Received: 9 June 2019 | Revised: 26 December 2019 | Accepted: 20 March 2020 | Published on: 10 April 2020

DOI: 10.1002/net.21941

RESEARCH ARTICLE

WILEY

A branch-and-cut approach for the least cost influence problem on social networks

Dilek Günneç[1] | **S. Raghavan**[2] | **Rui Zhang**[3]

[1]Department of Industrial Engineering, Ozyegin University, Istanbul, Turkey
[2]Robert H. Smith School of Business, Institute for Systems Research, University of Maryland, College Park, Maryland
[3]Leeds School of Business, University of Colorado, Boulder, Colorado

Correspondence
Rui Zhang, Leeds School of Business, University of Colorado, Boulder, CO 80309.
Email: rui.zhang@colorado.edu

Abstract

This paper studies a problem in the online targeted marketing setting called the least cost influence problem (LCIP) that is known to be NP-hard. The goal is to find the minimum total amount of inducements (individuals to target and associated tailored incentives) required to influence a given population. We develop a branch-and-cut approach to solve this LCIP on arbitrary graphs. We build upon Günneç et al.'s novel totally unimodular (TU) formulation for the LCIP on trees. The key observation in applying this TU formulation to arbitrary graphs is to enforce an exponential set of inequalities that ensure the influence propagation network is acyclic. We also design several enhancements to the branch-and-cut procedure that improve its performance. We provide a large set of computational experiments on real-world graphs with up to 155 000 nodes and 327 000 edges that demonstrates the efficacy of the branch-and-cut approach. This branch-and-cut approach finds solutions that are on average 1.87% away from optimality based on a test-bed of 160 real-world graph instances. We also develop a heuristic that prioritizes nodes that receive low influence from their peers. This heuristic works particularly well on arbitrary graphs, providing solutions that are on average 1.99% away from optimality. Finally, we observe that partial incentives can result in significant cost savings, over 55% on average, compared to the setting where partial incentives are not allowed.

KEYWORDS

exact method, influence maximization, integer programming, strong formulation

1 | INTRODUCTION

Online communication (through social networks, newspapers, blogs, shopping websites, etc.) has become one of the main resources for information sharing. A recent report (see [27]) shows that in the United States people consider online social networks to be one of the most effective ways for disseminating information, and two-thirds of the population use their online social networks as one of the channels for receiving information and news. Not surprisingly, people's decisions are affected by the information they receive through social media. While peer influence has been recognized as a role exerting an important impact in decision-making for a long time (see e.g., [1,2,9]), online social media provide a much easier and more convenient way to track the interaction of online customers based on their footprints. It opens an opportunity for researchers to understand social networks and manage their effects on purchasing decisions. The outcomes can be used as an essential part of creating successful online marketing strategies. As a result, there is an increasing interest in correctly identifying (targeting) customers that are most likely to help the spread of a product (or information) over a social network.

Indeed, Chen [3] initiated a stream of work in this area focused on identifying the fewest number of nodes to target in order to influence an entire network. However, the mathematical models studied by Chen and many other researchers for such influence maximization problems (IMPs) suffer from a significant practical shortcoming. They restrict the marketer to interventions where those selected for targeting receive the (entire) product gratis. Motivated by practical considerations the least cost influence

problem (LCIP) considered in this paper allows for individuals to be partially influenced by the use of tailored incentives (e.g., coupons that reduce the price of a product instead of receiving the product for free). The use of tailored (i.e., partial) incentives which allows for differentiated targeting is more natural (and prevalent) in a marketing setting. Thus the LCIP we study is considered in a deterministic setting, and seeks to minimize the cost of tailored incentives provided to individuals in a social network while ensuring that the entire network is "influenced."

From the algorithmic perspective, most of the previous work on IMPs is devoted to approximation algorithms and heuristics. In this paper, we focus on developing a branch-and-cut approach for the LCIP that can find high-quality solutions and bounds for large real-world networks.

1.1 | Problem definition

The LCIP was introduced by Günneç and Raghavan [11] in a product design and diffusion setting. Consider a social network represented as an undirected graph $G = (V, E)$, where node set $V = \{1, 2, \ldots, n\}$ denotes the set of individuals in the network and edge set E shows the connections between individuals on the social network. Let $a(i)$ be the set of node i's neighbors, and $deg(i) = |a(i)|$ denote its degree. Following a well-studied linear threshold model on the diffusion of innovations (see [9]), we use the term active if a node has adopted the product and the term inactive if it has not adopted the product. In the threshold model, for each node in the network, $i \in V$, there is a threshold, denoted by b_i. This threshold represents how easily a node can be influenced. We permit a payment p_i which is the tailored incentives for a node $i \in V$. Each inactive node $i \in V$ is influenced by an amount d_i (referred to as the *influence factor*) by each of its neighboring nodes in $a(i)$ that are active (i.e., have already adopted the product). The assumption that all neighbors of a node exert equal influence is relevant especially when privacy concerns are present in social networks (an issue which has become increasingly important with regulators both in North America and Europe). In this way, the influence does not depend on the identity of the neighbor and the information on the strength of the relationship. Although an individual is affected equally by each neighbor, this influence factor may be different for each individual.

All nodes are inactive initially. Then, we decide the tailored incentives p_i for each node $i \in V$. Now, a node i becomes active immediately if $p_i \geq b_i$. That is, if the payment is greater than or equal to the threshold. Under linear scaling it is without loss of generality that the units of the payments (typically monetary units) and threshold (typically utility) are equivalent. For example if p_i payment units at node i were equivalent to $\gamma_i p_i$ threshold units at the node, then we would simply scale the threshold b_i and incoming influence d_i by dividing them by γ_i. After deciding the initial incentives, in each step, we update the states of nodes by the following rule: an inactive node i becomes active if the sum of the tailored incentive p_i and the total influence coming from its active neighbors is at least b_i. The process continues until there is no change in the state of the network (i.e., no additional nodes are becoming active). The goal is to find the minimum total payment (i.e., $\sum_{i \in V} p_i$) while ensuring the entire network is activated by the end of this process.

Note that the assumption that all nodes are inactive is without loss of generality. If some nodes were active at the outset, we can propagate their influence and reduce the problem to a smaller one where all nodes are inactive initially. We also note that in a deterministic setting there is no benefit to delaying the payment of the tailored incentive. Hence, all incentives paid to a node i may be viewed as being paid at the outset of the process.

1.2 | Related literature

Günneç et al. [12] considered the LCIP in the setting where neighbors of a node may exert unequal influence and where the goal is to influence only a given proportion (α) of the network and showed it to be NP-hard. They also addressed the complexity of the LCIP for a variety of special cases. In particular, when $0 < \alpha < 1$ they showed that even when neighbors of a node exert equal influence and the graph is bipartite the problem remains NP-hard. When the entire network must be influenced ($\alpha = 1$) and neighbors of a node exert equal influence (i.e., the LCIP as defined in Section 1.1) they showed the problem to be APX-hard. When $\alpha = 1$ and neighbors exert unequal influence they showed the LCIP to be NP-hard on trees. They then showed that the LCIP (as defined in Section 1.1) on tree networks is polynomially solvable—by describing two polynomial time algorithms as well as a totally unimodular (TU) formulation that only applies to the LCIP on trees. Cordasco et al. [4] studied the LCIP on complete graphs and trees under the assumption of an equal influence factor for the entire network, $d_i = d \; \forall i \in V$, and provided a nontrivial polynomial time algorithm for these two cases.

Subsequent to an earlier (working) version of our paper, Fischetti et al. [7] considered the LCIP in a general setting where neighbors of a node may exert unequal influence, the entire network need not be influenced (i.e., $0 < \alpha \leq 1$), and the influence structure can be nonlinear. They proposed a novel set covering based formulation for this version of the LCIP. Their formulation has an exponential number of variables as well as an exponential number of constraints to address these three aforementioned

issues. Using this formulation, they described two approaches. One is a branch-and-cut approach where all variables are enumerated first. This approach does not scale well since it needs all variables from the outset. The other one is a price-cut-and-branch approach that dynamically generates both columns (variables) and cuts (constraints). While it can be applied to larger problems, the price-cut-and-branch is a heuristic that is not guaranteed to solve the problem to optimality (the dual bounds obtained after the root node are not valid since the branching phase does not allow for the addition of columns in this approach; i.e., only the lower bound obtained at the root node is valid). In this more general setting, they are only able to apply their approach to simulated graph instances with up to 100 nodes with an average degree up to 16. Their approach finds optimal solutions when the average degree is 4, but the quality of the solutions rapidly deteriorates when the average degree increases. The optimality gap reaches a maximum of 94.8% when the average degree is 16. When they apply their formulation to the LCIP as considered in this paper (i.e., as defined in Section 1.1) they are able to address simulated graph instances with up to 100 000 nodes with an average degree of 4 obtaining solutions ranging from 0% to 53.2% from optimality.

The weighted target set selection (WTSS) problem is closely related to the LCIP. The difference is that the WTSS problem requires all nodes selected for targeting to be paid their threshold amount b_i (i.e., $p_i = b_i$ for nodes selected for targeting) whereas the LCIP allows for partial payments (i.e., $0 < p_i \leq b_i$ for nodes selected for targeting). Since the WTSS problem does not allow for partial payments, often the definition provides a critical value $g_i = \left\lceil \frac{b_i}{d_i} \right\rceil$ for each node, instead of the influence factor for each node. When all nodes have the same threshold (i.e., $b_i = b$, $\forall i \in V$) we obtain the (unweighted) TSS problem introduced by Chen [3]. Chen showed that the TSS problem is hard to approximate within a polylogarithmic factor. He also provided a polynomial algorithm for the TSS on trees. Raghavan and Zhang [24] described the polytope of the WTSS problem on trees and cycles. Raghavan and Zhang [23] presented a branch-and-cut approach for the WTSS problem on arbitrary graphs and apply it to 180 real-world graph instances (with up to approximately 155 000 nodes and 327 000 edges). Their branch-and-cut approach finds solutions that are on average 0.90% from optimality, and solves 60 out of the 180 instances to optimality.

Kempe et al. [15] were the first researchers to consider the IMP from an algorithmic perspective. They considered a budgeted version of the TSS problem (i.e., given a budget of k seed products identify the k individuals to target so as to maximize the adoption of the product in the social network) in a randomized setting and showed it is NP-hard to find the optimal initial set. Based on the submodularity property of the objective function (which is due to the particular randomized assumption in the problem data they make) they developed a $\left(1 - \frac{1}{e}\right)$-approximation algorithm for the problem. Leskovec et al. [19] developed a much faster algorithm by using the submodularity property to reduce the number of evaluations on the influence propagation outcome of a seed node. Many algorithmic approaches have been developed on several extensions of the IMP including cases where influence is limited to the nodes that are within a prefixed number of hops from the seed node (see [6]). More recently, Nguyen et al. [22] proposed a generalized problem, which allows nodes to have various costs for selection and have different benefits once they are selected. They presented an approximate algorithm which has $\left(1 - \frac{1}{\sqrt{e}}\right)$ guarantee for the general case and $\left(1 - \frac{1}{e}\right)$ guarantee for the case when nodes have uniform costs. They showed their algorithm can obtain a (heuristic) solution for a network with 1.5 billion edges within only a few minutes. Demaine et al. [5] considered the IMP (as defined by [15]) with the further assumption that nodes can be paid partial incentives (as in the LCIP). In theory, they showed that the fractional version of the IMP has the same computational complexity as the IMP. That is the submodularity property of the objective function holds (which is again due to the particular randomized assumption on the uniform distribution of thresholds) yielding the same $\left(1 - \frac{1}{e}\right)$-greedy approximation algorithm for the problem. In practice, they showed that the solutions of the two versions could be significantly different: the fractional allocation can influence a larger number of nodes.

1.3 | Our contributions

In this paper, we design and test a branch-and-cut approach for solving the LCIP on arbitrary graphs. We build upon the TU formulation for trees described in Günneç et al. [12]. In Section 2, we present formulations for the LCIP on arbitrary graphs. Along the way, we discuss the LCIP on directed acyclic graphs (DAGs). Section 3 discusses the branch-and-cut approach. The key observation is to enforce an exponential set of inequalities that ensure the influence propagation network is acyclic for arbitrary graphs. We also discuss several enhancements for the branch-and-cut procedure. Section 4 presents our computational experiments on a large set of real-world graphs with up to 155 000 nodes and 327 000 edges that demonstrates the efficacy of the branch-and-cut approach. The branch-and-cut approach finds solutions that are on average 1.87% away from optimality over a test-bed of 160 real-world graph instances. However, without our enhancements, the naive implementation with the state-of-the-art commercial solver cannot guarantee feasible solutions for all instances. We also find one of the heuristics developed in the paper (called "influence greedy") works particularly well on arbitrary graphs (this heuristic is an optimal procedure for the LCIP on trees). This influence greedy heuristic finds the node with the smallest influence factor

among the inactive nodes and pays it the corresponding threshold to make it active. It then updates the graph by propagating influence from the active nodes (i.e., reducing the thresholds of the inactive nodes by the amount of incoming influence). This is repeated until a feasible solution is found. Finally, we show that partial incentives can result in significant cost savings, over 55% on average, compared to the setting where partial incentives are not allowed. Section 5 provides concluding remarks.

