilp}_{\text{emphasis}} = \texttt{true}$ refers to setting the emphasis parameter to a value of five, indicating the highest heuristic emphasis level in which the solver strongly favors the improvement of its best solution found over bound calculations.

2. $\text{ilp}_{\text{warmstart}} \in \{\texttt{true}, \texttt{false}\}$ is another Boolean parameter indicating whether or not to provide the solver with the best-so-far solution S^{bsf} of CMSA at the start of each application. This is called a "warm start" in the context of CPLEX.
3. $\text{ilp}_{\text{abort}} \in \{\texttt{true}, \texttt{false}\}$ is a third Boolean parameter indicating whether or not to abort CPLEX whenever a solution is found that is better than the current best-so-far solution S^{bsf} of CMSA.

Note that all three CPLEX-related parameters are used in standard CMSA and in RL-CMSA. However, using $\text{ilp}_{\text{warmstart}}$ and $\text{ilp}_{\text{abort}}$ would not make sense in Adapt-CMSA, as in this CMSA variant, it is important to know if a sub-instance can be solved to optimality within the allotted computation time.

Parameter Tuning

All three CMSA variants underwent parameter tuning with the `irace` tool [25]. Specifically, each CMSA variant was tuned once for the complete benchmark set, with an `irace` budget of 3000 algorithm runs. Additional problem instances were created to serve as tuning instances. Specifically, for each combination of $|V|$ and graph density, exactly one tuning instance was generated by each of the 3 network models, resulting in a total of 24 tuning instances. The computation time limit was set to 150 CPU seconds for all graphs with $|V| = 500$, 300 CPU seconds for graphs with $|V| = 1000$, 450 CPU seconds for graphs with $|V| = 1500$, and 600 CPU seconds for graphs with $|V| = 2000$.

The tuning results of the three CMSA variants are shown in Table 2. An interesting observation in examining these parameter settings is that standard CMSA performs best with the lowest possible determinism rate ($d_{\text{rate}} = 0.0$)

Table 2 Parameters, domains, and tuning results. "n.a." is the abbreviation for "not applicable"

Parameter	Domain	CMSA	Adapt-CMSA	RL-CMSA
n_{a}	$\{1, \ldots, 50\}$	17	n.a.	15
age_{max}	$\{1, \ldots, 10\}$	1	n.a.	4
d_{rate}	$[0.0, 0.99]$	0.0	n.a.	0.43
l_{size}	$\{3, \ldots, 100\}$	81	n.a.	n.a.
t_{SOLVER}	$\{1, \ldots, 20\}$	9	20	20
$\text{ilp}_{\text{emphasis}}$	{`true`, `false`}	`false`	`false`	`true`
$\text{ilp}_{\text{warmstart}}$	{`true`, `false`}	`false`	n.a.	`false`
$\text{ilp}_{\text{abort}}$	{`true`, `false`}	`false`	n.a.	`true`
α^{LB}	$[0.6, 0.99]$	n.a.	0.98	n.a.
α^{UB}	$[0.6, 0.99]$	n.a.	0.99	n.a.
α_{red}	$[0.01, 0.1]$	n.a.	0.04	n.a.
t_{prop}	$[0.1, 0.8]$	n.a.	0.54	n.a.
β	$[0.0, 2.0]$	n.a.	n.a.	0.16
b_{reset}	{`true`, `false`}	n.a.	n.a.	`false`
cf_{limit}	$[0.9, 1.0]$	n.a.	n.a.	0.9

for probabilistic solution construction. This sharply contrasts with the settings $\alpha^{\text{LB}} = 0.98$ and $\alpha^{\text{UB}} = 0.99$ in Adapt-CMSA, which cause newly constructed solutions to be heavily biased toward the best-so-far solution S^{bsf}. In contrast to the requirements of CMSA and Adapt-CMSA, RL-CMSA works best with an intermediate setting of $d_{\text{rate}} = 0.43$ concerning the determinism in constructing solutions. Another interesting observation concerns the setting of the ILP solver parameters. Whereas CMSA and Adapt-CMSA do not utilize any of the CPLEX features (such as heuristic emphasis, warm starts, or early termination upon improvement), RL-CMSA appears to gain an advantage from applying heuristic emphasis and the abort feature.

