International Journal of Computational Intelligence Systems (2022) 15:44
https://doi.org/10.1007/s44196-022-00098-1

RESEARCH ARTICLE

A Self-Adaptive Variant of CMSA: Application to the Minimum Positive Influence Dominating Set Problem

Mehmet Anıl Akbay[1] · **Albert López Serrano**[1] · **Christian Blum**[1]

Received: 31 January 2022 / Accepted: 20 June 2022

Abstract
Construct, merge, solve and adapt (CMSA) is a recently developed, generic algorithm for combinatorial optimisation. Even though the usefulness of the algorithm has been demonstrated by applications to a range of combinatorial optimisation problems, in some applications, it was observed that the algorithm can be sensitive to parameter settings. In this work, we propose a self-adaptive variant of CMSA, called Adapt-CMSA, with the aim of reducing the parameter sensitivity of the original version of CMSA. The advantages of this new CMSA variant are demonstrated in the context of the application to the so-called minimum positive influence dominating set problem. It is shown that, in contrast to CMSA, Adapt-CMSA does not require a computation time intensive parameter tuning process for subsets of the considered set of problem instances. In fact, after tuning Adapt-CMSA only once for the whole set of benchmark instances, the algorithm already obtains state-of-the-art results. Nevertheless, note that the main objective of this paper is not the tackled problem but the improvement of CMSA.

Keywords Combinatorial optimisation · Hybrid algorithm · Self-adaptation · Positive influence dominating set · Social networks

Abbreviations

CMSA	Construct, merge, solve and adapt
Adapt-CMSA	Self-adaptive construct, merge, solve and adapt
CO	Combinatorial optimisation
ILP	Integer linear programming
QAP	Quadratic assignment problem
MDS	Minimum dominating set
LNS	Large neighbourhood search
MPIDS	Minimum positive influence dominating set

1 Introduction

Algorithms for solving combinatorial optimisation (CO) problems [1] generally fall into two different categories: (1) exact techniques provide optimal solutions to the tackled problems in bounded time, and (2) approximate techniques are designed to provide good-enough solutions to the tackled problems within rather low computation times. The first category of approaches includes algorithms such as dynamic programming and mathematical programming techniques, as, for example, branch and bound or branch and cut. Most cutting edge mathematical programming techniques are implemented in commercial solvers such as CPLEX[1] and Gurobi.[2] These solvers exhibit a great performance, for example, for combinatorial optimisation problems that can be expressed in terms of integer linear programming (ILP) models.[3] However, with growing problem instance size and/or difficulty, these solvers start to fail. For some problems, this happens already for rather small problem instances [as an example consider the well-known quadratic assignment problem (QAP)], and for other problems, these solvers are able to solve surprisingly large instances to optimality in short computation times [as an example consider the minimum dominating set (MDS) problem]. In those cases in which exact solvers fail, approximate techniques are applied instead. This category of algorithms includes simple Greedy heuristics, but also more sophisticated metaheuristics [2, 3]. Examples of the latter ones are tabu search, iterated local search, evolutionary

✉ Christian Blum
christian.blum@csic.es

Mehmet Anıl Akbay
makbay@iiia.csic.es

Albert López Serrano
albert99n@gmail.com

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

[1] https://www.ibm.com/analytics/cplex-optimizer.

[2] http://www.gurobi.com/.

[3] Note that a wide range of combinatorial optimisation problems can be expressed in terms of ILPs.

Published online: 09 July 2022

algorithms, and ant colony optimisation. These algorithms perform often very well for instances of medium and even large size. However, in the context of large to very large instances, metaheuristics might get lost in the huge search spaces defined by these instances. For this reason, a popular trend in recent years concerns the hybridisation between exact techniques and metaheuristics [4–6]. The resulting hybrid algorithms often benefit from synergies between the exact and the approximate algorithm components of which they are composed. This has turned out to be beneficial, especially in the context of huge search spaces.

1.1 Background

One of the most popular hybrid techniques is *large neighbourhood search* (LNS) [7], respectively, large-scale neighbourhood search [8]. LNS is a method based on local search. In other words, the algorithm generally works with one incumbent solution per iteration and tries to identify a better solution in a predefined neighbourhood of the incumbent solution. The difference to a standard local search technique is that the neighbourhoods considered in LNS are much larger. In fact, in some cases, it might even be an NP-hard problem itself to find an improving neighbour in such a large neighbourhood. The main difference between existing LNS approaches is the way in which the large neighbourhoods are generated. However, many LNS approaches are based on the principle of *ruin-and-recreate* [9], also sometimes found as *destroy-and-recreate* or *destroy-and-rebuild*. In this type of LNS, the following is done at each iteration. First, the incumbent solution is partially destroyed, resulting in a partial solution. Second, a heuristic or an exact technique is used to search for an improving solution in the space of all feasible solutions that contain this partial solution. Examples of applications can be found in [10–12], just to name a few. Alternative ways of defining large neighbourhoods include local branching [13], the corridor method [14], and POPMUSIC [15].

A recent alternative to LNS is *construct, merge, solve and adapt* (CMSA) [16]. In principle, the idea of CMSA is similar to the one of LNS: at each iteration, the search space of a substantially reduced sub-instance of the tackled problem instance is searched for a better solution than the best solution found so far. The way, however, in which these reduced sub-instances are produced is conceptually very different. At each iteration, the algorithm probabilistically generates a set of solutions to the tackled problem instance. These solutions are then merged with an initially empty sub-instance, and an exact solver is applied to possibly find a best solution to the current sub-instance. Finally, the sub-instance is adapted based on this solution, and the algorithm proceeds with the subsequent iteration. CMSA has been successfully applied to a number of combinatorial optimisation problems. Some of the latest applications include the one to the maximum happy vertices problem [17], to route planning for cooperative air-ground robots [18], to refuelling and maintenance planning of nuclear power plants [19], and to the prioritised pairwise test data generation problem in software product lines [20].

1.2 Contribution

An overly high sensitivity to changes in parameter values is a recognised problem in research on metaheuristics [21]. Hereby, a metaheuristic is generally said to be *parameter sensitive*, if (1) the algorithm performance for specific instances or instance groups strongly depends on the parameter values and if (2) the required parameter values for different instances or instance groups are rather different to each other. When algorithms are too sensitive to parameter settings, this is seen as a rather negative aspect in the research community. Unfortunately, such a high sensitivity to parameter values was noticed in some applications of CMSA in the literature. One of the examples concerns the preliminary application of CMSA to an NP-hard CO problem known as the *minimum positive influence dominating set* (MPIDS) problem [22]. Therefore, in this paper, we propose a self-adaptive variant of CMSA, called Adapt-CMSA, with the aim of obtaining an algorithm less sensitive to parameter values. As a test case, we use the above-mentioned MPIDS problem. The obtained results show that Adapt-CMSA has several advantages over standard CMSA in the context of the MPIDS problem. First, Adapt-CMSA does, indeed, not require specific parameter tuning for subsets of the considered benchmark set. After applying parameter tuning once, Adapt-CMSA works very well for the whole benchmark set containing instances of very different sizes. Second, Adapt-CMSA clearly outperforms standard CMSA in the context of large networks for which even a specialised tuning does not enable CMSA to compete with Adapt-CMSA. We would expect a similar advantage of Adapt-CMSA over standard CMSA in most applications in which standard CMSA shows a high parameter sensitivity.