2 | FORMULATIONS FOR THE LCIP

Günneç and Raghavan [11] introduced the following time-indexed integer programming model. It tracks the order of activation with the binary u_{it} variable which is equal to 1 if node i is active in time period t and 0 otherwise.

$$\text{TimeIndex:} \quad \text{Min} \sum_{i \in V} p_i \tag{1}$$

$$\text{Subject to:} \quad u_{i0} = 0 \quad \forall i \in V, \tag{2}$$

$$p_i + \sum_{j \in a(i)} d_i u_{j(t-1)} \geq b_i u_{it} \quad i \in V, \ \ t = 1, 2, \ldots, T, \tag{3}$$

$$\sum_{i \in V} u_{iT} = |V|, \tag{4}$$

$$u_{it} \in \{0, 1\}, \ \ p_i \geq 0 \quad \forall i \in V, \ \ t = 1, 2, \ldots, T. \tag{5}$$

In this model, T is set as $|V| - 1$ since it takes at most $|V| - 1$ periods to finish the propagation process. The objective (1) is to minimize the total incentives given. All nodes are inactive in period 0 (constraint set (2)). A node becomes active if the summation of the incentive and the influence from active neighbors is greater than or equal to the threshold for that node (constraint set (3)). Lastly, the entire network should be activated (constraint set (4)). As we will see in Section 4, this formulation is weak and even the LP relaxation cannot be solved for large networks.

Günneç et al. [12] described a formulation based on influence propagation over arcs specific to trees. We enhance that model to apply it to arbitrary graphs. In this model, ARC, binary variable y_{ij} equals 1 if node i influences node j, and 0 otherwise. The objective (6) minimizes the total payments. Constraint set (7) requires the direction of influence over an arc should either be from i to j or from j to i. Constraint set (8) states that the summation of incentives given to node i and the incoming influence from active neighbors to i should be greater than or equal to the threshold for node i. Constraint set (9), called k-dicycle inequalities, enforces the condition that the directed influence propagation network formed by $\mathbf{y}$, $G(\mathbf{y})$, must be a DAG.

$$\text{ARC:} \quad \text{Min} \sum_{i \in V} p_i, \tag{6}$$

$$\text{Subject to:} \quad y_{ij} + y_{ji} = 1 \quad \forall \{i, j\} \in E, \tag{7}$$

$$p_i + \sum_{j \in a(i)} d_i y_{ji} \geq b_i \quad \forall i \in V, \tag{8}$$

$$\sum_{\{i,j\} \in C} y_{ij} \leq |C| - 1 \quad \forall \text{ dicycles } C \text{ in } G(\mathbf{y}), \tag{9}$$

$$p_i \geq 0 \quad \forall i \in V, \tag{10}$$

$$y_{ji} \in \{0, 1\} \quad \forall j \in V, \ \ i \in a(j). \tag{11}$$

This mixed integer programming formulation has exponentially many constraints and is generally much larger than the TimeIndex model. Furthermore, its linear relaxation does not provide integral solutions on trees (note that on trees constraint set (9) is deleted) which we know to solve polynomially.

We describe a third formulation for the LCIP, HLZ, that builds on the TU formulation for trees introduced by Günneç et al. [12]. This model further characterizes the incoming influence at a given node i. If no incentives are given to a node, it can become active only if at least $g_i = \left\lceil \frac{b_i}{d_i} \right\rceil$ neighbors become active. This allows us to characterize the influence on an incoming arc to node i as follows: H with incoming influence d_i, L with incoming influence $l_i = b_i - (g_i - 1)d_i$ and Z with incoming influence 0, so that the total incoming influence to a node is exactly equal to the difference between its threshold b_i and its payment p_i.

Specifically, we need to consider two situations in this categorization. First, consider the situation where $g_i \geq 2$ for node i as the example in Figure 1. We have $b_i = 15$, $d_i = 6$, $g_i = 3$, and $l_i = 3$. There are $g_i + 1 = 4$ possible scenarios. If the node receives no payment (i.e., $p_i = 0$), it should receive incoming influence of type H on $g_i - 1 = 2$ arcs, and an incoming influence of type L on one arc. Any remaining incoming arcs are of type Z. Otherwise, the node receives a payment of $l_i + \lambda d_i$ (where

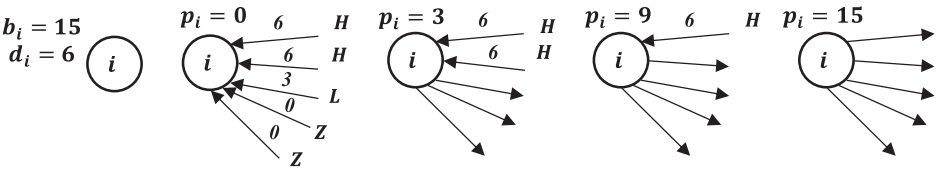

FIGURE 1 Categorization of incoming influence when $g_i \geq 2$

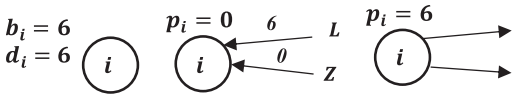

FIGURE 2 Categorization of incoming influence when $g_i = 1$

$\lambda = 0, \ldots, g_i - 1$) and has exactly $g_i - 1 - \lambda$ incoming arcs of type H. Figure 1 shows these scenarios with $p_i = 0, 3, 9, 15$, respectively. Second, consider the situation where $g_i = 1$ for node i as the example in Figure 2. We have $b_i = 6$, $d_i = 6$, $g_i = 1$, and $l_i = d_i$. There are $g_i + 1 = 2$ possible scenarios. If the node receives no payment, it must receive an incoming influence of type L on one arc. Any remaining incoming arcs are of type Z. Otherwise, the node receives a payment of $p_i = l_i$ and has no incoming arcs. Figure 2 shows these scenarios with $p_i = 0, 6$, respectively. Overall, we observe that when there is no payment to a node, there are exactly $(g_i - 1)$ arcs with the incoming influence of type H and one arc with the incoming influence of type L. When there is a payment (notice the payment set is discrete), the incoming arcs can only provide influence of type H, and there are at most $(g_i - 1)$ of them.

We use these facts to decompose the binary variable y_{ji} in ARC into three binary variables and develop the model HLZ. In this model, binary variables x^H_{ji}, x^L_{ji}, and x^Z_{ji} represent the type of influence coming over arc (j, i) into node i, and they are equal to 1 when the type is H, L, Z respectively, and 0 otherwise. The objective coefficients are $c^H_i = d_i$, $c^L_i = l_i$, $c^Z_i = 0$. The objective (12) minimizes the total payment. Constraint set (13) represents the maximum number of H arcs that can influence node i over its neighbor set $a(i)$. Constraint set (14) guarantees that there is at most one incoming arc with type L. Constraint set (15) ensures that an arc can be characterized with only one type of influence (H, L, or Z) and matches direction with that of the binary y variables. Constraint sets (16) and (17) are identical to the ARC model. Notice that when the underlying graph is a tree, constraint set (16) becomes redundant because constraint set (17) rules out all cycles with two arcs and there are no cycles with three or more arcs when the underlying graph is a tree. Thus, if we substitute constraint set (15) into constraint set (17), we obtain the TU formulation in Günneç et al. [12].

$$\text{HLZ:} \quad \text{Min} \sum_{i \in V} b_i - \sum_{i \in V} \sum_{j \in a(i)} \sum_{k \in \{H,L,Z\}} c^k_i x^k_{ji}, \tag{12}$$

$$\text{Subject to:} \quad \sum_{j \in a(i)} x^H_{ji} \leq g_i - 1 \quad \forall i \in V, \tag{13}$$

$$\sum_{j \in a(i)} x^L_{ji} \leq 1 \quad \forall i \in V, \tag{14}$$

$$\sum_{k \in \{H,L,Z\}} x^k_{ji} = y_{ji} \quad \forall i \in V,\ j \in a(i), \tag{15}$$

$$\sum_{\{i,j\} \in C} y_{ij} \leq |C| - 1 \quad \forall \text{ dicycles } C \text{ in } G, \tag{16}$$

$$y_{ij} + y_{ji} = 1 \quad \forall \{i,j\} \in E, \tag{17}$$

$$x^k_{ji} \in \{0,1\} \quad \forall i \in V,\ j \in a(i),\ k \in \{H,L,Z\}, \tag{18}$$

$$y_{ij} \in \{0,1\} \quad \forall i \in V,\ j \in a(i). \tag{19}$$

2.1 | Remark on the LCIP on DAGs

All three models presented in the previous section are defined for arbitrary graphs. We detour for a moment to comment on the LCIP on a DAG. On a DAG, we already have the direction of the arcs and thus influence propagation. For a node $i \in V$ on a DAG, let set $I(i)$ provide the nodes that have arcs directed into node i. Then, based on this information, we can apply HLZ formulation to the LCIP on DAGs. Notice that constraint sets (16) and (17) in HLZ become redundant because the influence propagation is known and is a DAG. Also, we do not need **y** variables any more. Thus, the LP relaxation (that we refer to as

LPDAG) of the resulting formulation can be obtained as below:

$$\text{LPDAG:} \quad \text{Min} \sum_{i\in V} b_i - \sum_{i\in V}\sum_{j\in I(i)}\sum_{k\in\{H,L,Z\}} c_i^k x_{ji}^k, \tag{20}$$

$$\text{Subject to:} \quad \sum_{j\in I(i)} x_{ji}^H \le g_i - 1 \quad \forall i \in V, \tag{21}$$

$$\sum_{j\in I(i)} x_{ji}^L \le 1 \quad \forall i \in V, \tag{22}$$

$$\sum_{k\in\{H,L,Z\}} x_{ji}^k = 1 \quad \forall i \in V,\ j \in I(i), \tag{23}$$

$$x_{ji}^k \ge 0 \quad \forall i \in V,\ j \in I(i),\ k \in \{H,L,Z\}, \tag{24}$$

We remove upper bound constraints $x_{ji}^k \le 1\ \ \forall i \in V, j \in I(i), k \in \{H,L,Z\}$ because they are redundant due to constraint set (23).

Theorem 1. *The constraint matrix of LPDAG is TU.*

Proof. Let A denote the constraint matrix of constraint sets (21), (22), and (23). A is a 0-1 matrix that has at most two nonzero elements in each column. We can partition the rows of A into two subsets J_1 and J_2 such that J_1 contains constraint set (21) and (22) and J_2 contains constraint set (23). Then, columns that have two nonzero elements have one of the nonzero coefficients in J_1 and one of the nonzero coefficients in J_2. Thus, A is a TU matrix due to Corollary 2.8 in Nemhauser and Wolsey ([21], p. 544). ∎

As a consequence, LPDAG optimally solves the LCIP on a DAG. In fact, the following combinatorial algorithm trivially solves the LCIP on DAGs. Consider the nodes of a DAG in any order. For a node $i \in V$ that has g_i or more incoming arcs in the DAG, set $p_i = 0$. For a node $i \in V$ that has $g_i - 1$ or fewer incoming arcs in the DAG the payment is set as $p_i = b_i - |I(i)|d_i$. It is easy to ascertain this solution from LPDAG. For a node with g_i or more incoming arcs, the amount of incoming influence will equal b_i. For a node with $g_i - 1$ or fewer incoming arcs a payment is necessary, and thus all incoming arcs have the high type of influence. We need to scan adjacent arcs of all nodes in V. Thus, the time complexity of this algorithm is $O(|E|)$.

3 | THE BRANCH-AND-CUT APPROACH

In this section, we develop a branch-and-cut approach based on HLZ for solving the LCIP. To solve the LP relaxation of HLZ, it is necessary to solve the separation problem for the exponential size set of k-dicycle inequalities (16). Grötschel et al. [10] present a separation procedure for k-dicycle inequalities (in the context of the linear ordering problem) which is based on the shortest path algorithm. We implement this separation procedure in our approach.

Separation procedure for k-dicycle inequalities: Given a current solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$, we define a weight $w_{ij} = 1 - \hat{y}_{ij}$ for each arc (i, j) in the graph. If C is a cycle, then, $\sum_{(i,j)\in C}\hat{y}_{ij} = \sum_{(i,j)\in C}(1 - w_{ij}) = |C| - \sum_{(i,j)\in C} w_{ij}$. Thus, $\sum_{(i,j)\in C}\hat{y}_{ij} \le |C| - 1$ if and only if $\sum_{(i,j)\in C} w_{ij} \ge 1$. This means we can check whether $\hat{y}$ violates the k-dicycle inequalities by finding a cycle whose weight is strictly less than 1. To do so, for each node in the graph we find the minimum cost cycle that contains it. Specifically, for each node i in V, we create a modified graph G^i by adding a dummy node i' and adding an arc between node j and node i' with $w'_{ji} = w_{ji}$ if arc (j, i) exists in G for each node j in $V\backslash\{i\}$. Then, we find a shortest path between node i and node i' on G^i. If the length of the shortest path is strictly smaller than 1, we add the corresponding k-dicycle inequality.

3.1 | Addressing symmetry

An integer program (IP) is symmetric if its variables can be permuted without changing the structure of the problem. Symmetry could cause trouble in a computational sense (i.e., the branch-and-cut approach can take a long time) because the search procedure wastes effort and time on eliminating symmetric solutions. Figures 3 and 4 show examples of two types of symmetry that can lead to multiple optimal solutions in the LCIP. We use objective perturbations to break symmetry in the problem (see [8]). The idea is simple. We modify the objective coefficients of some variables (typically those that exhibit symmetry) by a small nonnegative perturbation so that the total perturbation is strictly less than 1. Given that the rest of the problem data is integer this ensures that the set of optimal solutions for the perturbed problem is a subset of the optimal solutions for the original problem (i.e., with no perturbations). By judiciously choosing values for the perturbations (to try to create some ordering amongst the optimal solutions), the symmetry in the problem can be significantly reduced.