Results

The five algorithmic techniques (Greedy, CPLEX, CMSA, Adapt-CMSA, and RL-CMSA) were applied once to each problem instance from the benchmark set. The computation time limit for CPLEX and the three CMSA variants were the same as that used for tuning (described earlier). The results are presented as box plots in Figs 2, 3 and 4. Note that each network model has exactly one graphic (Erdös-Rényi, Watts-Strogatz, and Barabási-Albert). Each graphic features a 4×4 grid of box plots, where the rows show results (from top to bottom) for graphs of increasing size, and the columns (from left to right) show results for graphs of increasing density.

To facilitate the analysis of the results concerning statistical significance, critical difference (CD) plots are provided separately for the graphs of each of the three network models in Figs 5, 6 and 7. In short, a CD plot displays the average ranking of several algorithms across a set of problem instances, with the whiskers of two algorithms connected by a bold horizontal bar if their results are statistically equivalent. Each CD plot figure consists of five graphics. The first graphic (at the top) presents statistical information for the entire set of graphs corresponding to the respective network model. The remaining four graphics display statistical information for all graphs of a specific density.

When studying the results, the following main observations can be made:

1. First of all, the three CMSA variants clearly outperform Greedy, which is—as expected—for all network models the worst-performing technique.
2. The three CMSA variants also outperform CPLEX with statistical significance. This is with the exception of rather sparse graphs and, in general, Barabási-Albert graphs where CPLEX performs rather strongly. In fact, CPLEX outperforms standard CMSA with statistical significance on Barabási-Albert graphs (see Fig 7a). Remember that these graphs are scale-free; that is, their node degrees follow a power-law distribution. This means that some hubs are strongly connected to other nodes, while many others have only very few connections.
3. When comparing the three CMSA variants, it is clear that RL-CMSA is the best-performing approach across the graphs of all three network models (different sizes and densities). As shown by the CD plots in Figs 5, 6 and 7, there is only one subset of graphs (the densest Erdös-Rényi graphs) for which RL-CMSA is not the

