http://pubsonline.informs.org/journal/ijoo

INFORMS JOURNAL ON OPTIMIZATION
Vol. 1, No. 4, Fall 2019, pp. 304–322
ISSN 2575-1484 (print), ISSN 2575-1492 (online)

A Branch-and-Cut Approach for the Weighted Target Set Selection Problem on Social Networks

S. Raghavan,[a] Rui Zhang[b]

[a] Robert H. Smith School of Business and Institute for Systems Research, University of Maryland, College Park, Maryland 20742;
[b] Leeds School of Business, University of Colorado, Boulder, Colorado 80309

Contact: raghavan@umd.edu, http://orcid.org/0000-0002-9656-5596 (SR); rui.zhang@colorado.edu, http://orcid.org/0000-0002-4029-6585 (RZ)

Received: April 29, 2018
Revised: October 15, 2018; December 17, 2018
Accepted: December 26, 2018
Published Online in Articles in Advance: July 8, 2019

https://doi.org/10.1287/ijoo.2019.0012

Abstract. The study of viral marketing strategies on social networks has become an area of significant research interest. In this setting, we consider a combinatorial optimization problem, referred to as the weighted target set selection (WTSS) problem. Earlier research has focused on approximation algorithms for the unweighted case of the problem, which is known to be NP-hard. Motivated by the desire to develop a better understanding of the fundamental problems in social network analytics, we seek to develop mathematical programming approaches to solve the WTSS problem exactly. We build upon a tight and compact extended formulation for the WTSS problem on trees described in a companion paper and show that it is also a tight and compact extended formulation for directed acyclic graphs (DAGs). On the basis of the observation that the influence propagation network in any arbitrary graph is acyclic, we add an exponential set of inequalities that enforce this condition and show how to apply the extended formulation for DAGs to arbitrary graphs. Using this idea, we develop and implement a branch-and-cut approach to solve the WTSS problem on arbitrary graphs. Our computational experience on a test-bed of 180 real-world graph instances (with up to approximately 155,000 nodes and 327,000 edges) demonstrates the quality and efficacy of our solution approach. The branch-and-cut approach finds solutions that are on average 0.90% from optimality and solves 60 of the 180 instances to optimality. On the other hand, the best heuristic solutions generated are on average 5.46 times worse than the solutions generated by the branch-and-cut approach.

Keywords: branch-and-cut • influence maximization • integer programming • social networks • strong formulation

1. Introduction

Social networks play a fundamental role in the spread of influence, information, and ideas. Consequently, there is significant interest in understanding the dynamics of adoption within a social network, which may yield clues to better marketing strategies. Recently, the following viral marketing problem has generated significant interest from algorithmic researchers. Suppose we want to promote a new product over a given social network, and we would like this product to be adopted by everyone in this network. We can initialize the diffusion process by "targeting" certain influential people. Other people start to adopt this product owing to the influence they receive from these early adopters. Ideally, as the number of people who have adopted the product grows, a cascade will result, encompassing the entire network. Yet, how should we select these influential people who are targeted initially?

Kempe et al. (2003) first considered this problem, using a well-known "threshold" model from mathematical sociology (see Granovetter 1978) that explicitly represents the step-by-step dynamics of adoption. They considered a budgeted version of the 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 demonstrated that it is NP-hard. On the basis of the submodularity property of the objective function, which is due to the particular randomized assumption in the problem data they make, they developed a $(1-\frac{1}{e})$-approximation algorithm. Their work led to subsequent research that mostly focused on speeding up the algorithm (because their algorithm has a costly simulation in each step). Initiated by Chen (2009), another stream of work has focused on the problem in a deterministic setting with a cost minimization aspect (i.e., instead of the marketer being given a budget k, the desire is to find the minimum number of nodes to target in the network so that the entire network is influenced). The Chen et al. (2013) monograph nicely summarizes most of the relevant work in the area.

Chen (2009) initiated the study of the *Target Set Selection* (TSS) problem. Following the literature on the diffusion of innovations (Granovetter 1978), we will indicate that a node is active (or influenced) if it has

adopted the product and will indicate that a node is inactive (or not influenced) if it has not adopted the product. In the TSS problem, we are given a connected undirected graph $G = (V, E)$, where for each node $i \in V$ there is a threshold g_i that is between 1 and $\deg(i)$ (the degree of node i). All nodes are inactive initially. We select a subset of nodes, the target set, which become active. Afterward, in each step, we update the state of the nodes with the following rule: an inactive node i becomes active if at least g_i of its neighbors are active in the previous step. The threshold represents how easily a node is influenced by its neighbors. A higher threshold indicates that it requires a larger number of active neighbors to be influenced and become active. The goal is to find the minimum cardinality target set, while ensuring that all nodes are active by the end of this activation process. In this paper, we consider the *weighted TSS* (WTSS) problem. In the WTSS problem, for each node $i \in V$, there is a positive weight, denoted by b_i, which models the situation that different nodes might require differing levels of effort to become initial adopters.

Chen (2009) showed that it is hard to approximate the TSS problem within a polylogarithmic factor. He also provided a polynomial time algorithm for the TSS problem on trees. Chiang et al. (2013) provided a linear-time algorithm for the TSS problem on block-cactus graphs. They also showed that the problem is polynomially solvable on chordal graphs when $g_i \leq 2$ and on Hamming graphs when $g_i = 2$. Ben-Zwi et al. (2011) showed that for a graph with a treewidth bounded by ω, the TSS problem can be solved in V^{ω} time. Ackerman et al. (2010) provided some combinatorial bounds for the TSS problem under majority ($g_i \geq \frac{\deg(i)}{2}$) and strict majority ($g_i > \frac{\deg(i)}{2}$) influences. In passing, they also described an integer program (IP) model for the TSS problem. Spencer and Howarth (2013) consider the problem of providing incentives to consumers to promote "green" (i.e., environmentally friendly) behavior. In that context, they consider the TSS problem, although they refer to it as the Min-Cost Complete Conversion problem. To model the influence propagation process, they use a time-indexed IP formulation with as many time periods as the number of nodes in the network. This model grows very rapidly and is not a viable model from a computational perspective. In their experiments, they were only able to solve problems with 30 nodes and 75 edges. The same time-indexed IP is described in Shakarian et al. (2013). A serious issue with the previously discussed IP formulations in Ackerman et al. (2010), Spencer and Howarth (2013), and Shakarian et al. (2013) is the fact that they are weak (we empirically demonstrate this in Section 4.2). In other words, the gap between the objective value of the optimal solution to the IP formulation and the objective value of the optimal solution to its linear programming (LP) relaxation is large. Even on trees, their LP relaxation provides fractional solutions. In a parallel paper (Raghavan and Zhang 2018b), we propose a polynomial-time dynamic programming algorithm for the WTSS problem on trees. Further, we present a tight and compact extended formulation for the WTSS problem on trees. We project the extended formulation onto the space of the natural node variables, yielding the polytope of the WTSS problem on trees. The projection has an exponentially sized set of valid inequalities whose polynomial time separation is also discussed.

Shakarian et al. (2013) proposed the best-known heuristic for the TSS problem, which we discuss and adapt to the WTSS problem in Section 4 of this paper. Simultaneously to our work, Cordasco et al. (2015) discussed a heuristic for the WTSS problem that essentially applies the same idea as Shakarian et al. (2013) with a different weighting function in the selection procedure. For complete graphs, under the assumption that $b_i \geq b_j$ whenever $g_i \geq g_j$ for any two nodes i and j, Cordasco et al. show that their heuristic provides an optimal solution. In the Appendix to this paper, we show that the same holds for our adaptation of Shakarian et al.'s heuristic. Interestingly, Cordasco et al.'s heuristic actually performs very poorly (and significantly worse than our adaptation of Shakarian et al.'s heuristic) for the WTSS problem. In fact (as will be evident later), on the real-world graph instances tested, even our adaptation of Shakarian et al.'s heuristic performs poorly. On average, Shakarian solutions are 5.49 times larger than the lower bound and are 5.46 times larger than the solutions (i.e., upper bounds) generated by the branch-and-cut approach. This demonstrates (i) the significant improvement of our branch-and-cut approach (in terms of the quality of solutions provided) over the existing solution approaches in the literature, and (ii) the need for the research community to devise better heuristic procedures for the WTSS problem.

