Pesquisa Operacional (2023) 43: e275468 p.1-23
doi: 10.1590/0101-7438.2023.043.00275468

Printed version ISSN 0101-7438 / Online version ISSN 1678-5142
www.scielo.br/pope
ARTICLES

COMBINATORIAL DUAL BOUNDS ON THE LEAST COST INFLUENCE PROBLEM

Renato Silva de Melo[1], André Luís Vignatti[2*],
Flávio Keidi Miyazawa[3] and Matheus Jun Ota[4]

Received June 7, 2023 / Accepted July 25, 2023

ABSTRACT. The Least Cost Influence Problem is a combinatorial optimization problem that appears in the context of social networks. The objective is to give incentives to individuals of a network, such that some information spreads to a desired fraction of the network at minimum cost. We introduce a problem-dependent algorithm in a branch-and-bound scheme to compute a dual bound for this problem. The idea is to exploit the connectivity properties of sub-graphs of the input graph associated with each node of the branch-and-bound tree and use it to increase each sub-problem's lower bound. Our algorithm works well and finds a lower bound tighter than the LP-relaxation in linear time in the size of the graph. Computational experiments with synthetic graphs and real-world social networks show improvements in using our proposed bounds. The improvements are gains in running time or gap reduction for exact solutions to the problem.

Keywords: combinatorial optimization, social networks, diffusion of information.

1 INTRODUCTION

In the context of diffusion of information in social networks, early studies on sociology (Rogers, 2010) observe that information spreads through a social system like a contagious process, starting with a small group and spreading to other individuals in the system through the relationship between them. One relevant problem that emerges from this background is identifying a good set of individuals to target, the *early adopters*, and hoping that the chosen individuals can persuade their friends to adopt a new behavior, which also influences friends of friends by generating a

*Corresponding author
[1]Department of Computer Science, Federal University of Paraná, Curitiba, PR, Brazil – E-mail: rsmelo@inf.ufpr.br – https://orcid.org/0000-0001-8268-5215
[2]Department of Computer Science, Federal University of Paraná, Curitiba, PR, Brazil – E-mail: vignatti@inf.ufpr.br – https://orcid.org/0009-0000-6969-9233
[3]Institute of Computing, University of Campinas, Campinas, SP Brazil – E-mail: fkm@ic.unicamp.br – https://orcid.org/0000-0002-1067-6421
[4]Institute of Computing, University of Campinas, Campinas, SP Brazil – E-mail: matheus.ota@students.ic.unicamp.br – https://orcid.org/0000-0002-3730-6420

chain of adoption. For example, in applications like viral marketing, the early adopters can be influenced by receiving products for free or by getting discounts for buying a new product.

Kempe et al. (Kempe et al., 2003) approached this problem from the discrete optimization perspective. They considered a maximization version where the goal is to find a fixed size set of individuals such that the expected influence of these individuals is the largest possible. They show that the INFLUENCE MAXIMIZATION PROBLEM is NP-hard in stochastic diffusion models and propose a $(1 - \frac{1}{e})$-approximate greedy algorithm. They also have shown that the influence function is submodular and monotone in randomized settings. The approximation ratio was originally proved by Nemhauser et al. (Nemhauser et al., 1978) and is valid for every submodular and monotone maximization function. Furthermore, computing the expected influence of a given set of individuals is a #P-hard problem (Chen et al., 2010a,b).

Chen (Chen, 2009) studied the minimization version of the INFLUENCE MAXIMIZATION PROBLEM, which seeks a target set of minimum size, ensuring the activation of a given fraction of the network. The literature refers to this problem as the TARGET SET SELECTION (TSS). Besides being NP-hard to solve, this problem is also hard to approximate. Chen (Chen, 2009) proved that TSS cannot be approximated within a poly-logarithmic ratio. Even with explicitly deterministic thresholds, the problem is NP-hard to approximate within a ratio of $n^{1-\varepsilon}$ for every $\varepsilon > 0$ (Chen, 2009; Kempe et al., 2003). This problem becomes tractable in some restricted class of graphs. For example, the problem can be solved in linear time if the underlying graph is a tree (Chen, 2009), a block-cactus graph (Chiang et al., 2013), or a bounded treewidth graph (Ben-Zwi et al., 2011).

In this work, we investigate an extension of the TSS problem called LEAST COST INFLUENCE PROBLEM (LCIP). Rather than searching a group of individuals to start a propagation, this problem offers incentives for individuals to adopt new behaviors and then trigger a cascade that spreads to a given fraction of the network. Demaine et al. (Demaine et al., 2014) extend the INFLUENCE MAXIMIZATION PROBLEM by proposing a fractional version that incorporates the idea of offering discounts instead of making a binary choice of individuals. Günneç et al. (Günneç et al., 2020a,b) focused on mathematical programming models for the LCIP and described algorithms for the problem on trees. Besides that, they observe that the LCIP is a generalization of the TSS problem, so the LCIP has at least the same complexity as the TSS problem from a theoretical point of view. Fischetti et al. (Fischetti et al., 2018) present a generalization of the LCIP and introduce the concept of *activation functions*, which extends the commonly used *threshold functions*. They also propose a mathematical heuristic and an exact algorithm based on column generation.

1.1 Contributions

While previous works provided relevant exact solutions (Ackerman et al., 2010; Fischetti et al., 2018; Günneç et al., 2020b) and heuristic algorithms that can be used as upper bounds for this

problem (Chen et al., 2009; Cordasco et al., 2015; Demaine et al., 2014; Kempe et al., 2003), nothing beyond the standard LP-relaxation was proposed to compute lower bounds on the LCIP.

Observing that the influence propagation network is a directed acyclic graph (DAG), we derive a problem-dependent relaxation algorithm. The proposed algorithm exploits the connectivity properties of graphs to obtain a lower bound for the problem. Furthermore, we prove that the algorithm is correct and show experimentally that our lower bounds are tighter than the linear programming relaxation, providing smaller optimality gaps. To the best of our knowledge, there are no works on combinatorial lower bounds for this problem. The main objective of our relaxation is for fathoming in a branch-and-bound algorithm and helping reduce the computational effort to obtain exact solutions. We provide theoretical analysis on the complexity of our algorithm when dealing with a particular case of the problem, where the diffusion needs to reach only a fraction of the network instead of the whole network. In this case, our analysis leads to a related problem whose optimal solution is a dual bound for the original problem. This related problem is NP-hard, but it paves the way for improving the dual bound. To make our dual bound stronger, we also propose a branching rule that prioritizes the exploration of some branches of the decision tree. Such a prioritization leads to disconnected sub-graphs associated with the sub-problems of the branch-and-bound tree, which in turn improves the quality of our lower bounds.

The rest of this paper is organized as follows. Section 2 contains a brief overview of the social network diffusion process and the problem definition. Section 3 describes the mathematical programming formulation of the problem and the exact method to solve it. Section 4 proposes an algorithm to find lower bounds on the problem and presents a theoretical analysis of our findings. We introduce a branching rule to strengthen our dual bound algorithm in Section 5. Computational experiments are presented in Section 6. We conclude the paper in Section 7.

2 PROBLEM DEFINITION

Let G be a directed graph that models a social network. The vertex set $V(G)$ represents individuals, and the arc set $E(G)$ corresponds to the relationships between these individuals. We denote the vertex set and arc set by V and E, respectively, when the context is explicit. Each arc $(i, j) \in E$ has an associated weight $d_{ij} > 0$ that models the influence of i over j.

To model the diffusion of influence between the individuals in a social network, we consider a well-known model called the *threshold model*, presented by (Granovetter, 1978). In this model, we say that a vertex i is *active* when persuaded to adopt new behavior and *inactive*, otherwise. Every $i \in V$ has a threshold $t_i > 0$ which indicates the amount of influence needed to activate i, coming from i's neighbors. The activation process is *progressive*, i.e., each vertex can be active or inactive, and a vertex can change from inactive to active, but not the other way around. Initially, a subset $A_0 \subseteq V$ is chosen to be active. Then, the vertices in A_0 send influence to their inactive neighbors. These neighbors might become active in the next iteration and give rise to a new set $A_1 \subseteq V$ of active vertices. This process is repeated until no vertex can be activated. Let $\{A_\tau\}_{\tau=0}^{T}$ be the sequence of vertices activated during the diffusion process, where A_T is the first set in the

sequence that cannot activate any other vertex in the graph. We say that any vertex $i \in A_T \setminus A_0$ has been *influenced* by A_0.