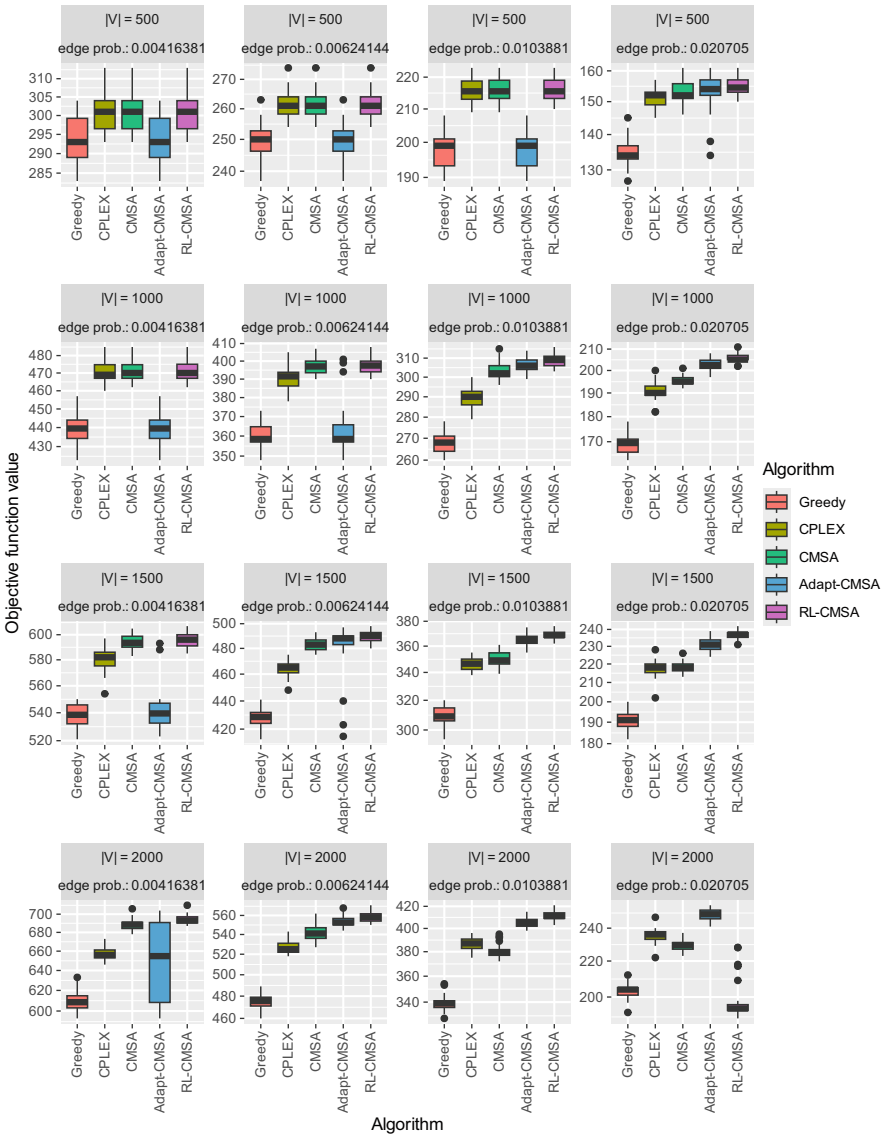

Fig. 2 Results for Erdös-Rényi graphs

best-performing method. This is because, for some reason, RL-CMSA does not work well for the largest and densest Erdös-Rényi graphs (see the bottom-right box plot in Fig 2.

4. Another interesting aspect is that Adapt-CMSA outperforms standard CMSA in the context of Watts-Strogatz and Barabási-Albert graphs. This is with statistical significance as shown in Figs 6 and 7. However, in the case of Erdös-Rényi

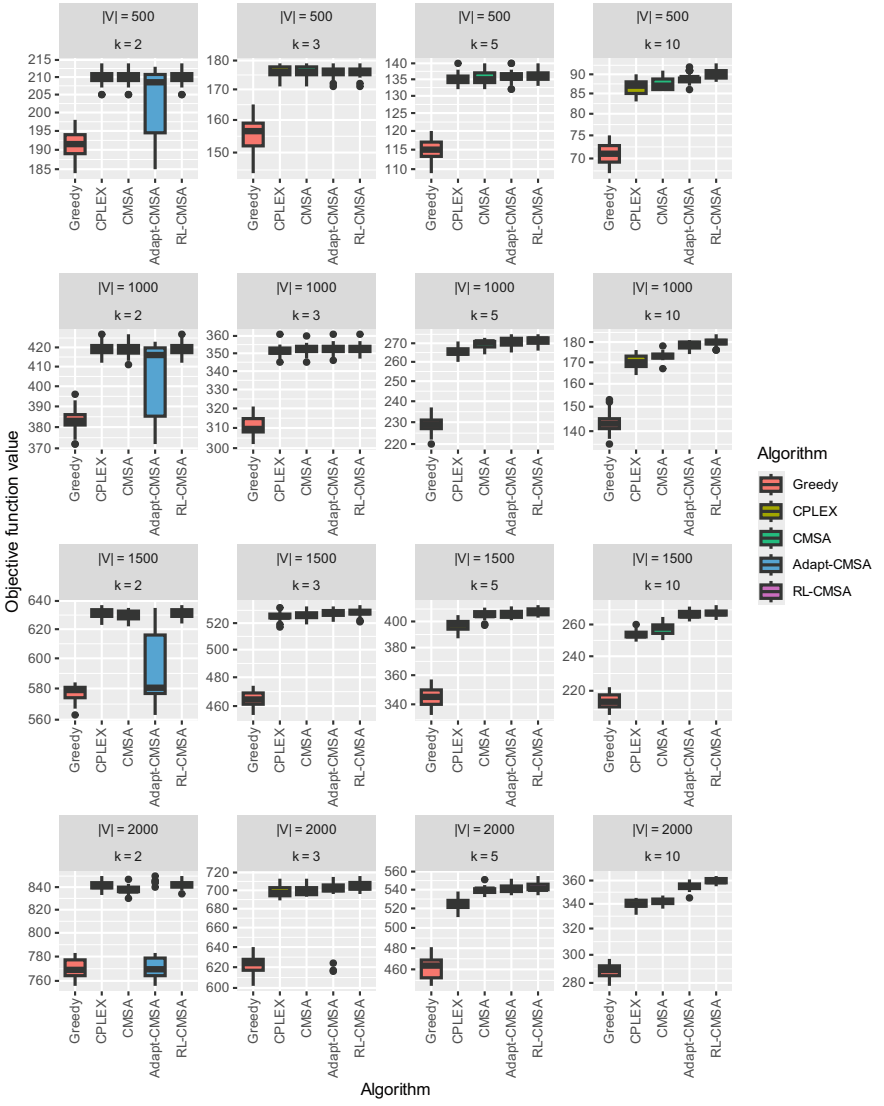

Fig. 3 Results for Watts-Strogatz graphs

graphs, Adapt-CMSA—when considering all of these graphs—is outperformed by CMSA. When studying the box plots from Fig 2, it becomes clear that this is the case due to a very low performance of Adapt-CMSA for the sparsest graphs (see the boxplots in the first column of that graphic). Remember that the parameter setting of Adapt-CMSA chosen by `irace` was such that the construction of new solutions is very much biased toward the best-so-far solution. This setting, even though working very well for the three groups of graphs of