1.1. Our Contributions

To our knowledge, much of the previous work involving influence maximization problems has focused on heuristics and approximation. As our computational results will demonstrate, existing formulations do not provide quality upper or lower bounds for the WTSS problem and are computationally burdensome (large running times, poor optimality gaps, and large memory requirements). Our goal is to develop a quality optimization-based approach for this exciting viral marketing application on social networks that is able to solve to near-optimality instances of a meaningful (and large) size. The rest of this paper is organized as follows. Section 2 reviews formulations for the WTSS problem, including Raghavan and Zhang's (2018b) tight and compact extended formulation for trees. Section 3 develops a branch-and-cut approach. We show that on

a directed acyclic graph (DAG), Raghavan and Zhang's (2018b) formulation is tight (i.e., its LP relaxation provides integer solutions on the target set selection variables). Using the observation that the influence propagation network must be a DAG, the extended formulation can be embedded into a formulation on arbitrary graphs, where an additional exponentially sized set of constraints is added to ensure that the arcs selected form a DAG. We use this to design and implement a branch-and-cut approach for the WTSS problem on arbitrary graphs. Section 4 provides our computational study, in which we are able to obtain high-quality solutions for simulated instances on real-world graphs, with up to approximately 155,000 nodes and 327,000 edges within a two-hour time limit. In fact, on a test-bed of 180 simulated instances on real-world graphs, our branch-and-cut approach finds solutions that are on average 0.9% from optimality and solves 60 of the 180 instances to optimality, whereas the best heuristic produces solutions are on average 5.46 times larger.

2. Formulations

In this section, we first discuss existing formulations for the TSS problem in the literature. They apply to the WTSS problem with the addition of weights in the objective function. Next, we present Raghavan and Zhang's (2018b) tight and compact extended formulation for the WTSS problem on trees (that is a starting point for the branch-and-cut approach developed in this paper).

A combinatorial optimization problem can be formulated as an IP in multiple ways. The standard way (see Nemhauser and Wolsey 1988, Conforti et al. 2014) to compare different IP formulations for the same problem is to solve their LP relaxations (because this has implications for the computational tractability of a formulation). Then, these IP formulations are evaluated by the optimal objective values of their LP relaxations. Given two different IP formulations, A and B, of a given minimization problem, let z_A^{LP} and z_B^{LP} be the optimal objective value of their LP relaxations, respectively. We say that formulation A is stronger (or better) than formulation B if $z_A^{LP} > z_B^{LP}$. Typically, a stronger formulation is more computationally efficient (Barnhart et al. 1993).

Both Spencer and Howarth (2013) and Shakarian et al. (2013) proposed a time-indexed formulation. To model the order in which nodes become active, an artificial time index t is created, taking values from 1 to $|V|$. The formulation uses a binary variable $x_{i,t}$, which is set to 1 if node i is active in time period t and is 0 otherwise. For any node $i \in V$, let $n(i)$ denote the set of node i's neighbors. The model "TimeIndexed" is as follows:

$$\text{(TimeIndexed)} \quad \text{Minimize} \quad \sum_{i \in V} b_i x_{i,1} \tag{1}$$

$$\text{subject to:} \quad x_{i,|V|} = 1 \tag{2}$$

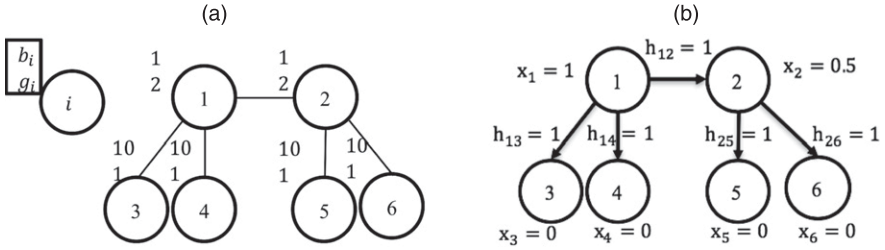
$$g_i x_{i,t-1} + \sum_{j \in n(i)} x_{j,t-1} \geq g_i x_{i,t} \quad \forall i \in V, t \in \{2, 3, \ldots, |V|\} \tag{3}$$

$$x_{i,t} \in \{0,1\} \qquad \forall i \in V, t \in \{1, 2, \ldots, |V|\}. \tag{4}$$

The objective function (1) minimizes the total weight of the nodes activated in time period 1 (i.e., the nodes selected in the target set). Constraint (2) makes sure that all nodes are active in time period $|V|$ (i.e., by the end of the diffusion process). Constraint (3) says that a node i is active in time period t only if it was active in time period $t-1$, or if it has at least g_i neighbors that were active in time period $t-1$.

Figure 1(a) describes a WTSS problem instance, for which the LP relaxation of TimeIndexed has a fractional optimal solution with $x_{1,1} = x_{2,1} = 0.0334$ and $x_{3,1} = x_{4,1} = x_{3,1} = x_{4,1} = 0$, and an objective value of 0.0668. However, the optimal integer solution selects nodes 1 and 2 (i.e., $x_{1,1} = x_{2,1} = 1$ and $x_{3,1} = x_{4,1} = x_{3,1} = x_{4,1} = 0$) and has an objective value of 2. In fact, we can analytically show that the LP relaxation of the TimeIndexed formulation can be arbitrarily weak. Consider a WTSS problem instance on a complete graph. For each node i in V, $b_i = 1$ and $g_i = |V| - 1$. Consider the LP relaxation of TimeIndexed. For this instance, a feasible solution to the LP relaxation has $x_{i,t} = (\frac{1}{2})^{|V|-t}$ for all i in V and $t \in \{1, 2, \ldots, |V| - 1\}$ and $x_{i,|V|} = 1$ for all i in V. The

Figure 1. (a) WTSS Problem Instance and (b) Fractional Optimal Solution to ACK



objective value of this fractional solution is $|V|(\frac{1}{2})^{|V|-1}$, which decreases to 0 in the limit as the size of the graph increases, whereas the optimal integer objective has the value $|V| - 1$.

Ackerman et al. (2010) introduced a different formulation. Here, for each node $i \in V$, a binary variable x_i is created denoting whether node i is selected in the target set. Then, to model the (implicit) order in terms of when nodes become active, a linear order is created among the nodes. For any two distinct nodes i and j in V, two binary variables h_{ij} and h_{ji} are created, with $h_{ij} = 1$ if node i is before node j in the linear order, and $h_{ji} = 1$ if node j is before node i in the linear order. Notice that the pair of variables h_{ij}, h_{ji} is created for every pair of nodes in the graph, regardless of whether they have an edge between them in the graph. The model "ACK" is as follows:

$$\text{(ACK)} \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \tag{5}$$

$$\text{subject to:} \quad h_{ij} + h_{ji} = 1 \qquad \forall i \neq j \in V \tag{6}$$

$$\sum_{j \in n(i)} h_{ji} + g_i x_i \geq g_i \qquad \forall i \in V \tag{7}$$

$$h_{ij} + h_{jk} + h_{ki} \leq 2 \qquad \forall i \neq j \neq k \in V \tag{8}$$

$$x_i \in \{0,1\} \qquad \forall i \in V \tag{9}$$

$$h_{ij}, h_{ji} \in \{0,1\} \qquad \forall i \neq j \in V. \tag{10}$$

The objective function (5) minimizes the total weight of the nodes selected in the target set. Constraint (6) makes sure that either node i is before node j or node j is before node i in the linear order. Constraint (7) says that a node $i \in V$ must be selected in the target set, or at least g_i of its neighbors become active before node i. Constraint (8) ensures that there is linear ordering.