Also, we consider the offer of external influences. These influences, which we call incentives, aim to break an individual's resistance in being influenced in the activation process. The incentives are represented by a vector $\mathbf{y} \in \mathbb{Z}^{|V|}$, where each coordinate $y_i \in \mathbb{N}_0$ denotes the amount of incentive given to a vertex $i \in V$. Applying the incentive y_i on a vertex i decreases its threshold t_i and makes it more susceptible to activation. This incentive is added with the influence coming from the other vertices. Thus, the initial set of active vertices is given by $A_0 = \{i \in V : y_i \geq t_i\}$. The vertices in A_0 begin the process as active and all the others as inactive. Time progresses in discrete steps $\tau = 0, 1, \ldots, T$, and an inactive vertex i becomes active at time τ if the total influence of its active in-neighbors plus its incentive exceeds the threshold t_i, i.e., if

$$\sum_{j \in N_i \cap A_{\tau-1}} d_{ji} \geq t_i - y_i,$$

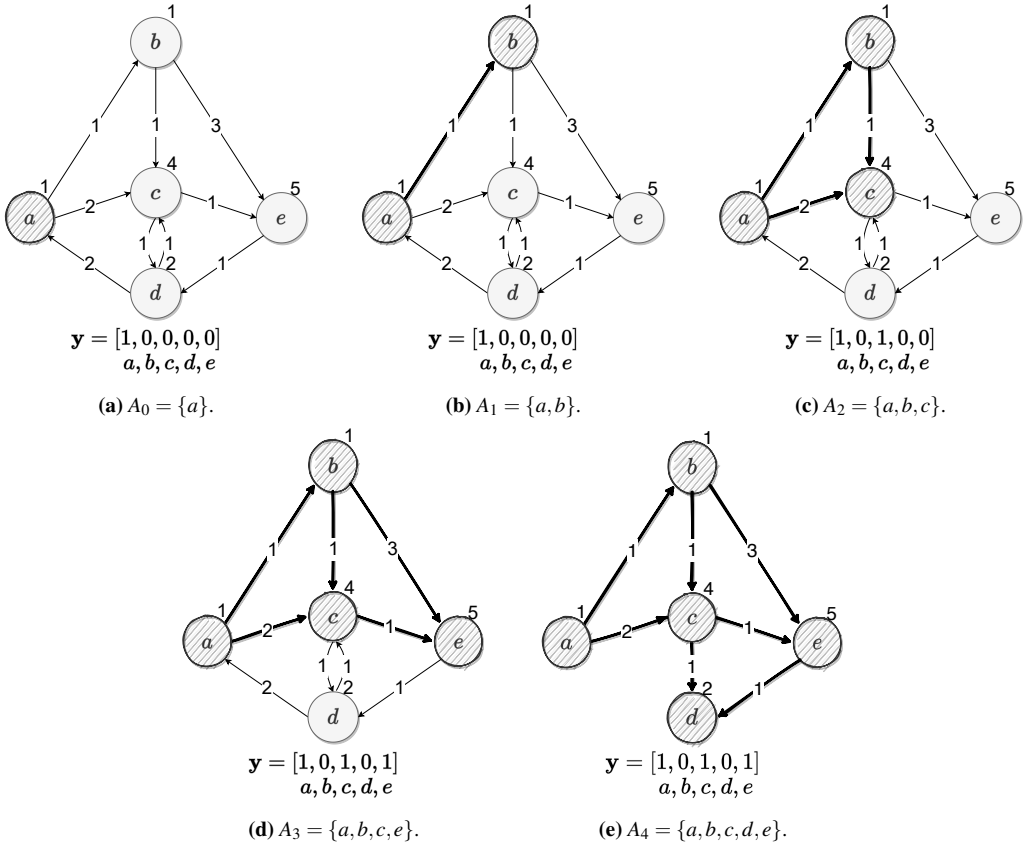
where N_i denotes the set of in-neighbors of i.

Figure 1 illustrates the activation process in the threshold model with incentives. Again, we have a directed graph with thresholds on the vertices and weights of influence on the arcs. Suppose that we want to activate 100% of the vertices. The process starts by setting the vertex a as active and all the others as inactive. We are paying enough incentive to achieve a's threshold without the influence of the neighbors, so vertex a receives 1 unit of incentive. We will try to activate the remaining vertices using incentives to decrease their threshold. Vertex b has threshold 1 and will be activated only by a in the next step, so no incentive is necessary. Next, in Figure 1c, vertex c has two active neighbors in which the weight of incoming arcs sums to 3 units. To achieve c's threshold, we must pay one incentive unit, so c is activated. The process continues as a cascade until all vertices are activated. Hence, in this example, we paid a total of 3 units of incentive to activate the whole network, that is, $y_a = y_c = y_e = 1$ and $y_b = y_d = 0$.

The problem consists of offering incentives to the vertices to trigger a cascade of influence that spreads to a given fraction α of the whole network. The goal is to minimize the sum of incentives given to individuals on the social network. The definition is given below.

Problem 1 (Least Cost Influence Problem (LCIP)). *We are given a real number $\alpha \in [0,1]$, a directed graph G with weights d_{ij} on each arc $(i,j) \in E$, and a threshold t_i, for each vertex $i \in V$. The goal is to find a vector $\mathbf{y} \in \mathbb{Z}^{|V|}$ of incentives that minimizes the sum $\sum_{i \in V} y_i$, ensuring that at least $\lceil \alpha |V| \rceil$ vertices are activated by the end of the activation process.*

In the remainder of this text, we will refer explicitly to the graph associated with a solution $\mathbf{y}$. Thus we introduce the following notation. For time steps $\tau = 0, 1, \ldots, T-1$, let $E_\tau = \{(i,j) \in E : i \in A_\tau, j \in A_{\tau+1} \setminus A_\tau\}$, be the set of arcs in which the influence was exerted at time τ. Consider the set $E^* = \bigcup_{\tau=0}^{T-1} E_\tau$. We say that the *propagation graph* $G^* = (A_T, E^*)$ is the graph induced by the solution $\mathbf{y}$. The graph in Figure 1e is an example of a propagation graph. It follows from the definition of the activation process that the propagation graph is acyclic.



(a) $A_0 = \{a\}$. **(b)** $A_1 = \{a,b\}$. **(c)** $A_2 = \{a,b,c\}$.

(d) $A_3 = \{a,b,c,e\}$. **(e)** $A_4 = \{a,b,c,d,e\}$.

Figure 1 – Activation process in the threshold model considering incentives, starting from a target set $S = \{a\}$. The number attached to each vertex represents its threshold. Sets A_τ contains the active vertices at time τ, for $\tau = 0,...,4$. Labels in each arc (i,j) denote the weight of influence d_{ij}. Highlighted arcs indicate who influences whom in the process. The vector $\mathbf{y}$ indexed by the vertices contains the incentive offered to each vertex.

3 INTEGER LINEAR PROGRAMMING FORMULATION

Different integer linear programming (ILP) formulations express the propagation using variables on the arcs (Ackerman et al., 2010; Günneç et al., 2020a,b; Raghavan & Zhang, 2019). The following formulation is a particular case of the model proposed by Fischetti et al. (Fischetti et al., 2018). For each vertex $i \in V$, let x_i be a binary variable indicating whether i is active at the end of the diffusion process. Similarly, for each arc $(i,j) \in E$, let z_{ij} be a binary variable

indicating whether i exerts influence over j. As mentioned previously, the integer variable y_i is the incentive to be paid to a vertex $i \in V$.

$$\min \sum_{i \in V} y_i$$

$$\text{s.t.} \sum_{i \in N_j} z_{ij} d_{ij} \geq t_j x_j - y_j \qquad \forall j \in V, \tag{1}$$

$$\sum_{(i,j) \in C} z_{ij} \leq \sum_{i \in V(C) \setminus \{k\}} x_i \qquad \forall k \in V(C), \text{cycle } C \subseteq E, \tag{2}$$

$$z_{ij} \leq x_i \qquad \forall (i,j) \in E, (j,i) \notin E, \tag{3}$$

$$\sum_{i \in V} x_i \geq \lceil \alpha |V| \rceil, \tag{4}$$

$$x_i \in \{0,1\} \qquad \forall i \in V, \tag{5}$$

$$y_i \in \mathbb{N}_0 \qquad \forall i \in V, \tag{6}$$

$$z_{ij} \in \{0,1\} \qquad \forall (i,j) \in E. \tag{7}$$

The objective function minimizes the total incentive offered to influence a given portion of the network. Constraints (1) models the condition that a vertex $i \in V$ gets active only when the total influence received from its active neighbors plus its incentive is greater than or equal to its threshold. The cycle elimination constraints in (2) generalize the classic cycle elimination from (Grötschel et al., 1985) and impose that the propagation graph must be acyclic. Meaning that the number of chosen arcs in a cycle C cannot be greater than the number of active vertices in $V(C) \setminus \{k\}$, where $V(C)$ is the set of vertices in the cycle. Constraints (3) ensures that an arc (i,j) can be chosen only if vertex i is activated. Note that if there is an arc (j,i) in G, constraints (2) imply that $z_{ij} + z_{ji} \leq x_i$, and therefore, $z_{ij} \leq x_i$ is redundant. Finally, constraints (4) impose that, by the end of the diffusion process, the number of active vertices is at least $\lceil \alpha |V| \rceil$.

