Artificial Intelligence Research and Development
M. Villaret et al. (Eds.)

doi:10.3233/FAIA210112

Application of CMSA to the Minimum Positive Influence Dominating Set Problem

Mehmet Anıl AKBAY [a,1] and Christian BLUM [a]
[a] *Artificial Intelligence Research Institute (IIIA-CSIC), Campus UAB, Bellaterra, Spain*

Abstract. Construct, Merge, Solve & Adapt (CMSA) is a recently developed algorithm for solving combinatorial optimization problems. It combines heuristic elements, such as the probabilistic generation of solutions, with an exact solver that is iteratively applied to sub-instances of the tackled problem instance. In this paper, we present the application of CMSA to an NP-hard problem from the family of dominating set problems in undirected graphs. More specifically, the application in this paper concerns the minimum positive influence dominating set problem, which has applications in social networks. The obtained results show that CMSA outperforms the current state-of-the-art metaheuristics from the literature. Moreover, when instances of small and medium size are concerned CMSA finds many of the optimal solutions provided by CPLEX, while it clearly outperforms CPLEX in the context of the four largest, respectively more complicated, problem instances.

Keywords. Construct, merge, solve & adapt, minimum positive influence dominating set, hybrid metaheuristics

1. Introduction

When faced with a hard combinatorial optimization problem, the related literature generally offers both exact and approximate techniques for solving the problem. Exact techniques guarantee to find an optimal solution to a given problem instance in bounded computation time. Therefore, instances up to a problem-specific size and/or difficulty are usually solved by using an exact technique. Hereby, the term exact technique might refer to a specialized algorithm or to a general-purpose tool such as, for example, an integer linear programming (ILP) solver. Examples for ILP solvers include CPLEX and Gurobi, just to name the currently most powerful ones. The computation time required by an exact technique generally starts to explode when reaching a problem-specific instance size and/or difficulty. When this happens researchers and practitioners usually resort to using approximate techniques for obtaining solutions to their problem. Examples range from simple greedy heuristics to more sophisticated metaheuristics [2]. In order to take profit from the valuable optimization expertise that has gone into the development of exact optimization tools such as CPLEX and Gurobi, in the last two decades some researchers have focused on the development of algorithms that allow to take profit from

[1] Corresponding Author; E-mail: makbay@iiia.csic.es

these tools even in the context of problem instances that are too large to be solved directly by them. Algorithms from this line of research are called *hybrid metaheuristics* or *matheuristics* [5]. Prominent examples include algorithms such as large neighborhood search (LNS) [13] and construct, merge, solve & adapt (CMSA) [1].

In this paper we demonstrate the application of CMSA to the so-called minimum positive influence dominating set (MPIDS) problem [16,17]. The MPIDS problem is an NP-hard combinatorial optimization problem with applications in social networks. Each vertex in such a network represents an individual—that is, a person—and edges indicate relationships, respectively interactions, between those individuals. The background of the MPIDS problem is that information propagated in social networks can have a significant, either positive or negative, impact on the respective parts of the society. From social norms theory it is known that the behavior of individuals can be affected by the perception of others' thoughts and behaviors [6]. This makes it possible to exploit the relationships among people in social networks in order to obtain great benefits for both the economy and society. The aim of the MPIDS problem is to identify a small subset of influential individuals (or key individuals) in order to accelerate the spread of positive influence in a social network [10,7]. Alternative applications of the MPIDS problem can be found in e-learning software [18], online business [14], drinking, smoking, and other drug-related problems [16].

The remaining part of this paper is organized as follows. A technical description of the MPIDS problem, together with a standard ILP model, is provided in Section 2. The application of CMSA to the MPIDS problem is described in Section 3. Finally, an experimental evaluation, including a comparison to the state-of-the-art from the literature, can be found in Section 4, and conclusions as well as an outlook to future lines of research are provided in Section 5.

2. MPIDS problem

In technical terms, the MPIDS problem can be described as follows. Given a simple that does not contain any loops and parallel edges, connected, undirected graph $G = (V, E)$, the problem requires to find a subset S^* of V of minimum cardinality such that the following two conditions are fulfilled:

1. S^* is a dominating set of G. Remember that a subset $S \subseteq V$ of the vertices of an undirected graph G is called a dominating set, if and only if each vertex $v \in V$ forms either part of S or has at least one neighbor that forms part of S.
2. At least half of the neighbors of each vertex $v \in V$ form part of S^*.