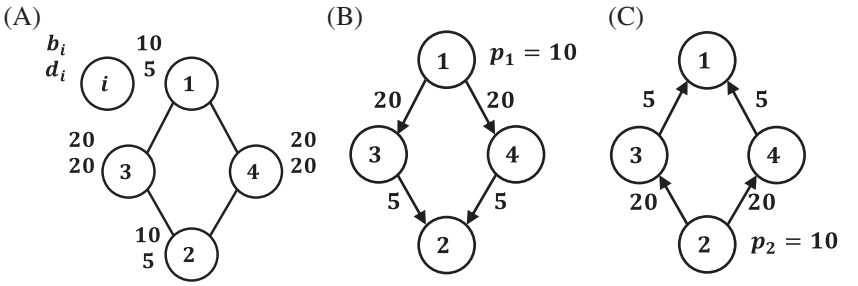

FIGURE 3 Example of influence direction symmetry. (A) An LCIP instance. (B) Solution 1. (C) Solution 2

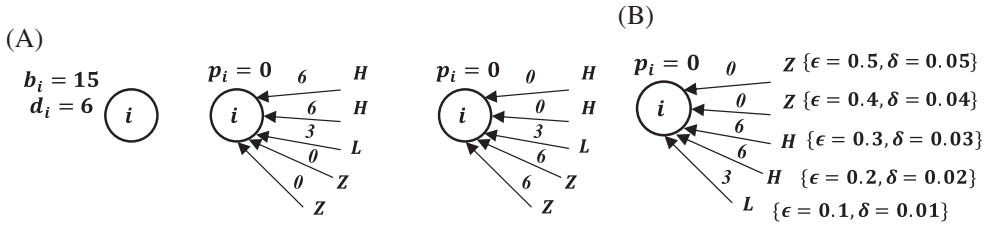

FIGURE 4 Example of influence allocation symmetry. (A) Two solutions. (B) After perturbations are subtracted

Influence direction symmetry (y perturbations): Given the LCIP instance in Figure 3, we have two solutions with the same objective value, 10, but different influence directions. In Solution 1 node 1 is paid its threshold of 10 units ($p_1 = b_1 = 10$) which causes it to become active, with the influence propagation process activating the rest of the network. The following variables are nonzero in Solution 1 to HLZ: $x^L_{13} = x^L_{14} = x^L_{32} = x^H_{42} = y_{13} = y_{14} = y_{32} = y_{42} = 1$. In Solution 2 node 2 is paid its threshold of 10 units ($p_2 = b_2 = 10$) which causes it to become active, with the influence propagation process activating the rest of the network. The following variables are nonzero in Solution 2 to HLZ: $x^L_{23} = x^L_{24} = x^L_{31} = x^H_{41} = y_{23} = y_{24} = y_{31} = y_{41} = 1$. To reduce this kind of symmetry, we provide a cost coefficient θ_{ij} to each variable y_{ij} which has a value of zero originally. The value of θ_{ij} satisfies the following three conditions:

1. $\forall\{i,j\} \in G(\mathbf{y})\ \theta_{ij} > 0$ when $j > i$ and $\theta_{ij} = 0$ otherwise,
2. all $\theta_{ij} > 0$ are distinct, and
3. $\sum_{\{i,j\}\in G(\mathbf{y})} \theta_{ij} < 1$.

These conditions ensure that all θ are nonnegative and the total perturbation is strictly less than 1. Thus, the set of optimal solutions for the perturbed problem is a subset of the optimal solutions for the original problem. Furthermore, condition 1 gives higher preference to the direction from node i to node j when $i > j$ and eliminates the symmetry between these two directions. It also fixes the value of θ_{ij} to zero when $i > j$ and allows us to focus on assigning the value to the remaining θ_{ij} where $j > i$. However, if all $\theta_{ij} > 0$ have the same value, the two solutions in Figure 3B,C will have the same objective value even after we add the perturbations. Condition 2 further reduces symmetry in this case by ensuring the perturbations are distinct. For example, if we set $\theta_{13} = 0.05$, $\theta_{14} = 0.10$, $\theta_{23} = 0.15$, $\theta_{24} = 0.20$, and all other $\theta = 0$ in Figure 3; then Solution 1 becomes the unique optimal solution. (Notice with the perturbations, Solution 1 has value 10.15 and Solution 2 has value 10.35.)

In order to satisfy these three conditions for the y perturbations, we suggest setting up the θ as follows. Consider the perturbations $\frac{1}{|E|^2}, \frac{2}{|E|^2}, \frac{3}{|E|^2}, \ldots, \frac{|E|-1}{|E|^2}, \frac{|E|}{|E|^2}$. For each edge $\{i,j\} \in E$ use one of these distinct perturbations (recall, we only use the perturbations in one of the two directions). It is easy to see that conditions 1 and 2 are satisfied with this choice. For condition 3, the maximum increment caused by θ in the objective value is $\frac{1}{|E|^2}(1+2+\cdots+|E|) = \frac{1}{|E|^2}\frac{|E|(|E|+1)}{2} = \frac{|E|+1}{2|E|} < 1$ since $|E| > 1$.

Influence allocation symmetry (x perturbations): Figure 4 gives an example with a node which allocates influences as described in Section 2. We do not pay this node any incentives. There are multiple ways (actually 30 ways!) to allocate the influence to its incoming arcs. Figure 4A displays two of them. To try and eliminate this symmetry we perturb (subtract) the cost coefficient of x^L_{ij} by ϵ_{ij} and the cost coefficient of x^H_{ij} by δ_{ij}. In particular, for a node i with $deg(i) > 1$ the perturbations satisfy the following conditions:

1. $\epsilon_{ji} > 0$, $\delta_{ji} > 0\ \forall j \in a(i)$ and ϵ_{ji} and δ_{ji} are distinct $\forall j \in a(i)$,
2. $\epsilon_{mi} + \sum_{j\in M} \delta_{ji} < \frac{1}{|V|}$ where $m = \arg\max_j\{\epsilon_{ji}\}$ is the index of the largest ϵ_{ji} and M is the set of indexes of the first $(g_i - 1)$ largest δ_{ji}, and

3. $|\epsilon_{ji} - \epsilon_{ki}| > |\delta_{ji} - \delta_{ki}|\ \forall j \neq k \in a(i)$.

The first condition ensures that all incoming arcs to a node have distinct perturbations. The second condition ensures that the largest amount of perturbation subtracted at a node in a feasible solution is strictly less than $\frac{1}{|V|}$. This is because a node has at most one L type influence arc (that subtracts an ϵ perturbation) and $(g_i - 1)$ H type influence arcs (that subtract δ perturbations). Thus, the total perturbation subtracted to a feasible (integer) solution is strictly less than 1 for the entire graph. The third condition creates an ordering among the incoming arcs for L and H type influence. It ensures that among the incoming arcs, the one with the lowest ϵ perturbation is selected as the L type arc (when the solution has an L type arc). Then, among the remaining incoming arcs, the ones with the lowest δ perturbation values are selected as the H type arcs (as many as the solution has). The rest (if any incoming arcs remain) are selected as Z type arcs. This breaks the symmetry in the problem. Consider Figure 4 as an example. In this situation, we need to assign one L type, two H type, and two Z type influences to the five incoming arcs. Consider the counter-clockwise ordering of the arcs in the figure and increase the perturbation values of ϵ and δ in this order. The specific ϵ values are from 0.1 to 0.5 in increments of 0.1, and δ values are from 0.01 to 0.05 in increments of 0.01. Notice, in the optimal solution the L type influence is allocated to the bottom arc which has the smallest ϵ value, while the H type influence is allocated to the second and third from the bottom arcs (that have the smallest δ values among the remaining 4 arcs), with the top two arcs allocated Z type influence. This solution in Figure 4B with objective value $15 - 14.85 = 0.15$ is the unique optimal solution (breaking the symmetry from the 30 possible solutions) after the perturbations are subtracted.

In order to satisfy these three conditions for the x perturbations, we suggest setting up the ϵ and δ perturbations for the incoming arcs at each node i as follows. Let $\beta = \frac{(\deg(i)+1)^2}{2\deg(i)} + 1$, $\tau = \frac{1}{|V|(\deg(i)+1)}$, and $\omega = \frac{1}{\beta|V|\deg(i)^2}$. Consider the distinct (ϵ, δ) perturbation pairs: $(1\tau, 1\omega), (2\tau, 2\omega), (3\tau, 3\omega), \ldots, ((deg(i) - 1)\tau, (deg(i) - 1)\omega), (deg(i)\tau, deg(i)\omega)$. For each of the $deg(i)$ incoming arcs use one of these distinct $deg(i)$ perturbation pairs. It is easy to see that condition 1 is satisfied. Condition 2 is satisfied because the maximum increment caused by ϵ and δ in the objective value is $\frac{\deg(i)}{|V|(\deg(i)+1)} + \frac{\deg(i)+(\deg(i)-1)+(\deg(i)-2)+\cdots+(\deg(i)-g_i+1)}{\beta|V|\deg(i)^2} < \frac{\deg(i)}{|V|(\deg(i)+1)} + \frac{1+2+\cdots+\deg(i)}{\beta|V|\deg(i)^2} = \frac{\deg(i)}{|V|(\deg(i)+1)} + \frac{\deg(i)+1}{2\beta|V|\deg(i)} < \frac{\deg(i)}{|V|(\deg(i)+1)} + \frac{1}{|V|(\deg(i)+1)} = \frac{1}{|V|}$. Since $\tau > \omega$, condition 3 is satisfied.

Finally, we note that to address both types of symmetry simultaneously, we can simply divide all suggested perturbation values by two. In our implementation, we use double-precision which is allowed by both Python and CPLEX. Given the size of our instances, numerical issues do not arise. Nonetheless, we note that caution may be required for even larger instances, and a more sophisticated way for setting up the perturbations could be necessary.

3.2 | Branch-and-cut enhancements

We now discuss some additional features of the branch-and-cut approach for solving the LCIP on arbitrary graphs.

Priority branching: We observe that in HLZ it suffices to define the y variables as binary (i.e., integrality can be relaxed on the x variables). This allows us to give the y variables higher branching priority in the branch-and-bound tree (we will refer to this prioritization as *priority branching*).

Lemma 1. *Integrality of the x variables can be relaxed in HLZ.*

Proof. Given a feasible vector y, once the y variables are fixed to either 0 or 1, the remaining constraint matrix is TU. This is easily seen from the proof of Theorem 1. ∎

Initial feasible solution: To provide the branch-and-cut approach with an initial feasible solution, we considered two greedy heuristics. The first greedy heuristic, that we refer to as *threshold greedy*, identifies the node with the smallest threshold value and pays it the entire threshold value to activate it. It then updates the graph by propagating influence from this newly activated node (i.e., reducing the threshold of each of the inactive neighbors of this newly activated node by their influence factors, checking if any nodes are activated, and repeating the process for each activated node). This is repeated until the entire graph is active (i.e., a feasible solution is found). The second greedy heuristic, that we refer to as *influence greedy*, operates in a somewhat counterintuitive fashion to the threshold greedy heuristic. It selects the node with the smallest influence factor among the inactive nodes in each iteration and pays it the threshold value. While it seems counterintuitive to select the node whose neighbors exert the smallest influence (on it), Günneç et al. [12] showed that the influence greedy heuristic solves the LCIP on trees optimally (indicating this strategy can indeed be effective). Over all our test instances, the influence greedy almost always gives a strictly better solution (236 out of 240 instances). Thus, we use the influence greedy heuristic as the initial feasible solution in our branch-and-cut approach.

Conservative separation: We realize that the separation procedure is expensive. Therefore, we modify the branch-and-cut approach in the following way. In the root node, we do our best to find violated inequalities in order to achieve a better dual

TABLE 1 Source and size of real-world graphs

Graph name	Source	# of nodes	# of edges
G04	SNAP	10 876	39 994
G05	SNAP	8846	31 839
G06	SNAP	8717	31 525
G08	SNAP	6301	20 777
G09	SNAP	8114	26 013
B-Alpha	SNAP	3783	24 186
B-OTC	SNAP	5881	35 592
AS01	SNAP	10 670	22 002
AS02	SNAP	10 900	31 180
Ning	BGU	9727	40 570
Escorts	Konect	10 106	39 016
Anybeat	N.R.	12 645	49 132
Gplus	Konect	23 613	39 182
Facebook1	BGU	39 439	50 222
Facebook2	Konect	2888	2981
Douban	N.R.	154 908	327 162

bound (i.e., we focus on the cutting plane method). However, once we enter the branching phrase, the separation procedure is only invoked when the integrality constraints are satisfied at a node.

Feasibility lift and inactive induced subgraph: In the formulation HLZ, we use binary variables to model the influence propagation process. Thus, the IP search process spends a significant amount of time on these variables. Actually, we are only interested in the payment vector (i.e., the natural payment variables p which can be easily decided from x variables). Consequently, for a given $\mathbf{p}$, we consider the feasibility of the solution and determine the set of nodes that can be activated using this payment vector. We call this process *feasibility lift*. If all nodes are activated by this payment vector, we have obtained a feasible solution, and the current node of the branch-and-bound tree can be fathomed. Otherwise, we can continue the branch-and-cut search as usual. One advantage of feasibility lift is that we can focus the separation procedure on the subgraph induced by the inactive nodes (because there must be some cycles to help them satisfy their thresholds). We refer to this subgraph as the *inactive induced subgraph*. In this way, we have a smaller supporting graph and can add fewer violated inequalities in the branch-and-cut approach. In our preliminary computational testing, we found that separation on the inactive induced subgraph is about 10 times faster than that on the original graph.