1.3 Paper Outline

The remainder of this paper is organised as follows. In Sect. 2, first, an introduction to standard CMSA is given, before the new self-adaptive CMSA variant is presented. Subsequently, in Sect. 3, the application of both standard CMSA and Adapt-CMSA to the MPIDS problem are outlined. Finally, a comprehensive experimental evaluation is provided in Sect. 4, while conclusions and an outline of future work can be found in Sect. 5.

2 Construct, Merge, Solve and Adapt

In this section, we first describe the standard version of CMSA in the context of CO problems that can be modelled in terms of binary ILPs. Subsequently, we introduce the self-adaptive variant of CMSA, henceforth labelled Adapt-CMSA. For the description of both CMSA variants, we assume to be tackling a CO problem which can be modelled in term of an ILP where $f()$ is the objective function to be minimised, and $x_i \in \{0, 1\}$ $(i = 1, \dots, n)$ is the set of binary decision variables used to model the objective function and the constraints of the problem. Note that many NP-hard CO problems fall into this category of problems. Examples include the well-known travelling salesman problem (TSP) and the quadratic assignment problem (QAP), just to name two emblematic problems.

In the general case as described above, we introduce a solution component c_i^0 and a solution component c_i^1 for each binary variable $x_i, i = 1, \dots, n$. Hereby, c_i^0 corresponds to $x_i = 0$, while c_i^1 corresponds to $x_i = 1$. Moreover, $C = \{c_1^0, \dots, c_n^0, c_1^1, \dots, c_n^1\}$ is the complete set of $2n$ solution components. Any *candidate solution* s is a subset of C with $|s| = n$. In addition, it is required that s contains exactly one of the two components c_i^0 and c_i^1 for each $i = 1, \dots, n$. Finally, a candidate solution s is a valid solution if it fulfils all the constraints of the tackled problem.

2.1 Standard CMSA

Algorithm 1 Pseudo-code of standard CMSA

```
1:  input 1: input graph G and the set of solution components C
2:  input 2: values for CMSA parameters n_a, age_max, and t_ILP
3:  input 3: values for solution construction parameters d_rate, l_size
4:  s_bsf := GenerateGreedySolution(C)
5:  C' := s_bsf
6:  age[c] := 0 for all c ∈ C
7:  while CPU time limit not reached do
8:      for i := 1, ..., n_a do
9:          s := ProbabilisticSolutionConstruction(C, l_size, d_rate)
10:         for all c ∈ S and c ∉ C' do
11:             age[c] := 0
12:             C' := C' ∪ {c}
13:         end for
14:     end for
15:     s'_opt := SolveSubinstance(C', t_ILP)
16:     if f(s'_opt) < f(s_bsf) then s_bsf := s'_opt end if
17:     Adapt(C', s'_opt, age_max)
18: end while
19: output: s_bsf
```

Algorithm 1 provides the pseudo-code of a standard CMSA for binary optimisation problems. Note that all functions in the pseudo-code are indicated with a special font as, for example, in GenerateGreedySolution(C). This function is used to initialise the best-so-far solution s_{bsf} with the solution generated by a greedy algorithm, as outlined in detail below. This is done at the start of the algorithm. Moreover, the sub-instance C', which is solved by an ILP solver at each iteration, is initialised to s_{bsf}. Note that, alternatively, s_{bsf} might be initialised to null and C' to the empty set. Each solution component $c \in C$ maintains a so-called age value $age[c]$. These age values are all initialised to zero. Note that the purpose of the age value of a solution component c is to count the number of consecutive CMSA iterations for which c forms part of C', without being included in the ILP-solution to the reduced problem instance generated on the basis of C'. At each iteration, CMSA iterates through four algorithmic steps. In the *construct* step, n_{a} valid solutions to the tackled problem are probabilistically constructed in function ProbabilisticSolutionGeneration(C). In the *merge* step, those solution components that (1) are found in at least one of the constructed solutions from the *construct* step, and (2) do currently not form part of C', are added to C' and their age value is set to zero. Next, the *solve* step first generates a reduced problem instance on the basis of C', which is done by adding—for all $i = 1, \dots, n$—the following constraints to the original ILP model of the tackled problem:

1. If $c_i^0 \in C'$ and $c_i^1 \notin C'$: add constraint $x_i = 0$ to the ILP model
2. If $c_i^0 \notin C'$ and $c_i^1 \in C'$: add constraint $x_i = 1$ to the ILP model

Note that, the more of these constraints are added to the original ILP, the smaller is the search space of the resulting sub-instance. Afterwards the extended ILP is solved in function SolveSubinstance(C', t_{ILP}), for example, by the application of an ILP solver with a CPU time limit of t_{ILP} seconds. Note that a variable x_i is only free in the extended ILP, if both solution components c_i^0 and c_i^1 form part of C'. Note also that the output of function SolveSubinstance(C', t_{ILP}) is—due to the computation time limit—not necessarily an optimal solution to the extended ILP. In those cases in which $f(s'_{\text{opt}}) < f(s_{\text{bsf}})$, the output of function SolveSubinstance(C', t_{ILP}) is set as s_{bsf}. Finally, in the *adapt* step, sub-instance C' is adapted in function Adapt($C', s'_{\text{opt}}, \text{age}_{\text{max}}$) depending both on s'_{opt} and on the age values of the solution components. This is done by increasing the age values of all components in $C' \setminus s'_{\text{opt}}$ by one, and by re-initialising the age values of all components in s'_{opt} to zero. The final action in the *adapt* step consists in removing all those components from C' whose age value has reached the maximum allowed age of age_{max}. This is done in order to prevent components that never appear in s'_{opt} to slow down the ILP solver in subsequent iterations.

2.2 Self-Adaptive CMSA

Algorithm 2 Pseudo-code of self-adaptive CMSA: Adapt-CMSA

1: **input 1:** input graph G and the set of solution components C
2: **input 2:** values for CMSA parameter t_{prop}, t_{ILP}
3: **input 3:** values for solution construction parameters α^{LB}, α^{UB}, α_{red}
4: $s_{\text{bsf}} := \text{GenerateGreedySolution}(C)$
5: $n_{\text{a}} := 1$; $\alpha_{\text{bsf}} := \alpha^{\text{UB}}$; $C' := s_{\text{bsf}}$
6: **while** CPU time limit not reached **do**
7: **for** $i := 1, \ldots, n_{\text{a}}$ **do**
8: $s := \text{ProbabilisticSolutionConstruction}(C, s_{\text{bsf}}, \alpha_{\text{bsf}})$
9: $C' := C' \cup s$
10: **end for**
11: $(s'_{\text{opt}}, t_{\text{solve}}) := \text{SolveSubinstance}(C', t_{\text{ILP}})$ ▷ This function returns two objects: (1) the obtained solution (s'_{opt}), (2) the required computation time (t_{solve})
12: **if** $t_{\text{solve}} < t_{\text{prop}} \cdot t_{\text{ILP}}$ and $\alpha_{\text{bsf}} > \alpha^{\text{LB}}$ **then** $\alpha_{\text{bsf}} := \alpha_{\text{bsf}} - \alpha_{\text{red}}$ **end if**
13: **if** $f(s'_{\text{opt}}) < f(s_{\text{bsf}})$ **then**
14: $s_{\text{bsf}} := s'_{\text{opt}}$
15: $n_{\text{a}} := 1$
16: **else**
17: **if** $f(s'_{\text{opt}}) > f(s_{\text{bsf}})$ **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: $C' := s_{\text{bsf}}$
24: **end while**
25: **output:** s_{bsf}

The pseudo-code of self-adaptive CMSA (Adapt-CMSA) is provided in Algorithm 2. The first noticeable difference to standard CMSA is the absence of the age values. This is because Adapt-CMSA works with a fixed maximum age of one, that is, after each iteration all solution components apart from those that form part of the best-so-far solution s_{bsf} are removed from the sub-instance C' (see line 23). Another difference can be seen in function ProbabilisticSolutionConstruction(C, s_{bsf}, α_{bsf}) for the probabilistic generation of solutions at each algorithm iteration (see line 8). Note that this latter function receives, apart from the set of all possible solution components (C), the currently best-so-far solution s_{bsf} and a parameter α_{bsf} (where $0 \leq \alpha_{\text{bsf}} < 1$) as input. This parameter biases the construction of new solutions towards the best-so-far solution s_{bsf}. More specifically, the higher the value of α_{bsf}, the higher will be the similarity of the solutions constructed in ProbabilisticSolutionConstruction(C, s_{bsf}, α_{bsf}) to s_{bsf}.

