Preventing disinformation spread across network clusters by strategic edge deletions: a Lagrangian heuristic

Martina Cerulli[1,*], Manlio Gaudioso[2], Domenico Serra[3] and Carmine Sorgente[3]

[1]*Department of Computer Science, University of Salerno, Fisciano, Italy*
[3]*Department of Computer Engineering, Modeling, Electronics and Systems, University of Calabria, Rende, Italy*
[3]*Department of Mathematics, University of Salerno, Fisciano, Italy*

Abstract
The rapid spread of fake news through dense clusters in networks jeopardizes trust and stability. Traditional approaches targeting harmful content often overlook the structural network dynamics that drive disinformation. The Cluster Deletion Problem addresses this by removing edges that bridge communities, disrupting disinformation flow while preserving intra-group communication. Indeed, given a graph representing the considered network, the Cluster Deletion Problem aims to transform it into a cluster graph via minimal edge deletions. In this paper, we address this problem with a tailored Lagrangian heuristic based on a dual descent algorithm, incorporating both a subgradient method and an ad-hoc polynomial-time repair heuristic. The proposed approach is tested on existing instances, in most cases providing better lower bounds with respect to the ones existing in the literature.

Keywords
cluster deletion, cybersecurity, fake news, Lagrangian relaxation

1. Introduction

The rapid dissemination of fake news and disinformation has emerged as a critical challenge for online platforms, with significant consequences for public trust, political stability, cyber security, and social well-being. Disinformation often spreads through dense clusters of interconnected users in social networks, amplifying its reach across large populations. While traditional techniques focus on detecting and removing harmful content, these approaches often fail to address the underlying network structures that enable the viral spread of false information. The Cluster Deletion Problem (CDP) provides an innovative solution to this issue. In the context of social networks, the CDP involves identifying and strategically removing edges – the connections between users – that link communities where disinformation is likely to spread. By focusing on these “boundary” users and their bridging connections, the CDP can disrupt the flow of fake news between disparate groups, containing its impact. This strategy limits the cross-community propagation of fake news, a key vector for disinformation. In this paper, we propose a novel solution approach to the CDP, which can be formally defined as follows: given a graph $\mathcal{G} = (V, E)$, find the minimum number of edge deletions required to transform $\mathcal{G}$ into a cluster graph, that is, a disjoint union of complete subgraphs, or equivalently, a graph where each connected component is a clique. In the decision version of the problem, an integer k is also considered, and the aim is to decide whether at most k edge deletions are enough to produce a cluster graph.

As already said, the CDP is frequently used in social networks to identify communities by clustering highly interconnected groups, which may represent social circles or influential subgroups. Shamir et al. (2004) applied the CDP to biological networks, particularly in protein-protein interaction networks, where clusters often correspond to protein complexes or functional modules in cellular processes. Sharan and Shamir (2000) leveraged similar clustering techniques in gene expression analysis, where

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03-8, 2025, Bologna, IT
*Corresponding author.
mcerulli@unisa.it (M. Cerulli); manlio.gaudioso@unical.it (M. Gaudioso); dserra@unisa.it (D. Serra); csorgente@unisa.it (C. Sorgente)
0000-0002-6341-8143 (M. Cerulli); 0000-0002-3022-7041 (M. Gaudioso); 0000-0002-6787-2220 (D. Serra); 0000-0001-6053-7318 (C. Sorgente)

clusters indicate genes with correlated expression patterns. Furthermore, Charikar et al. (2005) explored the use of edge deletion for clustering with qualitative constraints, showing its relevance to both social and biological networks where groupings based on connectivity patterns are of interest. Natanzon (1999) showed that the CDP is NP-complete and Shamir et al. (2004) that there exists some constant $\epsilon > 0$ such that it is also NP-hard to approximate the CDP within a factor of $1 + \epsilon$. The NP-completeness of the CDP, however, can already be extracted from the work of (Yannakakis, 1981) on hereditary graph properties. This problem has been proven to be NP-hard even when restricted to graphs with maximum degree 4 (Komusiewicz and Uhlmann, 2012). Due to its intractability, research has largely focused on developing approximation algorithms, and fixed-parameter tractable approaches (Crespelle et al., 2023). Gramm et al. (2005) addressed the decision version of the CDP, restricting the problem to allow at most k edge deletions, and provided a fixed-parameter algorithm with a runtime of $O(1.77^k + |V|^3)$. Subsequent improvements reduced this to $O(1.53^k + |V|^3)$ (Gramm et al., 2004), $O(1.47^k + |V|^3)$ (Damaschke, 2009), and finally $O(1.415^k + |V|^3)$ (Böcker and Damaschke, 2011). This advancement leverages the fact that an optimal cluster deletion set of k edges can be computed in polynomial time on graphs structured as a clique plus a constant number of additional nodes attached to it. As with many NP-complete problems, the CDP has also been studied on specific graph classes to identify instances where efficient solutions are achievable. For example, Gao et al. (2013) proposed a polynomial-time algorithm for solving the CDP on cographs. Regarding approximation algorithms, the first approach was introduced by Charikar et al. (2005) for complete graphs, using a linear programming relaxation to achieve a factor-4 approximation guarantee. More recently, Ailon et al. (2008) developed a randomized algorithm with an expected 3-approximation guarantee. Deterministic algorithms have also been proposed (van Zuylen and Williamson, 2009), culminating in a 2-factor approximation algorithm by Veldt et al. (2018). A problem with strong connections to the CDP is the Strong Triadic Closure. Grüttemeier and Komusiewicz (2020) showed that both problems are fixed-parameter tractable w.r.t. $|E| - k$, but do not admit a polynomial kernel parametrized by $|E| - k$. As regards exact approaches, Grötschel and Wakabayashi (1989) proposed the first integer linear formulation for the CDP, which was then made more compact by Martins (2016) exploiting the sparsity of the graph. To the best of our knowledge, the only heuristic procedure to address the CDP has been proposed in (Ambrosio et al., 2025), based on iterative edge contraction operations.

In this paper, we tackle the CDP through a Lagrangian relaxation approach, a powerful mathematical optimization technique, based on the machinery of the nondifferentiable optimization (Gaudioso and Monaco, 1992, Gaudioso et al., 2020), used to address hard combinatorial problems. We solve it through a Dual Descent (DD) Algorithm, which explores the solution space, finding valid lower bounds on the CDP optimal value. Within this algorithm, we incorporate both a subgradient method and an ad-hoc polynomial-time repair heuristic which, given an infeasible CDP solution, *repairs* it into a feasible one. We test our approach on existing instances, showing its effectiveness. The rest of the paper is organized as follows. In Section 2, we present the CDP edge-formulation, which is relaxed through a Lagrangian approach in Section 3. The DD Algorithm which is used to solve this relaxation is described in Section 4. Finally, computational results are discussed in Section 5, and Section 6 concludes the paper.