4 | COMPUTATIONAL EXPERIMENTS

We now discuss our computational experience with the branch-and-cut approach on both real-world and simulated social networks. Our computational experiments have several goals. We examine the strength of the HLZ formulation (to evaluate the benefit obtained by our formulation), the effect of branch-and-cut enhancements in terms of improving its performance, the quality of several heuristics for the LCIP, the effect of graph density on the branch-and-cut approach, and the benefit of partial incentives. For our branch-and-cut implementation, we use CPLEX 12.8 with the Python API and run our tests on a machine with an Intel Xeon E5-2630V4, 64 GB ram, under the Ubuntu 14.04 operating system.

4.1 | Data sets

We use three data sets. The first one is based on 16 real-world graphs: Gnutella (five graphs), Bitcoin (two graphs), Autonomous Systems (two graphs), Ning, Escorts, Anybeat, Google Plus, Facebook1, Facebook2, and Douban. They are taken from several online repositories, including the Stanford Large Network Data set Collection (SNAP, [18]), the BGU Social Networks Security Research Group (BGU, [20]), the Koblenz Network Collection (KONECT, [17]), and the Network Repository (N.R., [26]). All graphs are undirected (i.e., any directed graph is converted to an undirected one). If multiple connected components exist in a graph, we use the largest connected component in our computational experiments. Table 1 lists each graph with its source and the number of nodes and edges it contains.

Gnutella is a large file sharing peer-to-peer network (and the first decentralized peer-to-peer network of its kind). The graphs G04, G05, G06, G08, and G09 are snapshots of the Gnutella network (the nodes represent hosts in the network topology and the edges are connections between these hosts) on August 4, 5, 6, 8, and 9 of 2002. *Bitcoin Alpha* and *Bitcoin OTC* are platforms for trading Bitcoin. The two graphs are who-trusts-whom networks of people who trade using Bitcoin on these two platforms, which are shown in rows "B-Alpha" and "B-OTC". *Autonomous Systems* is the graph of routers comprising the Internet. Autonomous Systems is interesting because it represents a communication network of "who talks to whom". Rows "AS01" and "AS02" provide information on two Autonomous Systems graphs from March 31, 2001. *Ning* is an online platform for people and organizations to create custom social networks. The Ning graph is a snapshot of the friendship and group affiliations harvested during September 2012. *Escorts* is a bipartite network of an escort service, with nodes representing buyers and their escorts. We treat buyers and escorts in the same way and do not distinguish between them when we select a node to provide incentives. *Anybeat* is an online community, a public gathering place where individuals can interact with people from around their neighborhood or across the world. *Gplus* is a snapshot of a small portion of the major social network Google+. *Facebook1* is a compound Facebook social network of all participants in the LetsDoIt system, a group decision support system prototype for leisure actives (in the source repository, Facebook1 is referred to as *LetsDoIt*). *Facebook2* contains Facebook user-user friendships collected manually, starting from a set of ten selected users (in the source, Facebook2 is referred to as *Facebook(NIPS)*).

In addition to these 15 graphs, we also have one very large real-world social network based on Douban.com. This website, launched on March 6, 2005, is a Chinese Web 2.0 web site providing user interactions for sharing opinions on movies, books, and music. It is one of the largest online communities in China. The graph *Douban* contains the friendship network crawled in December 2010, which has 154 908 nodes and 327 162 edges. For each of these 16 real-world graphs, 10 LCIP instances are generated by simulating values of the threshold b_i and the influence factor d_i. Thus, there are 160 instances in total for the first data set.

The second and third data sets are based on simulated graphs. In their pioneering work on social network analytics Watts and Strogatz [28] describe a method to generate simulated social networks. Hagberg et al. [13] provide a Python package *NetworkX* that implements Watts and Strogatz's method. In particular, it includes a function *connected_watts_strogatz_graph* that generates connected social networks that we use. The rewiring probability p in this function is set as 0.3 because this value corresponds most closely with the social networks they studied in Watts and Strogatz [28]. The second data set consists of simulated graphs with 200 nodes and varied graph density, where the number of edges takes various values {400, 600, 800, 1000, 1200}. For each setting, ten graph instances are generated, and there are 50 small simulated graph instances in total. The third data set consists of large simulated graphs with 2500, 5000, and 10 000 nodes, respectively, and exactly 20 000 edges. For each setting, ten graph instances are generated, providing 30 instances in total.

For these three data sets (real-world, small simulated, and large simulated), we create a LCIP instance by first randomly generating node type g_i from a discrete uniform distribution between $[1, deg(i)]$ and influence factor d_i from a discrete uniform distribution between [1, 50]. Then, the threshold for a node i is calculated as $b_i = d_i(g_i - 1) + s_i$, where s_i is generated from a discrete uniform distribution between $[1, d_i]$. By this method, we ensure that if all neighbors of a node are active, the node becomes active as well.

4.2 | Comparing the LP relaxations of the formulations

We study the strength of the LP relaxations of TimeIndex, ARC, and HLZ on the real-world graphs, except for Douban (since it is much larger, we analyze it separately). We do not include any computational experiments with TimeIndex. First, we can analytically show that it is weak. Consider an LCIP instance on a complete graph with n nodes. For each node i in V, let $b_i = n-1$ and $d_i = 1$. Consider the LP relaxation of TimeIndex. For this instance, a feasible solution to the LP relaxation has $p_i = \frac{n-1}{n}$, and $y_{it} = \frac{t}{n}$ for all i in V and $t \in \{1, 2, \ldots, n\}$ (note that $T = n$). The objective value of this fractional solution is $n-1$. We now derive the optimal solution to TimeIndex for this instance. Initially, a node must be picked to start the influence propagation process (and will be paid $b_i = n-1$). Then, a complete graph with $(n-1)$ nodes is left with $b_i = n-2$ for all nodes. This is the case regardless of the node picked to start the influence propagation process; implying any node can be picked. Therefore, we can repeat this procedure until we have only two nodes left, and we have to pick one of them with cost 1. This optimal solution has objective value $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$. Thus the ratio of the objective value of the LP relaxation of TimeIndex to the optimal objective value of TimeIndex is $\frac{2}{n}$; which decreases to 0 in the limit as the size of the graph increases. In fact, when we implement TimeIndex, CPLEX is unable to solve the LP relaxation of any of the real-world instances within a 10-minute time limit. Providing more time does not help with these large real-world instances. We test a subset of them, which remain unsolved even after a time limit of 2 hours. Although TimeIndex is a compact formulation, it has $O(|V|^2)$ variables and $O(|V|^2)$ constraints, which makes the formulation grow rapidly. Needless to say, when the LP relaxation of an IP formulation cannot be solved, the model is not viable because the branch-and-bound

TABLE 2 Relative improvement (Rel. Imp. %) of the LP relaxation of HLZ over that of ARC on real-world instances

Rel. Imp. (%)	1	2	3	4	5	6	7	8	9	10	Avg.	Min.	Max.
G04	67.47	79.68	67.33	68.85	67.61	67.82	71.59	76.81	60.24	72.06	69.95	60.24	79.68
G05	58.47	84.26	59.70	74.11	71.81	65.27	67.56	75.35	72.54	72.33	70.14	58.47	84.26
G06	81.51	70.80	73.22	63.39	72.79	66.76	67.43	75.77	67.26	60.82	69.98	60.82	81.51
G08	54.45	62.97	63.53	69.45	53.80	58.39	57.01	58.46	59.84	57.56	59.55	53.80	69.45
G09	55.54	65.16	63.20	58.88	62.00	51.85	45.48	55.10	54.45	59.60	57.13	45.48	65.16
B-Alpha	33.74	47.22	40.25	63.47	62.29	51.75	43.13	52.44	54.86	35.64	48.48	33.74	63.47
B-OTC	35.77	47.84	29.25	57.06	57.09	25.36	50.06	44.10	62.11	37.13	44.58	25.36	62.11
AS01	25.36	33.57	38.19	49.83	18.32	43.69	38.23	41.41	43.96	50.26	38.28	18.32	50.26
AS02	43.83	35.49	27.73	41.65	57.76	33.63	31.95	27.54	49.55	46.00	39.51	27.54	57.76
Ning	42.62	48.68	58.72	62.99	44.76	51.79	40.56	52.07	45.80	39.12	48.71	39.12	62.99
Escorts	78.26	59.46	75.14	77.32	64.06	64.41	61.35	61.01	72.97	81.19	69.52	59.46	81.19
Anybeat	33.39	36.01	43.43	20.19	18.62	40.43	42.66	40.63	40.33	24.37	34.01	18.62	43.43
Gplus	11.67	14.75	11.52	13.39	12.94	13.54	17.12	12.84	13.03	12.97	13.38	11.52	17.12
Facebook1	4.61	5.01	5.31	3.41	4.92	4.60	4.66	4.63	5.40	5.79	4.84	3.41	5.79
Facebook2	1.34	0.50	0.93	1.51	1.13	0.76	0.55	2.55	2.29	0.88	1.24	0.50	2.55

TABLE 3 Relative improvement (Rel. Imp. %) of LP relaxations of ARC and HLZ on 200-node simulated instances

	1	2	3	4	5	6	7	8	9	10	Avg.	Min.	Max.
400	71.50	133.22	127.91	148.87	102.20	53.63	53.94	62.05	71.12	129.66	95.41	53.63	148.87
600	116.88	92.15	101.11	79.99	64.66	69.28	92.58	96.21	97.62	94.80	90.53	64.66	116.88
800	101.59	68.75	128.41	61.66	136.28	83.86	139.38	62.37	159.67	133.41	107.54	61.66	159.67
1000	41.59	46.77	52.18	214.00	101.98	111.64	43.30	132.71	78.57	30.09	85.28	30.09	214.00
1200	58.34	79.29	59.92	64.00	69.45	82.29	135.67	63.46	63.02	64.77	74.02	58.34	135.67

approach is predicated on solving the initial LP relaxation. Consequently, TimeIndex is not a viable formulation for the LCIP problem.

We now compare the LP relaxations of ARC and HLZ on real-world graphs. Table 2 shows the relative improvement of the LP relaxation of HLZ over that of ARC on real-world instances. Let $z^{\text{LP}}_{\text{HLZ}}$ and $z^{\text{LP}}_{\text{ARC}}$ denote the optimal objective value of LP relaxations of HLZ and ARC, respectively. The relative improvement is calculated as $\frac{z^{\text{LP}}_{\text{HLZ}} - z^{\text{LP}}_{\text{ARC}}}{z^{\text{LP}}_{\text{ARC}}} \times 100$. The results show that the improvement with HLZ is quite remarkable! Among these 150 real-world instances, on average, the relative improvement is about 44.62%; with the largest relative improvement over 84%. While the relative improvement on Facebook1 and Facebook2 instances is small, they turn out to be easy instances. If we exclude these easy instances, the relative improvement is almost 50% on average. Therefore, HLZ is a much stronger formulation than ARC. In terms of average running time over the 150 real-world instances, the LP relaxations of ARC and HLZ need 83 and 51 seconds (well under the 10-minute time limit we imposed), respectively. It turns out that more violated k-dicycle inequalities are found in ARC than HLZ. Thus, ARC invokes more iterations of the separation procedure and has longer running time in general. We also compare the LP relaxations of ARC and HLZ on the 50 small simulated instances in the second data set. The results are shown in Table 3. The relative improvement is even more significant on these 50 instances. The relative improvement is 90.56% on average; with the largest relative improvement over 200%. The average running time is less than 1 second for both the LP relaxations of ARC and HLZ for these 50 small (200-node) simulated instances. Consequently, it should be clear that HLZ can improve the LP relaxation significantly and has better computational efficiency compared to ARC.

In order to further demonstrate the quality of the lower bounds provided by the LP relaxation of HLZ, we compare its objective value to the optimal value of the 50 small simulated instances solved optimally by using our branch-and-cut procedure (with the best setting that we discuss in the next section). The optimality gap is calculated as $\left(1 - \frac{z^{\text{LP}}_{\text{HLZ}}}{z^*}\right) \times 100$ where z^* denotes the optimal value. Similarly, the optimality gap of ARC is calculated as $\left(1 - \frac{z^{\text{LP}}_{\text{ARC}}}{z^*}\right) \times 100$. Table 4 displays the results. The average optimality gap of HLZ is 4.85% while ARC is 48.28% which is almost 10 times larger.

