A Construct Merge Solve and Adapt Heuristic for Least Cost Influence Maximization

Felix Odelga(✉), Enrico Iurlano, and Günther R. Raidl

Algorithms and Complexity Group, TU Wien, Austria
{felix.odelga, eiurlano, raidl}@{student, ac}.tuwien.ac.at

Abstract. We propose a variant of the Construct, Merge, Solve and Adapt (CMSA) metaheuristic for solving the Least Cost Influence Maximization Problem (LCIP). This problem models influence diffusion in a social network. It seeks to determine payments to a selection of vertices to ensure a certain minimum portion of all vertices is influenced, i.e., they adopt a given idea, while minimizing the total cost. Influenced individuals may subsequently influence their neighbors, potentially triggering a cascading diffusion process. We develop several construction heuristics and introduce a strategy that combines them to derive bounded payment intervals, thereby restricting the search space. Furthermore, we present a Mixed Integer Linear Programming (MILP) formulation, which is employed both as a standalone exact approach and as the embedded solver within the CMSA framework. Computational experiments on benchmark instances derived from real-world social graphs with 30,398 to 154,908 vertices demonstrate that the proposed CMSA approach consistently outperforms both the standalone MILP model and the construction heuristics.

1 Introduction

We study the *Least Cost Influence Maximization Problem* (LCIP) [8], which models the spread of influence in a social network with different incentive costs for each individual. The objective is, by employing incentive payments, to influence a specific share of the entire network while minimizing the total cost of incentive payments. This generalizes the *Target Set Selection Problem* (TSS) [9] by allowing partial incentives instead of binary activation decisions.

We propose a variant of the Construct, Merge, Solve & Adapt (CMSA) metaheuristic for heuristically solving the LCIP and show that it is effective when compared with Mixed Integer Linear Programs (MILPs) and heuristic construction methods. While MILP formulations can find optimal solutions for small instances, they scale poorly to larger networks. Construction heuristics scale better, but sacrifice solution quality. The CMSA metaheuristic tries to combine the best of both worlds by utilizing MILP solving iteratively on reduced search spaces induced by diverse solutions obtained from construction heuristics.

The primary challenge in applying CMSA to LCIP lies in defining suitable solution components for merging. Restricting payments to only take values already

This document is a preprint accepted for MIC 2026 — version dated May 7, 2026.

present in the solution pool can be overly restrictive and is difficult to encode in a MILP. We therefore investigate two merging strategies: restricting payments to vertices that receive nonzero incentives in constructed solutions, and bounding each vertex's payment by the minimum and maximum payment observed in the solution pool.

Another issue encountered is the lack of benchmark instances containing both vertex thresholds and edge influence weights. To address this, we generate instances based on real-world social network graphs using both a trivalency model [5] as well as uniform random weight assignments.

The main contributions of this paper are a CMSA metaheuristic for the LCIP with two merging strategies for defining solution components, and a set of new benchmark instances based on real-world social network graphs.

The next section formally defines the LCIP, while Section 3 reviews related work. Section 4 describes the proposed CMSA approach, including our MILP formulation, construction heuristics, and the solution merging strategies. Experimental results are presented in Section 5, and we conclude with a summary and outlook in Section 6.

2 The Least Cost Influence Maximization Problem

The LCIP was introduced in [8] and we follow the original definition. We are given a directed graph $G = (V, E)$ representing a social network, where each vertex $v \in V$ has a threshold $b_v > 0$, and each directed edge $(v, w) \in E$ has an associated influence weight $d_{vw} > 0$, which is asymmetric in general.

The intention is to spread some opinion in the network by providing incentive payments to some vertices for influencing others. A vertex can attain two states, either *inactive* or *active*; the latter has the interpretation that the respective individual has been influenced, i.e., adopted the opinion. A solution is represented by the incentive payment $p_v \geq 0$ for each vertex $v \in V$. Initially, all vertices are inactive. At each discrete time step, a vertex v becomes active if

$$p_v + \sum_{w \in A(v)} d_{wv} \geq b_v,$$

where $A(v)$ denotes the set of active neighbors of v. This influence-propagation process terminates after at most $|V|$ steps as vertices remain active once activated.

Given a so-called minimum penetration factor $\alpha \in (0, 1]$, the objective is to minimize the total payment $\sum_{v \in V} p_v$ such that at least $\alpha|V|$ vertices are active at termination.

The more general case in which a selection of vertices is already active at the beginning can be easily reduced to the above by removing the initially active vertices and readjusting neighboring incentive thresholds [8]. Moreover, we remark that the timestep at which incentives are payed can be fixed to the very first timestep; this never worsens the solution quality and reduces the search space [8].

3 Related Work

Influence maximization and diffusion processes in social networks have been studied extensively, motivated by applications such as viral marketing: one of the first studied problems in this area is the *Target Set Selection* (TSS) problem [9], where the goal is to activate all vertices by selecting a minimum-size initial vertex set under threshold-based propagation. Variants of the TSS have been theoretically studied on restricted graph classes, see e.g. [4] for trees and complete graphs, and in practice with alternative optimization objectives, such as influence maximization under budget constraints [13].