The dynamic change of the value of α_{bsf} is one of the aspects that is handled in a self-adaptive way in Adapt-CMSA. First of all, Adapt-CMSA requires a lower bound α^{LB} and an upper bound α^{UB} for the value of α_{bsf} as input. Moreover, the step size α_{red} for the reduction of α_{bsf} must also be given as input. Adapt-CMSA starts with setting α_{bsf} to the highest possible value α^{UB}; see line 5.[4] In case the resulting ILP can be solved in a computation time t_{solve} which is below a proportion t_{prop} of the maximally possible computation time t_{ILP}, the value of α_{bsf} is reduced by α_{red}; see line 12. The rationale behind this step is the following one. In case the resulting ILP can be solved easily, the search space of the ILP is too small due to a rather low number of free variables. In order to have more free variables in the ILP, the solutions constructed in ProbabilisticSolutionConstruction(C, s_{bsf}, α_{bsf}) should be more different to s_{bsf}, which can be achieved by reducing the value of α_{bsf}.

The second aspect which is handled in a self-adaptive way in Adapt-CMSA is the number of solution constructions per iteration (n_a); see lines 13–22. The algorithm starts with a value

[4] Remember that this means that solutions constructed in this way will be more similar to s_{bsf} than with lower values of α_{bsf}.

of $n_a = 1$; see line 5. Moreover, in case the solution of the reduced ILP (s'_{opt}) improves over the best-so-far solution s_{bsf}, n_a is set back to one; see line 15. If, however, the solution of the reduced ILP (s'_{opt}) is strictly worse than the best-so-far solution s_{bsf}, the corresponding sub-instance was clearly too large and/or complex in order to be solved by the ILP solver within t_{ILP} seconds. In this case, if $n_a = 1$ the value of α_{bsf} is slightly increased (by $\frac{\alpha_{\text{red}}}{10}$); resp. n_a is set back to one, otherwise. In the remaining case ($f(s'_{\text{opt}}) = f(s_{\text{bsf}})$), n_a is incremented by one; see line 20. This is done because the sub-instance did not contain a better solution than s_{bsf}. At the same time, the sub-instance was solved within the allowed computation time of t_{ILP} seconds, which means that the size of the sub-instance should be increased.

Finally, note that functions SolveSubinstance(C', t_{ILP}) are exactly the same in both version of CMSA (standard CMSA and Adapt-CMSA).

3 Application to the MPIDS Problem

The only problem-dependent part of both standard CMSA and Adapt-CMSA is the construction of feasible solutions. For the purpose of describing the solution construction procedures, we first need to introduce the tackled problem. As mentioned in the introduction, both standard CMSA and Adapt-CMSA are applied to an NP-hard combinatorial optimisation problem known as the *minimum positive influence dominating set (MPIDS)* problem. This problem is known for its applications in the context of social networks. Imagine that the nodes and edges in such a social network represent individuals (persons) and relationships/interactions between those individuals, respectively. In general, information propagated in social networks has the potential to have a significant impact, which might be either positive or negative, on (parts of) the society. As social norms theory shows that the behaviour of individuals can be affected by the perception of others' thoughts and behaviours [23], relationships among people in social networks may be exploited in order to obtain economical and/or societal benefits. In this sense, the aim of the MPIDS problem is to identify a small subset of influential individuals (or key individuals) for speeding up the spread of positive influence in a social network [24, 25]. Alternative applications of the MPIDS problem can be found in e-learning software [26], online business [27], drinking, smoking, and other drug-related problems [28].

In the following, the MPIDS problem is described in a technical way. Let $G = (V, E)$ be an undirected graph without loops and without parallel edges. Any subset $S \subseteq V$ that fulfils the following condition is a valid solution to the problem: at least half of the neighbours of each vertex $v \in V$ must form part of S. Note that, if G is connected, any valid solution S is also a dominating set of G. The MPIDS problem aims to find a valid solution $S^* \subseteq V$ of minimum size. In other words, given a valid solution $S \subseteq V$, the objective function value of S is $f(S) := |S|$. Note that $S := V$ is a trivial solution to the problem. The MPIDS problem is NP-hard.

From an algorithmic point of view, the efforts of the research community initially focussed on the development of well-working greedy heuristics [29–34]. In fact, until 2021, the best available approach was our own greedy method from [34]. The development of successful metaheuristic approaches seemed much harder. This is shown by the results of the first two metaheuristics—an ILP-based memetic algorithm [35] and a swarm intelligence based algorithm [36]—whose results are inferior to the greedy approach from [34]. The first metaheuristic that was able to improve over [34] is the iterated carousel approach from [37]. Finally, the currently best metaheuristics are our own approaches: a negative learning ant colony optimisation approach from [38] and the preliminary standard CMSA approach from [22]. Both approaches perform on a comparable level.

Note that, for the application of CMSA and Adapt-CMSA, we make use of the following ILP model which is well known from the related literature. This model is based on a binary variable x_i for each vertex $v_i \in V$.

$$\textbf{min} \quad \sum_{i=1}^{n} x_i, \tag{1}$$

$$\textbf{s.t.} \quad \sum_{v_j \in N(v_i)} x_j \geq \left\lceil \frac{\deg(v_i)}{2} \right\rceil \quad \forall v_i \in V, \tag{2}$$

$$x_i \in \{0, 1\}. \tag{3}$$

Hereby, $N(v_i)$ denotes the neighbourhood of v_i in the input graph G. Moreover, $\deg(v_i)$ is the degree of vertex v_i, where $\deg(v_i) := |N(v_i)|$. Equation (2) forces any feasible solution to contain at least half of the neighbours of each vertex $v_i \in V$.

3.1 Solution Construction in CMSA and Adapt-CMSA

In the following, we outline the remaining aspect of CMSA and Adapt-CMSA: the construction of solutions in function ProbabilisticSolutionConstruction ($C, l_{\text{size}}, d_{\text{rate}}$) in the case of CMSA, respectively in function ProbabilisticSolutionConstruction ($C, s_{\text{bsf}}, \alpha_{\text{bsf}}$) in the case of Adapt-CMSA. Both functions make use of the solution construction mechanism of the greedy procedure from [34]. They only differ in the way in which this procedure is made probabilistic. For the following discussion, remember that a vertex $v \in V$ is called *covered* with respect to a (partial) solution s if and only if at least half of its

neighbours form part of S. In the opposite case, v is labelled as *uncovered*.