TABLE 4 Optimality gaps of LP relaxations of ARC and HLZ on 200-node simulated instances

Optimality gap(%) of LP relaxation of ARC													
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Avg.**	**Min.**	**Max.**
400	42.99	60.39	58.02	61.37	52.41	37.60	38.43	39.95	42.27	58.67	49.21	37.60	61.37
600	55.50	52.45	53.57	45.70	44.60	43.25	54.80	51.80	52.51	51.24	50.54	43.25	55.50
800	52.86	45.28	59.77	40.95	60.86	49.87	60.66	41.01	64.29	59.60	53.52	40.95	64.29
1000	31.62	34.67	37.62	70.02	51.96	54.77	30.63	60.19	47.30	24.14	44.29	24.14	70.02
1200	39.04	46.17	40.47	41.81	42.70	47.22	58.54	41.25	40.37	40.89	43.85	39.04	58.54
Optimality gap (%) of LP relaxation of HLZ													
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Avg.**	**Min.**	**Max.**
400	2.23	7.63	4.32	3.86	3.77	4.14	5.22	2.68	1.22	5.09	4.02	1.22	7.63
600	3.49	8.63	6.63	2.26	8.78	3.94	12.95	5.43	6.14	5.01	6.33	2.26	12.95
800	4.97	7.66	8.10	4.55	7.52	7.84	5.84	4.21	7.27	5.71	6.37	4.21	8.10
1000	3.18	4.11	5.08	5.86	2.98	4.28	0.59	7.35	5.90	1.31	4.06	0.59	7.35
1200	3.48	3.49	4.80	4.57	2.90	3.79	2.30	3.96	2.79	2.61	3.47	2.30	4.80

4.3 | Testing the proposed branch-and-cut approach

In the second set of experiments, we test the proposed branch-and-cut approach on the 160 real-world instances. To test the efficiency of the branch-and-cut approach and compare the advantage of HLZ over ARC, we create four settings. We first create two settings to evaluate the performance of the direct implementations of HLZ and ARC formulations with CPLEX. These two settings are called "HLZ-Basic-CCuts" and "ARC-Basic-CCuts" which are the straight implementations of HLZ formulation and ARC formulation, respectively, with k-dicycle separation under the default setting of CPLEX. Next, we create two settings to evaluate the value of the enhancements described in Section 3. The third setting is called "HLZ-Full." In addition to k-dicycle separation, HLZ-Full includes the enhancements: initial feasible solution, x perturbations, y perturbations, priority branching, feasibility lift, inactive induced subgraph separation, and conservative separation. We note that the enhancements applied to HLZ-Full can also be applied to ARC formulation except for x perturbations and priority branching. We want to see if ARC (with similar enhancements as HLZ-Full) would be capable of solving LCIP problem instances and also to gauge the differences in terms of computational performance between these two formulations. Consequently, we create the fourth setting "ARC-Full," that is, ARC executed with all of the enhancements applied to HLZ-Full except for x perturbations and priority branching. This also provides a fairer comparison between HLZ and ARC. Furthermore, in our branch-and-cut implementation, we remove the constraint $y_{ij} + y_{ji} = 1$ and use only variables y_{ij} where $i < j$ (so y_{ji} is replaced by $1 - y_{ij}$ in the model). In this way, we reduce the size of the model by $|E|$ constraints and $|E|$ variables. In order to isolate the effect of our enhancements, we turn off CPLEX's cuts in HLZ-Full and ARC-Full. Other than that, we keep the default setting for CPLEX. In these four settings, running time is capped at 10 minutes for each instance.

We first run our experiment on all of the 150 real-world graphs, except for Douban instances. We evaluate the optimality gap, calculated as $\frac{\text{ub}-\text{lb}}{\text{ub}} \times 100$ (where ub denotes the upper bound and lb denotes the lower bound obtained by each setting), to compare across the four settings. HLZ-Basic-CCuts performs badly in our experiments. Within the 10-minute time limit, HLZ-Basic-CCuts cannot find a feasible solution for most of the 150 real-world instances. In our experiment, only 10 out of the 150 instances obtain a feasible solution. These 10 instances are the easier Facebook2 instances. ARC-Basic-CCuts also performs poorly in our experiments. Within the 10-minute time limit, only 129 out of the 150 instances obtain a feasible solution with ARC-Basic-CCuts. Furthermore, while a feasible solution is obtained, the optimality gap is large. Over all instances with feasible solutions found by ARC-Basic-CCuts, the average optimality gap and the maximum optimality gap is over 85% and 98%, respectively. If we exclude the easy Facebook2 instances, the average optimality gap is over 92%. Based on these outcomes, we conclude that neither HLZ-Basic-CCuts nor ARC-Basic-CCuts is able to provide a satisfactory performance. Thus, if we only rely on a straight implementation and hand the job over to CPLEX, the branch-and-cut procedure of CPLEX stagnates because the LCIP instances present strong symmetry. Even with the built-in symmetry breaking techniques, CPLEX has difficulty finding feasible solutions and closing the gap. The cuts generated by CPLEX do not help in this situation. (We have run a subset of instances with a much longer time limit. The 10 Ning instances are used here because they have the largest average gap with our best setting as shown later. With a 2-hour time limit, the story remains similar for both settings. HLZ-Basic-CCuts cannot find feasible solutions for any instance. ARC-Basic-CCuts is able to obtain feasible solutions for 9 out of 10 instances but has an optimality gap of over 93% on average.)

TABLE 5 Optimality gap (%) of HLZ-Full on real-world instances with a 10-minute time limit

Opt gap (%)	1	2	3	4	5	6	7	8	9	10	Avg.	Min.	Max.
G04	0.23	0.49	0.23	0.54	0.29	0.31	0.48	0.38	0.28	0.36	0.36	0.23	0.54
G05	0.48	0.31	0.31	0.70	0.41	0.58	0.75	0.61	0.48	0.61	0.52	0.31	0.75
G06	0.61	0.76	0.85	0.46	0.67	0.48	0.72	0.63	0.41	0.42	0.60	0.41	0.85
G08	0.66	0.72	1.23	1.01	0.73	0.34	0.83	0.63	0.89	0.76	0.78	0.34	1.23
G09	0.46	0.98	0.54	0.56	0.55	0.75	0.52	0.66	0.70	0.78	0.65	0.46	0.98
B-Alpha	3.37	3.20	3.03	4.83	6.16	3.25	3.55	7.22	3.80	2.90	4.13	2.90	7.22
B-OTC	2.62	4.49	2.33	5.48	7.12	2.38	3.30	4.08	5.43	3.04	4.03	2.33	7.12
AS01	1.11	2.55	1.34	2.21	0.78	2.45	4.37	2.41	4.67	3.07	2.50	0.78	4.67
AS02	5.11	3.70	2.64	4.05	6.42	4.91	4.02	3.49	4.36	4.95	4.37	2.64	6.42
Ning	3.93	4.81	6.73	6.46	4.91	5.29	4.65	4.33	4.23	4.68	5.00	3.93	6.73
Escorts	0.62	0.72	1.21	0.56	0.66	0.66	0.55	0.86	0.89	0.58	0.73	0.55	1.21
Anybeat	2.77	4.08	6.26	2.21	2.02	5.27	8.70	5.17	5.96	2.74	4.52	2.02	8.70
Gplus	0.75	1.62	0.40	4.51	3.63	3.46	3.10	2.44	1.94	2.37	2.42	0.40	4.51
Facebook1	0.31	0.03	0.06	0.11	0.45	0.04	0.04	0.06	0.63	0.15	0.19	0.03	0.63
Facebook2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Next, we present the results of the settings with our enhancements. We want to point out that in our preliminary study, we test out various combinations of the branch-and-cut enhancements (see Section 3.2) and try to isolate the contribution of each individual enhancement. We find that separately the performance impact of each enhancement varies, sometimes they are beneficial, sometimes they are not. Furthermore, the combination of two beneficial enhancements sometimes yields worse results than the application of each individual enhancement. Thus, there is *not* always a compelling case to make for each enhancement separately. Instead, when all of the enhancements are applied together, they seem to complement each other and work well in unison. When we apply all of the components, the performance improves significantly and is more robust and consistent. We discuss in greater detail some of our findings on the impact of the various enhancements in the Appendix.

We present the results of HLZ-Full in Table 5. The first column gives the identifier for each graph. Then, it contains the optimality gap for each instance from column "1" to column "10." Furthermore, we report the average, minimum, and maximum gap for each graph under the columns "Avg," "Min," and "Max," respectively. As shown in Table 5, HLZ-Full performs very well in our experiment. Not only is a feasible solution obtained for each of the 150 instances, but they are also proven to be high quality solutions. Over all 150 instances, the average optimality gap is 2.05% and the largest gap is 8.70%. Furthermore, 10 out of the 150 instances (all Facebook2 instances) are solved to optimality.

Table 6 describes the results on the optimality gap obtained with the ARC-Full setting. Compared to HLZ-Full the results are quite poor. The average optimality gap is about 29.69% and the maximum optimality gap is about 45.37%. ARC-Full is unable to produce comparable results to HLZ-Full. This clearly demonstrates that HLZ-Full has a much better performance than ARC-Full.

Next, we compare both the upper bounds and lower bounds from HLZ-Full and ARC-Full. The purpose is to understand whether the performance improvement comes from better upper or lower bounds. We start with the upper bounds by considering the absolute difference of HLZ-Full over ARC-Full. The difference is calculated as $z^{\text{UB}}_{\text{HLZ-Full}} - z^{\text{UB}}_{\text{ARC-Full}}$ where $z^{\text{UB}}_{\text{HLZ-Full}}$ and $z^{\text{UB}}_{\text{ARC-Full}}$ are the upper bounds of HLZ-Full and ARC-Full, respectively. Figure 5 presents the results. The negative value means HLZ-Full has a better objective value than ARC-Full. Over the 150 instances, HLZ-Full can find a strictly better upper bound in 19 instances while it has a strictly worse upper bound in 16 instances. The upper bounds are the same in the remaining instances. Over all instances, the average cost of HLZ-Full is strictly better than that of ARC-Full. Figure 6 compares the lower bounds. Regarding the lower bound, we look at the relative improvement of HLZ-Full over ARC-Full. The relative improvement is calculated as $\left(\frac{z^{\text{LB}}_{\text{HLZ-Full}}}{z^{\text{LB}}_{\text{ARC-Full}}} - 1\right) \times 100$ where $z^{\text{LB}}_{\text{HLZ-Full}}$ and $z^{\text{LB}}_{\text{ARC-Full}}$ are the lower bounds of HLZ-Full and ARC-Full, respectively. HLZ-Full is able to find a better lower bound than ARC-Full for all of the 150 real-world instances. The average improvement across the 150 instances is about 43.52%, and the largest improvement is about 82.48%.

While we present the results with a 10-minute time limit here, we also include the results with a 1-hour time limit in the Appendix. The narrative is similar with a 1-hour running time. The average and maximum gap of HLZ-Full are 1.99% and 8.70%, respectively. The average and maximum gap of ARC-Full are 29.57% and 45.24%, respectively. When we compare the upper bounds, HLZ-Full can find a strictly better upper bound in 84 instances while it has a strictly worse upper bound in five instances. Regarding the lower bound, the average improvement of HLZ-Full over ARC-Full across the 150 instances is about 43.33%, and the largest improvement is about 82.09% (slightly smaller than with the 10 minute run time). Thus, providing

TABLE 6 Optimality gap (%) of ARC-Full on real-world instances with a 10-minute time limit

Opt gap (%)	1	2	3	4	5	6	7	8	9	10	Avg.	Min.	Max.
G04	40.08	44.23	39.93	40.73	40.13	40.18	41.61	43.19	37.40	41.69	40.92	37.40	44.23
G05	36.73	45.37	37.09	42.37	41.48	39.35	40.28	42.80	41.73	41.89	40.91	36.73	45.37
G06	44.51	41.40	42.34	38.51	42.00	39.85	40.22	42.98	40.00	37.53	40.93	37.53	44.51
G08	34.98	38.38	38.82	41.00	34.89	36.35	36.22	36.64	37.34	36.33	37.10	34.89	41.00
G09	35.46	39.44	38.53	36.69	38.09	34.15	31.14	35.46	35.21	37.34	36.15	31.14	39.44
B-Alpha	26.71	33.18	29.96	40.48	40.51	34.78	31.68	38.27	36.62	27.51	33.97	26.71	40.51
B-OTC	27.87	34.80	23.98	38.92	40.03	21.69	34.94	32.77	40.85	28.65	32.45	21.69	40.85
AS01	20.90	26.75	28.24	34.24	15.92	31.77	30.51	30.61	33.43	35.06	28.74	15.92	35.06
AS02	33.65	28.64	23.42	31.98	40.12	28.55	26.97	24.13	35.65	34.49	30.76	23.42	40.12
Ning	32.23	35.61	40.80	42.00	33.89	37.24	31.73	36.63	33.91	31.12	35.52	31.12	42.00
Escorts	43.72	37.36	43.11	43.36	39.02	39.11	38.00	38.01	42.24	44.62	40.86	37.36	44.62
Anybeat	26.85	29.24	34.65	18.45	17.24	32.33	36.00	32.33	32.63	21.58	28.13	17.24	36.00
Gplus	11.03	14.27	10.61	15.79	14.67	14.90	17.26	13.47	13.13	13.58	13.87	10.61	17.26
Facebook1	4.66	4.74	5.04	3.36	5.05	4.38	4.43	4.42	5.65	5.54	4.73	3.36	5.65
Facebook2	0.35	0.02	0.01	0.17	0.21	0.69	0.08	0.24	0.49	0.21	0.25	0.01	0.69

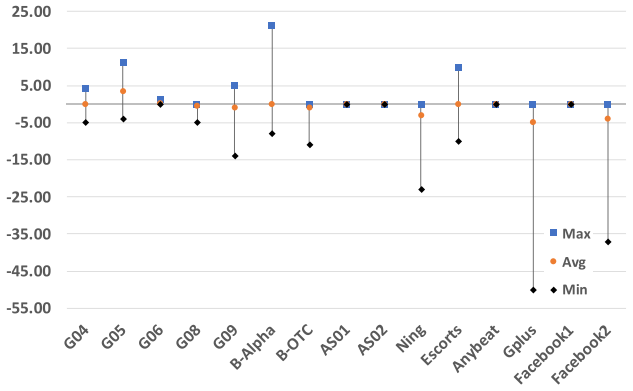

FIGURE 5 Comparison of HLZ-Full upper bound and ARC-Full upper bound on 150 real-world instances with a 10-minute time limit [Color figure can be viewed at wileyonlinelibrary.com]

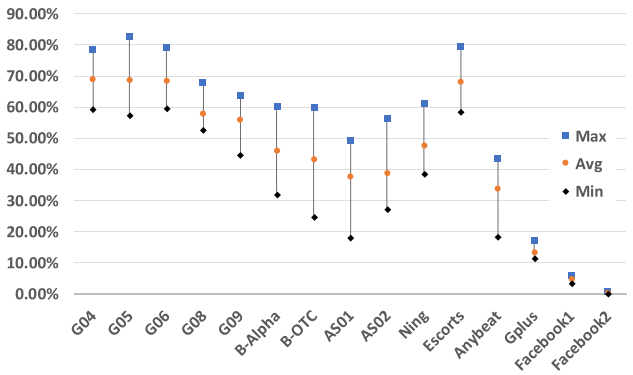

FIGURE 6 Improvement (%) in HLZ-Full lower bound over ARC-Full lower bound on 150 real-world instances with a 10-minute time limit [Color figure can be viewed at wileyonlinelibrary.com]

additional time helps HLZ-Full more than ARC-Full on the upper bound. While ARC-Full might benefit more than HLZ-Full on the lower bound with a longer running time, its lower bound is still much worse than that of HLZ-Full. In a nutshell, HLZ-Full is able to find better upper and lower bounds than ARC-Full. While the difference in the upper bounds is marginal, the difference in the lower bounds is significant and strongly contributes to the superior performance of HLZ-Full.