An important direction of research has been the change from the binary choice of including a vertex in the initial set towards partial incentives, which provide a more realistic model of influence. Demaine et al. [5] examine both integral and fractional models of influence propagation, presenting several greedy heuristics for selecting vertices under these models. Further heuristic approaches include for example simple path-based constructions [6].

The specific problem addressed here is based on work by Günneç et al. [8]. They proved NP-completeness via a reduction from *Independent Set* and analyzed the problem under restrictions such as uniform influence weights, full penetration, i.e. $\alpha = 1$, and tree-structured graphs. For the general case, they proposed a time-indexed MILP formulation. For trees and uniform influence, a totally unimodular formulation and a branch-and-cut approach for general graphs were presented in [7]. Melo et al. [12] derived dual bounds exploiting acyclic influence propagation.

Construct, Merge, Solve and Adapt (CMSA) is a prominent metaheuristic framework for combinatorial optimization introduced in [2]. It iteratively *constructs* diversified heuristic solutions, defines a reduced search space based on components appearing in the heuristic solutions (*merging step*) and applies a typically exact technique like a MILP solver to *solve* the obtained subproblem. Each solution component is associated with an age that increases every iteration but is reset to zero when the component is used in the incumbent solution. Components that remain unused and reach a certain maximum age are removed in the *adapt step* to keep the component pool for merging and thus the further induced subproblems reasonably sized.

4 Proposed CMSA Approach

We start by proposing a new MILP formulation for the LCIP and then detail our CMSA approach including the solution construction methods and the way how the merging step is realized.

4.1 MILP Formulation

Günneç et al. [8] suggested a straight-forward MILP formulation in which they consider time steps $t = 1, \ldots, T$ and set $T = |V|$ to guarantee a sufficiently high

duration for all propagation scenarios. This formulation clearly does not scale well enough to graphs with thousands of vertices.

To solve this issue, the following alternative view of the problem is helpful: Instead of considering the propagation of influence over time, we search for a directed subgraph $G' = (V', E')$ of G that models the influence propagation. This subgraph must be acyclic, since it is not possible for a vertex to influence itself, i.e. the activation of a vertex must not depend on other vertices that are activated by it [3]. A topological ordering of G' can be seen as the order in which vertices are activated. This leads to the following MILP, which uses Miller-Tucker-Zemlin (MTZ) inequalities based on [11] to ensure the acyclic property of the propagation subgraph:

$$\min \quad \sum_{v \in V} p_v \tag{1}$$

$$\text{s.t.} \quad \sum_{v \in V} y_v \geq \alpha |V| \tag{2}$$

$$x_{vw} \leq y_v \qquad \forall (v,w) \in E \tag{3}$$

$$b_v y_v \leq p_v + \sum_{(v,w) \in E} d_{wv} x_{wv} \qquad \forall v \in V \tag{4}$$

$$u_v - u_w + |V| x_{vw} \leq |V| - 1 \qquad \forall (v,w) \in E \tag{5}$$

$$x_{vw} + x_{wv} \leq 1 \qquad \forall (v,w) \in E \tag{6}$$

$$x_{vw} \in \{0,1\} \qquad \forall (v,w) \in E \tag{7}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \tag{8}$$

$$p_v \geq 0 \qquad \forall v \in V \tag{9}$$

$$0 \leq u_v \leq |V| - 1 \qquad \forall v \in V \tag{10}$$

Binary variables y_v indicate the activation status of vertex v and binary variables x_{vw} are introduced for each (directed) edge $(v,w) \in E$ to select it for the influence propagation subgraph, i.e. $x_{vw} = 1$ iff $(v,w) \in E'$. Variables p_v indicate the payment to vertex v, and variables u_v are used to ensure the acyclicity of the subgraph. The objective (1) minimizes the total payment, while constraint (2) ensures that enough vertices are activated. Inequalities (3) allow only active vertices to be the source of influence. The propagation of influence is modelled in inequalities (4) by allowing a vertex to be active only if there is enough incoming influence. To ensure the acyclicity of the subgraph, each vertex v has an associated potential u_v that grows along the edges of the subgraph. This is modelled in constraint (5): If no edge is present between v and w ($x_{vw} = 0$), the values of u_v and u_w are irrelevant. If an edge is present ($x_{vw} = 1$), the inequality can be transformed to $u_v + 1 \leq u_w$, meaning the potential of w must be at least one unit larger than the potential of v; this forbids any cycles for the subgraph. Inequalities (6) strengthen the formulation by directly avoiding trivial cycles.

4.2 Construction Heuristics

As a baseline for comparison, a randomized solution construction method is used. At each step, a random vertex is chosen and paid enough to activate it, after which its influence is propagated. This is repeated until the desired penetration is achieved.

In order to get more meaningful heuristic solutions, a greedy algorithm based on properties of vertices, e.g., their degree, can be used. In each iteration, the vertex with the highest score according to some evaluation function f (ties broken randomly) is paid the required amount to activate it, after which its influence is spread. This is repeated until the desired fraction α of vertices is active.