3.1 Solving the LCIP using branch-and-cut

Due to the number of possible cycles in the graph G, the number of constraints in (2) grows exponentially. The exact standard procedure for solving integer linear programs with exponential constraints is the branch-and-cut method, which combines LP-based branch-and-bound and constraint generation techniques. For completeness, we will take a brief look at the branch-and-bound approach. If the reader is already familiar with the concepts in this algorithm, you can go straight to Section 4.

The branch-and-bound method solves the problem by dividing it into smaller sub-problems. The principle is to split the feasible space into successively smaller subsets so that distinct subsets can be evaluated directly by implicit enumeration until the best solution is found. The method employs a tree structure consisting of nodes and branches to manage the subdivisions of the feasible region. In this tree, each node represents a sub-problem. To generate the solutions and efficiently explore the feasible region, two problem-specific routines are required, the *branch* and the *bound*.

- **Branching** is the procedure that splits a parent node into smaller sub-problems generating child nodes. In our case, we chose a binary variable x_i (or z_{ij}) and solve two sub-cases, namely the case $x_i = 0$ and the case $x_i = 1$.

- **Bounding** is the computation of lower and upper bounds used to avoid the complete exploration of all sub-trees. Since we are looking for the optimality conditions that will provide stopping criteria, an important task is to find a lower (dual) bound $\underline{z} \leq z^*$ and an upper (primal) bound $\bar{z} \geq z^*$, where z^* is the optimum value for the objective function of the original problem. If $\underline{z} \geq \bar{z}$, we do not need to consider the current sub-problem. Every feasible solution provides an upper bound (we can use heuristics to find it). The dual bound is an optimistic estimation of the objective function for the region represented by the node at hand. The most common approach to get dual bounds is by relaxing the integrality constraints of the original problem.

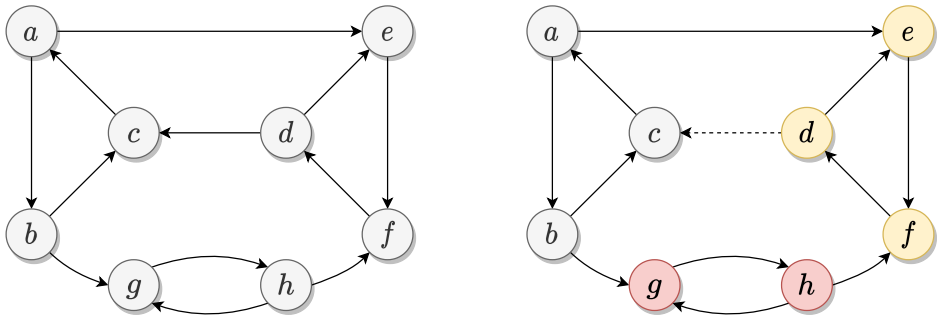
The set of all cycle elimination constraints in the model (1)-(7) is too large to write down. So, at each node of the branch-and-bound tree, violated inequalities are dynamically generated by solving the separation problem associated with inequalities in (2). In this approach, we implement the separation procedure proposed by (Grötschel et al., 1985). In short, the procedure adapts a shortest path algorithm to find a cycle that violates the cycle elimination constraints.

The branch-and-bound efficiency depends on how close to the optimal solution are the bounds of the sub-problems. Thus, we propose a combinatorial relaxation algorithm that can be used at each node of the branch-and-bound tree to obtain lower bounds.

4 LOWER BOUND ALGORITHM

Consider the following aspects of the LCIP. The associated propagation graph is a DAG for every solution $\mathbf{y}$, with at least one vertex with no incoming arcs (source). Thus, at least one vertex needs to be paid the total threshold value for any solution. The vertices chosen to receive the total incentive have the minimum threshold value in the best case.

We introduce a combinatorial algorithm to obtain a dual (lower) bound for this problem. The idea is to use connectivity properties of a sub-graph of the input graph at each node of the branch-and-bound tree. In the branch-and-bound, the recursive decomposition of a problem into sub-problems generates a decision tree where its root corresponds to the original problem, and each node corresponds to a smaller sub-problem. A natural branching rule in a branch-and-bound algorithm is by variable fixing. In the case of the formulation (1)-(7), we can fix the values of the binary variables $\mathbf{x}$ and $\mathbf{z}$. Suppose a binary variable, say z_{ij}, is selected to be the branching variable. Then two sub-problems are generated by fixing $z_{ij} = 0$ in one branch and $z_{ij} = 1$ in the other. We observe that fixing some arc variables z_{ij} (or vertex variables x_i) at zero means that these arcs (or vertices) were not chosen, which can disconnect the sub-graph related to that node of the decision tree. We are interested in using this information to increase the lower bound of each sub-problem during the branch-and-bound algorithm.



(a) strongly connected graph

(b) not strongly connected graph

Figure 2 – The directed graph in Figure (b) is obtained by removing the arc (d,c) of the graph in Figure (a). The vertex colors indicate the strongly connected components.

To illustrate the idea behind this strategy, consider the following example. Let the directed graph in Figure 2a be the input graph of the LCIP. We start the branch-and-bound tree by fixing some arc variables. Figure 3 shows the first levels of such structure, where the black node represents the sub-problem obtained by fixing the arc variables in $z_{ae} = 1$ and $z_{dc} = 0$. As the propagation graph of this sub-problem cannot contain the arc (d,c), we can represent the sub-graph associated with this node by removing the arc (d,c) from the original graph. In this way, we arrive at the sub-graph in Figure 2b, which is not strongly connected and is made of three different strongly connected components.

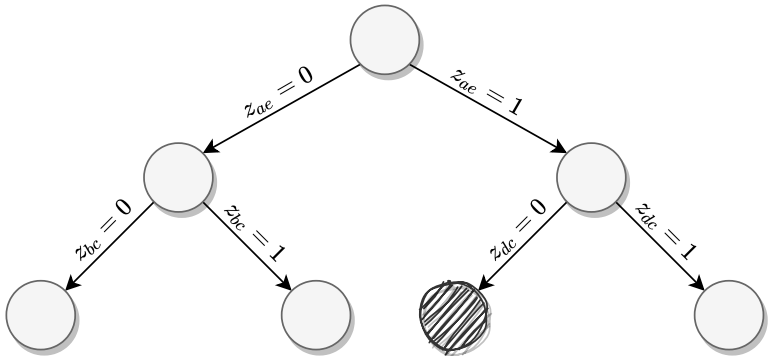


Figure 3 – Example of a decision tree with branching decisions on the binary variables. The black node represents the sub-problem obtained by fixing variables $z_{ae} = 1$ and $z_{dc} = 0$.

Our algorithm uses the concept of *condensed component graph*.

Definition 1 (Condensed Component Graph). *A condensed component graph H of a directed graph G is obtained by contracting the strongly connected components (s.c.c.) of G. Formally, each $v \in V(H)$ is associated with a s.c.c. C_v of G and there is an arc $(u,v) \in E(H)$ if and only if an arc exists from a vertex $i \in V(C_u)$ to a vertex $j \in V(C_v)$, where C_u and C_v are the s.c.c.'s associated with u and v, respectively.*

A different sub-graph G' of the input graph G is considered at each branch-and-bound tree node. We obtain G' from G by removing arcs and vertices in which the corresponding variables were fixed at zero by the decision tree. That is, $V(G') = V(G) \setminus \{i \in V(G) : x_i \text{ is fixed in zero}\}$ and $E(G') = E(G) \setminus \{(i,j) \in E(G) : z_{ij} \text{ is fixed in zero}\}$, at the current node of the branch-and-bound tree.