Next, we conduct experiments on a subset of the 150 real-world instances to see if we can close the small residual optimality gaps with HLZ-Full. We apply HLZ-Full with a 4-hour time limit to the first instance of all graphs except for Facebook2. The

TABLE 7 Optimality gap (%) of HLZ-Full with a 4-hour time limit on the first instance of each graph

Opt gap (%)	G04	G05	G06	G08	G09	B-Alpha	B-OTC	AS01	AS02	Ning	Escorts	Anybeat	Gplus	Facebook1
10 min	0.22	0.45	0.60	0.64	0.45	3.30	2.51	1.11	5.11	3.94	0.62	2.77	0.75	0.31
1 h	0.20	0.30	0.57	0.64	0.40	3.21	2.51	1.05	5.11	3.93	0.62	2.77	0.75	0.30
4 h	0.18	0.22	0.48	0.64	0.35	3.01	2.41	1.00	5.09	3.81	0.59	2.77	0.71	0.22

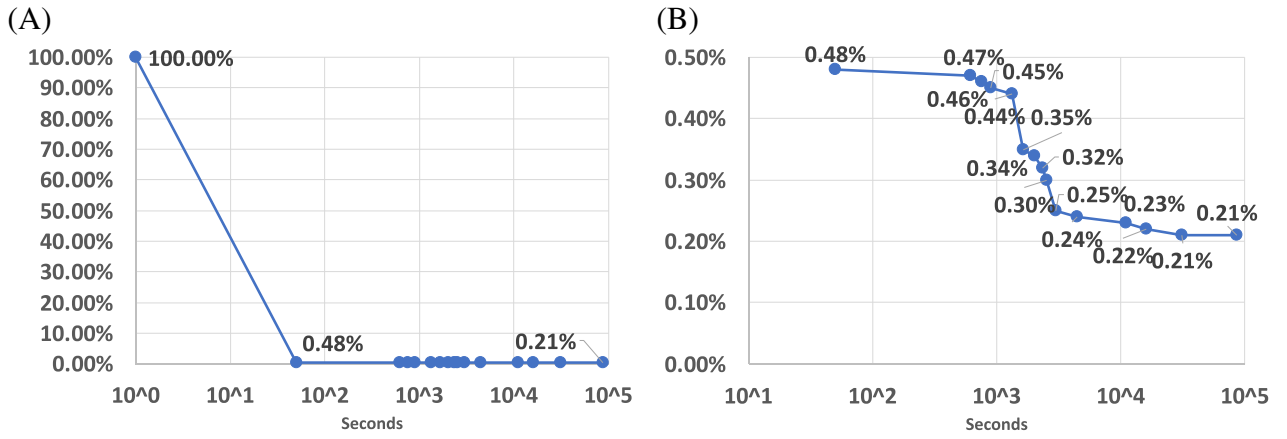

FIGURE 7 Optimality gap of the G05 instance over 24 hours. (A) The optimality gap over 24 hours. (B) The optimality gap after root node [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 8 Results of HLZ-Full on the Douban graph

Opt gap (%)	1	2	3	4	5	6	7	8	9	10	Avg.	Min.	Max.
Douban	0.13	0.16	0.19	0.19	0.15	0.18	0.12	0.17	0.16	0.19	0.16	0.12	0.19

resulting optimality gap is presented in Table 7. For ease of reference, we include the optimality gap after 10 minutes and 1 hour, as well. Overall, the improvement is marginal compared to the results of 1-hour of running time. On average, the optimality gap is reduced by 0.06%. Two instances (G08 and Anybeat) have no improvement. The largest improvement is 0.20% from the B-Alpha instance, while the B-Alpha instance has the third largest gap (3.21%) among these instances after 1 hour. Then, we run the G05 instance for 24 hours. We select the G05 instance because, relatively, its optimality gap has been reduced the most (51.11%) in 4 hours. Unfortunately, the optimality gap only decreases to 0.21% after 24 hours. Figure 7A plots the optimality gap during the course of the 24 hours, where the time (horizontal) axis is shown in logarithmic scale. Figure 7B starts with the optimality gap after the root node and the change over time thereafter. The optimality gap is 0.22% after about 3 and a half hours; but only improves by 0.01% and reaches 0.21% over the next 20+ hours. Given the tailing off effect of the branch-and-cut search, it is extremely difficult to completely close the gap. Clearly, although our branch-and-cut approach can find provably high quality solutions quickly, more research effort is needed to reach optimality.

Now, we focus on the 10 Douban instances. As the Douban graph is significantly larger than the other 15 real-world graphs, we modify the HLZ-Full setting as follows: first, we increase the running time to 1 hour. Second, even in the root node, we apply conservative separation. That is, we invoke the k-dicycle separation procedure only for an integer solution (at the root node and elsewhere). Table 8 presents our findings applying HLZ-Full. Although none of the ten instances are solved optimally, HLZ-Full has average, minimum, and maximum optimality gaps of 0.16%, 0.12%, and 0.19%, respectively.

In summary, over the 160 real-world graph instances, the average optimality gap of HLZ-Full is 1.87%. This clearly demonstrates that the proposed branch-and-cut approach is able to find high quality solutions and lower bounds for LCIP instances based on large real-world social networks.

Before concluding this section, we discuss Fischetti et al.'s [7] experience with their approach when applied to the LCIP as defined in this paper. There are significant contrasts between our computational experiences. As reported in their paper, their approach makes marginal improvements on sparse instances, but it has a much larger gap on dense instances. Specifically, for the instances with 2500 nodes and 20 000 edges (average degree 16) in their paper, they are only able to apply the heuristic price-cut-and-branch approach (and not their exact branch-and-cut approach). They report that after 2 hours of running time their heuristic price-and-cut approach has an average optimality gap of 53.2% while our approach (in their implementation with a 2-hour running time) has an average optimality gap of 12.9% (they generated different instances than the ones in our

TABLE 9 Optimality gaps (%) for heuristics on 200-node simulated instances

	Influence greedy		Threshold greedy		CGRV		Random	
Edges	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
400	2.77	6.04	6.82	10.32	48.63	64.50	63.89	68.17
600	6.74	11.46	12.87	16.99	54.11	58.34	68.91	72.67
800	7.10	13.01	12.04	17.05	59.69	68.48	70.83	74.96
1000	6.87	13.04	15.84	26.09	58.64	65.69	69.66	78.00
1200	5.65	8.77	21.28	26.45	66.63	69.84	73.57	77.51

paper). Furthermore, it is not clear whether they applied the branch-and-cut enhancements (described in Section 3.2) in their experiments. As we demonstrate earlier, these enhancements are crucial and tremendously improve the performance of our branch-and-cut approach.

4.4 | Evaluating heuristics for the LCIP

Next, we focus on a set of experiments on small simulated graphs with 200 nodes where we evaluated the cost of the optimal solution against heuristic solutions. Recall, in these simulated instances we varied graph density and created 50 instances where the number of edges takes values {400, 600, 800, 1000, 1200}. These instances are solved to optimality using HLZ-Full. As will be evident in Section 4.5, the LCIP becomes harder as the graph density increases. To solve these 50 instances optimally, the average running times are 0.75, 14.55, 140.58, 799.48, and 1068.26 seconds for instances with 200, 400, 600, 800, and 1200 edges, respectively. The maximum running times are 1.51, 50.48, 936.08, 6233.80, and 6760.31 seconds, respectively.

We consider four heuristics. The first two are influence greedy and threshold greedy. The third one is based on Cordasco et al.'s [4] algorithm for a specialized version of the LCIP where for a node i, b_i takes a value between 1 and $deg(i)$, and d_i is 1. We have adapted their algorithm to our (more general) version of the LCIP, which we now describe. In their algorithm, which we refer to as CGRV, the measure $\frac{g_i(g_i+1)}{\deg(i)(\deg(i)+1)}$ is computed for each node $i \in V$. At each iteration the node with the highest value of this measure (say j) is removed from the graph. Then, all neighbors of node j remaining in the graph are updated as follows. First, their degree is reduced by one. Next, we check whether $g_i > deg(i)$ for any of these nodes. If so, a partial payment of l_i is made the first time this occurs to node i, and a partial payment of d_i is made for each subsequent occurrences. Further g_i is reduced by one (indicating it needs one less neighbor to influence it because it has received a partial payment). Finally, the measure $\frac{g_i(g_i+1)}{\deg(i)(\deg(i)+1)}$ is updated for all remaining nodes in the graph. The process is repeated until all nodes are removed from the graph. The fourth one is called "Random" whose solution is generated randomly. In this heuristic, we randomly select a node among inactive nodes and pay the corresponding threshold to make it active. Then, we update the graph by propagating the influence from this newly active node and update the thresholds on the graph. We repeat this process until a feasible solution is obtained (i.e., all nodes are activated). For each instance, we find 100 random solutions and select the one with the smallest cost.

Table 9 summarizes the results providing the gap between the heuristic solution and the optimal solution value (the gap is calculated as the difference between the objective value of the heuristic and the optimal objective value divided by the objective value of the heuristic). The column "Influence Greedy" shows that the solutions obtained by influence greedy have the smallest gaps. The average gap is about 6% over all 50 instances. The threshold greedy has the second best performance. Its average gap is about 14%. The adapted CGRV algorithm performs quite badly in this more general LCIP setting as its average gap is about 58%. Not surprisingly, the random heuristic has the worst performance. The average gap of the random heuristic is about 69%. Notice an average gap of 69% means the objective value of the random heuristic is on average over three times that of the optimal objective value.

Since influence greedy had the best performance on the small simulated instances, we decided to also run it on the 160 real-world instances. Let $z^{\mathrm{H}}_{\mathrm{IG}}$ denote the objective value of the influence greedy heuristic. As a basis for comparison we used the lower bound of HLZ-Full, which we denote by $z^{\mathrm{LB}}_{\mathrm{HLZ\text{-}Full}}$. The optimality gap is then calculated as $\frac{z^{\mathrm{H}}_{\mathrm{IG}} - z^{\mathrm{LB}}_{\mathrm{HLZ\text{-}Full}}}{z^{\mathrm{H}}_{\mathrm{IG}}} \times 100$. Figure 8 displays the maximum, average and minimum optimality gap over the ten instances for each graph. The results show that, over these 160 real-world instances, the solution found by the influence greedy heuristic is on average at most about 1.99% away from optimality.

4.5 | The effect of graph density

In the next experiment, we investigate the role graph density plays in the difficulty of solving a problem. For this we use the large simulated networks since it is generally easier to control the graph density with larger networks. Recall, we generated 10

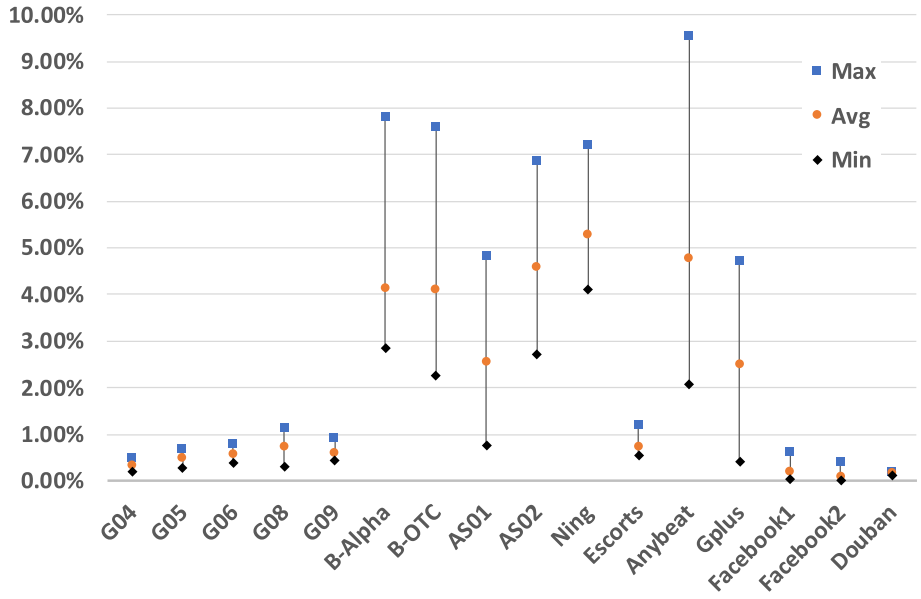

FIGURE 8 Optimality gap (%) of the influence greedy heuristic on real-world instances [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 10 Analyzing the effect of graph density on the branch-and-cut approach

	Optimality gap (%)												
Nodes	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Avg.**	**Min.**	**Max.**
10 000	0.16	1.22	1.21	1.09	1.47	1.18	0.92	1.38	1.23	1.39	1.13	0.16	1.47
5000	9.47	10.47	11.55	9.52	11.68	10.86	10.92	12.70	10.93	9.27	10.74	9.27	12.70
2500	14.48	16.06	16.25	15.59	16.18	15.79	16.30	16.08	18.03	13.87	15.86	13.87	18.03

instances with 10 000 nodes and average degree 4, 10 instances with 5000 nodes and average degree 8, and 10 instances with 2500 nodes and average degree 16. We use the "HLZ-Full" setting in the branch-and-cut approach and limit the running time to 10-minutes for all instances. Table 10 shows the optimality gap for these instances. The last two columns of the table show average and maximum values for these gaps over the 10 instances. We can see that as the graph gets denser the optimality gap gets bigger, implying that the problems get harder to solve for the branch-and-cut approach.