2. The CDP edge-formulation

To model the CDP, we consider the edge-formulation introduced in (Martins, 2016) in the binary variables x_{ij}, which, for each edge $\{i, j\} \in E$, is 1 if i and j belong to the same clique. Since minimizing the number of edges between clusters is equivalent to maximizing the number of edges inside the clusters (Dessmark et al., 2007), the CDP is formulated as:

$$\max_{x \in \{0,1\}^{|E|}} \sum_{\{i,j\} \in E} x_{ij} \tag{1a}$$

$$\text{s.t. } x_{ij} + x_{ik} \leq x_{jk} + 1 \qquad \forall\, i \in V,\ j, k \in \delta_i : j < k, \{j, k\} \in E \tag{1b}$$

$$x_{ij} + x_{ik} \leq 1 \qquad \forall\, i \in V,\ j, k \in \delta_i : j < k, \{j, k\} \notin E \tag{1c}$$

with δ_i defined as the set of neighbors of node $i \in V$. The objective function (1a) aims at maximizing the number of edges remaining in the graph. Given a triplet of nodes (i, j, k) such that both j and k belong to the neighborhood of i, constraints (1b) and (1c) impose that at most one between edges $\{i, j\}$ and $\{i, k\}$ stays in the graph, if either edge $\{j, k\}$ is removed from the graph ($x_{jk} = 0$) or does not exist, respectively.

3. The Lagrangian relaxation

Many optimization problems include constraints that make finding an exact solution computationally challenging. Lagrangian relaxation addresses this by relaxing some of these constraints and adding them to the objective as penalties. This simplifies the problem, which can be solved more efficiently while providing insights into the structure of the solution space. Specifically, in Lagrangian relaxation, each relaxed constraint is multiplied by a non-negative parameter, known as Lagrange multiplier, which can be tuned to iteratively approach feasible solutions. We construct the Lagrangian relaxation of the CDP formulation (1) by introducing multipliers

$$\lambda_{ijk}, \quad \forall\, i \in V,\ j,k \in \delta_i : j < k, \{j,k\} \in E, \quad \text{and} \qquad \mu_{ijk}, \quad \forall\, i \in V,\ j,k \in \delta_i : j < k, \{j,k\} \notin E$$

associated to constraints (1b) and (1c), respectively. We come out with the following problem:

$$z_{LR}(\lambda,\mu) = \max_{x \in \{0,1\}^{|E}} \left\{ \sum_{\{i,j\} \in E} x_{ij} + \sum_{i \in V} \sum_{\substack{j,k \in \delta_i: \\ j<k, \\ \{j,k\} \in E}} \lambda_{ijk}(1 + x_{jk} - x_{ij} - x_{ik}) + \sum_{i \in V} \sum_{\substack{j,k \in \delta_i: \\ j<k, \\ \{j,k\} \notin E}} \mu_{ijk}(1 - x_{ij} - x_{ik}) \right\}.$$

Let us define, for each edge $\{i, j\} \in E$, the sets

$$A_{ij} = \delta_i \cap \delta_j, \quad C_{ij} = \delta_i \setminus \{\{j\} \cup A_{ij}\}, \quad C_{ji} = \delta_j \setminus \{\{i\} \cup A_{ij}\}, \tag{2}$$

which allow us to write the cost coefficient c_{ij} associated to each variable x_{ij} in the objective function of $LR(\lambda, \mu)$ (see Appendix A) as:

$$c_{ij} = c_{ij}(\lambda,\mu) = 1 + \sum_{k \in A_{ij}} (\lambda_{kij} - \lambda_{ijk} - \lambda_{jik}) - \sum_{k \in C_{ij}} \mu_{ijk} - \sum_{k \in C_{ji}} \mu_{jik}. \tag{3}$$

Finally, isolating the constant term independent on x_{ij}, problem $LR(\lambda, \mu)$ is rewritten as:

$$z_{LR}(\lambda,\mu) = \left(\sum_{i \in V} \sum_{\substack{j,k \in \delta_i: \\ j<k\, \{j,k\} \in E}} \lambda_{ijk} + \sum_{i \in V} \sum_{\substack{j,k \in \delta_i: \\ j<k\, \{j,k\} \notin E}} \mu_{ijk} \right) + \max_{x \in \{0,1\}^{|E|}} \left\{ \sum_{\{i,j\} \in E} c_{ij} x_{ij} \right\}. \tag{4}$$

Notice that an optimal solution $x(\lambda, \mu)$ of $LR(\lambda, \mu)$ is obtained by simple inspection of the sign of the coefficients c_{ij} as

$$x_{ij} = \begin{cases} 1 & \text{if } c_{ij} \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

It is clear that, even relaxing the integrality constraint on x (i.e., replacing $x \in \{0,1\}^{|E|}$ with $x \in [0,1]^{|E|}$), the solution of the LR problem obtained as in Eq. (5) is still integer. Hence, the Lagrangian relaxation $LR(\lambda, \mu)$ of problem (1) has the *integrality property*, which means that the Lagrangian relaxation cannot provide a better bound on the original problem than the one given by the Linear Programming (LP) relaxation of the problem (Geoffrion, 1974).

Furthermore, observe that, once such a solution $x(\lambda, \mu)$ of $LR(\lambda, \mu)$ has been found, a subgradient

$g(\lambda, \mu)$ of $z_{LR}(\lambda, \mu)$ is immediately available. For any triplet (i, j, k), it is given by

$$g_{ijk}(\lambda, \mu) = \begin{cases} 1 + x_{jk} - x_{ij} - x_{ik}, & i \in V,\ j, k \in \delta_i : j < k, \{j, k\} \in E \\ 1 - x_{ij} - x_{ik}, & i \in V,\ j, k \in \delta_i : j < k, \{j, k\} \notin E. \end{cases} \tag{6}$$

Minimizing the Lagrangian function $z_{LR}(\lambda, \mu)$ with respect to the Lagrangian multipliers yields the Lagrangian dual problem. This dual problem provides an upper bound to the original (primal) CDP. The Lagrangian dual LD associated to our Lagrangian relaxation LR is

$$z_{LD} = \min_{\lambda, \mu \geq 0} z_{LR}(\lambda, \mu). \tag{7}$$

4. The Dual Descent method

This section describes a dual descent procedure to tackle the LD Problem (7). The DD method is an iterative approach to solving the Lagrangian dual problem while achieving the best possible bound on the original problem. As already noticed in Section 3, this bound cannot improve the LP one. However, the DD method can still be useful in exploring the solution space and giving insights into the structure of the problem.