To randomize this greedy construction heuristic, instead of picking the vertex v with highest score $f(v)$, a restricted candidate list is constructed as the set $R(f, r) = \{v \in V \setminus A \mid f(v) \geq r \cdot \max_{v \in V} f(v)\}$, where A is the set of already activated vertices and $r \in [0, 1]$ is a parameter. Some vertex is then picked uniformly at random from this list. Note that for $r = 1$ we obtain a deterministic construction heuristic (up to tiebreaking), while for $r = 0$ the algorithm behaves like the random construction introduced before.

For example, consider the following randomized greedy construction method based on the *DiscountHeuristic* from [5]: For each vertex v we take the number of inactive neighbors as score $f(v)$, which is then used to construct the respective set $R(f, r)$, from which a vertex is picked at random. The selected vertex is then activated and has its influence propagated, changing $f(v)$ for some vertices. The pseudocode for this heuristic can be seen in Algorithm 1.

Another heuristic called *ImpactHeuristic* generalizes the *DiscountHeuristic* to also take weights into account by using the scoring function $f(v) \coloneqq \sum_{w \in V \setminus A} d_{vw}$ (assuming $d_{xy} = 0$ when vertices x, y are not connected). In the case that all weights are 1, this behaves the same as the *DiscountHeuristic*, but prioritizes vertices with higher-weight edges when edge weights differ.

Algorithm 1 Greedy Heuristic

input: graph G, penetration factor $\alpha \in [0, 1]$ and randomization parameter $r \in [0, 1]$
$A \coloneqq \emptyset$
$p_v := 0,\ v \in V$ ▷ payments
while $|A| < \alpha |V|$ **do**
    $R \coloneqq \{v \in V \setminus A \mid f(v) \geq \max_{v \in V \setminus A} f(v) \cdot r\}$
    $v \coloneqq$ random vertex from R
    ActivateVertex(v)
    PropagateInfluence()
end while
return $p_v,\ v \in V$

4.3 Solution Merging

Next we present two different approaches for realizing the solution merging in the CMSA framework.

We call our first approach *non-zero payments*. Given a pool of so-far constructed solutions S, we partition the vertex set into two subsets: those with a non-zero payment in any solution and those whose payment is always zero (either activated solely through propagation or not at all). In the merging subproblem, the former vertex set still has unrestricted decision variables $p_v \geq 0$ for the payment, while the latter have their payments fixed to zero.

More formally, the *merging step* takes a solution set S and determines subset

$$P = \{v \in V \mid p_v^s > 0 \text{ for some } s \in S\},$$

where p_v^s is the payment to vertex v in solution s. For solving the merging subproblem, the MILP is then augmented by

$$p_v = 0 \quad \forall v \in V \setminus P. \tag{11}$$

Thus, within the CMSA we consider P to be our pool of current solution components and adapt it over the iterations.

To remove old unused solution components in the *adaptation step*, every vertex $v \in P$ has an associated age $a_v \geq 0$. When a vertex enters P, a_c is set to 0. If the vertex is not *used* in the solution found by the solver in the *solving* step, i.e., it receives no payment, a_c is incremented by one, and if a_c reaches a defined threshold $\text{age}_{\max}$, the component is removed from set P.

Early experiments showed that the performance of this approach depends strongly on the nature of the problem instance, for example if the influence weights are too low in comparison with activation costs and densities, there might not be any vertices with non-zero payments, in which case the merging degenerates into applying the MILP to the original problem.

Our second approach, called *payment range*, is a generalization of the first one. For each vertex $v \in V$ a set $P_v = \{s_v \mid s \in S\}$ of payments occurring in all solutions in S is maintained, and the allowed payment for the vertex in the merging subproblem is restricted to the range

$$\min P_v \leq p_v \leq \max P_v.$$

In addition to constraining the same vertices as in the first approach, this also restricts the merging subproblem by fixing the payment of vertices that received the same payment in all solutions in S to this amount and in general reduces the domains of payments for other vertices significantly.

For the *adaptation step*, each entry in each P_v has associated an individual age that is incremented by one if it is not *used*, and values reaching age $\text{age}_{\max}$ are removed from P_v. To determine whether a component is considered *used*, we construct an interval around the solution returned by the solver. Any component within this interval is counted as *used*. The interval extends by 25% of the total range $[\min P_v, \max P_v]$ in both directions around the solvers solution.

5 Computational Experiments

We first describe the benchmark instances we use for the experimental evaluation, then give some information on the software implementation and used hardware before diving into obtained results.

5.1 Benchmark Sets

As experiments on artificially generated graphs can be quite misleading, we opted for using benchmark instances from real-world social networks. Specifically, we selected graphs from Netzschleuder [14] and Konect [10] collections representing social interaction such as friendship status or messaging activity. Table 1 provides an overview of the used instances. Note that the graphs have about 30 000 to 155 000 vertices and up to over two million edges. The average degree $\overline{\deg}$ varies between about four and 47.