Figure 1(a) provides a WTSS problem instance, and Figure 1(b) describes a fractional optimal solution to the LP relaxation of ACK. This solution has $x_1 = 1$, $x_2 = 0.5$, with all other x decision variables equal to zero. The nonzero h variables for the edges in the graph are shown in Figure 1(b). For the remaining node pairs $h_{15} = h_{16} = h_{23} = h_{24} = h_{34} = h_{35} = h_{36} = h_{45} = h_{46} = h_{56} = 1$ and the rest are zero. The objective value is 1.5. Although ACK is stronger than TimeIndexed, it is still weak, and solving its LP relaxation does not provide integer solutions for the WTSS problem on trees.

Although the above two formulations are valid for the WTSS problem, they are weak on trees in the sense that their LP relaxations do not provide integral solutions. Raghavan and Zhang (2018b) proposed a tight and compact extended formulation for the WTSS problem on trees that we now discuss. This is done by creating a new graph G^t, from the input graph G, by subdividing each edge. For each edge $\{i,j\} \in E$, insert a dummy node d. Let D denote the set of dummy nodes. Because the dummy nodes have effectively split each edge into two in the original graph, each of the original edges $\{i,j\} \in E$ is replaced by two edges $\{i,d\}$ and $\{d,j\}$ in the new graph G^t. Let E^t denote the set of edges in G^t ($G^t = (V \cup D, E^t)$). The dummy nodes cannot be selected in the target set, and all have a threshold of 1. Thus, if one of its neighbors is activated, the dummy node will become activated and will propagate the influence to the other neighbor. As before, for each node $i \in V$, the binary variable x_i denotes whether node i is selected in the target set (these are the natural node variables). For each edge $\{i,d\} \in E^t$, where $i \in V$ and $d \in D$ (notice that G^t is bipartite and E^t only contains edges between the nodes in V and D), create two binary arc variables y_{id} and y_{di} to represent the direction of influence propagation. If node i sends influence to node d, y_{id} is 1, and 0 otherwise. As before, for any node $i \in V \cup D$, $n(i)$ denotes the set of node i's neighbors. Figure 2 illustrates the outcome of the procedure described above. It shows a WTSS problem instance, the transformed graph G^t, and a solution represented in G^t (a shaded node is in the target set, and directed arcs represent the influence directions formed by the target set). Then, we can write the following tight and compact extended formulation "BIP_{tree}" for the WTSS problem on trees.

$$(\text{BIP}_{\text{tree}}) \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \tag{11}$$

$$\text{subject to:} \quad \sum_{i \in n(d)} y_{id} \geq 1 \qquad \forall d \in D \tag{12}$$

$$x_i \leq y_{id} \qquad \forall i \in V, d \in n(i) \tag{13}$$

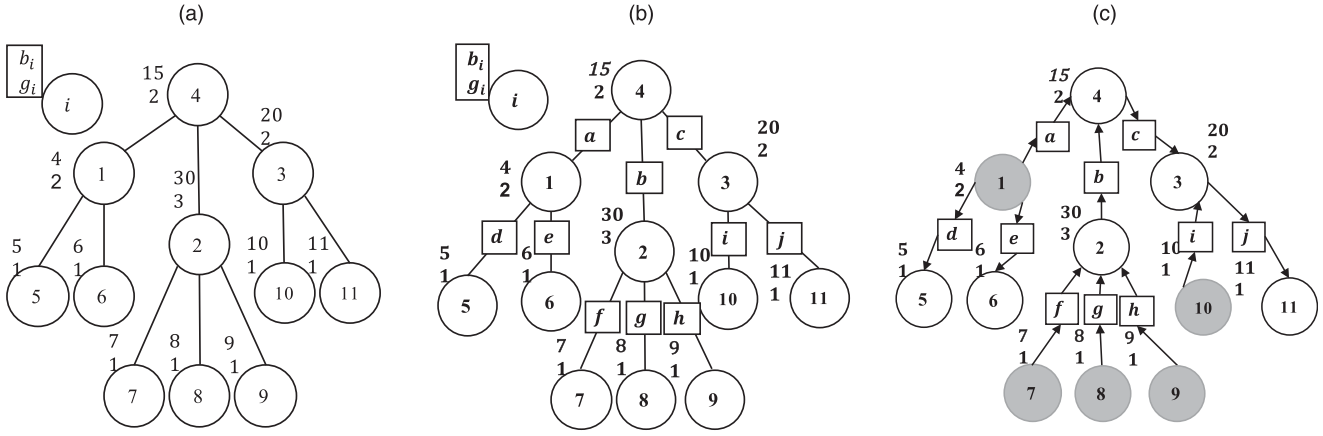
$$y_{id} + y_{di} = 1 \qquad \forall \{i,d\} \in E^t \tag{14}$$

$$\sum_{d \in n(i)} y_{di} + g_i x_i \geq g_i \qquad \forall i \in V \tag{15}$$

$$x_i \in \{0,1\} \qquad \forall i \in V \tag{16}$$

$$y_{id}, y_{di} \in \{0,1\} \qquad \forall \{i,d\} \in E^t. \tag{17}$$

Figure 2. (a) WTSS Problem Instance, (b) G^t, and (c) A Valid Solution to BIP_{tree}



Constraint (12) says that each dummy node has at least one incoming arc, because dummy nodes cannot be selected and have a threshold of 1. Constraint (13) says that if a node is selected, then it sends out influence to all of its neighbors. Notice that this type of constraint (e.g., $x_i \le h_{ij}$) would not be valid in ACK, because there could be two neighboring nodes that are in the target set. Constraint (14) makes sure that on each edge, influence is only propagated in one direction. Constraint (15) says that a node $i \in V$ must be selected or must have g_i or more incoming arcs. If a node $i \in V$ is selected, then it has no incoming arcs owing to constraint (13), and constraint (15) is satisfied. On the other hand, if a node $i \in V$ is not selected, it must have at least g_i incoming arcs.

The LP relaxation of BIP_{tree} provides integral solutions for the example in Figure 2. Raghavan and Zhang (2018b) show that (for trees) the projection of the feasible region of the LP relaxation of BIP_{tree} onto the space of the x variables provides the convex hull of target set vectors.

3. Strengthened Formulation and a Branch-and-Cut Approach for Arbitrary Networks

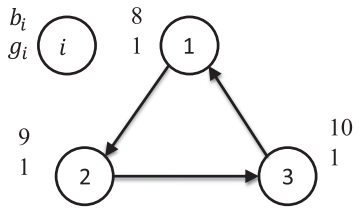
The extended formulation BIP_{tree} (discussed in Section 2) can be embedded into a larger model and applied to arbitrary graphs. This idea is based on the simple observation that the influence propagation process can be modeled as a DAG. We note that BIP_{tree} is not a valid formulation on any graph that contains a cycle. Figure 3 provides an example, illustrating the problem caused by a cycle. In this figure, we have a directed influence cycle of nodes 1, 2, and 3. Each of them has a threshold of 1 and has one directed incoming arc. In other words, all three nodes become active by being influenced by the others. However, in this cycle, nothing indicates which node is the first active one to initialize the influence propagation process. Put differently, the target set is empty, and this is not a feasible solution.

We now show that BIP_{tree} is a valid formulation for a DAG (with appropriate modifications, because the original graph is a DAG, as opposed to a bidirected graph), and its LP relaxation returns integral solutions on DAGs. We observe that the WTSS problem can be solved trivially on DAGs.