Let H be the condensed component graph of G'. From now on, we consider that H has the arc weights and vertex thresholds defined as follows. Consider C_u and C_v the s.c.c.'s associated with $u, v \in V(H)$ and $E_{uv} = \{(i,j) \in E(G') : i \in V(C_u) \text{ and } j \in V(C_v)\}$. We set the weight of each arc $(u,v) \in E(H)$ to be the sum of all arc weights that go from C_u to C_v, that is, $d_{uv} = \sum_{(i,j) \in E_{uv}} d_{ij}$. Furthermore, for each vertex $v \in V(H)$ we set $t_v = \min_{i \in V(C_v)} \{t_i\}$.

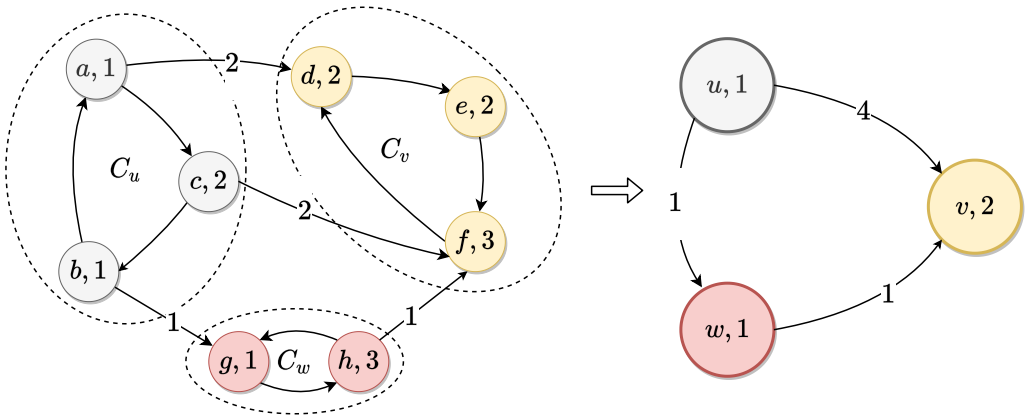


Figure 4 – Example of a condensed component graph. The original graph is on the left, and the condensed component graph is on the right.

Figure 4 presents an example for the condensed component graph of a small graph. The labels in each vertex of the figure denote the name and the threshold, respectively. For instance, the vertex in upper left corner has name a and threshold $t_a = 1$. In the leftmost graph, there are three strongly connected components C_u, C_w and C_v. For simplicity, we only show the arc weights between different components. In the second graph, we have the condensed component graph with the new thresholds and weights on arcs. For instance, the arc (u,v) has weight $d_{uv} = d_{ad} + d_{cf} = 4$, and the vertex v has threshold $t_v = \min\{2,2,3\}$.

The algorithm to find a lower bound for the LCIP follows.

Algorithm 1: Combinatorial Lower Bound

Input: G', influence **d** on arcs, thresholds **t** on vertices, and α

Output: A lower bound l for the LCIP

begin

  if G' *is strongly connected* ***or*** $\alpha < 1$ **then**

    return $\min_{i \in V(G')} \{t_i\}$.

  else

    Let H be the condensed component graph of G'.

    `// Return the optimum value of the LCIP on` H

    return $\sum_{v \in V(H)} \max\{0, t_v - \sum_{u \in N_v} d_{uv}\}$

Let l^{LP} be the lower bound obtained at the current node of the branch-and-bound tree by standard LP-relaxation, and let l be the lower bound obtained by the procedure described in Algorithm 1. When updating the lower bound $\underline{l}$ at the current node, we do $\underline{l} = \max\{l^{LP}, l\}$.

Now, we show that Algorithm 1 indeed returns a valid lower bound. If the sub-graph G' is strongly connected, the lower bound l is trivial by the observations at the beginning of this section. We return this trivial lower bound if $\alpha < 1$, even if G' is not strongly connected. Finding a tighter lower bound in this case may be computationally costly. Consequently, the algorithm would no longer be scalable. In Section 4.1, we explain this situation in detail.

When G' is not strongly connected and $\alpha = 1$, we can improve the trivial lower bound by examining the condensed component graph H structure. In this case, computing the total cost for activating all the vertices in the graph H (step (6) of Algorithm 1) is a sub-problem. As H is a condensed component graph of G', H is directed and acyclic. Therefore, we apply the algorithm proposed by (Günneç et al., 2020a) to get an optimal solution of the LCIP in DAGs in linear time.

The proof of correctness of our combinatorial relaxation follows.

Lemma 1. *Let G' be a sub-graph of a not strongly connected graph G and H be the condensed component graph of G'. If $\alpha = 1$, the optimum of the LCIP on H is a lower bound of the optimum of the LCIP on G'.*

Proof. Let $\underline{\mathbf{y}}$ and $\mathbf{y}^*$ be the optimal solutions to the LCIP on H and G', respectively. We will prove that for each $v \in V(H)$ and its associated s.c.c. C_v of G', we have that

$$\underline{y_v} \leq \sum_{i \in V(C_v)} y_i^*.$$

If v is a source in H, we are obligated to pay the threshold value of at least one vertex in C_v. Hence, it follows that

$$\sum_{i \in V(C_v)} y_i^* \geq \min_{i \in V(C_v)} \{t_i\} = t_v = \underline{y_v}.$$

Now suppose that v is not a source in H. Let G^* be the propagation graph associated with $\mathbf{y}^*$ and let G_v^* be the sub-graph of G^* induced by $V(C_v)$. Since the propagation graph is acyclic, there must exist a vertex $j \in V(C_v)$ that is a source in G_v^*. Hence,

$$\sum_{i \in V(C_v)} y_i^* \geq y_j^* \geq \max\left\{0, t_j - \sum_{u \in N_v} d_{uv}\right\} \geq \max\left\{0, t_v - \sum_{u \in N_v} d_{uv}\right\} = \underline{y_v},$$

where the second inequality follows from the fact that j receives influence only from vertices outside of C_v, and the third inequality follows the definition of t_v.

Altogether, we conclude that

$$\sum_{v \in V(H)} \underline{y_v} \leq \sum_{v \in V(H)} \sum_{i \in V(C_v)} y_i^* = \sum_{i \in V(G')} y_i^*.$$

□

Theorem 1. *The solution obtained by the Algorithm 1 is a lower bound for the* LEAST COST INFLUENCE PROBLEM *in the sub-graph G'.*

Proof. For the first case, the result is direct by the observations at the beginning of this section. The case in which G' is not strongly connected and $\alpha = 1$ (step 6 of the algorithm) holds by Lemma 1. □

Below we check that our algorithm has polynomial time complexity.

Theorem 2. *Given a directed graph G with n vertices and m arcs, Algorithm 1 takes $O(n+m)$ time.*

Proof. To construct the condensed component graph, we can use Kosaraju-Sharir's algorithm (Sharir, 1981), which runs in time $O(n+m)$. Line 3 can be executed in $O(n)$ time, and line 6 in $O(n+m)$. □

4.1 The case of $\alpha < 1$

Algorithm 1 is general and finds a dual bound for every case of the LCIP. However, there is room for improvement in the case where $\alpha < 1$ and the sub-graph G' is not strongly connected. The idea is to do something similar to the case of $\alpha = 1$ and solve the problem on the condensed component graph, which leads to a new combinatorial problem, as we explain next.

To improve the lower bound for the case in which we are not interested in achieving 100% adoption, we propose assigning a weight to each vertex of the condensed component graph H and using the total weight of the activated vertices to satisfy the portion of achieved vertices on the original problem. In this way, we arrive at a new variant of the LCIP, defined in Problem 2.

To assign weights to the vertices, we proceed as follows. Let H be the condensed component graph of G'. For each $v \in V(H)$, define a weight $w(v) = |V(C_v)|$, where $V(C_v)$ is the set of vertices in the component C_v of G' associated with the vertex v of H and fix a value $\kappa = \lceil \alpha |V(G)| \rceil$.

Problem 2 (Weighted Least Cost Influence Problem (WLCIP)). *Given a directed graph H with weight $w(v)$ on each vertex $v \in V(H)$, weight of influence $d_{uv} > 0$ on the arcs $(u,v) \in E(H)$, and an integer κ. Find a vector $\mathbf{y} \in \mathbb{Z}^{|V|}$ of incentives which minimizes the sum $\sum_{v \in V(H)} y_v$, ensuring that the total weight of active vertices is at least κ by the end of the activation process.*

As we are assuming solving the WLCIP on DAGs, a natural integer linear programming formulation for this problem on DAGs follows.

$$\min \sum_{v \in V(H)} y_v$$

$$\text{s.t.} \sum_{v \in V(H)} w(v) x_v \geq \kappa, \qquad \forall v \in V(H), \tag{8}$$

$$\sum_{u \in N_v} d_{uv} x_u + y_v \geq t_v x_v, \qquad \forall v \in V(H), \tag{9}$$

$$x_v \in \{0,1\}, \qquad \forall v \in V(H), \tag{10}$$

$$y_v \in \mathbb{N}_0, \qquad \forall v \in V(H), \tag{11}$$

where the binary decision variables x_v indicate that v is active, and the integer variable $y_v \geq 0$ represents the incentive assigned to v.