Most of the research efforts concerning the MPIDS problem have been focused on greedy heuristics [17,15,4,12,3]. Moreover, a swarm intelligence based algorithm [9] and an ILP-based memetic algorithm [8] were presented in the literature. The latter one is currently state-of-the-art for the MPIDS problem.

Note that the MPIDS problem can easily be stated in terms of an ILP. The model is based on a binary variable x_i for each vertex $v_i \in V$.

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

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

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

Hereby, $N(v_i)$ is the neighborhood of v_i in input graph G, and $deg(v_i)$ is the degree of vertex v_i, where $deg(v_i) := |N(v_i)|$. Equation (2) ensures that a feasible solution contains at least half of the neighbors of each vertex $v_i \in V$. In the context of the CMSA algorithm outlined in the next section, the objective function value $f(S)$ of a feasible solution $S \subseteq V$ is $f(S) := |S|$. Note that $S := V$ is a trivial solution to the problem.

3. The CMSA Algorithm

The general structure of the CMSA developed for the MPIDS problem is presented in Algorithm 1. The algorithm starts by taking an instance represented by a simple, connected, undirected graph $G = (V,E)$ as input. $S \subseteq V$ denotes a feasible MPIDS solution of G. At first, the best-so-far solution S_{bsf} is initialized to the trivial solution V. Then, two vector data structures, called $age_0[]$ and $age_1[]$, are initialized to value -1 for all $v \in V$; see line 4. Note that values $age_0[v]$ and $age_1[v]$ may range between -1 and a fixed positive integer value called $\text{age}_{\max}$. Hereby, $\text{age}_{\max}$ is one of CMSA's important parameters. These data structures are modified in two parts of the algorithm, namely (1) on the basis of the generated solutions and (2) on the basis of solving the so-called sub-instance. This is explained in detail below. After the initialization of data structures $age_0[]$ and $age_1[]$, a pre-processing procedure from [3] is applied to determine the set S_{par} of vertices that must form part of an optimal solution; see line 5. At each iteration, the algorithm probabilistically generates n_a solutions by applying function ProbablisticSolutionGeneration(S_{par}) in line 8. In order to probabilistically generate a solution S, a recent greedy algorithm from [3] is applied in a probabilistic way. This is also explained in Section 3.1. Afterwards, a sub-instance is generated on the basis of the current values in data structures $age_0[]$ and $age_1[]$.For a detailed explanation of the data structures used for defining a sub-instance, see Section 3.2. This sub-instance is then solved by CPLEX with a CPU time limit of t_{ILP} seconds by applying function SolveSubinstance($age_0[], age_1[], t_{\text{ILP}}$) ; see line 14. The result is a solution S_{opt}, which is a solution to both the sub-instance and the original problem instance. Note that t_{ILP} seconds may, or may not, be enough time for CPLEX to solve the sub-instance to optimality. In case t_{ILP} is not enough time, S_{opt} is a sub-optimal solution to the sub-instance. Next, the best-of-far solution S_{bsf} is updated with S_{opt} in case $f(S_{opt}) < f(S_{bsf})$; see line 15. Finally, the values of data structures $age_0[]$ and $age_1[]$ are modified on the basis of solution S_{opt} as shown in Section 3.3 in detail. The algorithm stops once the CPU time limit is reached. In the following, the remaining parts of the algorithm are outlined in more detail.