As edge and vertex weights for the social relations were not given for these graphs, we added them using two different random sampling strategies:

1. **Uniform**: edge weights are chosen uniformly at random from a given interval, i.e. $[1.0, 3.0]$,
2. **Trivalency**: following [5], edge weights are chosen uniformly at random from a given set of three values, i.e. $\{1.2, 1.6, 3.0\}$.

Concerning the vertex weights, the average of the sum of incoming edge weights to each vertex is calculated, which is then multiplied with a factor randomly chosen from $\{0.3, 0.5, 0.8\}$ mimicking easy, normal, or hard to influence vertices, respectively. The in this way augmented graphs are available online[1].

[1] `https://www.ac.tuwien.ac.at/research/problem-instances/#LCIP`

Table 1: Benchmark instances used in the evaluation.

Name	$\lvert V\rvert$	$\lvert E\rvert$	$\overline{\deg}$
`munmun_digg`	30 398	87 627	5.76
`enron`	36 692	367 662	20.04
`wiki_talk_eu`	40 993	58 120	2.84
`deezer_RO`	41 773	125 826	6.02
`deezer_HU`	47 538	222 887	9.38
`artist`	50 515	819 306	32.44
`deezer_HR`	54 573	498 202	18.26
`brightkite`	58 228	214 078	7.35
`marker_cafe`	69 413	1 644 849	47.39
`slashdot-08-11`	77 360	905 468	23.41
`slashdot-09-02`	82 168	948 464	23.09
`livemocha`	104 103	2 193 083	42.13
`douban`	154 908	327 162	4.22

Graphs were grouped into smaller and larger instances according to the number of vertices, with the small instance group including graphs of up to 60 000 vertices.

From each graph, four benchmark instances were derived by sampling edge and vertex weights twice with each weight generation method. In the following we denote these individual instances by the graph name and the suffixes `tri` and `uni` for the trivalency and uniform weight generation methods, respectively, and the numbers one and two for the two different samplings.

The required penetration factor α was set to 0.5 for all experiments.

5.2 Implementation and Tuning

The MILP models were implemented in Julia 1.11.7 using the JuMP library with Gurobi 11.0.2[2] as backend. All experiments were run on an Intel Xeon E5–2640 (2.4 GHz, 10-core) with up to 38 GB of available memory. To ensure fairness between different settings where different times are spent in generating solutions and running the MILP solver, Gurobi was restricted to a single thread and also the heuristic solution generation was not parallelized.

The irace library[3] was used to tune the parameters for the CMSA separately for the two instance groups, i.e., the construction heuristic to use including its diversification parameter, the number of solutions to generate per iteration, the time limit for the MILP solver per iteration, and the solution merging strategy. A tuning budget of 300 experiments was used. The overall time limit for each CMSA run as well as for solving the original MILP was set to two hours.

5.3 Results on Small Instances

From the smaller instances, `munmun_digg`, `wiki_talk_eu`, and `deezer_HR` were used as tuning instances. The resulting configuration employed the *DiscountHeuristic* with parameter 0.75, generating 20 solutions per iteration, using a solver time limit of 14.5 minutes per CMSA iteration and utilizing the *payment range* variant for solution merging. This variant proved better in manual testing and was clearly preferred during tuning compared to the *non-zero payments* version.

The CMSA was then performed ten times on each small instance excluding the tuning instances. Obtained results can be seen in Table 2. We can observe that the CMSA was able to outperform the MILP on all but three instances. Only on graph `enron`, which has the smallest number of vertices, Gurobi worked well in three out of four cases, in which it obtained best solutions in conjunction with small remaining optimality gaps of less than 10%. When comparing the results on instances `enron_uni1` and `enron_uni2`, one can see that the performance of the MILP approach can be drastically different on even the same graph but different weights assignments.

[2] `https://www.gurobi.com`
[3] `https://github.com/MLopez-Ibanez/irace`

Table 2: CMSA and MILP results on small instances.

Instance	MILP		CMSA			MILP/CMSA
	Obj.	%-Gap	Best	Avg	StdDev Obj.	%-Gap
enron_tri1	**22 504**	6.11	25 867	25 950	57.86	-15.31
enron_tri2	**23 797**	9.89	26 357	26 480	86.88	-11.28
enron_uni1	**24 425**	7.03	28 170	29 292	2 812.03	-19.93
enron_uni2	45 644	50.07	28 309	**28 404**	55.11	37.77
deezer_RO_tri1	2 648	100.00	1 504	**1 953**	243.13	26.27
deezer_RO_tri2	2 713	100.00	1 644	**2 127**	177.93	21.62
deezer_RO_uni1	2 158	100.00	1 157	**1 551**	182.04	28.10
deezer_RO_uni2	1 709	100.00	1 277	**1 417**	55.77	17.10
deezer_HU_tri1	1 775	100.00	1 493	**1 571**	39.09	11.47
deezer_HU_tri2	1 566	100.00	1 272	**1 334**	27.84	14.82
deezer_HU_uni1	1 306	100.00	1 073	**1 111**	30.43	14.89
deezer_HU_uni2	1 276	100.00	1 018	**1 085**	36.12	15.01
artist_tri1	94 278	100.00	93 530	**93 735**	141.15	0.58
artist_tri2	93 280	100.00	92 789	**93 018**	137.74	0.28
artist_uni1	96 539	100.00	95 893	**96 203**	209.03	0.35
artist_uni2	97 450	100.00	96 748	**96 914**	127.55	0.55
brightkite_tri1	38 584	70.27	25 467	**25 522**	34.72	33.85
brightkite_tri2	38 355	70.29	25 458	**25 537**	44.38	33.42
brightkite_uni1	39 733	69.96	27 039	**27 470**	1 031.54	30.86
brightkite_uni2	39 585	78.27	26 984	**27 488**	929.29	30.56