In this model, the objective function minimizes the incentives paid to the vertices of graph H. The cover constraints in (8) ensure that the total weight of the active vertices is at least the parameter κ. Note that the value of κ relates to the original graph G because we still want to achieve the portion α of the original network. If we use $\kappa = \lceil \alpha |V(G')| \rceil$ instead, we have a valid lower bound, but with $\lceil \alpha |V(G)| \rceil$, the lower bound can be higher (which is better). If we have $\kappa > \sum_{v \in V(H)} w(v)$ for some reason during the branch-and-bound search, the problem (WLCIP) is infeasible, and we must cut off the current node. Constraints in (9) respect the thresholds, the total of influence coming from active neighbors of v together with an incentive need to be at least the threshold of v if it is active. Constraints (10) and (11) ensure the integrality of the variables.

When $\alpha = 1$, we have a restricted case of the WLCIP in which we want to activate all the vertices. Therefore, we can ignore the weights of vertices of graph H, and the problem becomes the LCIP again.

Theorem 3 states that an optimal solution of the WLCIP on H provides a valid lower bound for the LCIP on G'.

Theorem 3. *Let G' be a sub-graph of G and H a condensed component graph of G'. The optimum of the WLCIP on H is a lower bound of the optimum of LCIP on G'.*

Proof. Let $\underline{y_v}$ and y_i^* be the optimal incentive given for each $v \in V(H)$ and $i \in V(G')$ for the WLCIP and the LCIP, respectively. We will show that it is always possible to construct a feasible solution $\mathbf{y}$ for the WLCIP on H from an optimal solution of the LCIP on G', such that $\sum_{v \in V(H)} y_v \leq \sum_{i \in V(G')} y_i^*$. We will see that this is sufficient to prove what we want.

Let $A_{G'} \subseteq V(G')$ be the set of active vertices related to $\mathbf{y}^*$. Define a set of active vertices in H as $A_H = \{v \in V(H) : C_v \cap A_{G'} \neq \emptyset\}$, where C_v is the strongly connected component associated with a condensed vertex v. It means that we activate a vertex v of H if C_v has at least one active vertex. This way, we can compute the incentives to be paid as

$$y_v = \max\left\{0, t_v - \sum_{u \in N_v \cap A_H} d_{uv}\right\}$$

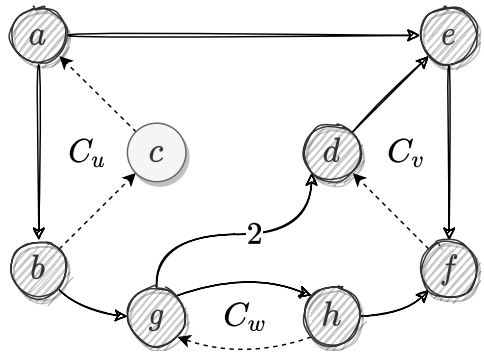
for every $v \in V(H)$.

It remains to compare the new solution $\mathbf{y}$ with $\mathbf{y}^*$ (see Figure 5 for a concrete example of the notation used in the rest of the proof). So, let G^* be the propagation graph of G' associated with $\mathbf{y}^*$ and denote as G_v^* the sub-graph of G^* induced by the vertices of $C_v \cap A_{G'}$. As G_v^* is acyclic, consider a source vertex j' of G_v^*. We have that

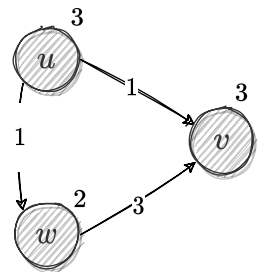
$$\begin{aligned} y_v &\leq \max\left\{0, t_{j'} - \sum_{u \in N_v \cap A_H} d_{uv}\right\} && (12)\\ &\leq \max\left\{0, t_{j'} - \sum_{i \in \{N_{j'} \setminus C_v\} \cap A_{G'}} d_{ij'}\right\} && (13)\\ &= y_{j'}^* \\ &\leq \sum_{j \in V(C_v)} y_j^*. \end{aligned}$$

Inequality 12 holds because $t_v \leq t_{j'}$ by the definition of the thresholds of H. Inequality 13 holds because

$$\sum_{u \in N_v \cap A_H} d_{uv} \geq \sum_{i \in \{N_{j'} \setminus C_v\} \cap A_{G'}} d_{ij'}. \quad (14)$$



(a) Propagation graph of the LCIP in a directed graph G' for $\alpha = 0.75$. The thresholds are $t_i = 2$, for every $i \in V(G')$ and $d_{ij} = 1$, for every arc $(i,j) \in E(G')$, except for (g,d) which is $d_{gd} = 2$.



(b) Propagation graph of the WLCIP in a condensed component graph H of G' for $\kappa = 6$. The thresholds are $t_v = 2$, for every $v \in V(H)$, the vertex and arc weights are on the figure.

Figure 5 – The solution of the LCIP in Figure 5a has value 5, the incentives are: $y_a = 2, y_b = y_g = y_h = 1$ and $y_c = y_d = y_e = y_f = 0$. The solution of the WLCIP in Figure 5b has value 3, the incentives are: $y_u = 2, y_w = 1$ and $y_v = 0$. In Figure 5a, $A_{G'} = V(G') \setminus \{c\}, G_v^* = \{\{d,e\},\{e,f\}\}$, where d is the source of G_v^*. In Figure 5b, from the solution of the LCIP on G', we have $A_H = V(H)$. Considering $j' = d$, in this example, $\{N_{j'} \setminus C_v\} \cap A_{G'} = \{g\}$. Lastly, $\sum_{u \in N_v \cap A_H} d_{uv} = d_{uv} + d_{wv} = 4$ and $\sum_{i \in \{N_{j'} \setminus C_v\} \cap A_{G'}} d_{ij'} = d_{gd} = 2$.

To prove Inequality 14, let $E_{uv} = \{(i,j) \in E(G') : i \in V(C_u) \text{ and } j \in V(C_v)\}$ be the set of arcs going from C_u to C_v. So, by the definition of the weight of the arcs of H, we have that

$$\sum_{u \in N_v \cap A_H} d_{uv} = \sum_{u \in N_v \cap A_H} \sum_{(i,j) \in E_{uv}} d_{ij} \tag{15}$$

$$\geq \sum_{u \in N_v \cap A_H} \sum_{i \in \{N_{j'} \cap C_u\} \cap A_{G'}} d_{ij'} \tag{16}$$

$$= \sum_{i \in \{N_{j'} \setminus C_v\} \cap A_{G'}} d_{ij'} \tag{17}$$

for a source j' of G_v^*.

Therefore,

$$\sum_{v \in V(H)} \underline{y_v} \leq \sum_{v \in V(H)} y_v \leq \sum_{v \in V(H)} \sum_{i \in V(C_v)} y_i^* = \sum_{i \in V(G')} y_i^*,$$

where the first inequality comes from the optimality of $\underline{\mathbf{y}}$. The second inequality holds by Inequalities 12 and 13. □

We are interested in solving the WLCIP in condensed component graphs in this work. Despite this, the problem is general and is defined for any directed graph. It is a generalization of the LCIP where, for each vertex i of a given input graph, a weight $w(i)$ is attached to it. We can see the LCIP as a particular case where all vertices have the same weight. So, as the LCIP is NP-hard,

WLCIP is NP-hard as well. Besides being difficult to solve in general cases, unfortunately, this problem is difficult to solve in DAGs, as stated in the following theorem.

Theorem 4. *WLCIP is NP-hard on directed acyclic graphs.*

Proof. Suppose that there is a polynomial-time algorithm for the WLCIP on DAGs. We show that we would be able to solve the minimization version of the knapsack problem (min-knapsack) in polynomial time in such a case. However, min-knapsack is a notorious NP-hard problem (Carnes & Shmoys, 2008; Csirik, 1991).