The solution construction mechanism utilised by both functions is shown in Algorithm 3. First, each solution s to be constructed is initialised by a set $s_{\text{par}} \subset V$ of nodes that must form part of an optimal solution; see line 3. Note that s_{par} is obtained by the application of a pre-processing procedure described in [34]. Then, at each step of the solution construction mechanism, the following is done. First, the set of all uncovered vertices with respect to (partial) solution s is determined; see line 5. This set is labelled U. Second, a node $v \in U$ such that $\deg(v) \leq \deg(v')$ for all $v' \in U$ is selected (line 6). Third, nodes from $N(v) \setminus s$ are iteratively added to s while $|N(v) \cap s| < \left\lceil \frac{\deg(v)}{2} \right\rceil$; see lines 7–10. Hereby, exactly one vertex from $N(v) \setminus s$ is selected by function ChooseFrom$(N(v) \setminus s)$ at each entry of the while loop.

In standard CMSA, function ChooseFrom$(N(v) \setminus s)$ is implemented as follows. At first, a candidate list L is created. This list includes all vertices $v' \in N(v) \setminus s$. Each vertex v' in L is characterised by its *cover degree* cov_deg(v'), which is the number of uncovered adjacent vertices of v'. Note that vertices in L are sorted according to a non-increasing cover degree value. Then, a uniform random number r is generated from the interval [0, 1]. If $r \leq d_{\text{rate}}$ (where d_{rate} is the so-called determinism rate) the vertex with the highest cover degree is selected and added to s. Otherwise, a vertex is selected randomly from the restricted candidate list which contains the first l_{size} vertices of L. Hereby, l_{size} is the size of the restricted candidate list. All vertices in the restricted candidate list have an equal probability $\frac{1}{l_{\text{size}}}$ of being selected.

In contrast, in Adapt-CMSA, function ChooseFrom $(N(v) \setminus s)$ is implemented in the following way. First, each vertex $v' \in N(v) \setminus s$ such that $v' \in s_{\text{bsf}}$ obtains a value $q(v') := (\text{cov\_deg}(v') + 1) \cdot \alpha_{\text{bsf}}$, while all other vertices $v'' \in N(v) \setminus s$ receive a value $q(v'') := (\text{cov\_deg}(v'') + 1) \cdot (1 - \alpha_{\text{bsf}})$. A vertex $\hat{v}$ is then chosen from $N(v) \setminus s$ according to the following probabilities:

$$\mathbf{p}(v') := \frac{q(v')}{\sum_{v'' \in N(v) \setminus s} q(v'')} \quad \forall\, v' \in N(v) \setminus s. \tag{4}$$

In other words, the higher the value of parameter $\alpha_{\text{bsf}} \in [0, 1]$, the stronger is the bias towards the best-so-far solution s_{bsf}. This bias does not exist in standard CMSA.

Algorithm 3 Solution construction procedure (CMSA and Adapt-CMSA)

1: **CMSA input:** solution construction parameters d_{rate}, l_{size}
2: **Adapt-CMSA input:** solution construction parameter α_{bsf}
3: $s := s_{\text{par}}$ ▷ $s_{\text{par}} \subset V$ is obtained from a pre-processing procedure
4: **while** s is not a valid solution **do**
5: Let $U \subseteq V$ be the set of uncovered vertices
6: Choose $v \in U$ such that $deg(v) \leq deg(v')$ for all $v' \in U$
7: **while** $|N(v) \cap s| < \left\lceil \frac{deg(v)}{2} \right\rceil$ **do**
8: $\hat{v} :=$ ChooseFrom$(N(v) \setminus s)$
9: $s := s \cup \{\hat{v}\}$
10: **end while**
11: **end while**
12: **output:** valid solution s

4 Experimental Evaluation

All experiments reported in the following were performed on a cluster of machines with Intel® Xeon® 5670 CPUs with 12 cores of 2.933 GHz and a minimum of 32 GB RAM. Note that CPLEX version 20.1 was used in one-threaded mode both in a standalone manner and within CMSA and Adapt-CMSA for solving the respective sub-instances. Two sets of experiments were performed. A comprehensive experimentation in the context of a new set of 800 scale-free networks is described in Sect. 4.1. The second set of experiments makes use of small, medium and large problem instances that were already used in the related literature; see Sect. 4.2.

4.1 Experiments Regarding Scale-Free Networks

In order to be able to compare CMSA and Adapt-CMSA on a controlled set of benchmark instances with different features, we generated the following set of 800 scale-free networks using the `igraph` software package [39]. An undirected network is said to be scale-free—or, equivalently, to follow a power-law distribution—if the statistical distribution of the degrees of its nodes is as follows:

$$P(k) \sim k^{-\lambda}, \tag{5}$$

where $P(k)$ is the probability that a given node has exactly k neighbours. Specifically, we used the random power-law graph generator function called `igraph_static_power_law_game` to generate networks differing in the following parameters:

- Number of nodes, $|V| \in \{1.000, 10.000, 50.000, 100.000\ 250.000, 500.000, 750.000, 1.000.000\}$.
- Number of edges: $l \in \{5, 10, 20, 30\}$, where $l \cdot |V|$ is the number of edges
- Exponent for the power-law exponential distribution, $\lambda \in \{2, 2.25, 2.5, 2.75, 3\}$. Note that parameter λ establishes the pace at which the probability of having highly connected nodes decreases.

Note that five networks were generated for all combinations of the number of nodes, the number of edges, and the exponent for the power-law exponential distribution. This makes a total of 800 networks. Neither of the generated networks has self-loops or multiple edges between a pair of nodes. Note that, following the suggestion in the `igraph` package, the `finite_size_correction` mechanism [40] for the generation of the networks. Our reason for choosing power-law, scale-free networks for the comparison between CMSA and Adapt-CMSA is that they are generally accepted models for social networks [41, 42].[5]

For the purpose of parameter tuning, we separately generated one graph for each combination of $|V| \in \{50.000, 100.000, 500.000, 1.000.000\}$,$l \in \{5, 30\}$and $\lambda \in \{2, 3\}$. That is, 16 graphs were used for parameter tuning purposes. In particular, we used the scientific tuning software irace [43] for fine-tuning the parameters of CMSA and Adapt-CMSA. The parameters of CMSA (together with their domains allowed for tuning) are the following ones:

1. Number of solution constructions per iteration, $n_a \in \{1, 2, \ldots, 19, 20\}$.
2. Upper limit for the age values, $\text{age}_{\max} \in \{1, 2, \ldots, 9, 10\}$.
3. Time limit for CPLEX per call, $t_{\text{ILP}} \in \{1, 2, \ldots, 49, 50\}$ CPU seconds.
4. Determinism rate for solution construction, $d_{\text{rate}} \in [0.0, 0.99]$.
5. Length of the restricted candidate list, $l_{\text{size}} \in \{2, 3, \ldots, 9, 10\}$.

The parameters of Adapt-CMSA, together with their domains, are the following ones:

1. Time limit for CPLEX per call, $t_{\text{ILP}} \in \{1, 2, \ldots, 49, 50\}$ CPU seconds.
2. Lower bound for the bias towards the best-so-far solution, $\alpha^{\text{LB}} \in [0.6, 0.99]$.
3. Upper bound for the bias towards the best-so-far solution, $\alpha^{\text{UB}} \in [0.6, 0.99]$.
4. Step size for the reduction of this bias, $\alpha_{\text{red}} \in [0.01, 0.1]$.
5. Parameter used for determining when to reduce the bias, $t_{\text{prop}} \in [0.1, 0.8]$.