All parameter values at iteration h are associated with the superscript h. In particular, $x^h = x(\lambda^h, \mu^h)$ and $z^h_{LR} = z_{LR}(\lambda^h, \mu^h)$. The aim of the method, whose pseudo-code is reported in Algorithm 1, is creating a sequence of multipliers (λ^h, μ^h) such that $z_{LR}(\lambda^{h+1}, \mu^{h+1}) \leq z_{LR}(\lambda^h, \mu^h)$. This is pursued by acting on one multiplier at a time. We introduce some possible multiplier updating rules, based on selecting one multiplier and considering the value of the associate constraint in correspondence with the current solution of the relaxation x^h.

The effect of the increase/decrease of any multiplier λ_{ijk} or μ_{ijk} onto the cost coefficients of the variables x in problem $LR(\lambda, \mu)$ is highlighted by recalling that the dualization of the constraints (1b) and (1c) gives rise, respectively, to the terms

$$\lambda_{ijk}(1 + x_{jk} - x_{ij} - x_{ik}) \text{ and } \mu_{ijk}(1 - x_{ij} - x_{ik}). \tag{8}$$

Given the starting λ^0, μ^0 (line 1), the cost coefficients c^0 can be immediately found by Eq. (3) (line 2), and consequently an optimal solution of $LR(\lambda^h, \mu^h)$ can be obtained by (5), together with its value z^0_{LR} (line 3). This solution is used as a starting point x^h within the while loop in lines 5–24 of the algorithm. It may be feasible in problem (1) (lines 11–13), or not (lines 6–10). If it is infeasible, at least one of the constraints of type (1b) or (1c) is violated. In line 7, for each triple (i, j, k) the different options are considered by calling Algorithm 3 described in Appendix B. It returns the updated multipliers together with the corresponding decrease in the cost coefficients Δ. After this multipliers update through Algorithm 3, given the induced graph corresponding to the infeasible solution x^h, in line 9, we apply a repair heuristic procedure presented in Section 4.1, to reconstruct the feasibility, obtaining solution $\tilde{x}^h$. This lets us determine, at each iteration, not only an upper bound z^h_{LR}, but also a lower bound to the optimal value of the CDP (line 10).

Stopping upon finding a feasible solution is often sufficient, however, one can decide to continue with the descent. In order to proceed, the Lagrangian multipliers must be updated. This is what is done in line 13, where the algorithm calls multiple iterations of a tailored Subgradient Method (Algorithm 4) described in Appendix C. It requires two multiplier pairs and a maximum number of iterations p_{max} in input. We consider as multiplier pairs: the current pair (λ^h, μ^h) and the first of the previously generated pairs (λ^h, μ^h) s.t. its distance (in terms of Euclidean norm) is greater than a given threshold ρ.

Clearly, if, at a given iteration h, the Δ returned by Algorithm 3 is too small (smaller than a threshold $\epsilon > 0$), the algorithm cannot generate a different solution in the following iterations, remaining blocked at the solution x^h. For this reason, only if Δ is greater than ϵ, we set the new objective function value to $z^h_{LR} - \Delta$ in line 15. Otherwise, we perform some iterations of the Subgradient Algorithm 4 (line 21) in order to generate different multipliers and a corresponding different z_{LR} value. We remark that,

Algorithm 1: Dual Descent algorithm for the Lagrangian Relaxation of the CDP

```
Input: Maximum number of DD iterations $h_{max}$, maximum number of subgradient iterations
       $p_{max}$, tolerance parameters $\epsilon, \rho > 0$, heuristic parameter $0 < \tau \leq 1$.
Output: Lower bound on the CDP optimal solution.
1  $LB \leftarrow 0$ and $(\lambda^0, \mu^0) \leftarrow (0, 0)$;
2  Compute cost coefficients $c^0_{ij}$ for each edge $\{i, j\} \in E$ according to Eq. (3);
3  Compute $x^0_{ij}$ for each edge $\{i, j\} \in E$ according to Eq. (5), and $z^0_{LR}$ to Eq. (4);
4  $h \leftarrow 0$;
5  while $h < h_{max}$ do
6      if $x^h$ is infeasible then
7          $\{(\lambda^{h+1}, \mu^{h+1}), \Delta\} \leftarrow$ Algorithm 3 $(h, \lambda^h, \mu^h, c^h)$;
8          Let $\tilde{\mathcal{G}}$ be a graph with node set $V$ and edge set $\tilde{E} = \{\{i, j\} \in E : x^h_{ij} = 1\}$;
9          $\tilde{x}^h \leftarrow$ Algorithm 2 $(\tilde{\mathcal{G}}, h, \tau)$;
10         if $\sum_{\{i,j\} \in E} \tilde{x}^h_{ij} > LB$ then $LB \leftarrow \sum_{\{i,j\} \in E} \tilde{x}^h_{ij}$;
11     else
12         if $\sum_{\{i,j\} \in E} x^h_{ij} > LB$ then $LB \leftarrow \sum_{\{i,j\} \in E} x^h_{ij}$;
13         $\{(\lambda^{h+1}, \mu^{h+1}), z^{h+1}_{LR}\} \leftarrow$ Algorithm 4 $(\lambda^h, \mu^h, p_{max})$;
14     if $\Delta > \epsilon$ then
15         Update $z^{h+1}_{LR} \leftarrow z^h_{LR} - \Delta$
16     else
17         $k \leftarrow h$ and $\delta \leftarrow 0$;
18         while $k > 0$ and $\delta < \rho$ do
19             $k \leftarrow k - 1$;
20             $\delta \leftarrow \left\| \begin{pmatrix} \lambda^h \\ \mu^h \end{pmatrix} - \begin{pmatrix} \lambda^k \\ \mu^k \end{pmatrix} \right\|$;
21         $\{(\lambda^{h+1}, \mu^{h+1}), z^{h+1}_{LR}\} \leftarrow$ Algorithm 4 $(\lambda^k, \mu^k, \lambda^h, \mu^h, p_{max})$;
22     Compute cost coefficients $c^{h+1}_{ij}$ for each edge $\{i, j\} \in E$ according to Eq. (3);
23     Compute $x^{h+1}_{ij}$ for each edge $\{i, j\} \in E$ according to Eq. (5);
24     $h \leftarrow h + 1$;
25 return $LB$
```

with this approach, no improvement of z_{LR} is guaranteed, but we can continue exploring the solution space in this way. According to Eq. (3) and (5) we compute the new cost coefficients c^{h+1} as well as the new point x^{h+1} in lines 22–23.