4.6 | Evaluating the benefit of partial incentives

In the final experiment, we evaluate the benefits of partial incentives. To do so, we evaluate the cost of an LCIP instance against a setting where no partial payments are allowed (i.e., either a node is paid b_i or 0). In other words we compare the cost of an LCIP instance against the cost of its WTSS counterpart. Recall in the WTSS problem, nodes must either be paid their full amount b_i or the sum of the influences from their neighbors must exceed their threshold. We convert an LCIP instance to a WTSS one by setting its threshold as b_i and the critical value g_i equal to $\left\lceil \frac{b_i}{d_i} \right\rceil$ for each i in V. Then, we calculate the relative savings of the LCIP as $\left(1 - \frac{z_{\text{LCIP}}}{z_{\text{WTSS}}}\right) \times 100$ where z_{LCIP} and z_{WTSS} are the optimal values of the LCIP and the WTSS instance, respectively. We convert all 50 small simulated instances into WTSS instances. Figure 9 displays the results. The plot indicates a significant saving with partial incentives. The average savings using partial payments compared to full payments are 71.47%. We also notice that the savings increase as the density of the graph increases. It increases from 64.90% to 76.50% when the number of edges increases from 400 to 1200.

Next, we make the same comparison to the 160 real-world instances. Because we can neither solve the resulting WTSS instances, nor the original LCIP instances to optimality, we compare the upper bound of an LCIP instance to the lower bound of the corresponding WTSS instance. That is, we calculate the relative saving of the LCIP instance solution as $\left(1 - \frac{\text{ub}_{\text{LCIP}}}{\text{lb}_{\text{WTSS}}}\right) \times 100$ where ub_{LCIP} is the upper bound of the LCIP instance and lb_{WTSS} is the lower bound of the WTSS instance. This is an underestimation of the potential savings generated by partial incentives. Figure 10 displays the results, indicating significant savings with partial incentives. While it varies among these real-world graphs, the average savings using partial payments

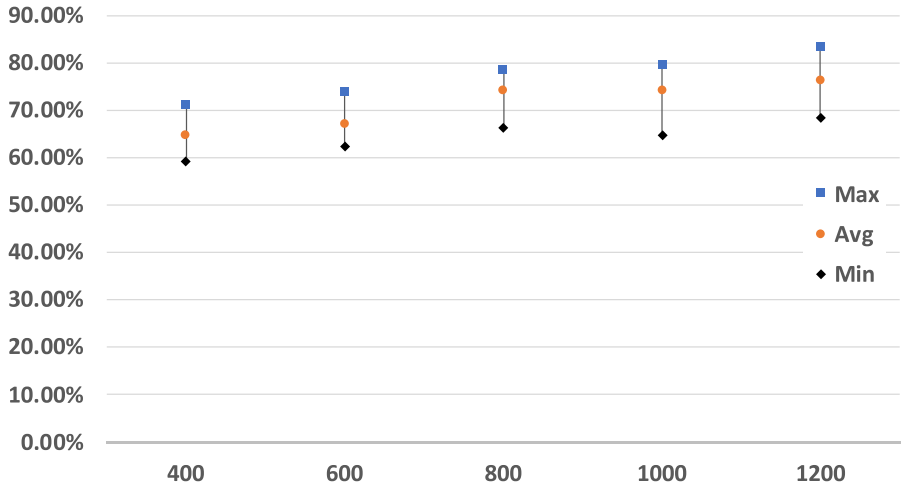

FIGURE 9 Relative savings of partial incentives on small simulated 200-node instances [Color figure can be viewed at wileyonlinelibrary.com]

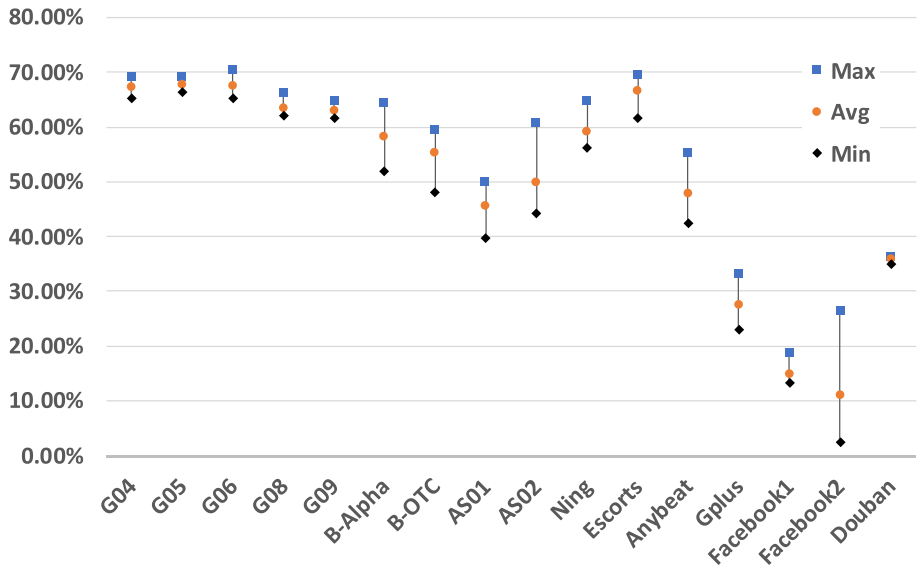

FIGURE 10 Relative savings of partial incentives on real-world instances [Color figure can be viewed at wileyonlinelibrary.com]

compared to full payments is 50.03%. The maximum savings can be as large as 70.34%. Over all the 210 instances, the average savings are over 55%. This emphasizes the importance and benefit of considering partial incentives in practice.

5 | CONCLUSIONS AND FUTURE WORK

The LCIP is a combinatorial optimization problem that arises in a variety of environments, from online marketing to epidemiology (see [19]). It is a fundamental problem in the diffusion of information and its extensions offer a rich set of problems for future research. In this paper, we focus on the case when all active neighbors of a node exert equal influence on it, and it is desired to activate the entire network. We design and test a branch-and-cut approach for the LCIP on arbitrary graphs. We build on Günneç et al.'s [12] novel TU formulation for the LCIP on trees. The key observation is to enforce an exponential set of inequalities that ensure the influence propagation network is acyclic for arbitrary graphs. We also design several enhancements. Then, we provide a large set of computational experiments on real-world graphs with up to 155 000 nodes and 327 000 edges that demonstrate the efficacy of the branch-and-cut approach. This branch-and-cut approach finds solutions that are on average 1.87% away from optimality based on a test-bed of 160 real-world graph instances. We also indicate the influence greedy heuristic works particularly well as a heuristic on arbitrary graphs. Finally, we show that partial incentives can result in significant cost savings, over 55% on average, compared to the setting where partial incentives are not allowed.

There are several potential directions for future work. Allowing competition and a defender-attacker setup in the problem (see [14]) and considering influence propagation over multiple social networks (see [16]) are among such directions. Although the TU formulation in Günneç et al. [12] provides a nice building block for developing an efficient approach on arbitrary graphs, it introduces a large number of variables by characterizing the incoming influence into several types. It would be interesting to see if it is possible to derive strong formulations based on the natural payment space. It could potentially reduce the number of variables and improve the computational efficiency. Our current research effort is addressing this issue.

Another natural direction is to allow for the influence structure to be nonlinear (e.g., diminishing influence or increasing influence from each additional neighbor). A third direction would be to consider scenarios where it is only necessary to influence a desired proportion α of the nodes in the network. Although the influence greedy algorithm can easily be adapted for finding solutions to these extensions, it is not straightforward to extend the TU based integer programming formulation to address this situation. The main problem is that the TU formulation is predicated on all nodes being activated at the end of the influence propagation process. When that is not the case, the formulation must add binary variables (one for each node) to keep track of the set of activated nodes at the end of the influence propagation process and embed the TU based integer programming formulation onto the subgraph defined by these activated nodes. In this regard, the recent paper by Fischetti et al. [7] takes a big first step.

The model we consider allows the influence propagation process to take as many steps/time periods as necessary. A fourth natural direction is to consider the influence maximization (e.g., the LCIP and the WTSS problem) with a constraint on the time allowed for the diffusion of influence (we call these latency constraints). Raghavan and Zhang [25] begin to address this question. They discuss the LCIP and WTSS problem in the scenario where there is only one time period for the influence propagation process.

Another natural future direction deals with the notion of an "effective value of an adopter"; meaning that each adopter (or node that is activated) may have a value (or prize) associated with it. The goal is then to capture the tradeoff between the cost of activating a portion of the network and the positive value gained by the nodes activated by the influence propagation process. This results in a "prize-collecting" version of the LCIP. Our work on the LCIP provides a big and important step along this research pathway.

ORCID

Dilek Günneç https://orcid.org/0000-0002-0749-2584
S. Raghavan https://orcid.org/0000-0002-9656-5596
Rui Zhang https://orcid.org/0000-0002-4029-6585

REFERENCES

[1] F. Bourne, "*Group influence in marketing and public relations*," *Some Applications of Behavioral Research*, UNESCO, Basil, Switzerland, 1957, pp. 207–225.
[2] J. Brown and P. Reingen, *Social ties and word-of-mouth referral behavior*, J. Consum. Res. **14** (1987), 350–362.
[3] N. Chen, *On the approximability of influence in social networks*, SIAM J. Discrete Math. **23** (2009), 1400–1415.
[4] G. Cordasco, L. Gargano, A.A. Rescigno, and U. Vaccaro, "*Optimizing spread of influence in social networks via partial incentives*," *Structural Information and Communication Complexity*, C. Scheideler (ed.), Springer, Cham, Switzerland, 2015, pp. 119–134.
[5] E.D. Demaine, M. Hajiaghayi, H. Mahini, D.L. Malec, S. Raghavan, A. Sawant, and M. Zadimoghadam, *How to influence people with partial incentives*, Proceedings of the 23rd International Conference on World Wide Web, 2014, pp. 937–948.
[6] T.N. Dinh, D.T. Nguyen, and M.T. Thai. *Cheap, easy, and massively effective viral marketing in social networks: Truth or fiction*? Proceedings of the 23rd ACM Conference on Hypertext and Social Media, ACM, 2012, pp. 165–174.
[7] M. Fischetti, M. Kahr, M. Leitner, M. Monaci, and M. Ruthmair, *Least cost influence propagation in (social) networks*, Math. Program. Ser. B **170** (2018), 293–325.
[8] A. Ghoniem and H.D. Sherali, *Defeating symmetry in combinatorial optimization via objective perturbations and hierarchical constraints*, IIE Trans. **43** (2011), 575–588.
[9] M. Granovetter, *Threshold models of collective behavior*, Am. J. Sociol. **83** (1978), 1420–1443.
[10] M. Grötschel, M. Jünger, and G. Reinelt, *On the acyclic subgraph polytope*, Math. Program. **33** (1985), 28–42.
[11] D. Günneç and S. Raghavan, *Integrating social network effects in the share-of-choice problem*, Decis. Sci. **48** (2017), 1098–1131.
[12] D. Günneç, S. Raghavan, and R. Zhang, Least-cost *influence maximization on social networks*, INFORMS J. Comput. **32** (2020), 289–302.
[13] A.A. Hagberg, D.A. Schult, and P.J. Swart, *Exploring network structure, dynamics, and function using NetworkX*, G. Varoquaux, T. Vaught, and J. Millman (eds.), Proceedings of the 7th Python in Science Conference, Pasadena, CA, 2008, pp. 11–15.
[14] M. Hemmati, J.C. Smith, and M.T. Thai, *A cutting-plane algorithm for solving a weighted influence interdiction problem*, Comput. Optim. Appl. **57** (2014), 71–104.
[15] D. Kempe, J. Kleinberg, and É. Tardos, *Maximizing the spread of influence through a social network*, Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, pp. 137–146.
[16] A. Kuhnle, M.A. Alim, X. Li, H. Zhang, and M.T. Thai, *Multiplex influence maximization in online social networks with heterogeneous diffusion models*, IEEE Trans. Comput. Soc. Syst. **5** (2018), 418–429.
[17] J. Kunegis, April 2017. KONECT network dataset – KONECT. http://konect.uni-koblenz.de/networks/konect.
[18] J. Leskovec and A. Krevl, June 2014. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

[19] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, *Cost-effective outbreak detection in networks*, Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2007, pp. 420–429.
[20] O. Lesser, L. Tenenboim-Chekina, L. Rokach, and Y. Elovici, "*Intruder or welcome friend: Inferring group membership in online social networks*," *Social Computing, Behavioral-Cultural Modeling and Prediction*, Springer, Heidelberg, Germany, 2013, pp. 368–376.
[21] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, Wiley, New York, 1988.
[22] H.T. Nguyen, T.N. Dinh, and M.T. Thai, *Cost-aware targeted viral marketing in billion-scale networks*, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE, 2016, pp, 1–9.
[23] S. Raghavan and R. Zhang, *A branch-and-cut approach for the weighted target set selection problem on social networks*, INFORMS J. Optim. **1** (2019), 304–322.
[24] S. Raghavan and R. Zhang, *Weighted target set selection on trees and cycles*, Working paper, University of Maryland, 2019.
[25] S. Raghavan and R. Zhang, *Influence maximization with latency requirements on social networks*, Working paper, University of Maryland, 2020.
[26] R.A. Rossi and N.K. Ahmed, *The network data repository with interactive graph analytics and visualization*, Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015. http://networkrepository.com.
[27] E. Shearer and K.E. Matsa, *News use across social media platforms 2018*, Technical report, Pew Research Center, 2018.
[28] D. Watts and S. Strogatz, *Collective dynamics of small-world networks*, Nature **393** (1998), 440–442.