Note that in the case of numerical parameters, the precision of irace was fixed to two positions behind the comma. Both for the tuning of CMSA and Adapt-CMSA irace was applied with a budget of 3.000 algorithm applications. The time limit for each problem instance was set to $|V|/100$ CPU seconds. The outcome of the tuning runs can be summarised as follows:

- CMSA parameter values: $n_a = 1$, $\text{age}_{\max} = 1$, $t_{\text{ILP}} = 38$, $d_{\text{rate}} = 0.53$, $l_{\text{size}} = 2$.
- Adapt-CMSA parameter values: $t_{\text{ILP}} = 44$, $\alpha^{\text{LB}} = 0.85$, $\alpha^{\text{UB}} = 0.97$, $\alpha_{\text{red}} = 0.05$, $t_{\text{prop}} = 0.13$.

Note that both parameter settings indicate that we are dealing with very large graphs. In the case of CMSA, for example, $n_a = 1$ and $\text{age}_{\max} = 1$ are set in this restrictive way because, otherwise, the size of the sub-instances would be too large to be solved by CPLEX. Similarly, the parameters of Adapt-CMSA are characterised by a strong bias towards the best-so-far solution in order to keep the sub-instance as small as possible, while still being able to find improving solutions.

With the final parameter settings as provided above, both CMSA and Adapt-CMSA were applied exactly once to each of the 800 problem instances. The computation time limit was the same as the one chosen for tuning, that is, $|V|/100$ CPU seconds. The results are shown in a summarised way in the graphic of Fig. 1. The graphic is composed of $4 \times 8 = 24$ sub-graphics for each combination of $|V| = n$ (rows) and l (columns). Each sub-graphic shows—for all five values of λ—the average improvement of Adapt-CMSA over CMSA (in percent). Note that those cases in which Adapt-CMSA improves over CMSA are additionally marked by bars in blue colour, while bars in red colour indicate the cases in which CMSA is better than Adapt-CMSA.

The following observations can be made. First, for smaller graphs (up to 50.000 nodes) not much difference between the two algorithms can be observed. However, starting from 100.000 nodes, Adapt-CMSA clearly outperforms CMSA. This

[5] Note that, due to the size and the number of the generated graphs, this instance set is very large and consumes a large amount of memory (more than 17 GB). It can be obtained by contacting the corresponding author by email.

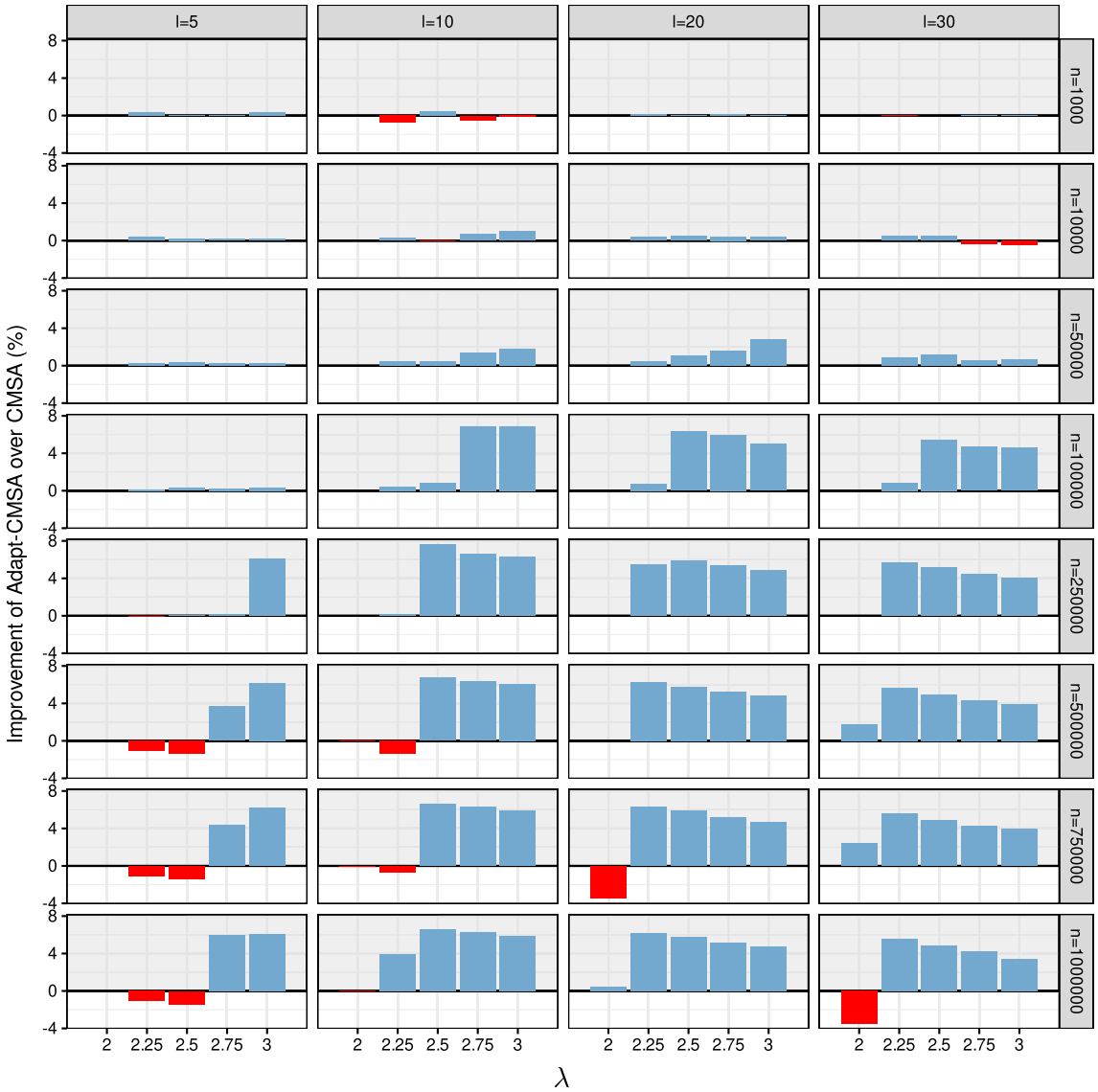

Fig. 1 Average improvement of Adapt-CMSA over standard CMSA (in percent)

holds especially with a growing number of nodes and a growing number of edges. Interestingly, for the smallest values of λ —that is, for $\lambda \in \{2, 2.25\}$ in the case of $l = 5$, respectively for $\lambda = 2$ in the case of the remaining values of l—CMSA often seems to have a slight advantage over Adapt-CMSA. In other words, the advantage of Adapt-CMSA over CMSA is higher for graphs with less nodes with high degrees. Nevertheless, these results provide a strong indication for the general superiority of Adapt-CMSA over CMSA.

4.2 Experiments Regarding Instances from the Literature

In our second set of experiments, we compare CMSA and Adapt-CMSA to the standalone application of CPLEX and to the best metaheuristic from the literature [37] (ICG). This is done in the context of 17 social networks that are partially used in the related literature on the MPIDS problem. These networks are of small and medium size, and contain between 34 and 36.692 nodes and between 788 and 198.050 edges. In addition, CPLEX and our CMSA variants were applied to 10 larger social networks from the SNAP library that contain between 37.700 and 1.134.890 nodes and between 2.289.003 and 3.387.388 edges (https://snap.stanford.edu/data/).