The algorithm continues until the maximum number of iterations h_{max} are performed, returning the best lower bound LB found. $z^{h_{max}}_{LR}$ will be the best upper bound found, which cannot be smaller than the LP relaxation value.

4.1 Repair heuristic

In this section, we describe a polynomial-time heuristic, which is called within the DD Algorithm whenever an infeasible solution x^h is found and operates on a graph $\tilde{G} = (V, E)$ obtained by considering only the edges associated with positive x^h_{ij}. The algorithm starts by receiving a graph, an iteration counter h and a threshold parameter τ between 0 and 1. Its goal is to transform the infeasible x^h into a feasible solution $\tilde{x}^h$ by iteratively processing the connected components of the graph.

While the graph is not empty, in line 2, the algorithm computes the connected components of graph $\tilde{G}$. For each connected component S, the nodes are ordered in descending order of their degree, i.e., by the number of neighbors within S (line 4). In the case of multiple nodes with the same number of

Algorithm 2: Repair heuristic algorithm.

```
   Input: Graph $\tilde{G} = (V, E)$, iteration counter $h$, parameter $0 < \tau < 1$.
   Output: Feasible CDP solution $\tilde{x}^h$.
1  while Graph $\tilde{G}$ is not empty do
2      Compute the connected components of $\tilde{G}$;
3      foreach connected component $S$ do
4          Order the nodes $v$ in $S$ by (descending) value of $\delta_i$;
5          $k^S \leftarrow \lceil \tau |S| \rceil$;
6          $K^S \leftarrow \emptyset$;
7          for $k \in 1 \ldots k^S$ do
8              $i \leftarrow S[k]$;
9              $K \leftarrow \{i\}$;
10             if $|\delta_i| > 0$ then
11                 foreach $j \in \delta_i$ do Compute sets $A_{ij} = \delta_i \cap \delta_j$ and $C_{ji} = \delta_j \setminus \{\delta_i \cup \{i\}\}$;
12                 Order the set $\delta_i$ in a non-increasing order of $A_{ij}$; in the case of ties, order in a
                     non-decreasing order of $C_{ji}$;
13                 while $\delta_i \neq \emptyset$ do
14                     $u \leftarrow \delta_i[0]$;
15                     $\delta_i \leftarrow \delta_i \setminus \{u\}$;
16                     if $K \subseteq \delta_u$ then $K \leftarrow K \cup \{u\}$ else continue;
17             if $|K| > |K^S|$ then $K^S \leftarrow K$;
18         foreach $u, v \in V : u \in K^S$ and $v > u$ do
19             if $\{u, v\} \in E$ and $v \in K^S$ then Set $\tilde{x}^h_{uv} \leftarrow 1$ else Set $\tilde{x}^h_{uv} \leftarrow 0$;
20         $V \leftarrow V \setminus K^S$;
21 return $\tilde{x}^h$
```

neighbors, they are ordered with respect to their index i. In lines 5-6, parameter k^S and set K^S are initialized. They are needed because we consider one by one the first k^S nodes in the ordered set S and construct k^S cliques starting from each of these nodes independently. K^S is the largest among the obtained cliques, initialized to the empty set, while k^S is set to a predefined fraction τ of $|S|$ (rounded up to the next integer).

After the initialization a for loop starts. At each iteration, the k-th node (with k from 1 to k^S) is selected in the ordered set of nodes in S as the starting point (line 8), and a clique K is initialized with just this node $i = S[k]$ (line 9). If the node i is not a singleton ($|\delta_i| > 0$), in lines 11–11, for each neighbor j of i, the sets A_{ij} and C_{ji} are computed as defined in (2). The set δ_i is then ordered in a non-increasing order of $|A_{ij}|$, breaking ties by $|C_{ji}|$ in non-decreasing order (line 12). This order will be used to determine how nodes from δ_i should be considered for insertion in the clique K. First of all, the more neighbors they have in common with i (i.e., the greater $|A_{ij}|$), the greater the potential final dimension of the clique. In the case of ties, we will prefer the nodes that are more *isolated* (i.e., with smaller $|C_{ji}|$) because it is less probable that they will be considered for other cliques in the future. The algorithm then enters a while loop (lines 13–16): provided there are still nodes left in δ_i, the algorithm iteratively adds nodes u from the ordered set δ_i to the clique K, removing them from δ_i. A node u is added to K if all members of K are also neighbors of u. If this condition is satisfied (u can belong to the clique K), K is updated. In line 17, if clique K is larger than the best clique K^S found for S so far, K^S is set to K. As soon as all the k^S first nodes are considered (the for loop in lines 7–17 ends) the best clique K^S found is considered and the solution $\tilde{x}^h$ is updated. For each pair of nodes $u, v \in K$, the edge $\{u, v\}$ is marked as part of the solution if it exists in the graph. Specifically, if $\{u, v\} \in E$ and both u and v are in K^S, set $\tilde{x}^h_{uv} = 1$, otherwise set $\tilde{x}^h_{uv} = 0$. After processing a component, the clique K^S is removed from the graph. This process repeats until the graph becomes empty, and the algorithm returns the feasible solution $\tilde{x}^h$.

5. Numerical results

To assess the performance of our proposed approach for the CDP, we conducted experiments using different datasets of instances: one derived from (Ambrosio et al., 2025), another from (Cerulli et al., 2023), and a third from the DIMACS graph coloring challenge[1]. The first dataset, from (Ambrosio et al., 2025), includes 120 networks generated according to the Barabási-Albert model, with varying sizes and densities, making them well-suited for representing real-world network scenarios. The second dataset, from (Cerulli et al., 2023), consists of 6 different networks selected from the social-networks literature (out of the 14 considered in the paper), each with fewer than 1000 nodes. Finally, the third dataset includes 40 instances, each containing fewer than 1000 nodes and 10000 edges. On these benchmark sets, we compare our approach against the edge contraction heuristic (referred to as "ECH") described in (Ambrosio et al., 2025).