Proposition 1. *The WTSS problem on a DAG can be solved in $O(|E|)$ time.*

Proof. Given a DAG, the optimal solution of the WTSS problem is trivial to find. For a node i, set $x_i = 1$ if the number of its incoming arcs is smaller than its threshold. Otherwise, set $x_i = 0$. Notice that the WTSS problem solution must contain these nodes (set to $x_i = 1$) at a minimum. We now argue that this is a feasible solution. Consider the nodes of the DAG in topological order. By design, when we reach a node, its predecessors are all

Figure 3. The Problem with Influence Propagation Around a Cycle



active. Therefore, if it has more than g_i incoming arcs, this node will become active. Otherwise, it has fewer than g_i incoming arcs, which means that we have selected it in the target set. Because we did not select any nodes other than those selected by necessity, this solution is optimal. The time complexity is $O(|E|)$ because we need to scan through all nodes and check their adjacent arcs. □

Given a DAG, we put this data into BIP_{tree} in the following way. For any arc (j,i) in the DAG, we set $y_{jd} = 1$ and $y_{dj} = 0$ (to ensure that in BIP_{tree}, node j does not receive influence from node i) and leave y_{di} and y_{id} as decision variables in the model. Notice that this implies that constraint (12) can be removed from BIP_{tree}, because it is always satisfied. Let $p(i)$ denote the set of dummy nodes, which are adjacent to node i, while considering all incoming arcs (j,i) to node i in the DAG. We refer to the resulting IP as BIP_{DAG}, whose linear relaxation (LP_{DAG}) is as follows.

$$(\text{LP}_{\text{DAG}}) \quad \text{Minimize} \quad \sum_{i\in V} b_i x_i \tag{18}$$

$$\text{subject to:} \quad x_i \le y_{id} \qquad \forall i \in V, d \in p(i) \tag{19}$$

$$y_{id} + y_{di} = 1 \qquad \forall i \in V, d \in p(i) \tag{20}$$

$$\sum_{d\in p(i)} y_{di} + g_i x_i \ge g_i \quad \forall i \in V \tag{21}$$

$$x_i \ge 0 \qquad \forall i \in V \tag{22}$$

$$y_{id}, y_{di} \ge 0 \qquad \forall i \in V, d \in p(i). \tag{23}$$

In the dual to LP_{DAG} (which we refer to as DLP_{DAG}), we have w_{id}, z_{id}, and v_i as dual variables for constraint sets (19), (20), and (21), respectively. DLP_{DAG} can be written as follows:

$$(\text{DLP}_{\text{DAG}}) \quad \text{Maximize} \sum_{\text{i}\in\text{V}} \text{g}_\text{i}\text{v}_\text{i} + \sum_{\text{i}\in\text{V}} \sum_{\text{d}\in\text{p(i)}} \text{z}_\text{id} \tag{24}$$

$$\text{subject to:} \quad w_{id} + z_{id} \le 0 \qquad \forall i \in V, d \in p(i) \tag{25}$$

$$v_i + z_{id} \le 0 \qquad \forall i \in V, d \in p(i) \tag{26}$$

$$g_i v_i - \sum_{d\in p(i)} w_{id} \le b_i \qquad \forall i \in V \tag{27}$$

$$v_i \ge 0 \qquad \forall i \in V \tag{28}$$

$$w_{id} \ge 0 \qquad \forall i \in V, d \in p(i). \tag{29}$$

Theorem 1. LP_{DAG} *has an optimal solution with the* x *and* y *variables binary.*

Proof. Proposition 1 provides an optimal target set, which can be converted to a feasible solution to LP_{DAG}. Let S be the set of selected nodes. Set $x_i = 1$, $y_{id} = 1$, and $y_{di} = 0$ for all d in $p(i)$ if node $i \in S$. Then, set $x_i = 0$, $y_{id} = 0$, and $y_{di} = 1$ for all d in $p(i)$ if node $i \notin S$. Finally, for the case $x_i = 0$, we can reverse an appropriate number of arcs to ensure that constraint (21) is always binding, if necessary. We now construct a dual feasible solution to DLP_{DAG} that satisfies the complementary slackness (CS) conditions. The CS conditions between LP_{DAG} and DLP_{DAG} are as follows:

$$(g_i - \sum_{d\in p(i)} y_{di} - g_i x_i) v_i = 0 \quad \forall i \in V \tag{30}$$

$$(x_i - y_{id}) w_{id} = 0 \quad \forall i \in V, d \in p(i) \tag{31}$$

$$(b_i - g_i v_i + \sum_{d\in p(i)} w_{id}) x_i = 0 \quad \forall i \in V \tag{32}$$

$$(-v_i - z_{id}) y_{di} = 0 \quad \forall i \in V, d \in p(i) \tag{33}$$

$$(-w_{id} - z_{id}) y_{id} = 0 \quad \forall i \in V, d \in p(i). \tag{34}$$

Because constraint (21) is always binding for the (primal) feasible solution we constructed, the CS condition (30) is always satisfied (for any dual feasible solution). We now discuss the remaining CS conditions.

For a node $i \in S$, $x_i = 1$. For these nodes $i \in S$, we also have $y_{id} = 1$ and $y_{di} = 0$ for all d in $p(i)$. Set $v_i = w_{id} = \frac{b_i}{g_i - |p(i)|}$ and $z_{id} = -\frac{b_i}{g_i - |p(i)|}$ for nodes $i \in S$. Observe that constraints (25), (26), and (27) are binding with this choice and $w_{id} = \frac{b_i}{g_i - |p(i)|} > 0$ because $|p(i)| < g_i$ (otherwise node i would not be selected), showing that the solution is dual feasible. Because constraints (25), (26), and (27) are binding, all of the CS conditions are satisfied, as well.

For a node $i \in V \setminus S$, $x_i = 0$. We set all dual variables associated with node $i \in V \setminus S$ to zero; that is, $v_i = w_{id} = z_{id} = 0$ for all $d \in p(i)$. Then, it is easy to verify that this choice of dual variables is feasible. Further conditions (31), (33), and (34) are always satisfied because the dual variables are zero, whereas condition (32) is satisfied because $x_i = 0$ (or all of the CS conditions are satisfied), proving the integrality of the x variables.

Next, we show that the y variables are binary, given that the x variables are binary. Assume that this is not true, and there is an extreme point $(\mathbf{x}^*, \mathbf{y}^*)$ where $\mathbf{x}^*$ is binary and $\mathbf{y}^*$ is fractional. Observe that when $x_i^* = 1$, $y_{id}^* = 1$ and $y_{di}^* = 0$ for all $d \in p(i)$ owing to constraints (19) and (20). Because the x variables are binary, it implies that the fractional y values must correspond to the arcs adjacent to the node $i \notin S$.

We look at the supporting graph, which is formed by the fractional $\mathbf{y}^*$ values. In other words, only arcs with fractional values of y are kept in the supporting graph. Consider a connected component of the supporting graph, as shown in Figure 4(a). When an end node is in V (node 1 in Figure 4(a)), it has one fractional incoming arc, and all other incoming arcs are integers. Given that g_i is an integer, constraint (21) must be satisfied as a nonbinding constraint. Additionally, when an end node is in D (e.g., c and d in Figure 4(a)), there are no constraints for it.

Consider any two end nodes, denoted as s and t, in a connected component of the supporting graph. Let $P_{st} = \{(s,l),\ldots,(k,t)\}$ and $P_{ts} = \{(t,k),\ldots,(l,s)\}$ denote the fractional paths from nodes s to t and from nodes t to s, respectively. We construct two new feasible solutions $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ and $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ as follows:

$$\overline{\mathbf{x}} = \mathbf{x}^*; \overline{y}_a = \begin{cases} y_a^* + \epsilon, & a \in P_{st}, \\ y_a^* - \epsilon, & a \in P_{ts}, \\ y_a^*, & a \in E^t \setminus \{P_{st} \cup P_{ts}\}, \end{cases} \quad \text{and } \tilde{\mathbf{x}} = \mathbf{x}^*; \tilde{y}_a = \begin{cases} y_a^* - \epsilon, & a \in P_{st}, \\ y_a^* + \epsilon, & a \in P_{ts}, \\ y_a^*, & a \in E^t \setminus \{P_{st} \cup P_{ts}\}, \end{cases}$$

where ϵ is a sufficiently small positive value (ϵ should be less than the slack in both the nonbinding constraints at the two end nodes s and t and should also satisfy $0 \le y_a^* \pm \epsilon \le 1$ for all $a \in \{P_{st} \cup P_{ts}\}$). Thus, $(\mathbf{x}^*, \mathbf{y}^*)$ is not an extreme point because $(\mathbf{x}^*, \mathbf{y}^*) = \frac{1}{2}(\overline{\mathbf{x}}, \overline{\mathbf{y}}) + \frac{1}{2}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$. □