On the other instances, the average %-gap between the solution qualities of the CMSA and the MILP reached up to 37.77% in favor of the CMSA. Just for the instances on graph `artist`, these gaps are rather small with values of less than one percent.

Additionally, Fig. 1 shows a graphical comparison of final objective values obtained by the MILP, CMSA, the construction heuristics without randomization and the random solution construction baseline. The runtimes of the construction heuristics average to 87.89ms for the random heuristic, 245.53ms for the discount heuristic, and 3772.18ms for the impact heuristic. Both the *DiscountHeuristic* and the *ImpactHeuristic* were clearly able to outperform the randomized baseline, with the *ImpactHeuristic* producing slightly better solutions, but having a significantly longer runtime. Clearly, however, CMSA consistently obtained significantly better solutions.

Figures 2a and 2b show the progress of the incumbent solution quality over time for ten individual CMSA runs as well as the MILP specifically on instances `deezer_RO_uni2` and `artist_tri1`. Each round point represents either the start/end of a CMSA iteration or an incumbent found by the MILP solver.

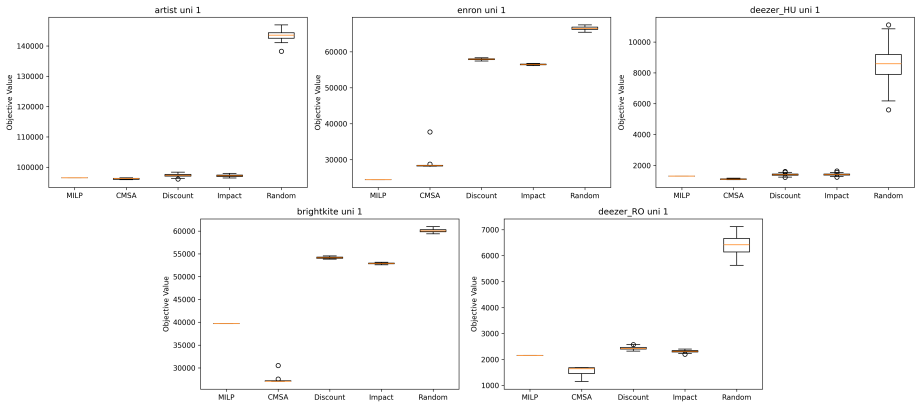
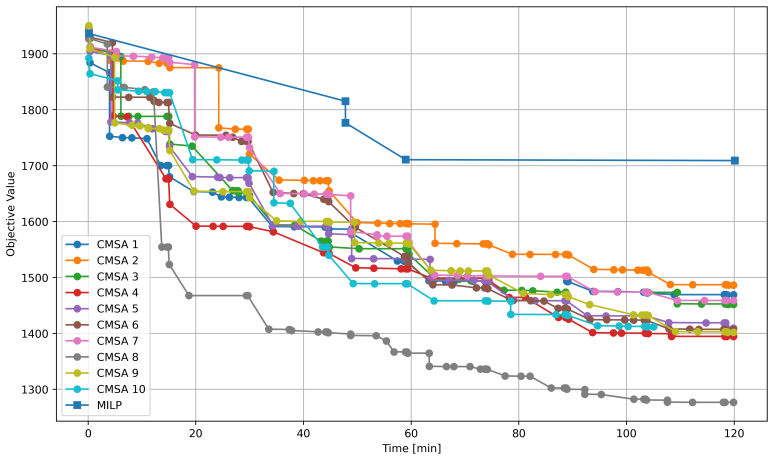
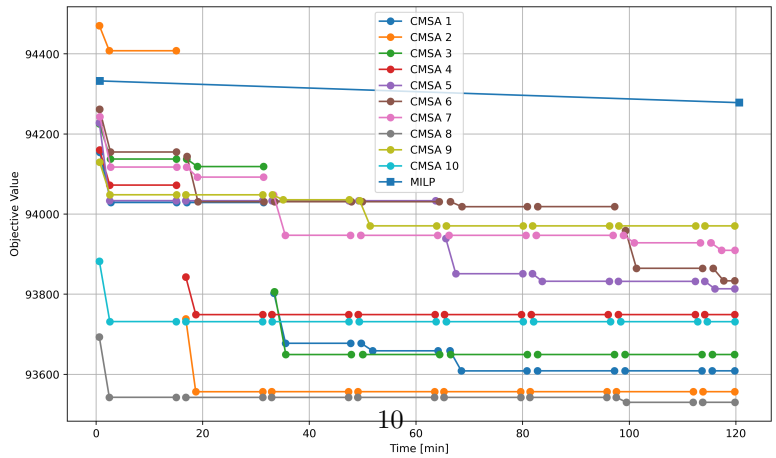


Fig. 1: Final objective values obtained by the MILP, CMSA as well as the faster construction heuristics on small instances.