Let (I, c, b, B) be an instance of min-knapsack, where for each item $i \in I, c(i) > 0$ is its cost and $b(i) > 0$ its size. We aim to find a subset $J \subseteq I$ such that $\sum_{i \in J} c(i)$ is minimum and $\sum_{i \in J} b(i) \geq B$. We can assume $0 < B \leq \sum_{i \in I} b(i)$.

Create the following instance for the WLCIP. We set $V = \{v_i : i \in I\}, E = \emptyset, w(v_i) = b(i)$ and $t_{v_i} = c(i)$ for all $i \in I$. Moreover, set $\alpha = \dfrac{B}{\sum_{i \in I} b(i)}$. Note that $\alpha \sum_{v_i \in V} w(v_i) = B$. Let $A \subseteq V$ be the set of activated vertices in an optimal solution. Observe $J = \{i : v_i \in A\}$ is an optimal solution for min-knapsack. □

Even though the WLCIP is NP-hard on DAGs, demanding a somewhat elaborate mathematical approach, it deserves further consideration. If we solve the WLCIP to optimality to obtain a lower bound for $\alpha < 1$, Algorithm 1 loses scalability. Furthermore, we observed in preliminary experiments that the LP-relaxation of the formulation (8)-(11) does not provide a better lower bound than $\min_{i \in V(G')} \{t_i\}$ (line 3 of Algorithm 1). Because of this, we do not solve the WLCIP in the experiments of Section 6. Attempts should be made to find lower bounds for the WLCIP tighter than the LP-relaxation of the formulation (8)-(11). For example, such attempts could be made by a combinatorial relaxation or by the LP-relaxation of a stronger formulation.

5 BRANCHING RULE

Observe that the greater the number of s.c.c. in G', the greater the lower bound can be. To take advantage of this, we formulate a branching rule that increases the number of strongly connected components in the sub-graphs associated with the child nodes sub-problems in the branch-and-bound tree.

The idea is to give higher priority to branch on the fractional variables associated with *strong bridges* (Definition 2) or *strong articulation points* (Definition 3) of G'. In this way, when we fix in zero a variable associated with a strong articulation point (or strong bridge) and remove the corresponding vertex (or arc) of G', the number of components in G' increases. Consequently, the number of vertices of the condensed graph increases too, and, in turn, the lower bound can also increase.

Definition 2 (Strong Bridge). *A strong bridge of a directed graph G is an arc whose removal increases the number of strongly connected components of G.*

Definition 3 (Strong Articulation Point). *A strong articulation point of a directed graph G is a vertex whose removal increases the number of strongly connected components of G.*

All the strong bridges and strong articulation points can be computed in $O(|V|+|E|)$ time for a directed graph $G=(V,E)$ (Italiano et al., 2012).

Algorithm 2 describes a branching rule based on these concepts. To describe it, we define the following notation. Denote by S the set of all *connectivity cuts* of G, that is, S contains all strong bridges and strong articulation points. Also, let $sc(G,s)$ represent the number of strongly connected components generated by removing s from G', where $s \in S$ can be an arc or a vertex of G. For a solution $(\hat{\mathbf{x}},\hat{\mathbf{y}},\hat{\mathbf{z}})$ of the continuous relaxation of a sub-problem (or feasible region) P, the set $F=\{i\in V(G):\hat{x}_i\notin\mathbb{Z}\}\cup\{(i,j)\in E(G):\hat{z}_{ij}\notin\mathbb{Z}\}$ contains the elements of G associated with the fractional variables of an LP-relaxation of the ILP model in Section 3. The list $\mathscr{L}$ represents the list of active nodes of the branch-and-bound tree. Each node represents a problem, and P denotes the current sub-problem.

Algorithm 2: Branching Rule

Input: G'

begin

- $S \leftarrow$ ConnectivityCuts(G')
- $S' \leftarrow S \cap F$; `/* fractional articulation cuts */`
- **if** $S' \neq \emptyset$ **then**
  - $s^* = \arg\max_{s\in S'}\{sc(G',s)\}$
  - `/* Branch on variable associated with` s^* `*/`
  - **if** $s^* \in V(G)$ **then** `/* branch on a vertex variable */`
    - $P' \leftarrow P \cap \{\hat{x}_{s^*} \leq 0\}$
    - $P'' \leftarrow P \cap \{\hat{x}_{s^*} \geq 1\}$
  - **else** `/* branch on an arc variable */`
    - $P' \leftarrow P \cap \{\hat{z}_{s^*} \leq 0\}$
    - $P'' \leftarrow P \cap \{\hat{z}_{s^*} \geq 1\}$
  - `/* Put` P' `and` P'' `in the list of active nodes */`
  - $\mathscr{L} \leftarrow \mathscr{L} \cup \{P'_i, P''_i\}$
- **else**
  - Use another branching rule.

In Algorithm 2, the first step is to compute all the strong bridges and strong articulation points. Subroutine ConnectivityCuts(G') represents this procedure which can be done using the algorithm presented in (Italiano et al., 2012). If there are fractional variables associated with the connectivity cuts, we choose the one that generates more strongly connected components (line

5). So we branch on the chosen variable by fixing it in zero for one child node and in one for the other.

6 COMPUTATIONAL EXPERIMENTS

We conducted computational experiments with the branch-cut-and-price framework SCIP 6.0 running in an Intel Core i5-3210M 2.50GHz with 4GB of RAM. We used Gurobi Optimizer 8.1 as the underlying LP-solver and implemented the algorithms in C++. The test set is composed of synthetic random directed graphs and real networks.

6.1 Synthetic graphs

As in (Fischetti et al., 2018; Günneç et al., 2020a), we use the generative model proposed by (Watts & Strogatz, 1998) for small-world random graphs. The rewiring probability parameter for the small world graph is limited to assume values $\beta \in \{0.1, 0.3\}$. The influence weights on the arcs are chosen uniformly at random from $\{1, ..., 10\}$. Let $N(\mu, \sigma)$ be a normally-distributed random variable, with mean μ and standard deviation σ. For every $i \in V$, we set $t_i = \max\{1, \min\{N(\mu, \sigma), d_i\}\}$, where $d_i = \sum_{(j,i)\in E} d_{ji}$, $\mu = 0.7d_i$ and $\sigma = \frac{d_i}{|N_i|}$. We restrict the experiments with synthetic graphs for $\alpha = \{1, 0.5, 0.1\}$.

Table 1 summarizes the difference in performance when we apply the lower bound in the branch-and-cut. Each value on the table is the average of 5 executions. Every execution generates a new graph of a given size and average degree. The first column contains the name of instances in format *n-deg* where *n* is the number of vertices, and *deg* is the average degree. The second column is the value of β, the rewiring probability of the generative random graph model. The third column is the value of α, the portion of the network to be activated in the LCIP. In the other columns, BC means the branch-and-cut algorithm using only the LP-relaxation, and BC+ means using our combinatorial relaxation to get the lower bound, including the branching rule (Algorithm 2). Next, we present the time in seconds for those that finished before the time limit. The time limit for these instances is set to 1800 seconds. Dashed cells in the column "time" indicate that the running time reached the time limit. We marked in boldface the best results. E.g., in the instance "50-4" with $\alpha = 1$, our method (BC+) required less running time to find the optimum. In column "dual bound", the higher, the better, while the contrary is in the "primal bound". Observe that our algorithm finds a better value for the dual bound in most instances. We have the corresponding average optimality gap between the dual and primal bounds in the last column. The symbol ∞ in the column "gap" denotes the gap is infinity or very large. The optimality gap is computed as defined in both solvers SCIP and Gurobi. Let l be the dual bound, and u be the primal bound. We set the gap to ∞ if $l = 0$. Otherwise, the gap is $(u - l)/l$. Recall that the values in Table 1 are averages, so the gaps on the table are the average gaps. The number in parentheses next to the infinity symbol is the number of instances where the lower bound is greater than zero.

Table 1 – Experiments with synthetic small-world graphs.