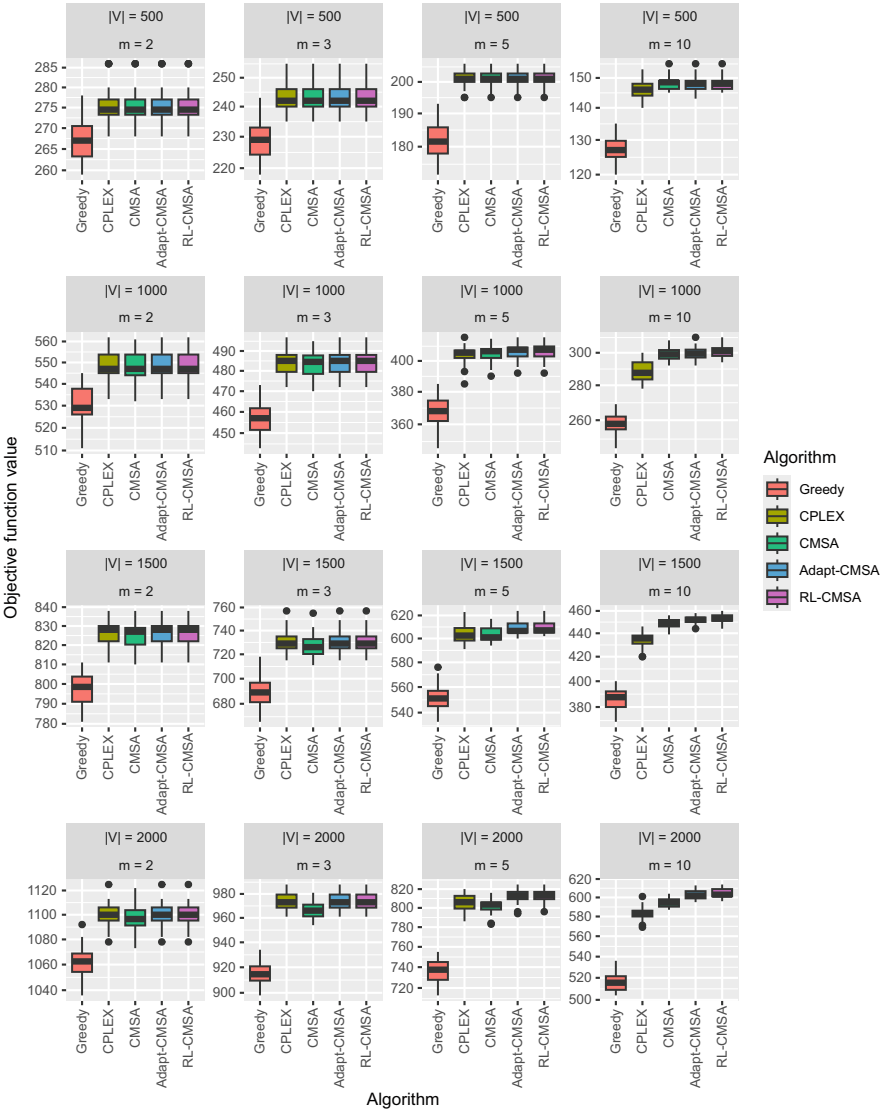

Fig. 4 Results for Barabási-Albert graphs

higher density (regardless of the network model), does not seem to work well at all when sparse graphs (at least the ones produced by the Erdös-Rényi and the Watts-Strogatz models) are concerned. We assume that, with a rather greedy solution construction, it is very hard to escape from local minima in these cases.