3.1. Probabilistic construction of solutions

Function ProbablisticSolutionGeneration(S_{par}) generates a valid solution S as follows. First, S is initialized to S_{par}. Note that, by initializing all solutions to be constructed by

Algorithm 1 CMSA for the MPIDS problem

input: a problem instance $G = (V, E)$
parameters: n_a, d_{rate}, l_{size}, $\text{age}_{\max}$, and t_{ILP}
$S_{bsf} := V$
$age_0[v] := -1$ and $age_1[v] := -1$ for all $v \in V$
$S_{par} :=$ PreProcessing(G)
while CPU time limit not reached **do**
  for $k := 1, \ldots, n_a$ **do**
    $S :=$ ProbablisticSolutionGeneration(S_{par})
    for all $v \in V$ **do**
      if $v \in S$ and $age_1[v] = -1$ **then** $age_1[v] := 0$
      if $v \notin S$ and $age_0[v] = -1$ **then** $age_0[v] := 0$
    end for
  end for
  $S_{opt} :=$ SolveSubinstance$(age_0[], age_1[], t_{\text{ILP}})$
  if $f(S_{opt}) < f(S_{bsf})$ **then** $S_{bsf} := S_{opt}$
  Adapt$(age_0[], age_1[], S_{bsf}, \text{age}_{\max})$
end while
return: S_{bsf}, the best solution found by the algorithm

S_{par}, the algorithm's performance is enhanced because the construction of solutions is accelerated. After this initialization, the set U of uncovered vertices with respect to S is determined. In this context, note 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 neighbors form part of S. In the opposite case, v is defined as *uncovered*. The following steps are then repeated until no uncovered vertices are left:

1. A vertex $v \in U$ with the smallest neighborhood size ($deg(v)$) is chosen. In other words, a vertex $v \in U$ is chosen such that $deg(v) \leq deg(v')$ for all $v' \in U$.
2. Afterward, vertices are iteratively chosen from $N(v) \setminus S$ and added to S until v is covered. The minimum number of adjacent vertices ($h_S(v)$) that need to be chosen and added to S is calculated using the following equation: $h_S(v) := \lceil \frac{deg(v)}{2} \rceil - |N_S(v)|$. Here, $N_S(v)$ refers to the set of neighbors of v that form already part of solution S. In contrast to the original greedy algorithm from [3], a vertex $v_i \in N(v) \setminus S$ may either be selected in a deterministic or in a probabilistic way. For this, we utilize two important parameters, namely the determinism rate d_{rate} and the candidate list size l_{size}. At first, a candidate list L is created. This list includes all the vertices $v' \in N(v) \setminus S$. Each vertex v' in L is characterized by its *cover degree*, 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 $r \leq d_{\text{rate}}$, 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. All vertices in the restricted candidate list have an equal probability $\frac{1}{l_{\text{size}}}$ of being selected.
3. The set U of uncovered vertices is re-computed.

3.2. Definition and solution of the sub-instance

Before describing the modification of data structures $age_0[]$ and $age_1[]$, we first explain how the values in these data structures are used for defining the sub-instance, which is obtained as an ILP model with additional restrictions. In other words, the sub-instance is obtained by adding additional constraints to the ILP model from Section 2. This ILP model is obtained as follows. First, the values $age_0[]$ and $age_1[]$ are used for splitting the set of vertices into three disjoint subsets: $V_{in} \subseteq V$ is the set of vertices that are forced to form part of any solution of the sub-instance. $V_{out} \subseteq V$ is the set of vertices that are excluded from any solution to the sub-instance. Finally, $V_{open} \subseteq V$ is the set of vertices that may, or may not, form part of a solution to the sub-instance.

- V_{in} contains all vertices $v \in V$ with $age_0[v_i] = -1$ and $age_1[v_i] \geq 0$.
- V_{out} contains all vertices $v \in V$ with $age_0[v_i] \geq 0$ and $age_1[v_i] = -1$.
- V_{open} contains all remaining vertices.

The corresponding ILP model is obtained by adding a constraint $x_i = 1$ for all $v_i \in V_{in}$, and a constraint $x_i = 0$ for all $v_i \in V_{out}$. After generating the restricted ILP which corresponds to the sub-instance, CPLEX is applied to the restricted ILP with a computation time limit of t_{ILP} seconds, resulting in a solution S_{opt}. Note that the more restricted a sub-instance is, the easier it is for CPLEX to derive an optimal solution to the sub-instance.

3.3. Modification of the data structures

As mentioned above, data structures $age_0[]$ and $age_1[]$ are modified (1) after the construction of a solution S (see lines 9-12 of Algorithm 1) and (2) after solving the sub-instance (line 16). Both cases are explained below.