While CPLEX was applied exactly once to each of these 27 problem instances, both CMSA and Adapt-CMSA were applied 10 times to each instance. A computation time limit of 2 h was given to each CPLEX run. On the contrary, much less time was given to the CMSA variants. In the case of the 17 small/medium size problem instances we allowed a computation time

Table 1 Numerical results for small to medium size instances

Network	Best	CPLEX		ICG		CMSA			Adapt-CMSA		
	known	q	Gap (%)	q	Avg	q	Avg	$\overline{t(s)}$	q	Avg	$\overline{t(s)}$
Karate	15	**15**	0.00	n.a.	n.a.	**15**	15.00	0.004	**15**	15.00	0.004
Dolphins	30	**30**	0.00	**30**	30.0	**30**	30.00	0.02	**30**	30.00	0.02
Football	63	**63**	0.00	64	64.0	**63**	63.70	3.06	**63**	63.80	3.07
Jazz	79	**79**	0.00	n.a.	n.a.	**79**	79.00	1.66	**79**	79.00	0.23
CA-AstroPh	6736[a]	6740	0.30	6808	6812.95	6751	6753.60	1311.32	**6738**	6739.40	1329.03
CA-GrQc	2587	**2587**	0.00	**2587**	2587.50	**2587**	2587.00	19.08	**2587**	2587.00	1.87
CA-HepPh	4718	**4718**	0.01	4743	4746.85	4726	4727.50	604.52	**4718**	4718.20	482.04
CA-HepTh	4471	**4471**	0.00	4481	4483.10	4474	4474.70	306.65	**4471**	4471.00	17.60
CA-CondMat	9584	**9584**	0.06	9625	9627.85	9593	9595.00	1773.06	9585	9586.20	1048.08
Email-Enron	11682	**11682**	0.00	11737	11740.65	11692	11693.40	1815.34	**11682**	11682.80	690.15
ncstrlwg2	2994	**2994**	0.00	n.a.	n.a.	2995	2995.00	20.92	**2994**	2994.20	213.07
actors-data	3092	**3092**	0.24	n.a.	n.a.	3099	3100.90	763.51	3093	3093.70	597.38
ego-facebook	1973	**1973**	0.00	**1973**	1973.25	1975	1975.00	5.553	**1973**	1973.00	95.42
socfb-Brandeis99	1397[a]	**1400**	1.41	1443	1445.05	1427	1428.90	324.00	1414	1416.30	340.99
socfb-nips-ego	1398	**1398**	0.00	n.a.	n.a.	**1398**	1398.00	0.04	**1398**	1398.00	0.03
socfb-Mich67	1327[a]	**1329**	1.56	n.a.	n.a.	1342	1344.60	241.66	1340	1342.70	288.82
soc-gplus	8244	**8244**	0.00	n.a.	n.a.	8250	8251.20	956.37	**8244**	8244.00	2.425
Average		**3552.88**				3558.59	3559.56		3554.35	3554.96	

[a]These best-known results were obtained by [22] (CA-AstroPh) and [38] (socfb-Brandeis99, socfb-Mich67)

Table 2 Numerical results for large SNAP networks

Network	Best	CPLEX		CMSA			Adapt-CMSA		
	known	q	Gap (%)	q	Avg	$\overline{t(s)}$	q	Avg	$\overline{t(s)}$
musae_git	9752	**9752**	0.00	9793	9796.90	356.24	9757	9758.00	361.02
loc-gowalla_edges	67617	**67617**	0.07	67723	67727.90	1902.18	67690	67695.20	1915.43
gemsec_facebook_artist	15194	**15194**	1.20	15319	15330.70	494.05	15256	15259.50	484.12
deezer_HR	22699	54573	95.68	22567	22605.10	541.96	**22338**	22354.50	523.39
com-youtube	351281	**351281**	0.00	351960	351972.50	11134.93	351422	351431.20	11053.78
com-dblp	120492	**120492**	0.08	120640	120647.00	3082.43	120566	120576.30	3086.46
Amazon0302	130378	262111	97.50	128913	128939.80	2606.64	**128587**	128624.40	2605.34
Amazon0312	180853	400727	95.41	183113	183113.00	1.27	**174495**	174832.50	3994.77
Amazon0505	183114	410236	95.19	185310	185310.00	1.32	**176882**	177175.90	4100.25
Amazon0601	179964	403394	96.94	182279	182279.00	1.405	**173509**	173929.10	4030.06
Average		209537.70		126761.70			**124050.20**		

of $|V|/10$ CPU seconds for each run. A relatively shorter computation time of $|V|/100$ CPU seconds was allowed for the application to the large instances from the SNAP library. The main reason for this difference is that $|V|/100$ s would have been a very short computation time for most of the small and medium size instances.

In a first experiment we applied both CMSA and Adapt-CMSA with the parameter values from the previous section to all 27 instances. The obtained results are shown in numerical form in Table 1 (small/medium size instances) and Table 2 (large instances). These tables have the following structure. The first column contains the instance name, and the second column provides information about the quality of the best solutions known to date. Columns with heading 'q' report on the quality of the best solutions found by the four approaches, and columns with heading 'avg' provide the respective average solution quality. Furthermore, columns with heading '$\overline{t(s)}$' indicate the average computation times of CMSA and Adapt-CMSA to find the best solutions in each run. Note that the information about average computation times was not provided in [37] for ICG. The authors, however, state that they chose a computation time

limit of $|V| \cdot 30/1000$, because this assured convergence of their algorithm in the case of all considered problem instances. In other words, ICG would not profit from a higher computation time limit. Finally, the gap (in percent) between the solution obtained by CPLEX and the best lower bound is indicated in the column with heading 'gap(%)'. Note that when the gap is zero, CPLEX was able to prove optimality. The best result for each instance is shown in bold font. Furthermore, in case the best solution known so far was improved, the respective result is underlined. Finally, in those cases in which none of the algorithms was able to reach the currently best-known solution, we provide at the bottom of the table an indication of the algorithm that obtained the respective best-known solution.

The following observations can be made. First, CPLEX performs strongly for small and medium size instances. Apart from instance `CA-AstroPh`, CPLEX obtains all best-known solutions. Only in six out of 17 cases, CPLEX is not able to prove optimality of these results. The performance of Adapt-CMSA is very similar to the one of CPLEX. In one case (instance `CA-AstroPh`) Adapt-CMSA outperforms CPLEX both in terms of best-performance and in average-performance. On the downside, in four other cases (instances `CA-CondMat`, `actors-data`, `socfb-Brandeis99` and `socfbMich67`) the results of Adapt-CMSA fall slightly short of those of CPLEX. On the other side, Adapt-CMSA clearly outperforms CMSA, which (with the parameter setting for scale-free networks) only matches the results of Adapt-CMSA for six out of 17 problem instances. Finally, note that both CMSA and Adapt-CMSA clearly outperform the most recent metaheuristic from the related literature (ICG). Concerning the large instances from the SNAP library—see Table 2—we can state that the standalone application of CPLEX clearly starts to fail with a growing problem instance size. In fact, in five out of 10 cases—see the ones with an optimality gap of more than 95%—CPLEX is only able to provide the trivial solution that simply contains all network nodes. In addition, it can also be observed that the standard CMSA approach fails for instances `Amazon0312`, `Amazon0505` and `Amazon0601`. In these three cases, standard CMSA is not able to improve over the initial solutions provided by the greedy approach. Adapt-CMSA, on the other side, works very well also for these large-size SNAP networks. In fact, Adapt-CMSA is able to obtain new best-known solutions in five out of 10 cases. Moreover, in those five cases in which CPLEX still works fine, the results of Adapt-CMSA are only slightly worse than those of CPLEX. Therefore, a first conclusion of this work is that Adapt-CMSA appears to be a CMSA variant that does not require to be specifically tuned for subsets of the considered benchmark set. It shows a high performance over the whole range of benchmark instances with one single parameter value set.