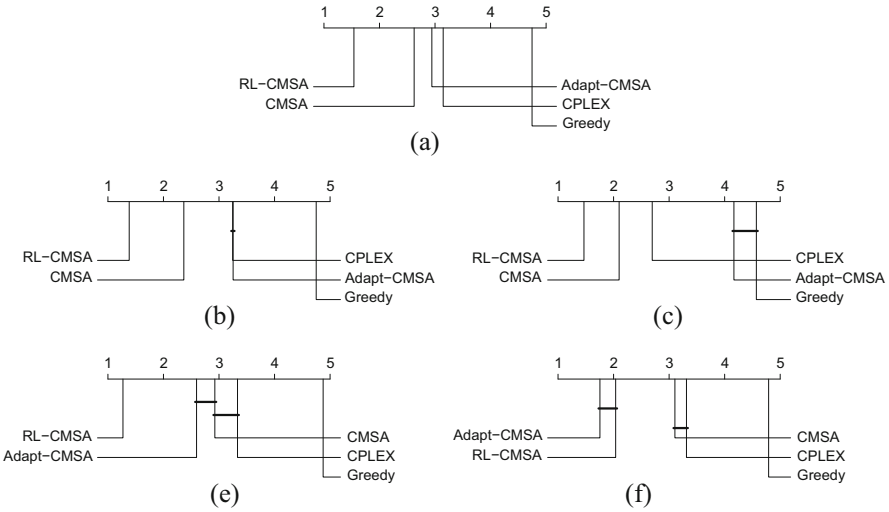

Fig. 5 Critical Difference (CD) plots concerning Erdös-Rényi graphs. (**a**) All graphs. (**b**) Density $p = 0.00624144$. (**c**) Density $p = 0.00416381$. (**d**) Density $p = 0.0103881$. (**e**) Density $p = 0.020705$

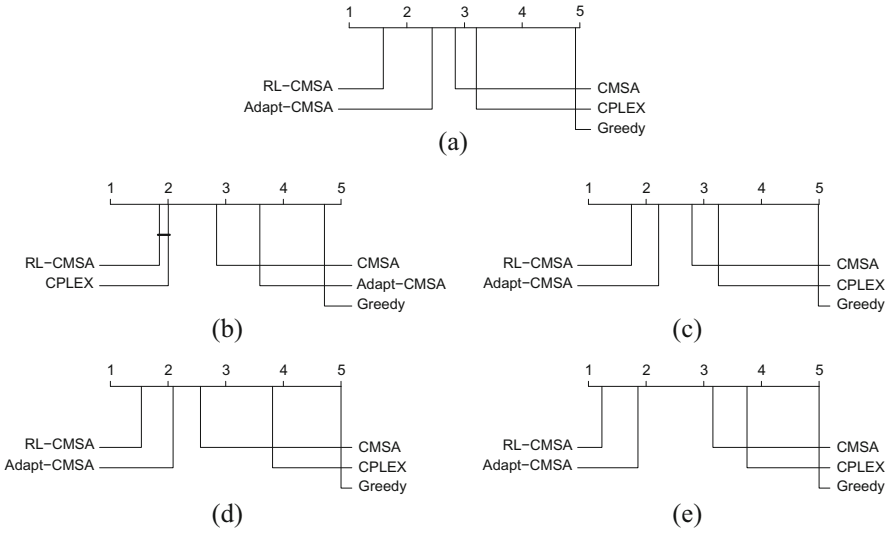

Fig. 6 Critical Difference (CD) plots concerning Watts-Strogatz graphs. (**a**) All graphs. (**b**) Density $k = 2$. (**c**) Density $k = 3$. (**d**) Density $k = 5$. (**e**) Density $k = 10$

Finally, the `STNWeb` tool [33] was used in an attempt to learn something about the changing behavior of the three CMSA variants when applied to graphs created with different network models. This tool produces graphics—called STN graphics—that visualize the trajectories of algorithms in the search space. We

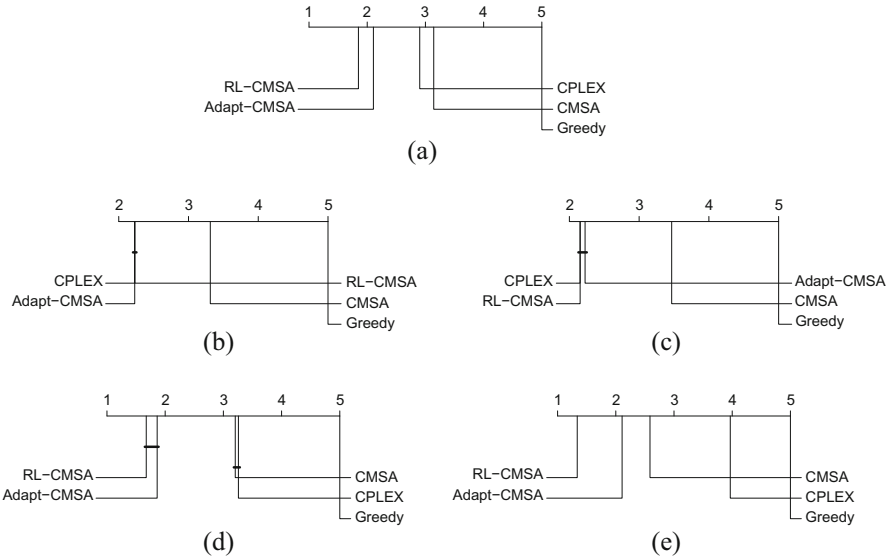

Fig. 7 Critical Difference (CD) plots concerning Barabási-Albert graphs. (**a**) All graphs. (**b**) Density $m = 2$. (**c**) Density $m = 3$. (**d**) Density $m = 5$. (**e**) Density $m = 10$