After the construction of a solution S in line 8 of Algorithm 1, the following modifications are performed for each $v \in V$:

- If $v \in S$ and $age_1[v] = -1$, then $age_1[v] := 0$. This means that if (1) v forms part of S and if (2) v is currently excluded from forming part of solutions to the sub-instance (due to $age_1[v] = -1$), then $age_1[v]$ is set to zero. This means that v can now be considered for the inclusion in solutions to the sub-instance.
- If, otherwise, $v \notin S$ and $age_0[v] = -1$, then $age_0[v] := 0$ This means that if (1) v does not form part of S and if (2) v is currently not excluded to form part of solutions to the sub-instance, it may now be considered for exclusion.

Next we describe the modification of the data structures after solving the current sub-instance, that is, after generating a solution S_{opt} in the current iteration. This modification is done in function $\mathsf{Adapt}(age_0[], age_1[], S_{bsf}, \text{age}_{\max})$ (see line 16 of Algorithm 1). The working of this function is pseudo-coded in Algorithm 2. If a vertex $v \in V_{open}$ is not chosen by CPLEX for solution S_{opt}, two actions are performed: first, $age_0[v]$ is set to zero, and second, $age_1[v]$ is increased by one. In case $age_1[v]$ reaches age_{max}, $age_1[v]$ is set to its default value -1, which means that vertex v is excluded from the sub-instance in the next iteration. In other words, the vertex is transferred from set V_{open} to set V_{out} since it has not been selected by CPLEX to form part of S_{opt} during the last $\text{age}_{\max}$ iterations. Similarly, if a vertex $v \in V_{open}$ is frequently chosen by CPLEX for solution S_{opt}, it is transferred from set V_{open} to set V_{in} as described in line 9 of Algorithm 2.

Algorithm 2 Function Adapt($age_0[], age_1[], S_{bsf}, \text{age}_{\max}$)

input: sub-instance (C')
for all $v \in V$ **do**
  if $v \in V_{open}$ **then**
    if $v \notin S_{bsf}$ **then**
      $age_0[v] := 0$ and increase $age_1[v]$ by 1
      if $age_1[v] = \text{age}_{\max}$ **then** $age_1[v] := -1$
    else
      $age_1[v] := 0$ and increase $age_0[v]$ by 1
      if $age_0[v] = \text{age}_{\max}$ **then** $age_0[v] := -1$
    end if
  else
    if $age_0[v] \geq 0$ **then** $age_0[v] := 0$
    if $age_1[v] \geq 0$ **then** $age_1[v] := 0$
  end if
end for
output: C', updated sub-instance

4. Experimental Evaluation

In the following we compare CMSA with the following approaches: (1) application of CPLEX 12.10 in one-threaded mode (with a computation time limit of 2 hours per problem instance); (2) IGA-PIDS, which is the currently best greedy approach from [3]; (3) HSIA, a hybrid swarm intelligence based algorithm from [9]; and (4) ILPMA, an ILP-based memetic algorithm from [8]. The experiments concerning CMSA, CPLEX and IGA-PIDS were performed on a cluster of machines with Intel® Xeon® CPU 5670 CPUs with 12 cores of 2.933 GHz and a minimum of 32 GB RAM. As in the standalone application of CPLEX, sub-instances in CMSA were solved using CPLEX 12.10 in one-threaded mode. The results for HSIA and ILPMA were taken from the respective publications. Unfortunately, they were not available for all problem instances studied in this work. CMSA, CPLEX and IGA-PIDS were applied to 17 social networks that are usually used in the literature on the MIPDS problem. These networks are of small and medium size that contain between 34 and 36692 nodes and between 788 and 198050 edges. In addition, the three algorithms were applied to 10 larger social networks from the SNAP library that contain between 37700 and 1134890 nodes and between 2289003 and 3387388 edges (`https://snap.stanford.edu/data/`).