All the experiments were performed on a 12th Gen Intel(R) Core(TM) i7-12700 2.10 GHz CPU with 16.0 GB RAM. For each run, we set a time limit of one hour, with a maximum of 3000 iterations (h_{max}) for the Algorithm 1 and 50 iterations (p_{max}) for the inner Subgradient Algorithm 4. Additionally, we used $\epsilon = 10^{-4}$, $\rho = 0.5$ and $\tau = 0.04$.

In Table 1, we report the average results obtained on the first dataset (Ambrosio et al., 2025) by grouping instances based on the number of nodes and edges, with each group representing the mean performance over instances sharing the same structural characteristics. For each group, we report the following information: number of nodes $|V|$ and edges $|E|$ of all the instances of the group; average value of the continuous relaxation (*UB*); average value of the solution identified by the ECH (*LB_{ECH}*); average value of the solution identified by running Algorithm 1 (*LB_{DD}*); average runtime of Algorithm 1 (*time_{DD}*) and number of explored solutions with different objective (*#sol_{DD}*). The left part of the table refers to groups of instances with $n \in [100, 400]$, while the right part to groups of instances with $n \in [600, 1000]$. In the last line of the table, we report the total averages of all columns, separately for these two groups.

Table 1
Results on instances from (Ambrosio et al., 2025).

$\vert V\vert$	$\vert E\vert$	UB	LB$_{ECH}$	LB$_{DD}$	Time$_{DD}$	#sol$_{DD}$	$\vert V\vert$	$\vert E\vert$	UB	LB$_{ECH}$	LB$_{DD}$	Time$_{DD}$	#sol$_{DD}$
100	**99**	49.5	34.2	34.2	30.4	34.0	**600**	**7056**	1782.0	378.0	377.4	3600.0	95.4
100	**196**	98.0	46.2	45.2	61.7	41.4	**600**	**10476**	3528.0	485.2	500.0	3600.0	47.0
100	**291**	145.5	58.0	57.8	87.2	52.8	**600**	**13824**	5238.0	586.8	606.6	3600.0	30.0
100	**384**	192.0	68.0	67.8	122.8	62.4	**600**	**3564**	6912.0	663.4	706.0	3600.0	30.4
200	**396**	198.0	89.6	87.8	125.0	75.4	**800**	**12544**	3168.0	533.8	540.4	3600.0	15.4
200	**784**	392.0	122.2	122.2	283.0	106.8	**800**	**18624**	6272.0	682.0	720.4	3600.0	20.6
200	**1164**	582.0	144.2	145.2	383.3	126.8	**800**	**24576**	9312.0	827.4	861.4	3600.0	16.8
200	**1536**	768.0	166.6	171.2	540.8	152.8	**800**	**6336**	12288.0	937.8	1006.2	3600.0	15.4
400	**4656**	792.0	229.6	224.0	981.1	176.0	**1000**	**9900**	4950.0	698.8	722.6	3600.0	10.8
400	**6144**	1568.0	297.8	298.4	1673.8	255.8	**1000**	**19600**	9800.0	907.6	955.0	3600.0	12.6
400	**1584**	2328.0	353.0	357.0	2371.0	205.0	**1000**	**29100**	14550.0	1072.6	1139.2	3600.0	8.0
400	**3136**	3072.0	397.0	420.8	3298.8	159.0	**1000**	**38400**	19200.0	1215.0	1323.6	3600.0	5.0
	Average	*848.75*	*167.2*	*169.3*	*829.9*	*120.7*		**Average**	*8083.3*	*788.2*	*770.1*	*3600.0*	*25.6*

In Table 2, we present the results obtained for the second dataset (Cerulli et al., 2023). Since it contains only six instances, the results are not averaged. Instead, each row reports the result of a specific instance, whose name is shown in the first column. The remaining column headings are the same as in Table 1. In Table 2, we present the results obtained for the second dataset (Cerulli et al., 2023). Since it contains only six instances, the results are not averaged. Instead, each row reports the result of a specific instance, whose name is shown in the first column. The remaining column headings are the same as in Table 1.

[1]https://mat.tepper.cmu.edu/COLOR/instances.html

Table 2
Average results on benchmark instances from (Cerulli et al., 2023).

instance	$\vert V\vert$	$\vert E\vert$	UB	$\mathbf{LB_{ECH}}$	$\mathbf{LB_{DD}}$	$\mathbf{Time_{DD}}$	$\mathbf{\#sol_{DD}}$
karate	**34**	**78**	39.0	24.0	24.0	101.4	21
dolphins	**62**	**159**	79.5	55.0	55.0	89.7	40
lesmis	**77**	**254**	150.0	136.0	127.0	240.7	66
polbooks	**105**	**441**	221.0	112.0	118.0	141.1	89
adjnoun	**112**	**425**	212.5	65.0	66.0	217.6	63
football	**115**	**613**	319.0	299.0	302.0	232.5	147
		Average	*170.2*	*115.2*	*115.3*	*170.5*	*71*

In Table 3 we show the average results obtained for the third dataset coming from the DIMACS graph coloring challenge. The instances are grouped based on the number of nodes (up to 50, up to 100, up to 150, up to 300, over 300), with each group representing the mean performance over the considered instances. The column headings are the same as in Table 1.

Table 3
Average results on benchmark instances from the DIMACS graph coloring challenge.

$\vert V\vert$	$\vert E\vert$	UB	$\mathbf{LB_{ECH}}$	$\mathbf{LB_{DD}}$	$\mathbf{Time_{DD}}$	$\mathbf{\#sol_{DD}}$
28.0	**179.7**	89.8	35.1	35.9	91.9	33.1
84.3	**792.3**	405.9	235.6	234.5	435.9	134.6
128.9	**1976.8**	1058.6	815.1	782.7	671.9	211.1
216.6	**4696.9**	2374.1	1060.1	1062.2	819.1	232.5
450.0	**7983.5**	3991.8	1404.4	1455.6	2532.5	267.5
	Average	*1608.3*	*729.6*	*732.9*	*933.5*	*179.8*

The results in the table emphasize the strong performance of the proposed algorithm in producing high-quality lower bounds, which are, in most of the cases, superior to the existing heuristic solution values. As the problem size increases with larger numbers of nodes, the computational time grows significantly, with runtimes reaching the time limit for the instances with $n \geq 600$ (right part of Table 1). Despite the increased computational effort, for those instances the bound found by the DD method is consistently higher than the one found by the ECH. An important positive indicator of the DD algorithm performance is the number of solutions explored. Higher values of $\mathbf{\#sol_{DD}}$ suggest that the algorithm is effectively exploring a broad solution space, which is beneficial for identifying strong lower bounds. This is particularly notable for mid-sized instances, where the algorithm shows high exploration rates. When considering larger instances, the number of explored solutions decreases as the algorithm does not stop for having performed h_{max} iterations, but because the time limit of one hour is reached.