applied CMSA, Adapt-CMSA, and RL-CMSA ten times to all problem instances and recorded every new best-so-far solution to produce these trajectories. In Fig 8, we show the resulting STN graphics for the first (out of 30) graphs with 2000 nodes from the third density level as a representative example. The three graphics in this figure refer to the corresponding Erdös-Rényi graph (see Fig 8a), the Watts-Strogatz graph (see Fig 8b), and the Barabási-Albert graph (see Fig 8c). Before delving into the analysis, please note that the displayed graphics result from—in STN jargon—search space partitioning, meaning that each node (dot, square, triangle) in these graphics might represent a single solution or clusters of similar solutions. Firstly, it is evident that the STN graphics for the Erdös-Rényi and Watts-Strogatz graphs display far more similarities with each other than they do with the STN graphic for the Barabási-Albert graph. In particular, they show that both CMSA and Adapt-CMSA start their search trajectories in a common area of the search space. However, while some CMSA trajectories are attracted to the same search space region, this does not hold true for Adapt-CMSA. Moreover, they show that RL-CMSA trajectories start in different search space regions. These two observations make sense as CMSA and Adapt-CMSA, in contrast to RL-CMSA, are biased by the same greedy heuristic that favors low-degree nodes. In general, both graphics do not show any overlaps between the search trajectories of different algorithms. This suggests that, in Erdös-Rényi and Watts-Strogatz graphs, good solutions might be very different from each other. In contrast, in the context of the

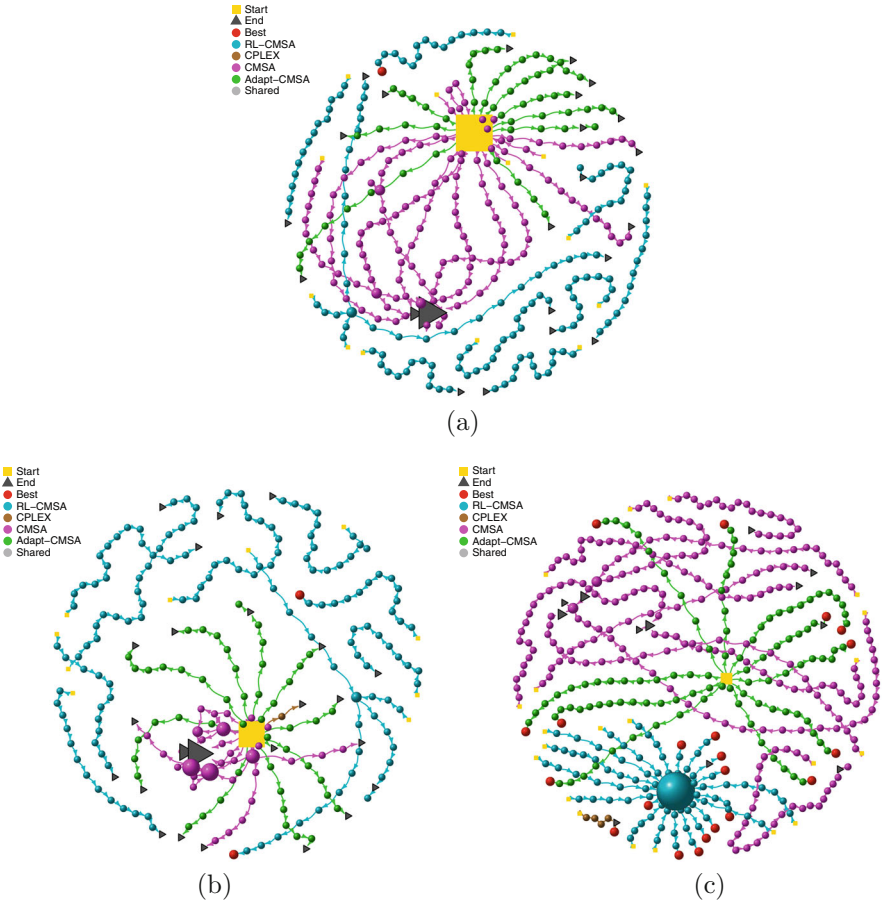

Fig. 8 STNWeb graphics. The three graphics show ten runs of all CMSA variants (in addition to the deterministic CPLEX run) for the first graph with 2000 nodes and the third density level. (**a**) shows results for the corresponding Erdös-Rényi graph, (**b**) for the Watts-Strogatz graph, and (**c**) for the Barabási-Albert graph