Graph	b	α	Time(s)		Dual bound		Primal bound		Gap	
			BC	BC+	BC	BC+	BC	BC+	BC	BC+
50-4	0.1	1	553.5	**400.6**	32.3	**35.7**	37.4	**35.7**	0.7	**0**
		0.5	**608.9**	1,036.1	**22.4**	21.7	**52.8**	66.9	**3.2**	4.2
		0.1	**1,288.6**	1,445.1	12.9	**11.3**	**24.7**	31.6	**1.2**	2.2
	0.3	1	**356.7**	525.6	**51.7**	48.4	51.7	51.7	**0**	0.1
		0.5	-	-	**15.2**	10.4	**74.8**	124.1	**5.9**	11.7
		0.1	1,081.6	**722.1**	9.5	**11.2**	21.2	**17.8**	∞ (4)	**0.8**
50-8	0.1	1	-	-	7.6	**18.4**	402.0	**389.7**	∞ (3)	**21.8**
		0.5	-	-	0	**18.4**	**284.2**	301.1	∞ (0)	**16.8**
		0.1	-	-	0.4	**18.4**	87.8	87.8	∞ (1)	**4**
	0.3	1	-	-	1.0	**16.2**	**339.5**	393.8	∞ (1)	**23.7**
		0.5	-	-	0	**15.9**	347.7	**325.1**	∞ (0)	**19.6**
		0.1	-	-	0	**15.9**	**75.4**	78.1	∞ (0)	**4**
75-4	0.1	1	1,468.3	**1,125.3**	29.1	**32.0**	**45.5**	48.5	**1.1**	1.7
		0.5	**1,149.9**	1,306.4	**22.9**	20.9	**36.1**	47.0	**0.9**	2.2
		0.1	-	**1,442.9**	2.4	**10.4**	40.8	**34.3**	∞ (3)	**2.6**
	0.3	1	982.1	**839.7**	64.1	**65.8**	95.6	**86.4**	1.2	**0.6**
		0.5	-	-	16.6	**22.7**	131.5	**94.2**	12.2	**4.8**
		0.1	**1,084.4**	1,092.4	11.7	**12.0**	28.5	**28.2**	1.8	1.8
75-8	0.1	1	-	-	6.2	**14.6**	**538.3**	560.4	∞ (2)	**37.2**
		0.5	-	-	0	**14.6**	**416.5**	449.7	∞ (0)	**29.8**
		0.1	-	-	0	**14.6**	124.9	**119.8**	∞ (0)	**7.2**
	0.3	1	-	-	11.4	**17.4**	**678.0**	685.8	∞ (3)	**41.8**
		0.5	-	-	0	**14.6**	**441.1**	444.8	∞ (0)	**29.4**
		0.1	-	-	0	**14.6**	**105.4**	108	∞ (0)	**6.4**
75-12	0.1	1	-	-	0	**35.7**	781.9	**779.2**	∞ (0)	**20.9**
		0.5	-	-	0	**35.7**	683.6	**663.5**	∞ (0)	**17.5**
		0.1	-	-	0	**35.7**	238.4	**235.1**	∞ (0)	**5.6**
	0.3	1	-	-	0	**32.7**	**818.9**	882.3	∞ (0)	**29**
		0.5	-	-	0	**32.7**	839.3	**813.1**	∞ (0)	**26.1**
		0.1	-	-	0	**32.7**	235.9	235.9	∞ (0)	**6.8**
100-4	0.1	1	-	-	18.4	**18.6**	**197.3**	230.4	**11.4**	13.4
		0.5	-	-	12.4	**12.6**	**121.8**	180.0	∞ (4)	**16.7**
		0.1	**678.4**	1,445.7	**12.1**	8.85	**23.6**	35.7	∞ (4)	**3**
	0.3	1	-	**1,793.6**	47.6	**50.7**	188.3	**112.0**	2.84	**1.5**
		0.5	-	-	**17.6**	16.27	**211.5**	254.5	**13.8**	17.6
		0.1	**1,442.3**	1,442.6	8.5	**11.6**	**39.1**	41.2	∞ (4)	**3**
100-8	0.1	1	-	-	13.3	**18.7**	859.4	**835.8**	∞ (3)	**48.0**
		0.5	-	-	0	**15**	**422.3**	466.6	∞ (0)	**30.6**
		0.1	-	-	0	**15**	149.7	149.7	∞ (0)	**9**
	0.3	1	-	-	3.8	**16.5**	**986.2**	1,010.6	∞ (2)	**60.2**
		0.5	-	-	0	**16.5**	687.2	**671.0**	∞ (0)	**39.6**
		0.1	-	-	0	**16.5**	147.6	147.6	∞ (0)	**8**
100-12	0.1	1	-	-	0	**37.7**	1136.3	**1099**	∞ (0)	**28.2**
		0.5	-	-	0	**37.7**	771.5	771.5	∞ (0)	**19.4**
		0.1	-	-	0	**37.7**	275.4	275.4	∞ (0)	**6.3**
	0.3	1	-	-	0	**25.2**	**1,166.6**	1,178.1	∞ (0)	**48.8**
		0.5	-	-	0	**25.2**	**1,075.4**	1,075.6	∞ (0)	**44.5**
		0.1	-	-	0	**25.2**	279.8	279.8	∞ (0)	**10.8**
100-16	0.1	1	-	-	0	**47.1**	**1,513.5**	1518	∞ (0)	**31.4**
		0.5	-	-	0	**47.1**	1281	1281	∞ (0)	**26.2**
		0.1	-	-	0	**47.1**	436.6	436.55	∞ (0)	**8.3**
	0.3	1	-	-	0	**50.1**	1,725.6	**1,698.3**	∞ (0)	**32.9**
		0.5	-	-	0	**50.1**	1,461.3	1,461.3	∞ (0)	**28.3**
		0.1	-	-	0	**50.1**	459.9	459.9	∞ (0)	**8.2**

The results exhibited in Table 1 illustrate our algorithm behavior for graphs of different sizes and densities. Our algorithm is not effective when the graphs are small (50-4, for example). In some cases, the running time is worse than the branch-and-cut using only the LP-relaxation (BC). However, as the density and the number of vertices increases, our algorithm (BC+) achieves better gaps. While the BC has lower bounds equal to zero in many instances, BC+ always provides a lower bound greater than zero, contributing to smaller gaps. In the vast majority of the instances, our algorithm provides smaller gaps.

6.2 Real world networks

We also performed experiments with real-world social networks to demonstrate the effects of applying the lower bound algorithm on real data. We used the datasets of the Koblenz Network Collection (Kunegis, 2013), human social network category.

Table 2 shows a short description of each social network used here. For each network, n is the number of vertices, and m is the number of arcs. The weights on the arcs are the original weights of the networks. On graphs with no arcs weights, we set the weights to 1. Lastly, the threshold t_i, for each vertex i, is defined in the same way as in the synthetic graphs (see Section 6.1).

Table 2 – Real world social networks.

Network	n	m	Description
High School	70	366	Contains friendships between boys in a small high school in Illinois. Arc weights show how often a boy chose another boy as a friend.
Residence	217	2,672	Contains friendship ratings between residents living at a residence hall. Arcs are weighted from strongest to weakest tie from 5 to 1.
Physicians	241	1,098	Captures innovation spread among physicians. Arcs are weightless.
Wiki-vote	889	2,914	Wikipedia who-votes-on-whom network. Vertices represent users, and an arc (i, j) represents that user i voted on user j.
Adolescent	2,539	12,969	Each student lists their ten best friends. Higher edge weights indicate more interactions between two students.
Advogato	6,539	47,135	The trust network of the online platform Advogato. The weight on arcs models the level of trust between the individuals.
DBLP	12,591	49,728	Citation network of DBLP. A database of scientific publications. Each arc represents a citation of a publication by another publication.
Cora Citation	23,166	91,500	Cora citation network. Vertices represent scientific papers. An arc between two vertices indicates that a vertex cites another.

Table 3 summarizes the results for real-world social networks. Dashed cells indicate that the running time reached the time limit (1800 seconds). Also, we enabled all the presolving methods of the SCIP framework for both algorithms BC and BC+. These presolving methods provide

gains in running time or gap reduction for our algorithm, except in the Residence hall and Advogato networks. In the column entitled "dual bound", the higher the number, the better. This column shows that our algorithm provides higher lower bounds in the majority of the networks for different values of α. It implies gains in the running time or gap reduction.

Table 3 – Experiments on real world based social networks for $\alpha = \{1, 0.5, 0.1\}$.