The comparison with the continuous relaxation, which provides the best possible benchmark in terms of upper bounds, further demonstrates the quality of the lower bounds. Even if it does not reach the upper bound found by the LP relaxation, the gap is in most of the cases reduced, underlining the strength of the proposed method. The overall trends across the table indicate consistent and reliable performance, with the algorithm adapting well to varying problem structures and scales. In total the DD algorithm is at least as good as the ECH for 115 on 166 instances. Out of these 115 instances, it is strictly better for 95 of them.

6. Conclusion

Links between dense clusters in networks are highly critical for disinformation spread. In this work, we study a combinatorial optimization problem, named Cluster Deletion, aimed at partitioning a network into clusters of fully connected nodes, by identifying the minimum number of bridge connections. Monitoring these connections allows for disrupting the disinformation flow while preserving intra-group communication. We introduce a Lagrangian heuristic to solve the problem, consisting of a dual descent algorithm equipped with a subgradient method and an ad-hoc repair heuristic. Computational experiments are conducted on benchmark networks from the literature, providing, in most cases, new best-known solutions.

Acknowledgments

The research of M. Cerulli was partially supported by the project "SEcurity and RIghts in the CyberSpace" SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. This support is gratefully acknowledged.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

R. Shamir, R. Sharan, D. Tsur, Cluster graph modification problems, Discrete Applied Mathematics 144 (2004) 173–182.

R. Sharan, R. Shamir, CLICK: A clustering algorithm with applications to gene expression analysis, in: Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB), 2000, pp. 307–316.

M. Charikar, V. Guruswami, A. Wirth, Clustering with qualitative information, Journal of Computer and System Sciences 71 (2005) 360–383. doi:10.1016/j.jcss.2004.10.012, learning Theory 2003.

A. Natanzon, Complexity and approximation of some graph modification problems, University of Tel-Aviv, 1999.

M. Yannakakis, Edge-deletion problems, SIAM Journal on Computing 10 (1981) 297–309. doi:10.1137/0210021.

C. Komusiewicz, J. Uhlmann, Cluster editing with locally bounded modifications, Discrete Applied Mathematics 160 (2012) 2259–2270. doi:10.1016/j.dam.2012.05.019.

C. Crespelle, P. G. Drange, F. V. Fomin, P. Golovach, A survey of parameterized algorithms and the complexity of edge modification, Computer Science Review 48 (2023) 100556. doi:10.1016/j.cosrev.2023.100556.

J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, Graph-modeled data clustering: Exact algorithms for clique generation, Theory of Computing Systems 38 (2005) 373–392. doi:10.1007/s00224-004-1178-y.

J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, Automated generation of search tree algorithms for hard graph modification problems, Algorithmica 39 (2004) 321–347. doi:10.1007/s00453-004-1090-5.

P. Damaschke, Bounded-degree techniques accelerate some parameterized graph algorithms, in: J. Chen, F. V. Fomin (Eds.), Parameterized and Exact Computation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 98–109. doi:10.1007/978-3-642-11269-0_8.

S. Böcker, P. Damaschke, Even faster parameterized cluster deletion and cluster editing, Information Processing Letters 111 (2011) 717–721. doi:10.1016/j.ipl.2011.05.003.

Y. Gao, D. Hare, J. Nastos, The cluster deletion problem for cographs, Discrete Mathematics 313 (2013). doi:10.1016/j.disc.2013.08.017.

N. Ailon, M. Charikar, A. Newman, Aggregating inconsistent information: Ranking and clustering, Journal of the ACM 55 (2008). doi:10.1145/1411509.1411513.

A. van Zuylen, D. P. Williamson, Deterministic pivoting algorithms for constrained ranking and clustering problems, Mathematics of Operations Research 34 (2009) 594–620. doi:10.1287/moor.1090.0385.

N. Veldt, D. F. Gleich, A. Wirth, A correlation clustering framework for community detection, in: Proceedings of the 2018 World Wide Web Conference, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2018, pp. 439–448. doi:10.1145/3178876.3186110.

N. Grüttemeier, C. Komusiewicz, On the relation of strong triadic closure and cluster deletion, Algorithmica 82 (2020) 853–880. doi:10.1007/s00453-019-00617-1.

M. Grötschel, Y. Wakabayashi, A cutting plane algorithm for a clustering problem, Mathematical Programming 45 (1989) 59–96. doi:10.1007/BF01589097.

P. Martins, Modeling the maximum edge-weight k-plex partitioning problem, 2016. doi:10.48550/arXiv.1612.06243. arXiv:1612.06243.

G. Ambrosio, R. Cerulli, D. Serra, C. Sorgente, U. Vaccaro, Exact and heuristic solution approaches for the cluster deletion problem on general graphs, Networks (2025). doi:10.1002/net.22267.

M. Gaudioso, M. Monaco, Variants to the cutting plane approach for convex nondifferentiable optimization, Optimization 25 (1992) 65–75. doi:10.1080/02331939208843808.

M. Gaudioso, G. Giallombardo, G. Miglionico, Essentials of numerical nonsmooth optimization, 4OR 18 (2020) 1–47. doi:10.1007/s10288-019-00425-x.

A. Dessmark, J. Jansson, A. Lingas, E.-M. Lundell, M. Persson, On the approximability of maximum and minimum edge clique partition problems, International Journal of Foundations of Computer Science 18 (2007) 217–226. doi:10.1142/S0129054107004656.

A. M. Geoffrion, Lagrangean relaxation for integer programming, in: M. L. Balinski (Ed.), Approaches to Integer Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 1974, pp. 82–114. doi:10.1007/BFb0120690.

M. Cerulli, D. Serra, C. Sorgente, C. Archetti, I. Ljubić, Mathematical programming formulations for the collapsed k-core problem, European Journal of Operational Research 311 (2023) 56–72. doi:10.1016/j.ejor.2023.04.038.

N. Z. Shor, Minimization methods for non-differentiable functions, Springer Science & Business Media, 2012. doi:10.1007/978-3-642-82118-9.

F. Carrabs, M. Gaudioso, G. Miglionico, A two-point heuristic to calculate the stepsize in subgradient method with application to a network design problem, EURO Journal on Computational Optimization (2024) 100092. doi:10.1016/j.ejco.2024.100092.