Barabási-Albert graph, the search trajectories of RL-CMSA seem to be attracted by a common area of the search space. After passing through this area, all of them find different solutions of the same quality (red dots). This behavior suggests that, in contrast to Erdös-Rényi and Watts-Strogatz graphs, in Barabási-Albert graphs, the MIS problem is characterized by a common backbone of nodes that must form part of high-quality solutions. As RL-CMSA is the only CMSA variant with a learning component, it is also the only one to learn to include the nodes from the backbone into solutions before finding the best solutions in each run.

Conclusion

This chapter introduced three variants of one of the most recent hybrid metaheuristics called "Construct, Merge, Solver & Adapt" (abbreviated as CMSA), which uses problem-specific solvers at each iteration to solve smaller sub-instances of the tackled problem instances. In addition to the standard CMSA variant, we described a self-adaptive CMSA variant called Adapt-CMSA and a CMSA variant called RL-CMSA that uses reinforcement learning. To showcase an application, these three CMSA variants were applied to solve the maximum independent set problem, a classical NP-hard problem in Computer Science. The RL-CMSA variant emerged as the best-performing technique, followed by the Adapt-CMSA variant. However, it was also evident that Adapt-CMSA might have problems escaping from local minima in case of an overly strong bias toward the best-so-far solution. In contrast, RL-CMSA showed a strong performance over the whole benchmark set of graphs produced by different network models.

We believe that one of the main research lines around CMSA in the near future will be concerned with strengthening the learning component of the algorithm. Recent applications of RL-CMSA [29] and Learn-CMSA [28] show the potential of this type of research. Another possibly fruitful research line concerns using specialized exact techniques (instead of black-box ILP solvers) for solving sub-instances.

Cross-References

▸ Matheuristics by Examples
▸ Matheuristics for Maritime Inventory Routing Problems

Acknowledgments The authors were supported by grants TED2021-129319B-I00 and PID2022-136787NB-I00 funded by MCIN/AEI/10.13039/501100011033.

Competing Interest Declaration The author(s) has no competing interests to declare that are relevant to the content of this manuscript.

References

1. Akbay MA, Kalayci CB, 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, Springer, pp 1–16
2. Akbay MA, López Serrano A, Blum C (2022) A self-adaptive variant of CMSA: application to the minimum positive influence dominating set problem. Int J Comput Intell Syst 15(1):44
3. Akbay MA, Kalayci CB, Blum C (2023) Application of Adapt-CMSA to the two-echelon electric vehicle routing problem with simultaneous pickup and deliveries. In: European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar), Springer, pp 16–33