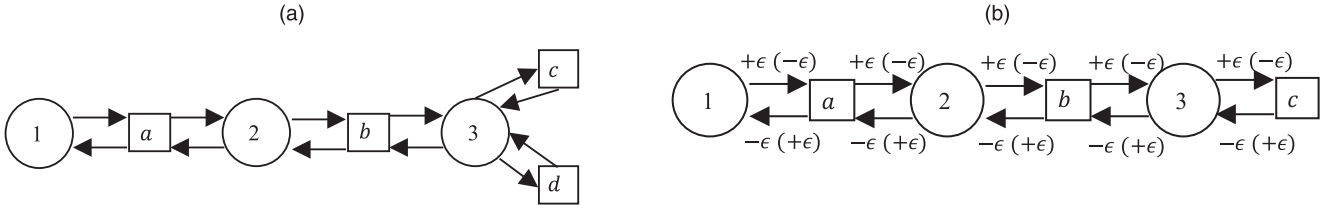
Our model for arbitrary graphs will introduce additional variables and constraints to model the fact that the influence propagation network implied by a feasible solution to the WTSS problem should be acyclic. In this model, we introduce a new variable set, h_{ij} and h_{ji}, for all $\{i,j\}$ in E (these are defined on the original graph). If $h_{ij} = 1$, it means that node i gives influence to node j. Otherwise, $h_{ij} = 0$. A new constraint set, which ensures that the directed graph formed by $\mathbf{h}$, and denoted by $G(\mathbf{h})$, is a DAG is needed. With this in hand, we have the following formulation, which we refer to as BIP:

$$\text{(BIP)} \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \tag{35}$$

$$\text{subject to:} \quad (13), (14), (15), (16), (17)$$
$$h_{ij} + h_{ji} = 1 \qquad \forall \{i,j\} \in E \tag{36}$$
$$\sum_{(i,j) \in C} h_{ij} \le |C| - 1 \qquad \forall \text{dicycles C in G} \tag{37}$$
$$h_{ij} \le y_{id}, \; h_{ji} \le y_{jd} \qquad \forall \{i,j\} \in E \& \{i,d\}, \{j,d\} \in E^t \tag{38}$$
$$h_{ij}, h_{ji} \in \{0,1\} \qquad \forall \{i,j\} \in E. \tag{39}$$

The objective function is the same as before. Constraint set (36) says that influence must go in one direction between two nodes in the original graph. Constraint set (37), called k-dicycle inequalities, ensures that influence cannot propagate around a directed cycle (dicycle). Here, k refers to the cardinality $|C|$ of the dicycle. For any dicycle C, the constraint says at most, $|C| - 1$ of the h_{ij} variables can be 1. Because the h variables are

Figure 4. Illustrating the Proof of Theorem 1



A connected component of the supporting graph formed by the fractional $\mathbf{y}^*$ values

Constructing two new feasible solutions using the fractional paths between end node 1 and c

defined on the original graph G, and the y variables are on the transformed graph G_t, we need constraint set (38) to serve as linking constraints, which synchronize the influence propagation process between these two graphs (notice that when $h_{ij} = 1$, $y_{id} = 1$, and when $h_{ji} = 1$, $y_{jd} = 1$). Note that this is how we obtained LP_{DAG} earlier. Constraint set (39) enforces the binary constraint on h variables. (Notice, as in the DAG, we have dropped constraint (12) because it is implied by constraint sets (36) and (38).)

To solve the LP relaxation of BIP, it is necessary to solve the separation problem for the exponential set of k-dicycle inequalities (37). Grötschel et al. (1985) present a separation procedure for k-dicycle inequalities, which is based on the shortest path algorithm. We implement this separation procedure in our approach. Further, we make an observation that in BIP, it is sufficient to define the h variables as binary (i.e., integrality can be relaxed on the x and y variables). This can be helpful in terms of branching.

Corollary 1. *Integrality on the x and y variables can be relaxed in BIP.*

Proof. Once the h variables are binary and satisfy constraints (37) and (36), we have a DAG, and the integrality of x and y follow as a direct consequence of Theorem 1. □

We take advantage of Corollary 1 by dropping constraints (16) and (17) from BIP. This leaves the h variables as the only binary variables in the model, on which we branch. We call this the H-branching rule. We note that it is possible to derive a formulation without the h variables and work directly with the x and y variables (i.e., a formulation that works on the transformed graph directly). This formulation would simply add the k-dicycle constraints defined on the transformed graph (i.e., with the y variables) to BIP_{tree}. However, there are three reasons for focusing on (working with) BIP. First, in terms of its LP relaxation, it is easy to show that any feasible solution to the LP relaxation of BIP is feasible to the LP relaxation of this alternate model (showing that in terms of LP relaxations, it is no worse than the alternate model). Second, BIP requires fewer binary variables (because the x and y variables need to be defined as binary variables in this alternate model), which makes a significant difference in the branch-and-cut approach. Last (because the k-dicycle constraints are defined on the h variables), BIP has a much smaller supporting graph for the separation procedure, which saves significantly on time. In fact, solving the LP relaxation of BIP is 10 times faster than this alternate model empirically according to a preliminary test set.

We now discuss some additional features of the branch-and-cut approach. To provide the branch-and-cut approach with an initial feasible solution, we use a greedy heuristic. The greedy heuristic selects the smallest weight node among the inactive nodes and adds it to the target set. It then updates the graph by propagating influence from the target set. This is repeated until a feasible solution is found (i.e., the target set can influence the entire network). In the extended formulation BIP, we use additional variables to model the influence propagation process. Thus, the IP search process spends a significant amount of time on these variables because of the binary requirement. However, we are actually only interested in the target set (i.e., the natural node (x) variables). Consequently, for a given $\mathbf{x}$, we consider the integral portion of the solution (i.e., those nodes with $x_i = 1$) and determine the set of nodes denoted by V_a that can be activated using this target set. We call this process *Feasibility Lift*. If all nodes are activated by these selected nodes ($V_a = V$), we have obtained an integral 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 the 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 Graph* and denote it as $G_{IIG} = G - V_a = G[V \setminus V_a]$. In this way, we have a much smaller supporting graph and can add fewer violated inequalities in the branch-and-cut procedure. Algorithm 1 presents the pseudocode of this procedure. In the worst case, we need to run the separation $|V|$ times. A shortest-path algorithm is executed each time. Thus, it depends on the specific shortest-path algorithm chosen here. In our implementation, we use Dijkstra's algorithm with a Fibonacci heap. Thus, the time complexity of Algorithm 1 is $O(|V|(|E| + |V| \log |V|))$.

Algorithm 1 (Feasibility Lift and Inactive Induced Graph)

Input: Graph G and the solution $(\mathbf{x}, \mathbf{y})$ in the current node.

1. Let $\mathbf{x}^1 = \{i \in V : x_i = 1\}$.
2. Let V_a be the set of nodes activated by the target set $\mathbf{x}^1$.
3. **if** $V_a = V$ **then**
4. A feasible target set is found and the current node is fathomed.
5. **else**
6. Let $G_{IIG} = G - V_a = G[V \setminus V_a]$ be the subgraph induced by the inactive nodes (Inactive Induced Graph).

7. Run the separation procedure on G_{IIG} for violated k-dicycle inequalities.
8. **end if**

We realize that the separation procedure is expensive. Therefore, we run the branch-and-cut procedure in the following way: In the root node, we do our best to find the violated inequalities to achieve a better dual bound (i.e., we focus on the cutting plane method). However, once we enter the branching phase, the separation procedure is invoked only when the integrality constraints are satisfied at a node. We refer to this as "lazy separation."

4. Computational Experiments

We now discuss our computational experience with the branch-and-cut approach. We start by describing our data sets: consisting of one real-world and two simulated data sets. We conducted four sets of experiments. First, we compared the strength of our proposed model BIP with the previous models in the literature by solving their LP relaxations. Next, we evaluated the performance of the proposed branch-and-cut approach on the real-world data sets. In the third set of experiments, we compared the optimal solution with a set of existing heuristics in the literature. In the final set of experiments, we study the effect of graph density on the proposed branch-and-cut approach. To conduct our experiments, we used CPLEX 12.7 with Python API and ran our tests on a machine with an Intel Xeon E5-2630V4 processor, 64 GB of RAM, and a 5 TB disk, under the Ubuntu 14.04 operating system.