(a) Instance `deezer_RO-uni2`



(b) Instance `artist-tri1`

Fig. 2: CMSA progress over time (small instances).

Figure 2a shows the more typical behavior we observed, with the CMSA improving the incumbent solution quality significantly and steadily over time. Yet, observe that there still is a significant variance in the final solution quality, and one CMSA run did significantly better than the others; also, the runs seem to not have completely converged in our time limit of two hours. This indicates that, in general even on our small graphs the instances are challenging and some room is left for further improvement.

In comparison, the plots in Figure 2b show the special case of graph `artist`. First, note that the y-axis covers here just a small range of objective values, meaning that in general all solution values are close together. What we primarily observe here is that the initial solution quality plays a more important role, i.e., is strongly correlated with the final solution quality. In most runs, CMSA could not substantially improve on the initial solutions. An exception is run CMSA-2, which started with the worst initial solution quality but nevertheless finished on second place. In this figure, the short pause for solution generation between MILP solver calls also is well visible as a usually small gap, and the graph for most iterations has either an L-shape, indicating that the solver quickly finds a better solution and nothing new afterwards, or is just a straight line, meaning that the MILP solver did not find any improvements. Sometimes the incumbent value changes significantly between iterations, which happens when the construction heuristic finds a new incumbent.

5.4 Results on Large Instances

For the larger instances, the graphs `livemocha` and `slashdot-08-11` were used as tuning instances, resulting in the configuration to use the *DiscountHeuristic* with parameter 0.63, generating 11 solutions per iteration, a solver time limit of 16 minutes and again utilizing the *payment range* version for solution merging. Again, CMSA was run ten times with a time limit of two hours and the MILP once with the same time limit of two hours per instance. Results can be seen in Table 3, with the CMSA finding better solutions than the MILP in eight out of twelve instances. Figure 3 shows again a graphical comparison of the MILP, CMSA and construction heuristics. Here the average runtimes of the construction heuristics are 253.21ms for the random heuristic, 698.87ms for the discount heuristic, and 21614.56ms for the impact heuristic.

Here our methods show quite different behavior on all three graphs. The MILP worked surprisingly well on the `slashdot-09-02` instances, which are the smallest in respect to the number of edges in our group of large graphs. Figure 4b shows for instance `slashdot-09-02-uni1` that the CMSA improves the solution quality substantially faster at the beginning, but is eventually is overtaken by the MILP. Another interesting observation is the difference between the *trivalency* and *uniform* weight generation methods, with *trivalency* instances proving easier for both the CMSA and MILP to obtain good solutions. Furthermore the CMSA solution costs for the *uniform* instances have a higher spread (standard deviation of 214 and 372 compared to 2796 and 5537) at the two-hours mark.

Table 3: CMSA and MILP results on large instances.

Instance	MILP Obj.	MILP %-Gap	CMSA Best	CMSA Avg	CMSA StdDev Obj.	MILP/CMSA %-Gap
`marker_cafe_tri1`	626 141	81.29	416 841	**423 362**	5964.58	32.39
`marker_cafe_tri2`	627 196	81.21	416 953	**421 669**	5 524.77	32.77
`marker_cafe_uni1`	645 494	81.28	439 203	**447 654**	6 350.74	30.65
`marker_cafe_uni2`	643 604	81.15	440 245	**447 133**	6 271.96	30.53
`slashdot-09-02_tri1`	**91 125**	12.19	98 069	98 585	372.32	-8.19
`slashdot-09-02_tri2`	**90 770**	11.61	97 344	97 869	213.85	-7.82
`slashdot-09-02_uni1`	**91 320**	91.66	102 061	105 582	2 796.21	-15.62
`slashdot-09-02_uni2`	**90 597**	33.76	101 843	107 439	5536.98	-18.59
`douban_tri1`	77 211	100.00	26 909	**26 914**	4.05	65.14
`douban_tri2`	76 788	100.00	26 706	**26 717**	21.07	65.21
`douban_uni1`	85 246	100.00	24 291	**24 294**	2.27	71.50
`douban_uni2`	84 516	100.00	24 227	**24 229**	1.65	71.33