CMSA requires well-working values for n_a (number of solution constructions per iteration), d_{rate} (determinism rate), l_{size} (candidate list size), $\text{age}_{\max}$ (upper limit for the age-values), and t_{ILP} (time limit for CPLEX per iteration). The scientific tuning software irace [11] was used for tuning these parameters. More specifically, irace was used for generating two parameter settings for CMSA: one of the 17 small/medium sized instances, and another one for the 10 large networks. Networks `CA-AstroPh`, `Email-Enron` and `socfb-Brandeis99` were used for the first tuning experiment, and networks `Amazon0312` and `Amazon0601` were used for the second one. Finally, for each of the two applications of irace the budget was fixed to 2000 algorithm runs, each

Table 1. Parameter values obtained for CMSA by tuning with irace.

Networks	n_a	d_{rate}	l_{size}	$\text{age}_{\max}$	t_{ILP}
Small/medium size	1	0.1	8	4	16
Large size	1	0.9	8	1	13

one with a time limit of 600 CPU seconds. The considered parameter value domains were as follows: $n_a \in \{1,\dots,20\}$, $d_{\text{rate}} \in \{0.0, 0.1, 0.2, \dots, 0.8, 0.9\}$, $l_{\text{size}} \in \{3,\dots,10\}$, $\text{age}_{\max} \in \{1,\dots,10\}$ and $t_{\text{ILP}} \in \{1,\dots,30\}$ (in seconds). The obtained parameter value settings are shown in Table 1. It is worth noting that the value of parameter $\text{age}_{\max}$ decreases and the value of parameter d_{rate} increases as instance size grows to keep the sub-instance small enough to be solved by CPLEX.

While CPLEX and IGA-PIDS were applied exactly once to each of the 27 problem instances, CMSA was applied 10 times to each instance. A computation time limit of 2 hours was given to each CPLEX run, while a limit of 600 seconds was applied to each run of CMSA.

The results, in comparison to those of CPLEX (with a time limit of 2 hours per instance), IGA-PIDS [3], HSIA [9], ILPMA [8], and are shown in Table 2 (small/medium instances) and Table 3 (large instances). Both tables have the following structure. The first column contains the instance name. Columns with heading 'q' report on the quality of the best solutions found be the five approaches, and columns with heading 'avg' provide the average solution quality obtained. Furthermore, columns with heading '$\overline{t(s)}$' indicate the average computation times of ILPMA and CMSA to find the best solutions in each run. In addition, the column with heading '$\overline{t(s)}$' shows the computation time of the greedy approach IGA-PIDS. 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.

The following observations can be made. First, CPLEX performs very strongly for all small/medium size instances, and for five of the large instances. However, for the remaining five large instances it fails to find any other than the trivial solution within 2 hours of computation time. CMSA performs very comparably to CPLEX for the small/medium size instances. In one case (`CA-AstroPh`) CMSA finds a better solution than CPLEX. In five other instances it provides results that are marginally worse than those of CPLEX. Moreover, CMSA obtains the best average solution quality in the case of the 10 large problem instances. In particular, CMSA finds solutions much better than the trivial ones in the case of the five instances for which CPLEX fails. Moreover, CMSA significantly outperforms the current state-of-the-art approaches from the literature (HSIA and ILPMA).

Finally, the results also show that there is surely the need to find a way to improve CMSA for very large instances such as the last three instances of Table 3. The greedy approach IGA-PIDS outperforms CMSA in the case of these three instances.

5. Conclusions and Outlook

In this study, one of the recent hybrid metaheuristics (construct, solve, merge & adapt) was proposed to solve the minimum positive influence dominating set problem. Costruct,

Table 2.: Numerical results for small to medium size instances.