4.1. Data Sets

We generated three data sets. The first one is based on 18 real-world graphs: Gnutella (five graphs), Bitcoin (two graphs), Autonomous Systems (two graphs), Ning, Hamsterster, Escorts, Anybeat, Advogato, Google Plus, Facebook1, Facebook2, and Douban. They are taken from several online repositories, including the Stanford Large Network Dataset Collection (SNAP; see Leskovec and Krevl 2014), the BGU Social Networks Security Research Group (BGU; see Lesser et al. 2013), the Koblenz Network Collection (KONECT; see Kunegis 2017), and the Network Repository (N.R.; see Rossi and Ahmed 2015). 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.

Gnutella is a large file-sharing peer-to-peer network (and the first decentralized peer-to-peer network of its kind). The Gnutella graphs G04, G05, G06, G08, and G09 are snapshots of the Gnutella network on August 4, 5, 6, 8, and 9 of 2002. The nodes represent hosts in the Gnutella network topology, and the edges represent connections between the Gnutella hosts. Row "G04" in Table 1 shows that the Gnutella network on August 4, 2002 is obtained from SNAP and has 10,876 nodes and 39,994 edges. *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." Bitcoin Alpha has 3,783

Table 1. Source and Size of Real-World Graphs

Graph Name	Source	No. of nodes	No. of edges
G04	SNAP	10,876	39,994
G05	SNAP	8,846	31,839
G06	SNAP	8,717	31,525
G08	SNAP	6,301	20,777
G09	SNAP	8,114	26,013
B-Alpha	SNAP	3,783	24,186
B-OTC	SNAP	5,881	35,592
AS01	SNAP	10,670	22,002
AS02	SNAP	10,900	31,180
Ning	BGU	9,727	40,570
Hamsterster	Konect	1,788	12,476
Escorts	Konect	10,106	39,016
Anybeat	N.R.	12,645	49,132
Advogato	N.R.	5,042	39,277
Gplus	Konect	23,613	39,182
Facebook1	BGU	39,439	50,222
Facebook2	Konect	2,888	2,981
Douban	N.R.	154,908	327,162

nodes and 24,188 edges. Bitcoin OTC has 5,881 nodes and 35,592 edges. *Autonomous Systems* is the graph of routers comprising the Internet. Given that the WTSS problem models the dissemination of ideas and influence, 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. AS01 has 10,670 nodes and 22,002 edges. AS02 has 10,900 nodes and 31,180 edges.

Ning is an online platform for people and organizations to create custom social networks. Snapshots of the friendship and group affiliation networks from Ning were harvested during September 2012, resulting in a graph with 9,727 nodes and 40,570 edges. *Hamsterster* contains friendships between users of the website hamsterster.com. It has 1,788 nodes and 12,476 edges. *Escorts* is the bipartite network of buyers and their escorts. The nodes are buyers and escorts. An edge denotes transactions between a male buyer and a female escort. We treat buyers and escorts in the same way and do not distinguish between them when we select a target set. Escorts has 10,106 nodes and 39,016 edges. *Anybeat* is an online community, a public gathering place where individuals can interact with people from around their neighborhood or across the world. The data represent the friendship network and have 12,645 nodes and 49,132 edges. *Advogato* is based on the friendship network of Advogato.org. It has 5,042 nodes and 39,277 nodes. *Gplus* is a snapshot of a small portion of the major social network Google+. It has 23,613 nodes and 39,182 edges. *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*). It contains 39,439 nodes and 50,222 edges. *Facebook2* contains Facebook user-user friendships collected manually, starting from a set of 10 selected users (in the source, Facebook2 is referred to as *Facebook(NIPS)*). The Facebook2 network has 2,888 nodes and 2,981 edges.

In addition to these 17 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 website 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 18 real-world graphs, 10 WTSS problem instances are generated by simulating values for b_i and g_i. The value of g_i is drawn from a discrete uniform distribution between $[1, |n(i)|]$, and weight b_i is drawn from a discrete uniform distribution between $[1, 100]$. Thus, there are 180 instances in total for the first data set.

The second and third data sets are based on simulated graphs, generated using Watts and Strogatz's (1998) method described in their pioneering work on social network analytics. Their method starts with a graph that is a cycle over n nodes. Next, each node in the cycle is connected to its k nearest neighbors (or $k-1$ neighbors if k is odd). Then with the "rewiring probability" p, each edge $\{u, v\}$ in this initial graph (cycle with each node connected to its k nearest neighbors) is replaced by an edge $\{u, w\}$, with the choice of node w being made uniformly randomly. We chose the rewiring probability p as 0.3 because Watts and Strogatz showed that this value corresponds most closely with the social networks they studied. Hagberg et al. (2008) provide a Python package *NetworkX* that implements Watts and Strogatz's method. It includes a function *connected_watts_strogatz_graph* that generates connected graphs. Because the Watts and Strogatz method is not guaranteed to generate connected graphs, *connected_watts_strogatz_graph* repeatedly applies the method (discarding graphs that are not connected) until a connected graph is generated. 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, 1,000, 1,200}. For each setting, 10 graph instances are generated, and there are 50 small simulated graph instances in total. The third data set consists of large simulated graphs with 2,500, 5,000, and 10,000 nodes, respectively, and exactly 20,000 edges. For each setting, 10 graph instances are generated, providing 30 instances in total. For each simulated graph instance (in the small simulated and large simulated data sets), we generated a WTSS problem instance by drawing g_i from a discrete uniform distribution between $[1, |n(i)|]$ and b_i from a discrete uniform distribution between $[1, 100]$. All 260 test instances used in this paper are available at http://dx.doi.org/10.17632/5635rkshm7.1.

4.2. Comparing the LP Relaxations of the Formulations

We compare the strength of the LP relaxations of TimeIndexed, ACK, and BIP. Recall that in Section 2 we analytically showed that the LP relaxation of the TimeIndexed formulation can be arbitrarily weak. In fact, when implemented, the LP relaxation of TimeIndexed could not solve any real-world instances, thus making it a nonviable option to solve the WTSS problem. The LP relaxation of the ACK formulation is also unable to solve any real-world instances. The problem with both TimeIndexed and ACK is the number of variables. Although ACK is a compact formulation, it has a variable for every node pair (or $O(|V|^2)$ variables), which makes the formulation grow very rapidly, with the result that even its LP relaxation cannot be solved for these

large real-world graphs. Needless to say, when the LP relaxation of an IP formulation cannot be solved, the model is not viable because the branch-and-bound approach is predicated on solving the initial LP relaxation. Consequently, TimeIndexed and ACK are not viable formulations for the WTSS problem.

To obtain a baseline to compare with the LP relaxation of BIP, we modify ACK to make its LP relaxation viable. First, instead of defining h_{ij} for every pair of nodes, we define only the h_{ij} and h_{ji} variables for edges $\{i, j\}$ in E. This greatly reduces the number of variables. To enforce the (implicit) ordering between nodes, we replace constraint (8) with constraint (37) (the k-dicycle inequalities). We refer to this formulation as ACKD. A natural question is the equivalence of ACKD to ACK. Following an identical procedure to theorem 1 in Newman and Vempala (2001), we can show that ACK and ACKD are equivalent in terms of the strength of the LP relaxation (i.e., their LP relaxations have the same objective value). Although ACKD has exponentially many k-dicycle constraints, we can add violated constraints on the fly and solve its LP relaxation.