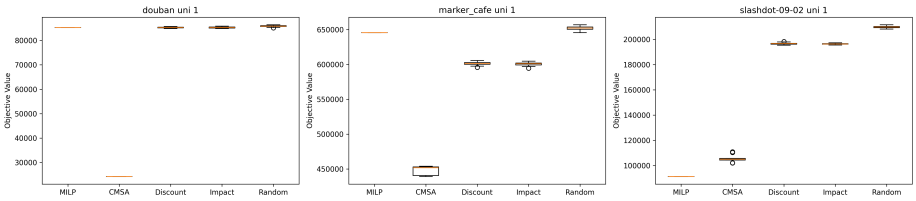
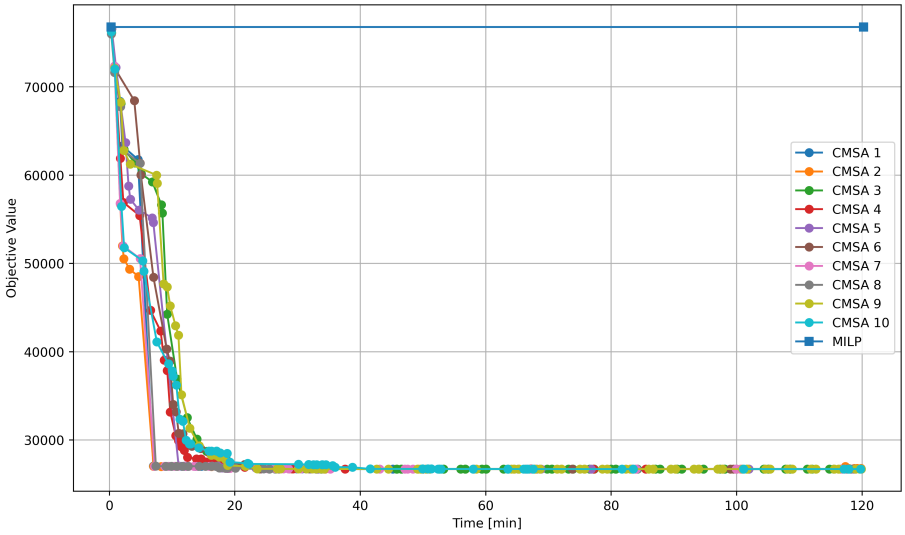


Fig. 3: Final objective values obtained by the MILP, CMSA as well as the faster construction heuristics on large instances.

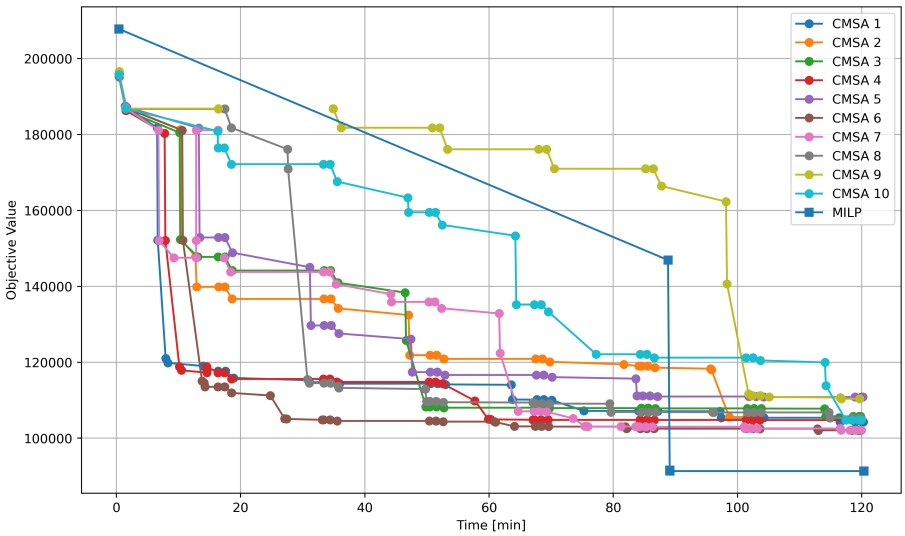
On the other instances, the CMSA again yielded substantially better final solutions with objective value gaps of 30% up to over 70% in comparison to the MILP.

On the `douban` instances, cf. 4b, the CMSA runs converge rather consistently in under 40 minutes to almost the same objective value (as indicated by the low standard deviation), while the MILP is not able to find any improvement on its initial solution.

For the `marker_cafe` instances, the MILP is again not able to find any improvement on its initial solution, and the CMSA seems to not have converged after two hours, but yielded significantly better solutions than the MILP. Looking at restricted MILP runs as part of the CMSA shows them finding large improvements at the start and no better solutions afterwards, indicating that these instances could benefit from shorter solver runtimes.



(a) Instance `douban-tri2`



(b) Instance `slashdot-09-02-uni1`

Fig. 4: CMSA Progress over Time (large instances)

6 Conclusions

We considered the Least Cost Influence Maximization Problem (LCIP), which is a generalization of the well-known Target Set Selection problem. In earlier work on the LCIP, only specific graph classes like trees and construction heuristics have been studied, and a simple time-indexed MILP model has been proposed, which does not scale at all to large graphs due to its time-indexed nature. In this work we proposed a new MILP formulation relying on the idea to find a directed acyclic influence subgraph. This model is applicable also to larger graphs as those we consider, and depending on the graph structure, Gurobi is able to find reasonable solutions with it. Still, its applicability is naturally restricted.