Network	CPLEX		IGA-PIDS		HSIA		ILPMA			CMSA		
	q	gap (%)	q	$\overline{t(s)}$	q	avg	q	avg	$\overline{t(s)}$	q	avg	$\overline{t(s)}$
Karate	**15**	0.00	**15**	0.0	n.a.	n.a.	**15**	**15.0**	0.03	**15**	15.00	0.0
Dolphins	**30**	0.00	31	0.0	n.a.	n.a.	**30**	**30.0**	0.13	**30**	30.00	0.011
Football	**63**	0.00	68	0.0	n.a.	n.a.	65	65.65	0.54	**63**	63.00	14.96
Jazz	**79**	0.00	81	0.0	n.a.	n.a.	n.a.	n.a.	n.a.	**79**	79.00	0.21
CA-AstroPh	6740	0.30	6953	0.031	6905	6906.6	6857	6865.45	300.41	**6736**	6739.90	539.18
CA-GrQc	**2587**	0.00	2607	0.0	2597	2598.4	2594	2596.05	45.07	**2587**	2587.00	3.17
CA-HepPh	**4718**	0.01	4817	0.015	4791	4792.4	4770	4773.85	157.43	**4718**	4718.10	183.01
CA-HepTh	**4471**	0.00	4544	0.0	4515	4516.2	4502	4506.25	107.93	**4471**	4471.00	10.89
CA-CondMat	**9584**	0.06	9748	0.015	9729	9734.0	9683	9689.6	506.37	9585	9585.60	460.42
Email-Enron	**11682**	0.00	11843	0.031	11865	11873.4	11814	11818.95	760.08	11683	11683.80	183.16
ncstrlwg2	**2994**	0.00	3010	0.015	3004	3005.4	3001	3002.85	65.69	**2994**	2994.00	14.57
actors-data	**3092**	0.24	3147	0.016	3143	3144.5	3130	3134.5	137.74	**3092**	3093.50	467.65
ego-facebook	**1973**	0.00	1975	0.078	1726[a]	1726.6[a]	1737[a]	1741.55[a]	56.91	**1973**	1973.00	59.16
socfb-Brandeis99	**1400**	1.41	1502	0.032	n.a.	n.a.	n.a.	n.a.	n.a.	1405	1408.20	377.66
socfb-nips-ego	**1398**	0.00	**1398**	0.016	n.a.	n.a.	n.a.	n.a.	n.a.	**1398**	1398.00	0.024
socfb-Mich67	**1329**	1.56	1427	0.015	n.a.	n.a.	n.a.	n.a.	n.a.	1336	1338.70	384.60
soc-gplus	**8244**	0.00	8289	0.031	n.a.	n.a.	n.a.	n.a.	n.a.	8253	8254.20	486.03
average	**3552.88**		3615.00							3554.00		

[a]: apparently, papers on HSIA and ILPMA have used a different ego-facebook instance

Table 3. Numerical results for large instances.

Network	CPLEX		IGA-PIDS		CMSA		
	q	gap (%)	q	$\overline{t(s)}$	q	avg	$\overline{t(s)}$
musae_git	**9752**	0.00	10386	1.98	10017	10027.20	586.25
loc-gowalla_edges	**67617**	0.07	69086	0.21	67931	67946.60	595.07
gemsec_facebook_artist	**15194**	1.20	16217	0.21	15405	15418.40	595.73
deezer_HR	54573	95.68	23738	0.14	**22713**	22747.80	592.43
com-youtube	**351281**	0.00	353387	2.98	352566	352587.00	593.50
com-dblp	**120492**	0.08	121940	0.31	120970	120993.60	599.43
Amazon0302	262111	97.50	134569	0.23	**130303**	130360.70	592.74
Amazon0312	400727	95.41	**180853**	0.67	183093	183103.90	282.99
Amazon0505	410236	95.19	**183114**	0.64	185230	185279.40	411.53
Amazon0601	403394	96.94	**179964**	0.66	182202	182231.30	252.42
average	209537.70		127325.40		**127043.00**		

Merge, Solve & Adapt is based on the probabilistic construction of solutions, which are used to extend the restricted sub-instance. This sub-instance is then solved by CPLEX at each iteration. Based on the results provided by CPLEX, the restricted sub-instance is modified and passed to the next iteration. Note that this procedure allows to make a beneficial use of high-performance ILP solvers such as CPLEX even in the context of problem instances that are too large for CPLEX to be applied directly.

Computational experiments were performed on 17 small/medium sized social networks and on ten larger benchmark instances from the SNAP database. The proposed approach was evaluated and compared to the state-of-the-art methods from the literature (including two metaheuristics and one greedy approach) and to the results obtained by the ILP solver CPLEX 12.10. The analysis of the results showed that construct, solve, merge & adapt outperforms the metaheuristics from the literature. Moreover, apart from the largest three problem instances, our approach outperforms the greedy approach. In comparison to CPLEX our approach obtains comparable results for small/medium sized instances, and starts to outperform CPLEX as the instance size grows.