A. Linking primal constraints with dual variables

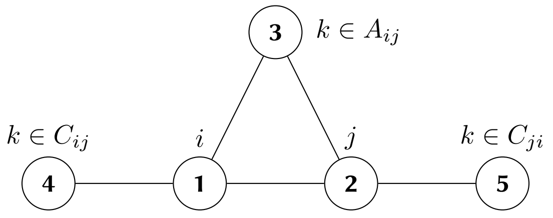

Figure 1: Example graph.

In Fig. 1, a graph with 5 nodes is shown, in order to better explain formula (3). Let i be node 1, j be node 2, and $k \in \{3, 4, 5\}$ (i.e., $i < j < k$). Observe that, in problem (1), variable x_{ij} appears in all constraints associated to the nodes i, j and $k \in \delta_i \cup \delta_j$. Consider first the case $k \in A_{ij}$, that is $\{i,j\},\{i,k\},\{j,k\} \in E$ (node 3 in Fig. 1). We have the following three constraints of type (1b) associated to the nodes (i,j,k) and involving the variable x_{ij}:

$$\begin{aligned} x_{ij} + x_{ik} - x_{jk} &\le 1 \quad \text{Node}\, i, \quad \textit{Multiplier}\, \lambda_{ijk}, \\ x_{ij} + x_{jk} - x_{ik} &\le 1 \quad \text{Node}\, j, \quad \textit{Multiplier}\, \lambda_{jik}, \\ x_{ik} + x_{jk} - x_{ij} &\le 1 \quad \text{Node}\, k, \quad \textit{Multiplier}\, \lambda_{kij}. \end{aligned}$$

Consequently, the contribution of the above constraints to the cost coefficient $c_{ij}(\lambda, \mu)$ is:
$\lambda_{kij} - \lambda_{ijk} - \lambda_{jik}$.

We remark that the indices of the variables are always written in such a way that the first index is smaller than the second one. Therefore, i will always appear before j, which, in turn, will always appear before k. This is why, for example, in the third constraint above, instead of having $x_{ki} + x_{kj} - x_{ij} \leq 1$, we have $x_{ik} + x_{jk} - x_{ij} \leq 1$. As for the multipliers, the first index is fixed to the node on which the corresponding constraint is defined, while the remaining two are ordered in such a way that the second index is smaller than the third one.

Consider now the case $k \in C_{ij}$ (node 4 in Fig. 1). The constraint to consider is

$$x_{ij} + x_{ik} \leq 1 \quad \text{Node}\, i, \quad \textit{Multiplier}\, \mu_{ijk}.$$

Instead, for $k \in C_{ji}$ (node 5 in Fig. 1) the constraint to be taken into account is

$$x_{ij} + x_{jk} \leq 1 \quad \text{Node}\, j, \quad \textit{Multiplier}\, \mu_{jik}.$$

The corresponding contributions to $c_{ij}(\lambda, \mu)$ are $-\mu_{ijk}$ and $-\mu_{jik}$, respectively.

B. Multipliers update in case of infeasibility

The pseudo-code of the algorithm which is used to update the Lagrangian multipliers in case an infeasible solution x^h is found is reported in Algorithm 3. It analyzes different constraint violations and updates the multipliers accordingly.

Algorithm 3: Multipliers update when an infeasible solution is found.

```
   Input: Iteration counter $h$, multipliers $\lambda^h, \mu^h$, and cost coefficients $c^h$.
   Output: Updated multipliers $(\lambda^{h+1}, \mu^{h+1})$.
1  foreach triplet $(i, j, k)$ do
2      if constraint (1b) is violated by $(i, j, k)$ then
3          $\Delta_{ijk} \leftarrow \min\{c^h_{ij}, -c^h_{jk}, c^h_{ik}\}$;
4      else if constraint (1c) is violated by $(i, j, k)$ then
5          $\Delta_{ijk} \leftarrow \min\{c^h_{ij}, c^h_{ik}\}$;
6  $\lambda^{h+1} \leftarrow \lambda^h$;
7  $\mu^{h+1} \leftarrow \mu^h$;
8  $(\bar{i}, \bar{j}, \bar{k}) \leftarrow \arg\max_{i,j,k} \Delta_{ijk}$ ;
9  if $(\bar{i}, \bar{j}, \bar{k})$ violates constraint (1b) then
10     $\lambda^{h+1}_{\bar{i}\bar{j}\bar{k}} \leftarrow \lambda^h_{\bar{i}\bar{j}\bar{k}} + \Delta_{\bar{i}\bar{j}\bar{k}}$;
11 else if $(\bar{i}, \bar{j}, \bar{k})$ violates constraint (1c) then
12     $\mu^{h+1}_{\bar{i}\bar{j}\bar{k}} \leftarrow \mu^h_{\bar{i}\bar{j}\bar{k}} + \Delta_{\bar{i}\bar{j}\bar{k}}$;
13 $\Delta \leftarrow \Delta_{\bar{i}\bar{j}\bar{k}}$;
14 return $(\lambda^{h+1}, \mu^{h+1})$ and $\Delta$
```

If, for a certain triplet $(\hat{i}, \hat{j}, \hat{k})$, constraint (1b) is violated, i.e., if $x^h_{\hat{i}\hat{j}} + x^h_{\hat{i}\hat{k}} > x^h_{\hat{j}\hat{k}} + 1$, it means that $x^h_{\hat{i}\hat{j}} = x^h_{\hat{i}\hat{k}} = 1$ and $x^h_{\hat{j}\hat{k}} = 0$. From (5), it holds that $c^h_{\hat{j}\hat{k}} < 0, c^h_{\hat{i}\hat{j}} \geq 0$ and $c^h_{\hat{i}\hat{k}} \geq 0$. We observe that, taking into account Eq. (8) and the remark in Appendix A, any increase of $\lambda^h_{\hat{i}\hat{j}\hat{k}}$ results in an increase of $c^h_{\hat{j}\hat{k}}$ and a decrease of both $c^h_{\hat{i}\hat{j}}$ and $c^h_{\hat{i}\hat{k}}$.

If instead, for a certain triplet $(\hat{i}, \hat{j}, \hat{k})$ constraint (1c) is violated, that is: $x^h_{\hat{i}\hat{j}} + x^h_{\hat{i}\hat{k}} > 1$, it means that $x^h_{\hat{i}\hat{j}} = x^h_{\hat{i}\hat{k}} = 1$, which implies, from (5), that $c^h_{\hat{i}\hat{j}}, c^h_{\hat{i}\hat{k}} \geq 0$. We observe that (8) indicates that any increase of $\mu^h_{\hat{i}\hat{j}\hat{k}}$ produces a decrease of both $c^h_{\hat{i}\hat{j}}$ and $c^h_{\hat{i}\hat{k}}$.