4. Akbay MA, Blum C, Kalayci CB (2024) CMSA based on set covering models for packing and routing problems. Ann Oper Res 343(1):1–38
5. Alves Viana LG (2022) Uma meta-heurística híbrida para o problema de cobertura de discos ponderados. Bachelor's thesis, Universidade Federal de Alagoas, Instituto de Computação, Maceió, Brazil
6. Arora D, Maini P, Pinacho-Davidson P, Blum C (2019) Route planning for cooperative air-ground robots with fuel constraints: an approach based on CMSA. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp 207–214
7. Barabási AL, Albert R (1999) Emergence of scaling in random networks. Science 286(5439):509–512
8. Blum C (2020) Minimum common string partition: on solving large-scale problem instances. Int Trans Oper Res 27(1):91–111
9. Blum C (2024) Construct, Merge, Solve & Adapt: A Hybrid Metaheuristic for Combinatorial Optimization. Springer Nature, Berlin
10. Blum C, Blesa MJ (2016) Construct, merge, solve and adapt: application to the repetition-free longest common subsequence problem. In: Evolutionary Computation in Combinatorial Optimization: 16th European Conference, EvoCOP 2016, Porto, March 30–April 1, 2016, Proceedings 16, Springer, pp 46–57
11. Blum C, Blesa MJ (2018) A comprehensive comparison of metaheuristics for the repetition-free longest common subsequence problem. J Heurist 24(3):551–579
12. Blum C, Gambini Santos H (2019) Generic CP-supported CMSA for binary integer linear programs. In: Hybrid Metaheuristics: 11th International Workshop, HM 2019, Concepción, Chile, January 16–18, 2019, Proceedings 11, Springer, pp 1–15
13. Blum C, Ochoa G (2021) A comparative analysis of two matheuristics by means of merged local optima networks. Eur J Oper Res 290(1):36–56
14. Blum C, Pereira J (2016) Extension of the CMSA algorithm: an LP-based way for reducing sub-instances. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, pp 285–292
15. Blum C, Pinacho P, López-Ibáñez M, Lozano JA (2016) Construct, merge, solve & adapt: a new general algorithm for combinatorial optimization. Comput Oper Res 68:75–88
16. de Oliveira EB, da Silva Batista M, Pinheiro RGS (2023) Uma abordagem híbrida CMSA para o problema da cadeia de caracteres mais próxima. In: Proceedings of the Simpósio Brasileiro de Pesquisa Operacional, vol 55
17. Djukanović M, Kartelj A, Blum C (2023) Self-adaptive CMSA for solving the multidimensional multi-way number partitioning problem. Expert Syst Appl 232:120762
18. Dupin N, Talbi EG (2021) Matheuristics to optimize refueling and maintenance planning of nuclear power plants. J Heurist 27(1–2):63–105
19. Erdös P, Rényi A (1959) On random graphs I. Publ math debrecen 6(290–297):18
20. Ferrer J, Chicano F, Ortega-Toro JA (2021) CMSA algorithm for solving the prioritized pairwise test data generation problem in software product lines. J Heurist 27:229–249
21. Ghirardi M, Salassa F (2022) A simple and effective algorithm for the maximum happy vertices problem. Top 30(1):181–193
22. Hawa A (2020) Exact and evolutionary algorithms for the score-constrained packing problem. PhD thesis, Cardiff University
23. Lewis R, Thiruvady D, Morgan K (2019) Finding happiness: an analysis of the maximum happy vertices problem. Comput Oper Res 103:265–276
24. Lizárraga E, Blesa MJ, Blum C (2017) Construct, merge, solve and adapt versus large neighborhood search for solving the multi-dimensional knapsack problem: which one works better when? In: Evolutionary Computation in Combinatorial Optimization: 17th European Conference, EvoCOP 2017, Amsterdam, April 19–21, 2017, Proceedings 17, Springer, pp 60–74
25. López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58

26. Martí R, Parreño F, Mortes J (2024) Mathematical models and solving methods for diversity and equity optimization. J Heurist 30(5):291–323
27. Pinacho-Davidson P, Bouamama S, Blum C (2019) Application of CMSA to the minimum capacitated dominating set problem. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp 321–328
28. Pinacho-Davidson P, Blum C, Pinninghoff MA, Contreras R (2024) Extension of CMSA with a learning mechanism: application to the far from most string problem. Int J Comput Intell Syst 17(1):109
29. Reixach J, Blum C (2024) How to improve "Construct, Merge, Solve and Adapt"? use reinforcement learning! Ann Oper Res In press
30. Rosati RM, 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, Springer, pp 254–267
31. Rosati RM, Bouamama S, Blum C (2023) Construct, merge, solve and adapt applied to the maximum disjoint dominating sets problem. In: Di Gaspero L, Festa P, Nakib A, Pavone M (eds) Metaheuristics. Springer International Publishing, Cham, pp 306–321
32. Rosati RM, Bouamama S, Blum C (2024) Multi-constructor CMSA for the maximum disjoint dominating sets problem. Comput Oper Res 161:106450
33. Sartori CC, Blum C, Ochoa G (2023) STNWeb: a new visualization tool for analyzing optimization algorithms. Softw Impacts 17:100558
34. Thiruvady D, Lewis R (2022) Recombinative approaches for the maximum happy vertices problem. Swarm Evolut Comput 75:101188
35. Thiruvady D, Blum C, Ernst AT (2019) Maximising the net present value of project schedules using CMSA and parallel ACO. In: Hybrid Metaheuristics: 11th International Workshop, HM 2019, Concepción, January 16–18, 2019, Proceedings 11, Springer, pp 16–30
36. Watts DJ, Strogatz SH (1998) Collective dynamics of 'small-world' networks. Nature 393(6684):440–442
37. Zhou J, Zhang P (2024) Simple heuristics for the rooted max tree coverage problem. In: Wu W, Guo J (eds) Combinatorial Optimization and Applications. Springer Nature Switzerland, Cham, pp 252–264