In a second experiment, we aimed at studying the change of performance of standard CMSA when specifically tuned for small and medium size problem instances on one side, and for large SNAP network on the other side. Again, we used irace for the purpose or parameter tuning. For small and medium size instances, the budget given to irace consisted of 1000 algorithm applications, and instances `CA-AstroPh` and `socfb-Brandeis99` were used for tuning. The same budget was used for the tuning run concerning large SNAP networks. In this case, instances `Amazon0505` and `Amazon0601` were used for tuning. The outcome of these two tuning experiments was the following one:

- Small/med. size instances: $n_a = 1$, $\text{age}_{\max} = 3$, $t_{\text{ILP}} = 16$, $d_{\text{rate}} = 0.09$, $l_{\text{size}} = 8$.
- Large SNAP networks: $n_a = 1$, $\text{age}_{\max} = 1$, $t_{\text{ILP}} = 39$, $d_{\text{rate}} = 0.95$, $l_{\text{size}} = 10$.

Clearly, the parameter settings for small and medium size instances result in much larger sub-instance sizes than those for large SNAP networks. With these new parameter settings we repeated the experiments of CMSA. The results, in comparison to the CMSA results obtained with the previous parameter values, are shown in Tables 3 and 4.

The new CMSA results improve substantially in the case of small and medium size problem instances (Table 3). In fact, CMSA is now able to generate for 14 out of 17 problem instances the best-known solutions. In one case—see instance `actors-data`—CMSA is even able to generate a new best known solution of value 3091. With the specialised parameter setting, CMSA is now even able to perform slightly better, on average, than Adapt-CMSA (compare to Table 1). The results for large SNAP instances, however, show that specialised tuning does not help in this case. Even though the results of CMSA improve over the original ones from Table 2 in the case of those problem instances that were used for tuning (`Amazon0505` and `Amazon0601`), they become worse for seven out of 10 problem instances. Moreover, even in those cases in which CMSA is able to improve with a specialised parameter setting, the results are still clearly inferior to those of Adapt-CMSA.

5 Conclusions and Outlook to Future Work

Construct, merge, solve and adapt (CMSA) is a recent matheuristic for the application to combinatorial optimisation problems. The algorithm is based on solving opportunely defined sub-instances of the original problem instances at each iteration by means of an exact solver such as, for example, an integer linear programming solver. One of the occasional disadvantages of CMSA is the need for repeated parameter tuning for subsets of the considered benchmark set. For dealing with this problem, we proposed in this work a self-adaptive variant

Table 3 Improvement of CMSA after specific tuning for small and medium size instances

Network	Best	CMSA			CMSA (special tuning)		
	known	q	Avg	$\overline{t(s)}$	q	Avg	$\overline{t(s)}$
Karate	15	**15**	15.00	0.004	**15**	15.00	0.01
Dolphins	30	**30**	30.00	0.02	**30**	30.00	0.03
Football	63	**63**	63.70	3.06	**63**	63.70	4.81
Jazz	79	**79**	79.00	1.66	**79**	79.00	0.36
CA-AstroPh	6736	6751	6753.60	1311.32	**6736**	6737.30	1181.44
CA-GrQc	2587	**2587**	2587.00	19.08	**2587**	2587.00	1.57
CA-HepPh	4718	4726	4727.50	604.52	**4718**	4718.10	332.57
CA-HepTh	4471	4474	4474.70	306.65	**4471**	4471.00	14.08
CA-CondMat	9584	9593	9595.00	1773.06	**9584**	9584.10	1001.96
Email-Enron	11682	11692	11693.40	1815.34	**11682**	11682.00	1471.90
ncstrlwg2	2994	2995	2995.00	20.92	**2994**	2994.00	26.65
actors-data	3092	3099	3100.90	763.51	**3091**	3092.30	521.21
ego-facebook	1973	1975	1975.00	5.553	**1973**	1973.00	25.26
socfb-Brandeis99	1397	1427	1428.90	324.00	**1406**	1407.90	215.55
socfb-nips-ego	1398	**1398**	1398.00	0.04	**1398**	1398.00	0.04
socfb-Mich67	1327	1342	1344.60	241.66	**1335**	1338.30	198.67
soc-gplus	8244	8250	8251.20	956.37	**8244**	8244.20	808.90
Average		3558.59	3559.56		3553.29	3553.82	

Table 4 Results of CMSA after specific tuning for the large SNAP networks

Network	Best	CMSA			CMSA (special tuning)		
	known	q	Avg	$\overline{t(s)}$	q	Avg	$\overline{t(s)}$
musae_git	9752	**9793**	9796.90	356.24	9890	9896.40	359.88
loc-gowalla_edges	67617	**67723**	67727.90	1902.18	67993	68008.30	1957.53
gemsec_facebook_artist	15194	**15319**	15330.70	494.05	15511	15526.70	498.52
deezer_HR	22699	**22567**	22605.10	541.96	22727	22744.50	538.43
com-youtube	351281	**351960**	351972.50	11134.93	352225	352245.80	11246.08
com-dblp	120492	**120640**	120647.00	3082.43	120970	120977.20	3131.91
Amazon0302	130378	**128913**	128939.80	2606.64	130386	130422.20	2600.53
Amazon0312	180853	183113	183113.00	1.27	**180438**	181974.30	2372.35
Amazon0505	183114	185310	185310.00	1.32	**182160**	183670.60	1527.61
Amazon0601	179964	182279	182279.00	1.405	**179584**	181282.40	2197.99
Average		126761.70	126772.19		**126188.40**	126674.84	

of CMSA, called Adapt-CMSA, that adjusts its parameters on the fly in order to be able to solve problem instances of very different sizes without the need of re-tuning. Experiments were performed in the context of the minimum positive influence dominating set (MPIDS) problem.

Based on the obtained results, we can say that Adapt-CMSA has several advantages over standard CMSA in the context of the MPIDS problem. First, Adapt-CMSA does not need to be specifically tuned for subsets of the considered benchmark set. After one single tuning run, Adapt-CMSA works very well for the whole benchmark set, which contains instances of very different sizes. Second, Adapt-CMSA clearly outperforms standard CMSA in the context of large networks for which even a specialised tuning does not enable CMSA to compete with Adapt-CMSA.

In future work, we aim at confirming the findings of this paper in the context of other hard combinatorial optimisation problems. We believe that making CMSA self-adaptive is a significant step towards improving the wide applicability of this approach.

Acknowledgements We acknowledge administrative and technical support by the Spanish National Research Council (CSIC).

Author Contributions Methodology, CB; programming, MAA; writing—original draft, MAA, CB; writing—review and editing, CB; data curation, ALS; supervision, CB; validation, ALS. All the authors have read and agreed to the published version of the manuscript.

Funding This work was supported by grant PID2019-104156GB-I00 funded by MCIN/AEI/10.13039/501100011033. M.A. Akbay was funded by the Ministry of National Education, Turkey (Scholarship program: YLYS-2019).

Availability of Data and Materials Our detailed results on scale-free networks can be obtained from the corresponding author on demand. The same holds for all problem instances used in this work.