On the negative side, we realized that the performance of our approach starts to degrade for the largest three problem instances. This means that, in these cases, even sub-instances are too large for CPLEX to be solved in the reduced amount of time given at each iteration. Therefore, one line of future research will deal with finding ways to overcome this problem, possibly by designing a self-adaptive version of the proposed algorithm. Note that parameter tuning might not be necessary anymore for such an algorithm version. In addition, introducing a learning mechanism that enables the CPLEX solutions to influence the probabilistic solution generation procedure is also considered as an important future research direction.

Acknowledgements

This work was funded by project CI-SUSTAIN, Spanish Ministry of Science and Innovation (PID2019-104156GB-I00). The corresponding author was funded by the Ministry of National Education, Turkey (Scholarship program: YLYS-2019).

References

[1] Christian Blum, Pedro Pinacho, Manuel López-Ibáñez, and José A Lozano. Construct, merge, solve & adapt: a new general algorithm for combinatorial optimization. *Computers & Operations Research*, 68:75–88, 2016.

[2] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[3] Salim Bouamama and Christian Blum. An improved greedy heuristic for the minimum positive influence dominating set problem in social networks. *Algorithms*, 14(3):79, 2021.

[4] MAI Fei and CHEN Weidong. An improved algorithm for finding minimum positive influence dominating sets in social networks. *Journal of South China Normal University*, 48(3):59–63, 2016.

[5] Martina Fischetti and Matteo Fischetti. *Matheuristics*, pages 121–153. Springer International Publishing, 2018.

[6] Angela K. Fournier, Erin Hall, Patricia Ricke, and Brittany Storey. Alcohol and the social network: Online social networking sites and college students' perceived drinking norms. *Psychology of Popular Media Culture*, 2(2):86, 2013.

[7] Dilek Günneç, Subramanian Raghavan, and Rui Zhang. Least-cost influence maximization on social networks. *INFORMS Journal on Computing*, 32(2):289–302, 2020.

[8] Geng Lin, Jian Guan, and Huibin Feng. An ilp based memetic algorithm for finding minimum positive influence dominating sets in social networks. *Physica A: Statistical Mechanics and its Applications*, 500:199–209, 2018.

[9] Geng Lin, Jinyan Luo, Haiping Xu, and Meiqin Xu. A hybrid swarm intelligence-based algorithm for finding minimum positive influence dominating sets. In Yong Liu, Lipo Wang, Liang Zhao, and Zhengtao Yu, editors, *Proceedings of ICNC-FSKD 2019 – Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery*, pages 506–511. Springer International Publishing, 2020.

[10] Cheng Long and Raymond Chi-Wing Wong. Minimizing seed set for viral marketing. In *2011 IEEE 11th International Conference on Data Mining*, pages 427–436. IEEE press, 2011.

[11] Manuel López-Ibánez et al. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43 – 58, 2016.

[12] Jiehui Pan and Tian-Ming Bu. 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)*, pages 360–364. IEEE, 2019.

[13] David Pisinger and Stefan Ropke. *Large Neighborhood Search*, pages 99–127. Springer International Publishing, 2019.

[14] Amir Afrasiabi Rad and Morad Benyoucef. Towards detecting influential users in social networks. In *International Conference on E-Technologies*, pages 227–240. Springer, 2011.

[15] Hassan Raei, Nasser Yazdani, and Masoud Asadpour. 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*, pages 253–257. IEEE, 2012.

[16] Feng Wang, Erika Camacho, and Kuai Xu. Positive influence dominating set in online social networks. In *International Conference on Combinatorial Optimization and Applications*, pages 313–321. Springer, 2009.

[17] Feng Wang, Hongwei Du, Erika Camacho, Kuai Xu, Wonjun Lee, Yan Shi, and Shan Shan. On positive influence dominating sets in social networks. *Theoretical Computer Science*, 412(3):265–269, 2011.

[18] Guangyuan Wang. *Domination problems in social networks*. PhD thesis, University of Southern Queensland, 2014.