How to cite this article: Günneç D, Raghavan S, Zhang R. A branch-and-cut approach for the least cost influence problem on social networks. *Networks.* 2020;76:84–105. https://doi.org/10.1002/net.21941

APPENDIX A: TESTING THE IMPACT OF THE BRANCH-AND-CUT ENHANCEMENTS

To test the impact of the branch-and-cut enhancements, we use the 10 instances from the second data set (small simulated instances) with 200 nodes and 1200 edges. In order to isolate the effect of our enhancements, we turn off CPLEX's cuts. Other than that, we keep the default setting for CPLEX. For each instance, running time is capped at 10 minutes. Table A1 contains the results concerning optimality gap in percentage. "BB" stands for the straight implementation with k-dicycles separation. "X" stands for x perturbations and "Y" stands for y perturbations. "F" is for feasibility lift and inactive induced subgraph. "G" stands for the initial feasible solution found by the influence greedy heuristic. "PB" stands for priority branching. Finally, the conservative separation is denoted by "C". If the combination cannot find a feasible solution within the time limit, the gap is denoted by "100.00%". We make several observations by using BB as the benchmark. First, enhancements X, Y, and F improve the gaps on some instances, but worsen the performance on average. Second, enhancements G and PB can improve the average gap. However, it is not necessarily the case if we look at one particular instance. For example, enhancement G has a bigger gap for instances 5, 8, and 10, and enhancement PB has a bigger gap for instances 5, 7, and 10. Third, enhancement C improves the gap (over BB) for each instance in this test set. Finally, when we apply all these enhancements, it has the best performance. It has the smallest average gap, the smallest median gap, and the smallest maximum gap. Figure A1 demonstrates the impact of these enhancements by plotting the average gap. While the test is only based on one set of instances, we observe similar behavior when we apply this test to other instances. The performance is most robust when all enhancements are applied. However, the impact of each enhancement may vary among different sets of instances, for example, "BB-G" is worse than "BB-PB" for the 10 instances with 200 nodes and 1200 edges while "BB-G" can have better results than "BB-PB" in another set of 10 instances.

TABLE A1 Optimality gap (%) of the branch-and-cut enhancements over 200-node-1200-edge instances with a 10-minute time limit

Opt gap (%)	1	2	3	4	5	6	7	8	9	10	Avg.	Min.	Max.	Median
BB	9.69	8.61	14.32	6.80	1.18	10.34	2.91	5.10	4.28	1.59	6.48	1.18	14.32	5.95
BB-X	7.81	9.68	100.00	8.25	4.89	100.00	4.20	5.58	1.34	10.25	25.20	1.34	100.00	8.03
BB-Y	4.91	8.15	100.00	5.68	3.84	12.85	5.56	3.39	1.97	5.00	15.14	1.97	100.00	5.28
BB-F	6.45	11.05	95.65	13.28	2.97	11.53	3.50	6.11	4.60	3.81	15.90	2.97	95.65	6.28
BB-G	5.68	7.81	5.37	6.69	3.03	9.40	2.21	5.11	3.76	3.37	5.24	2.21	9.40	5.24
BB-PB	3.95	6.52	5.38	3.62	1.89	3.94	4.03	3.38	1.80	2.34	3.69	1.80	6.52	3.78
BB-C	2.20	1.15	5.75	2.02	1.15	2.55	0.51	1.08	0.00	0.78	1.72	0.00	5.75	1.15
BB-XY-C-F-G-PB	0.98	0.78	3.17	2.17	0.91	1.86	0.92	1.16	0.00	1.10	1.31	0.00	3.17	1.04

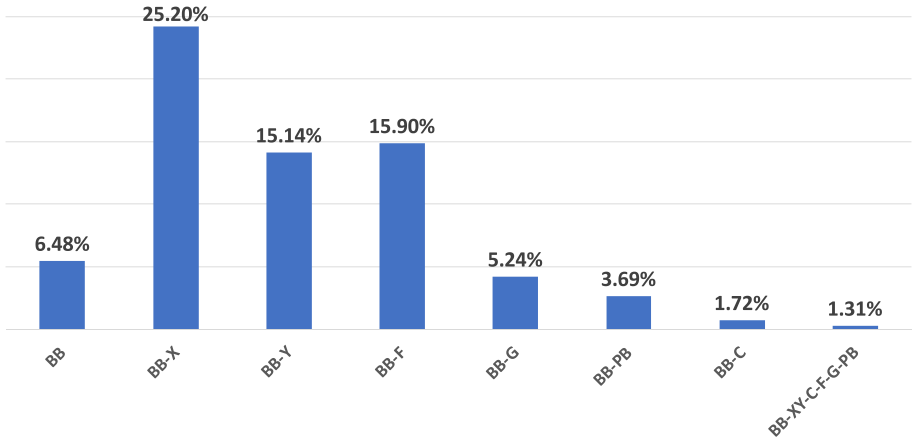

FIGURE A1 Average optimality gap (%) of the branch-and-cut enhancements over 200-node-1200-edge instances with a 10-minute time limit [Color figure can be viewed at wileyonlinelibrary.com]

APPENDIX B: RESULTS OF ARC-FULL AND HLZ-FULL WITH A 1-HOUR TIME LIMIT

TABLE B1 Optimality gap (%) of HLZ-Full on real-world instances with a 1-hour time limit

Opt gap (%)	1	2	3	4	5	6	7	8	9	10	Avg.	Min.	Max.
G04	0.20	0.40	0.20	0.46	0.27	0.27	0.47	0.32	0.25	0.34	0.32	0.20	0.47
G05	0.30	0.29	0.27	0.64	0.17	0.51	0.58	0.52	0.36	0.54	0.42	0.17	0.64
G06	0.57	0.73	0.76	0.41	0.63	0.38	0.68	0.61	0.33	0.32	0.54	0.32	0.76
G08	0.64	0.70	1.08	0.91	0.67	0.29	0.73	0.56	0.76	0.66	0.70	0.29	1.08
G09	0.40	0.91	0.49	0.50	0.52	0.69	0.42	0.56	0.65	0.63	0.58	0.40	0.91
B-Alpha	3.21	3.10	2.88	4.58	5.79	3.04	3.37	7.22	3.45	2.64	3.93	2.64	7.22
B-OTC	2.51	4.24	2.21	5.24	7.01	2.35	3.18	3.90	5.22	2.93	3.88	2.21	7.01
AS01	1.05	2.55	1.25	2.17	0.74	2.43	4.37	2.39	4.60	3.04	2.46	0.74	4.60
AS02	5.11	3.66	2.61	4.04	6.42	4.91	4.01	3.49	4.36	4.95	4.36	2.61	6.42
Ning	3.93	4.81	6.71	6.45	4.89	5.29	4.65	4.26	4.23	4.68	4.99	3.93	6.71
Escorts	0.60	0.69	1.16	0.56	0.63	0.66	0.54	0.81	0.85	0.57	0.71	0.54	1.16
Anybeat	2.77	4.08	6.26	2.21	2.02	5.26	8.70	5.17	5.93	2.74	4.51	2.02	8.70
Gplus	0.75	1.62	0.40	4.51	3.63	3.46	3.10	2.39	1.93	2.37	2.42	0.40	4.51
Facebook1	0.30	0.02	0.04	0.08	0.45	0.04	0.02	0.02	0.63	0.15	0.18	0.02	0.63
Facebook2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

TABLE B2 Optimality gap (%) of ARC-Full on real-world instances with a 1-hour time limit

Opt gap (%)	1	2	3	4	5	6	7	8	9	10	Avg.	Min.	Max.
G04	39.99	44.13	39.83	40.65	40.05	40.10	41.51	43.11	37.30	41.58	40.83	37.30	44.13
G05	36.67	45.24	37.00	42.26	41.36	39.21	40.22	42.73	41.62	41.77	40.81	36.67	45.24
G06	44.35	41.31	42.24	38.39	41.92	39.77	40.10	42.88	39.90	37.43	40.83	37.43	44.35
G08	34.84	38.28	38.71	40.87	34.80	36.24	36.10	36.55	37.23	36.24	36.99	34.80	40.87
G09	35.37	39.29	38.41	36.59	38.00	34.08	31.02	35.39	35.06	37.23	36.04	31.02	39.29
B-Alpha	26.56	33.01	29.81	40.29	40.11	34.59	31.51	38.02	36.33	27.39	33.76	26.56	40.29
B-OTC	27.79	34.70	23.84	38.74	39.84	21.60	34.81	32.68	40.68	28.51	32.32	21.60	40.68
AS01	20.86	26.69	28.17	34.10	15.88	31.67	30.43	30.51	33.33	34.93	28.66	15.88	34.93
AS02	33.51	28.59	23.34	31.91	39.96	28.46	26.92	24.09	35.57	34.39	30.67	23.34	39.96
Ning	32.13	35.47	40.64	41.85	33.79	37.14	31.61	36.48	33.80	31.04	35.40	31.04	41.85
Escorts	43.63	37.27	42.98	43.16	38.91	38.98	37.90	37.89	42.08	44.49	40.73	37.27	44.49
Anybeat	26.77	29.16	34.17	18.41	17.16	32.11	35.57	32.20	32.50	21.53	27.96	17.16	35.57
Gplus	11.01	14.10	10.58	15.65	14.54	14.86	17.09	13.39	13.09	13.33	13.76	10.58	17.09
Facebook1	4.64	4.71	5.02	3.35	5.03	4.36	4.42	4.40	5.63	5.52	4.71	3.35	5.63
Facebook2	0.32	0.01	0.00	0.16	0.07	0.23	0.08	0.22	0.41	0.09	0.16	0.00	0.41

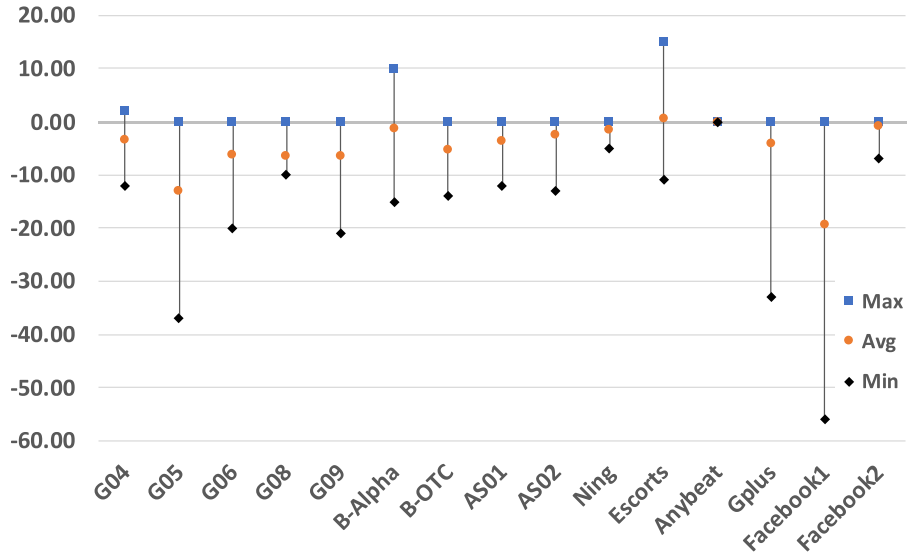

FIGURE B1 Comparison of HLZ-Full upper bound and ARC-Full upper bound on 150 real-world instances with a 1-hour time limit [Color figure can be viewed at wileyonlinelibrary.com]

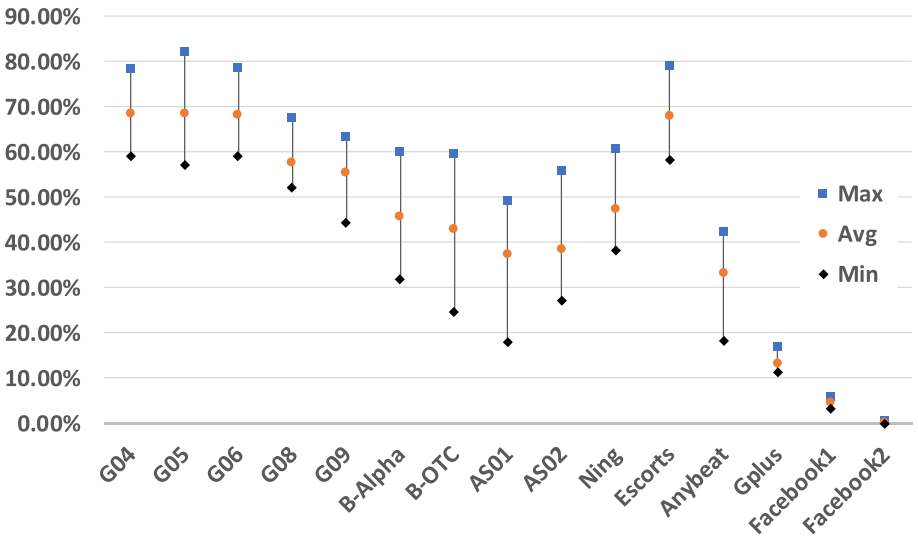

FIGURE B2 Improvement (%) in HLZ-Full lower bound over ARC-Full lower bound on 150 real-world instances with a 1-hour time limit [Color figure can be viewed at wileyonlinelibrary.com]