Network	α	Time(s)		Dual bound		Primal bound		Gap	
		BC	BC+	BC	BC+	BC	BC+	BC	BC+
	1.0	1.2	**0.1**	18	18	18	18	0	0
High School	0.5	-	**173.3**	0	**6**	21	**6**	106.6	**0**
	0.1	**29.2**	-	3	3	**3**	9	**0**	2
	1.0	-	-	18	18	**42**	114	**1.3**	5.3
Residence hall	0.5	-	-	0	**6**	408	408	∞	**67**
	0.1	-	-	0	**6**	60	60	∞	**9**
	1.0	-	-	**139.5**	138.4	162	**157.5**	0.16	**0.14**
Physicians	0.5	-	-	8.4	**15.5**	229.5	**81**	26.3	**4.2**
	0.1	-	**300.2**	4.9	**13.5**	18	**13.5**	2.7	**0**
	1.0	0.3	**0.1**	3,692	3,692	3,692	3,692.0	0	0
Wiki-vote	0.5	-	-	**221.3**	213.3	**273**	286	**0.23**	0.34
	0.1	**336.3**	472.3	52	52	52	52	0	0
	1.0	-	**38.9**	656.3	**661.5**	1,800.8	**661.5**	1.7	**0**
Adolescent	0.5	-	-	0	**5.3**	94.5	94.5	∞	**17**
	0.1	-	-	0	**5.3**	210	210	∞	**39**
	1.0	-	-	394,380	**395,640**	**997,794**	1,069,488	**1.5**	1.7
Advogato	0.5	-	-	0	**126**	801,864	801,864	∞	**6,363**
	0.1	-	-	0	**126**	82,404	82,404	∞	**653**
	1.0	17.4	**8.7**	40,645.5	40,645.5	40,645.5	40,645.5	0	0
DBLP	0.5	-	-	1,268.5	1,268.5	25,122	25,122	18.8	18.8
	0.1	-	-	51.1	51.1	15,128.5	15,128.5	294.8	294.8
	1.0	-	-	624,899	**624,954**	625,416	**625,086**	0	0
Cora Citation	0.5	-	-	80,096	80,096	1,643,268	1,643,268	19.5	19.5
	0.1	-	-	3,365.5	3,365.5	135,696	135,696	39.3	39.3

Regarding the primal bounds, the reader can see that the gains of BC+ are not so expressive compared to BC. Despite that, the gains with the lower bounds outweigh the losses with the primal bound. Moreover, lower bounds are commonly used to approximate the optimality gap of heuristic methods. Note that for three networks (Wiki-vote, DBLP, Cora Citation), there is a small benefit in applying the dual bound algorithm, i.e., the performance of the branch-and-cut is almost the same whether using the lower bound or not. These networks have in common that the problem was entirely solved in the root node of the branch-and-bound tree for both algorithms BC and BC+. We believe this happens because such networks are more sparse than the others. Thus no branching is performed on the variables, and the lower bound algorithm has no chance to exploit the connectivity of the sub-graphs. In this way, when there are few changes in the

structures of the sub-graphs obtained from the branches, it is expected that our algorithm cannot increase the dual bounds significantly.

7 CONCLUSION

We proposed and analyzed an algorithm to compute a lower bound for the Least Cost Influence Problem based on particular properties of the problem. In addition to the algorithm itself, some auxiliary strategies proved to be helpful to increase the lower bounds. We designed custom branching strategies to strengthen the lower bounds by using strong bridges and strong articulation points. We also provide computational experiments on large social networks to check for practical applicability, showing that our algorithm can out-perform the linear programming relaxation.

Our results show that the subject should be approached carefully, and we envision some space for improvements. For example, it is possible to improve the experimental results by finding a relaxed solution such that its value corresponds to the lower bound found by our algorithm. Also, in dense graphs, we observed that the bounds behave better when $\alpha = 1$ than when $\alpha < 1$. However, when we are not interested in influencing all the network individuals and the sub-graphs are not strongly connected, the task becomes significantly more complex. In this case, getting higher lower bounds implies solving a new NP-hard problem named Weighted Least Cost Influence Problem. Because of this, it may be preferable to keep the algorithm efficient and straightforward. Despite the theoretical conclusions about the WLCIP, we do not rule out the possibility of finding other methods for obtaining better dual bounds efficiently in the case of $\alpha < 1$. Therefore, seeking new ways to approach this case is an open question in this study. To conclude, we believe that our theoretical findings on the WLCIP can open up the path for research on deriving a strong formulation for the WLCIP and finding combinatorial algorithms to solve it.

Acknowledgements

The present article is an expanded version of the conference paper (de Melo et al., 2020).

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, CNPq (Proc. 314366/2018-0, 425340/2016-3) and FAPESP (Proc. 2015/11937-9).

References

ACKERMAN E, BEN-ZWI O & WOLFOVITZ G. 2010. Combinatorial model and bounds for target set selection. *Theoretical Computer Science*, **411**(44-46): 4017–4022.

BEN-ZWI O, HERMELIN D, LOKSHTANOV D & NEWMAN I. 2011. Treewidth governs the complexity of target set selection. *Discrete Optimization*, **8**(1): 87–96.

CARNES T & SHMOYS D. 2008. Primal-dual schema for capacitated covering problems. In: *International Conference on Integer Programming and Combinatorial Optimization*. pp. 288–302. Springer.

CHEN N. 2009. On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, **23**(3): 1400–1415.

CHEN W, WANG C & WANG Y. 2010a. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 1029–1038. ACM.

CHEN W, WANG Y & YANG S. 2009. Efficient influence maximization in social networks. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 199–208. ACM.

CHEN W, YUAN Y & ZHANG L. 2010b. Scalable influence maximization in social networks under the linear threshold model. In: *2010 IEEE International Conference on Data Mining*. pp. 88–97. IEEE.

CHIANG CY, HUANG LH, LI BJ, WU J & YEH HG. 2013. Some results on the target set selection problem. *Journal of Combinatorial Optimization*, **25**(4): 702–715.

CORDASCO G, GARGANO L, RESCIGNO AA & VACCARO U. 2015. Optimizing spread of influence in social networks via partial incentives. In: *International Colloquium on Structural Information and Communication Complexity*. pp. 119–134. Springer.

CSIRIK J. 1991. Heuristics for the 0-1 min-knapsack problem. *Acta Cybernetica*, **10**(1-2): 15–20.

DE MELO RS, VIGNATTI AL, MIYAZAWA FK & OTA MJ. 2020. Tighter Dual Bounds on the Least Cost Influence Problem. In: *Proceedings of the 52nd Brazilian Operational Research Symposium*. SBPO.

DEMAINE ED, HAJIAGHAYI MT, MAHINI H, MALEC DL, RAGHAVAN S, SAWANT A & ZADIMOGHADAM M. 2014. How to influence people with partial incentives. In: *Proceedings of the 23rd International Conference on World Wide Web*. pp. 937–948. ACM.

FISCHETTI M, KAHR M, LEITNER M, MONACI M & RUTHMAIR M. 2018. Least cost influence propagation in (social) networks. *Mathematical Programming*, pp. 1–33.

GRANOVETTER M. 1978. Threshold models of collective behavior. *American Journal of Sociology*, pp. 1420–1443.

GRÖTSCHEL M, JÜNGER M & REINELT G. 1985. On the acyclic subgraph polytope. *Mathematical Programming*, **33**(1): 28–42.

GÜNNEÇ D, RAGHAVAN S & ZHANG R. 2020a. A branch-and-cut approach for the least cost influence problem on social networks. *Networks*, **76**(1): 84–105.

GÜNNEÇ D, RAGHAVAN S & ZHANG R. 2020b. Least-cost influence maximization on social networks. *INFORMS Journal on Computing*, **32**(2): 289–302.

ITALIANO GF, LAURA L & SANTARONI F. 2012. Finding strong bridges and strong articulation points in linear time. *Theoretical Computer Science*, **447**: 74–84.

KEMPE D, KLEINBERG J & TARDOS É. 2003. Maximizing the spread of influence through a social network. In: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 137–146. ACM.

KUNEGIS J. 2013. KONECT – The Koblenz Network Collection. In: *Proc. Int. Conf. on World Wide Web Companion*. pp. 1343–1350. Available at: http://userpages.uni-koblenz.de/~kunegis/paper/kunegis-koblenz-network-collection.pdf.

NEMHAUSER GL, WOLSEY LA & FISHER ML. 1978. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, **14**(1): 265–294.

RAGHAVAN S & ZHANG R. 2019. A branch-and-cut approach for the weighted target set selection problem on social networks. *INFORMS Journal on Optimization*, **1**(4): 304–322.

ROGERS EM. 2010. *Diffusion of innovations*. Simon and Schuster.

SHARIR M. 1981. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, **7**(1): 67–72.

WATTS DJ & STROGATZ SH. 1998. Collective dynamics of 'small-world' networks. *Nature*, **393**(6684): 440.

How to cite

MELO RS, VIGNATTI AL, MIYAZAWA FK & OTA MJ. 2023. Combinatorial Dual Bounds on the Least Cost Influence Problem. *Pesquisa Operacional*, **43**: e275468. doi: 10.1590/0101-7438.2023.043.00275468.