More specifically, in line 3, we define

$$\Delta_{ijk} = \min\left\{c^h_{ij}, c^h_{ik}, -c^h_{jk}\right\} \geq 0,$$

for each triplet (i, j, k) violating constraint (1b) and, in line 5, we define

$$\Delta_{ijk} = \min\left\{c^h_{ij}, c^h_{ik}\right\} \geq 0,$$

for each triplet (i, j, k) violating constraint (1c). Then, we look for the tiplet $(\bar{i}, \bar{j}, \bar{k})$ corresponding to the greatest Δ value in line 8. If it violates constraint (1b), in line 10, we update the corresponding multiplier $\lambda_{\bar{i}\bar{j}\bar{k}}$ as follows

$$\lambda^{h+1}_{ijk} = \lambda^h_{\bar{i}\bar{j}\bar{k}} + \Delta_{\bar{i}\bar{j}\bar{k}}, \tag{9}$$

while the remaining multipliers λ_{ijk} for $(i, j, k) \neq (\bar{i}, \bar{j}, \bar{k})$ as well as μ_{ijk} for all (i, j, k) do not change (lines 6–7). If, instead, $(\bar{i}, \bar{j}, \bar{k})$ violates constraint (1c), we update the corresponding multiplier $\mu_{\bar{i}\bar{j}\bar{k}}$ in line 12 as:

$$\mu^{h+1}_{i,j,k} = \mu^h_{\bar{i}\bar{j}\bar{k}} + \Delta_{\bar{i}\bar{j}\bar{k}}, \tag{10}$$

while not changing the remaining multipliers (lines 6–7).

C. The subgradient algorithm

Within the DD algorithm, we call the subgradient method described in this section, which produces a different multiplier pair which can be used to update the z_{LR} value. Actually, the subgradient methods itself can be used as an alternative to the DD algorithm to solve the dual problem (7). Indeed, this method, introduced by Shor (2012), is aimed at solving the unconstrained optimization problem $\min_{y\in\mathbb{R}^n} f(y)$, with $f : \mathbb{R}^n \to \mathbb{R}$ convex and not necessarily smooth. The basic iteration scheme is

$$y^{p+1} = y^p - \alpha^p g^p,$$

where the subgradient g^p is an element of $\partial f(y^p)$, the subdifferential of f at point y^p.

The stepsize α^p commonly used in the literature is in the form $\alpha^p = \dfrac{t^p}{\|g^p\|}$, where the stepsize t^p ensuring the convergence of the sequence $\{t^p\}$ can be set in different ways.

Following Carrabs et al. (2024), we propose the following choice of the stepsize:

$$t^p = \frac{\|\delta^p\|^2\|g^p\|}{2(f(y^{p-1}) - f(y^p) + (g^p)^\top\delta^p)}, \tag{11}$$

where $\delta^p = y^p - y^{p-1}$. Such a stepsize choice is an extension of the well-known Barzilai and Borwein approach to convex nonsmooth optimization.

When applying the subgradient method to our problem (7), y represents the pair of multipliers (λ, μ) and $f(y)$ is the function $z_{LR}(\lambda, \mu)$. The corresponding pseudocode is shown in the Algorithm 4. It starts from two points (λ^0, μ^0) and (λ^1, μ^1), which are needed to determine at the first iteration $p = 1$ the value of the stepsize t^1. At the beginning, z_{best} is assigned the tighter value between the upper bounds associated with the two starting points (line 1). Then, new points (λ^p, μ^p) are found at each iteration $p > 1$, by using the stepsize t^p (lines 4–5) and the subgradient g^p (line 9). The subgradient method stops as soon as a point (λ_p, μ_p) associated with a tighter upper bound z^p_{LR} than the starting one is found (line 11). If no such a solution is produced during p_{max} iterations, then the procedure returns the last point, together with the associated upper bound (line 13). Thanks to this algorithm, we can modify the Lagrangian multipliers properly within the DD algorithm, also when a feasible solution is found.

Algorithm 4: Subgradient Algorithm for Lagrangian Relaxation of the CDP

```
    Input: $\lambda^0, \mu^0, \lambda^1, \mu^1 \geq 0$, maximum number of iterations $p_{max}$.
    Output: Dual solution $(\lambda_{best}, \mu_{best})$ and corresponding value $z_{best}$.
 1  $z_{best} \leftarrow \min\{z_{LR}(\lambda^0, \mu^0), z_{LR}(\lambda^1, \mu^1)\}$;
 2  $p \leftarrow 1$;
 3  while $p < p_{max}$ do
 4      $t^p \leftarrow \frac{\|\delta^p\|^2 \|g(\lambda^p, \mu^p)\|}{\epsilon + 2\left(z_{LR}(\lambda^p, \mu^p) - z_{LR}(\lambda^{p-1}, \mu^{p-1}) - g(\lambda^p, \mu^p)^\top (\lambda^p - \lambda^{p-1}, \mu^p - \mu^{p-1})\right)}$;
 5      $t^p \leftarrow \max\left\{\min\left\{t_p, \frac{1}{\log_{10}(1+p)}\right\}, 0\right\}$;
 6      $\lambda^{p+1}_{ijk} \leftarrow \max\left\{0, \lambda^p_{ijk} - \frac{t^p}{\|g(\lambda^p, \mu^p)\|} g_{ijk}(\lambda^p, \mu^p)\right\}, \quad i \in V,\ j, k \in \delta_i : j < k, \{j, k\} \in E$;
 7      $\mu^{p+1}_{ijk} \leftarrow \max\left\{0, \mu^p_{ijk} - \frac{t^p}{\|g(\lambda^p, \mu^p)\|} g_{ijk}(\lambda^p, \mu^p)\right\}, \quad i \in V,\ j, k \in \delta_i : j < k, \{j, k\} \notin E$;
 8      Compute $z_{LR}(\lambda^{p+1}, \mu^{p+1})$;
 9      Compute $g^{p+1}(\lambda^{p+1}, \mu^{p+1})$ according to Eq. (6);
10      if $z_{LR}(\lambda^{p+1}, \mu^{p+1}) < z_{best}$ then
11          return $(\lambda^{p+1}, \mu^{p+1}), z_{LR}(\lambda^{p+1}, \mu^{p+1})$
12      $p \leftarrow p + 1$.
13  return $(\lambda_p, \mu_p), z^p_{LR}$
```