We therefore further proposed a Construct, Merge, Solve & Adapt metaheuristic that combines iterative randomized solution construction with solution merging via our new MILP model. We suggested two ways of performing the merging step, the non-zero payment and the payment range strategies, of which the latter turned out to be more effective. The CMSA, along with the standalone MILP and the heuristic construction methods, was tested on multiple large instances, showing that CMSA outperforms the other approaches in terms of solution quality in most cases.

The evaluation revealed strong variations in performance across different instances, suggesting that graph-type specific tuning can be beneficial. One promising direction to achieve this would be the application of self-adaptive CMSA [1], which can adjust its own parameters during a run. Further improvements might come from examining further solution construction methods, and considering combinations of them to generate a more diverse pool of solutions and solution components.

Overall, we demonstrated that the CMSA metaheuristic is an effective approach for solving large instances of the LCIP, efficiently combining the advantages of greedy solution construction methods with a MILP formulation.

References

1. Akbay, M.A., Serrano, A.L., Blum, C.: A self-adaptive variant of CMSA: application to the minimum positive influence dominating set problem. Int. J. Comput. Intell. Syst. **15**(1), 44 (2022). https://doi.org/10.1007/S44196-022-00098-1
2. Blum, C., Davidson, P.P., Ibáñez, M.L., Lozano, J.A.: Construct, merge, solve & adapt A new general algorithm for combinatorial optimization. Comput. Oper. Res. **68**, 75–88 (2016). https://doi.org/10.1016/J.COR.2015.10.014
3. Chen, C., Pasiliao, E.L., Boginski, V.: A cutting plane method for least cost influence maximization. In: Chellappan, S., Choo, K.R., Phan, N. (eds.) Computational Data and Social Networks – 9th International Conference. LNCS, vol. 12575, pp. 499–511. Springer (2020). https://doi.org/10.1007/978-3-030-66046-8_41
4. Cordasco, G., Gargano, L., Rescigno, A.A., Vaccaro, U.: Optimizing spread of influence in social networks via partial incentives. In: Scheideler, C. (ed.) Structural information and communication complexity. LNCS, vol. 9439, pp. 119–134. Springer (2015)

5. Demaine, E.D., Hajiaghayi, M., Mahini, H., Malec, D.L., Raghavan, S., Sawant, A., Zadimoghaddam, M.: How to influence people with partial incentives. In: Chung, C., Broder, A.Z., Shim, K., Suel, T. (eds.) 23rd International World Wide Web Conference. pp. 937–948. ACM (2014). https://doi.org/10.1145/2566486.2568039
6. Goyal, A., Lu, W., Lakshmanan, L.V.S.: SIMPATH: an efficient algorithm for influence maximization under the linear threshold model. In: Cook, D.J., Pei, J., Wang, W., Zaïane, O.R., Wu, X. (eds.) 11th IEEE International Conference on Data Mining. pp. 211–220. IEEE Computer Society (2011). https://doi.org/10.1109/ICDM.2011.132
7. Günneç, D., Raghavan, S., Zhang, R.: A branch-and-cut approach for the least cost influence problem on social networks. Networks **76**(1), 84–105 (2020). https://doi.org/10.1002/NET.21941
8. Günneç, D., Raghavan, S., Zhang, R.: Least-cost influence maximization on social networks. INFORMS J. Comput. **32**(2), 289–302 (2020). https://doi.org/10.1287/IJOC.2019.0886
9. Kempe, D., Kleinberg, J.M., Tardos, É.: Maximizing the spread of influence through a social network. In: Getoor, L., Senator, T.E., Domingos, P.M., Faloutsos, C. (eds.) Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 137–146. ACM (2003). https://doi.org/10.1145/956750.956769
10. Kunegis, J.: KONECT: the Koblenz network collection. In: et al., L.C. (ed.) 22nd International World Wide Web Conference. pp. 1343–1350. ACM (2013). https://doi.org/10.1145/2487788.2488173, https://konect.cc, last accessed 2025-08-18
11. Melo, R.A., Queiroz, M.F., Ribeiro, C.C.: Compact formulations and an iterated local search-based matheuristic for the minimum weighted feedback vertex set problem. Eur. J. Oper. Res. **289**(1), 75–92 (2021). https://doi.org/10.1016/J.EJOR.2020.07.006
12. Melo, R., Vignatti, A., Miyazawa, F., Ota, M.: Tighter dual bounds on the least cost influence problem. In: Proceedings of the 2020 Brazilian Symposium on Operations Research (SBPO) (2020). https://doi.org/10.59254/sbpo-2020-122560
13. Nguyen, H., Zheng, R.: On budgeted influence maximization in social networks. IEEE J. Sel. Areas Commun. **31**(6), 1084–1094 (2013). https://doi.org/10.1109/JSAC.2013.130610
14. Peixoto, T.P.: The Netzschleuder network catalogue and repository (2020). https://doi.org/10.5281/zenodo.7839981