Table 2 shows the results of the LP relaxations of BIP and ACKD. Both relaxations were provided a 1-hour time limit (because that was the time limit used later for the branch-and-cut procedure), which turned out to be more than adequate for these real-world instances. The first column has the graph name. The two major columns "ACKD-LP" and "BIP-LP" are the objective values of the LP relaxation of ACKD and BIP, respectively. We report the average, minimum, and maximum value in the "Avg," "Min," and "Max" subcolumns, respectively. The relative improvement obtained by the LP relaxation of BIP over that of ACKD can be calculated as $\frac{z^{LP}_{BIP} - z^{LP}_{ACKD}}{z^{LP}_{ACKD}} \times 100$, where z^{LP}_{BIP} and z^{LP}_{ACKD} denote the optimal objective value of the LP relaxations of BIP and ACKD, respectively. This improvement is shown under the "Improvement" column. Again, we report the average, minimum, and maximum values across the ten instances for each real-world graph. Figure 5 plots the results shown in the column "Improvement." For each real-world graph, the plot shows the average, minimum, and maximum value of the improvement across the 10 instances. We observe that except for Facebook1 and Facebook2, all other graphs have an improvement of more than 50%, on average. Additionally, 13 of the 18 graphs have an LP relative improvement of more than 100% (meaning that the optimal objective value of the LP relaxation of BIP is at least twice that of the optimal objective value of the LP relaxation of ACKD), and 9 of them are more than 150%. Hamsterster and Advogato even have an improvement of more than 200%. As we will see in the next section, Facebook1 and Facebook2 are actually easy instances because both BIP and ACKD solved them optimally. However, Hamsterster and Advogato are much more difficult to solve. Therefore, as these experiments demonstrate, BIP is a much stronger formulation than ACKD. Its LP relaxation improves upon ACKD, especially for harder instances.

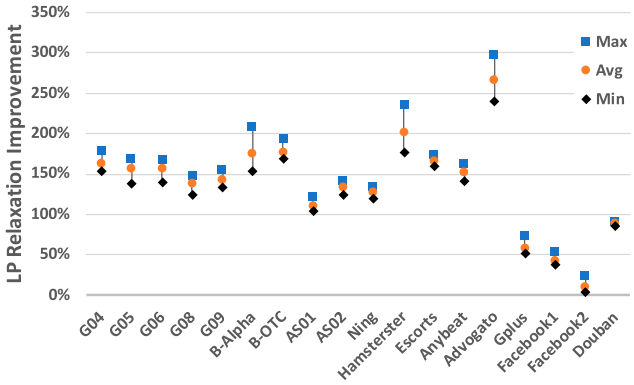
4.3. Testing the Proposed Branch-and-Cut Approach

In the second set of experiments, we test the proposed branch-and-cut approach on the 180 real-world instances. To test the benefits of the enhancements applied in our branch-and-cut approach, we compare three

Table 2. Improvement (%) of the LP Relaxation of BIP over That of ACKD on Real-World Graphs

	ACKD-LP			BIP-LP			Improvement (%)		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
G04	9,922.83	9,023.58	10,715.59	25,991.47	24,247.66	27,390.06	162.29	152.67	178.58
G05	8,345.33	7,373.18	9,716.12	21,327.74	18,481.31	23,150.24	155.99	137.79	168.40
G06	8,251.26	7,638.81	9,255.90	21,065.34	19,870.07	22,611.02	155.63	139.33	166.62
G08	6,804.89	6,371.56	7,667.45	16,181.10	15,191.50	17,154.47	137.95	123.73	147.75
G09	8,632.51	8,178.79	9,568.25	20,861.67	19,584.42	22,876.98	141.75	133.88	154.63
B-Alpha	2,466.83	2,178.47	2,751.39	6,759.28	5,860.88	7,648.75	174.14	152.92	207.09
B-OTC	3,325.21	2,962.34	3,739.51	9,196.27	8,146.95	10,194.25	176.71	168.50	194.08
AS01	8,672.15	8,211.01	9,579.22	18,146.10	17,319.84	19,476.00	109.41	103.32	121.29
AS02	6,986.76	6,412.83	7,524.49	16,245.14	15,100.61	17,367.70	132.57	124.60	140.33
Ning	10,756.15	9,896.38	12,073.17	24,474.32	22,664.96	27,177.40	127.62	118.90	132.88
Hamsterster	948.25	788.53	1,173.58	2,848.05	2,240.76	3,387.07	200.74	176.50	235.68
Escorts	8,930.92	8,224.33	10,021.93	23,652.64	22,496.00	26,000.47	164.98	159.44	173.53
Anybeat	6,058.66	5,585.71	6,594.29	15,257.20	14,077.53	16,031.22	151.93	140.63	162.79
Advogato	1,946.93	1,646.56	2,245.32	7,093.21	6,529.33	8,008.85	265.45	240.33	296.54
Gplus	2,221.62	1,941.61	2,577.00	3,486.18	3,028.00	3,999.00	57.03	51.52	72.90
Facebook1	2,932.41	2,479.41	3,458.00	4,166.94	3,691.00	4,825.00	42.40	38.16	53.71
Facebook2	385.59	300.25	453.78	423.60	330.00	484.00	10.18	3.66	23.88
Douban	83,597.19	81,330.72	84,957.35	157,496.25	153,347.38	161,591.53	88.41	85.39	90.20

Figure 5. (Color online) Improvement (%) of the LP Relaxation of BIP over That of ACKD on Real-World Graphs



settings. "BIP-Basic" is the direct implementation of the BIP formulation with k-dicycle separation. "BIP-Full" uses all of the enhancements discussed previously. Specifically, it adds to "BIP-Basic" the greedy initial solution, the H-branching rule, Feasibility Lift, Inactive Induced Graph separation, and lazy separation. To isolate the effect of our enhancements, we turn off CPLEX's cuts in both settings. The third setting is referred to as "BIP-Basic-CCuts." Here, CPLEX's cuts are turned on to compare our enhancements in BIP-Full against the addition of CPLEX's cuts. In our implementations, we removed the constraint $h_{ij} + h_{ji} = 1$ and used only variables h_{ij}, where $i < j$ in V. In this way, we reduced the size of the model by $|E|$ constraints and $|E|$ variables. Other than that, we keep the default settings for CPLEX. (Note that to implement a branch-and-cut procedure with an exponential number of constraints, one needs to use a control callback function in CPLEX. By default, CPLEX requires MIP models using control callback functions to run on a single thread.)

We first run our experiment on all of the real-world graphs, except for Douban (because it is much larger, we analyzed it separately), using a 1-hour time limit for each of the three branch-and-cut settings. We evaluate the optimality gap, calculated as $\frac{ub-lb}{ub} * 100$ (where ub denotes the upper bound and lb denotes the lower bound obtained by each setting), to compare across the three settings. BIP-Basic performs poorly in our experiments. Within the 1-hour time limit, BIP-Basic can find an upper bound (i.e., a feasible solution) in only 96 of 170 real-world instances. Furthermore, the average optimality gap between the upper and lower bounds exceeds 84% over these 96 instances. BIP-Basic-CCuts performance is similar to BIP-Basic, if not worse. It finds an upper bound in 74 instances and has an average gap of 81% across these instances. Hence, CPLEX's cuts were able to reduce the gap slightly when feasible solutions were obtained. On the other hand, BIP-Full performs very well in our experiment, clearly demonstrating the benefits of the enhancements applied in our branch-and-cut approach. Table 3 summarizes these results. Not only are feasible solutions obtained for all of the 170

Table 3. Optimality Gap of BIP-Full on Real-World Graphs

	Optimality gap (%)			Solved instances	
Graph Name	Avg	Min	Max	#	Avg Running Time (seconds)
G04	0.61	0.01	1.98	0	—
G05	0.58	0.00	3.68	3	867.90
G06	0.24	0.00	0.95	1	1,527.54
G08	0.07	0.00	0.61	8	588.72
G09	0.03	0.00	0.12	7	1,354.17
B-Alpha	0.16	0.00	0.72	7	1,759.66
B-OTC	0.68	0.13	1.95	0	—
AS01	0.06	0.00	0.55	7	117.34
AS02	1.53	0.07	4.72	0	—
Ning	2.89	1.92	4.10	0	—
Hamsterster	1.84	0.00	4.96	1	1,236.87
Escorts	0.46	0.00	1.34	1	1,386.08
Anybeat	1.73	0.24	3.35	0	—
Advogato	4.32	2.43	5.89	0	—
Gplus	0.79	0.00	4.19	5	84.79
Facebook1	0.00	0.00	0.00	10	72.12
Facebook2	0.00	0.00	0.00	10	0.43

instances, but they are also proven to be of high quality. Over the 170 instances, the average gap is 0.94%, and the largest gap is 5.89%. Furthermore, 60 instances are solved optimally, with an average running time of 587.19 seconds. Of the 110 instances that were not solved to optimality, the average gap is 1.47%.