Declarations

Conflict of Interest The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Wolsey, L.A., Nemhauser, G.L.: Integer and Combinatorial Optimization, vol. 55. Wiley, Hoboken (1999)
2. Gendreau, M., Potvin, J.-Y. (eds.): Handbook of Metaheuristics, pp. 57–97. Springer, Switzerland (2019)
3. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. **35**, 268–308 (2003)
4. Talbi, E. (ed.): Hybrid Metaheuristics. Studies in Computational Intelligence, vol. 434. Springer, Berlin (2013)
5. Blum, C., Raidl, G.R.: Hybrid Metaheuristics—Powerful Tools for Optimization. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, Switzerland (2016)
6. Boschetti, M.A., Maniezzo, V., Roffilli, M., Bolufé Röhler, A.: Matheuristics: Optimization, simulation and control. In: Blesa, M.J., Blum, C., Di Gaspero, L., Roli, A., Sampels, M., Schaerf, A. (eds.) Proceedings of HM 2009—6th International Workshop on Hybrid Metaheuristics. Lecture Notes in Computer Science, vol. 5818, pp. 171–177. Springer, Berlin (2009)
7. Pisinger, D., Ropke, S.: In: Gendreau, M., Potvin, J.-Y. (eds.) Large Neighborhood Search, pp. 99–127. Springer, New York (2019)
8. Ahuja, R.K., Orlin, J.B., Sharma, D.: Very large-scale neighborhood search. Int. Trans. Oper. Res. **7**(4–5), 301–317 (2000)
9. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: Record breaking optimization results using the ruin and recreate principle. J. Comput. Phys. **159**(2), 139–171 (2000)
10. Demir, E., Bektaş, T., Laporte, G.: An adaptive large neighborhood search heuristic for the pollution-routing problem. Eur. J. Oper. Res. **223**(2), 346–359 (2012)
11. Schmid, V.: Hybrid large neighborhood search for the bus rapid transit route design problem. Eur. J. Oper. Res. **238**(2), 427–437 (2014)
12. Eskandarpour, M., Dejax, P., Péton, O.: A large neighborhood search heuristic for supply chain network design. Comput. Oper. Res. **80**, 23–37 (2017)
13. Fischetti, M., Lodi, A.: Local branching. Math. Program. **98**(1), 23–47 (2003)
14. Caserta, M., Voß, S.: A corridor method based hybrid algorithm for redundancy allocation. J. Heuristics **22**(4), 405–429 (2016)
15. Lalla-Ruiz, E., Voß, S.: POPMUSIC as a matheuristic for the berth allocation problem. Ann. Math. Artif. Intell. **76**(1–2), 173–189 (2016)
16. Blum, C., Pinacho Davidson, P., López-Ibáñez, M., Lozano, J.A.: Construct, merge, solve & adapt: a new general algorithm for combinatorial optimization. Comput. Oper. Res. **68**, 75–88 (2016)
17. Lewis, R., Thiruvady, D., Morgan, K.: Finding happiness: an analysis of the maximum happy vertices problem. Comput. Oper. Res. **103**, 265–276 (2019)
18. Arora, D., Maini, P., Pinacho-Davidson, P., Blum, C.: Route planning for cooperative air-ground robots with fuel constraints: an approach based on CMSA. In: Proceedings of GECCO 2019–Genetic and Evolutionary Computation Conference, pp. 207–214. Association for Computing Machinery, New York (2019)
19. Dupin, N., Talbi, E.-G.: Matheuristics to optimize refueling and maintenance planning of nuclear power plants. J. Heuristics **27**(1), 63–105 (2021)
20. Ferrer, J., Chicano, F., Ortega-Toro, J.A.: CMSA algorithm for solving the prioritized pairwise test data generation problem in software product lines. J. Heuristics **27**(1), 229–249 (2021)
21. Tatsis, V.A., Parsopoulos, K.E.: Dynamic parameter adaptation in metaheuristics using gradient approximation and line search. Appl. Soft Comput. **74**, 368–384 (2019)
22. Akbay, M.A., Blum, C.: Application of CMSA to the minimum positive influence dominating set problem. In: Artificial Intelligence Research and Development, pp. 17–26. IOS Press, Amsterdam (2021)
23. Fournier, A.K., Hall, E., Ricke, P., Storey, B.: Alcohol and the social network: Online social networking sites and college students' perceived drinking norms. Psychol. Pop. Media Cult. **2**(2), 86 (2013)
24. Long, C., Wong, R.C.-W.: Minimizing seed set for viral marketing. In: 2011 IEEE 11th International Conference on Data Mining, pp. 427–436. IEEE Press (2011)
25. Günneç, D., Raghavan, S., Zhang, R.: Least-cost influence maximization on social networks. INFORMS J. Comput. **32**(2), 289–302 (2020)
26. Wang, G.: Domination problems in social networks. PhD thesis, University of Southern Queensland (2014)
27. Rad, A.A., Benyoucef, M.: Towards detecting influential users in social networks. In: International Conference on E-Technologies, pp. 227–240. Springer (2011)
28. Wang, F., Camacho, E., Xu, K.: Positive influence dominating set in online social networks. In: International Conference on Combinatorial Optimization and Applications, pp. 313–321. Springer (2009)

29. Wang, F., Du, H., Camacho, E., Xu, K., Lee, W., Shi, Y., Shan, S.: On positive influence dominating sets in social networks. Theor. Comput. Sci. **412**(3), 265–269 (2011)
30. Raei, H., Yazdani, N., Asadpour, M.: A new algorithm for positive influence dominating set in social networks. In: 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, pp. 253–257. IEEE (2012)
31. Fei, M., Weidong, C.: An improved algorithm for finding minimum positive influence dominating sets in social networks. J. South China Norm. Univ. **48**(3), 59–63 (2016)
32. Pan, J., Bu, T.-M.: A fast greedy algorithm for finding minimum positive influence dominating sets in social networks. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 360–364. IEEE (2019)
33. Chen, W., Zhong, H., Wu, L., Du, D.-Z.: A general greedy approximation algorithm for finding minimum positive influence dominating sets in social networks. J. Comb. Optim. 1–20 (2021)
34. Bouamama, S., Blum, C.: An improved greedy heuristic for the minimum positive influence dominating set problem in social networks. Algorithms **14**(3), 79 (2021)
35. Lin, G., Guan, J., Feng, H.: An ilp based memetic algorithm for finding minimum positive influence dominating sets in social networks. Phys. A **500**, 199–209 (2018)
36. Lin, G., Luo, J., Xu, H., Xu, M.: A hybrid swarm intelligence-based algorithm for finding minimum positive influence dominating sets. In: Liu, Y., Wang, L., Zhao, L., Yu, Z. (eds.) Proceedings of ICNC-FSKD 2019–Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery, pp. 506–511. Springer, Cham (2020)
37. Shan, Y., Kang, Q., Xiao, R., Chen, Y., Kang, Y.: An iterated carousel greedy algorithm for finding minimum positive influence dominating sets in social networks. IEEE Trans. Comput. Soc. Syst. (2021). (**in press**)
38. Serrano, A.L., Nurcahyadi, T., Bouamama, S., Blum, C.: Negative Learning Ant Colony Optimization for the Minimum Positive Influence Dominating Set Problem, pp. 1974–1977. Association for Computing Machinery, New York (2021)
39. Csardi, G., Nepusz, T., et al.: The igraph software package for complex network research. Int. J. Complex Syst. **1695**(5), 1–9 (2006)
40. Cho, Y.S., Kim, J.S., Park, J., Kahng, B., Kim, D.: Percolation transitions in scale-free networks under the achlioptas process. Phys. Rev. Lett. **103**(13), 135702 (2009)
41. Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999)
42. Fronczak, P.: In: Alhajj, R., Rokne, J. (eds.) Scale-Free Nature of Social Networks, pp. 2300–2309. Springer, New York (2018)
43. López-Ibánez, M., et al.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)