We note that the enhancements applied to BIP-Full (over BIP-Basic) can also be applied to ACKD formulation. We wanted to determine whether ACKD (with similar enhancements as BIP-Full) would be capable of solving WTSS problem instances and also to gauge the differences in terms of computational performance between these two formulations. Consequently, we created the setting "ACKD-Full," that is, ACKD executed with all of the enhancements applied to BIP-Full. This also provides for a fairer comparison between BIP and ACKD.

Figure 6 compares the average optimality gap of BIP-Full and ACKD-Full. Across the 170 instances, the average optimality gap of ACKD-Full is approximately 47% (and the maximum gap is approximately 69%), which is significantly worse than BIP-Full. Facebook1 and Facebook2 are solved to optimality, whereas none of the remaining 150 real-world instances are. These results establish that BIP-Full outperforms ACKD-Full with much smaller average optimality gaps.

To understand whether the performance improvement of BIP-Full over ACKD-Full comes from better upper or lower bounds, we compared both the upper bounds and lower bounds from BIP-Full and ACKD-Full. Figure 7 compares the upper bounds. The improvement is calculated as $\frac{z^{UB}_{\text{ACKD-Full}} - z^{UB}_{\text{BIP-Full}}}{z^{UB}_{\text{ACKD-Full}}} \times 100$, where $z^{UB}_{\text{BIP-Full}}$ and $z^{UB}_{\text{ACKD-Full}}$ denote the objective values of the upper bounds obtained by BIP-Full and ACKD-Full, respectively. BIP-Full is able to find a better upper bound (i.e., feasible solution) than ACKD-Full for each of the 170 real-world instances, except for the 20 easy instances solved optimally by both BIP-Full and ACKD-Full. The average improvement is approximately 9%, and the largest improvement is approximately 25%. Figure 8 compares the lower bounds. The improvement is calculated as $\frac{z^{LB}_{\text{BIP-Full}} - z^{LB}_{\text{ACKD-Full}}}{z^{LB}_{\text{ACKD-Full}}} \times 100$, where $z^{LB}_{\text{BIP-Full}}$ and $z^{LB}_{\text{ACKD-Full}}$ denote the lower bounds obtained by BIP-Full and ACKD-Full, respectively. BIP-Full is able to find a better lower bound than ACKD-Full for 144 of the 170 real-world instances. The remaining 26 instances have the same lower bounds and are the relatively easier Gplus, Facebook1, and Facebook2 instances. The average improvement across the 170 instances is approximately 87%, and the largest improvement is approximately 138%.

We now focus on the Douban instances. We increased the running time to two hours for both BIP-Full and ACKD-Full, because the Douban graph is significantly larger than the other 17 real-world graphs. Table 4 summarizes our findings. Although none of the 10 instances are solved optimally, BIP-Full has average, minimum, and maximum optimality gaps of 0.13%, 0.06%, and 0.18%, respectively, whereas ACKD-Full has gaps of 54.32%, 48.60%, and 90.71%, respectively. We find that the average, minimum, and maximum upper bound improvements of BIP-Full over ACKD-Full are 13.93%, 4.59%, and 82.54%. On the other hand, the average, minimum, and maximum lower bound improvements of BIP-Full over ACKD-Full are 88.14%, 84.92%, and 89.70%.

Figure 6. (Color online) Average Optimality Gap (%) of BIP-Full and ACKD-Full

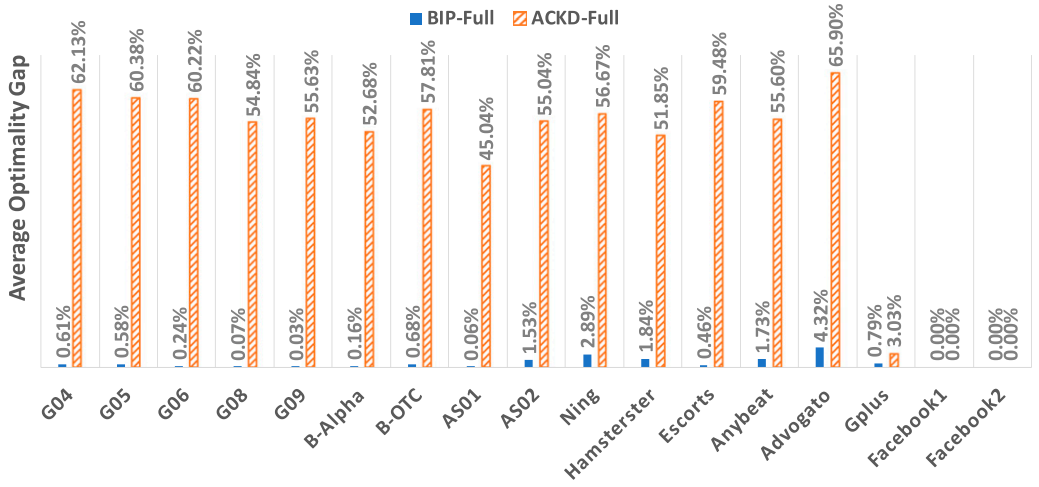
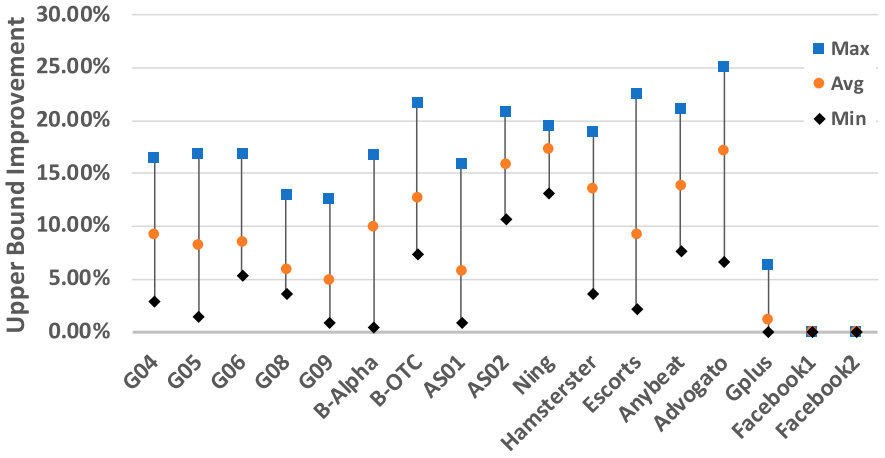


Figure 7. (Color online) Improvement (%) in BIP-Full Upper Bound over ACKD-Full Upper Bound on Real-World Graphs



To evaluate whether providing additional time to ACKD-Full would enable it to improve upon its optimality gaps, we ran it with a 20-hour time limit. Table 4 shows these results under the column headings "ACKD-Full (20 hr)." Notice that the average optimality gap reduces slightly to 48.92%. We note that after the first three hours, the gap went down very slowly (less than 2% in 17 hours), indicating very little progress in solving the problem using ACKD-Full. We would like to emphasize that longer runs result in an enormous space burden. ACKD-Full consumes an enormous amount of space. In the 20-hour ACKD-Full run, the size of the branch-and-cut search tree is more than 1 TB. On the other hand, in the 2-hour runs of BIP-Full, the size of the branch-and-cut search trees never exceeds 7 GB. Another point to note is that even with 10 times more time than BIP-Full, ACKD-Full is not able to improve upon the lower bounds (as the optimality gap stays at 48.92%), nor is it able to improve upon the upper bounds found by BIP-Full. BIP-Full's upper bounds are on average 5.51% better than the upper bounds found by ACKD-Full. This improvement stems from just 1 of the 10 instances, in which the 20-hour run of ACKD-Full was able to improve significantly upon the upper bound found in the two-hour run of ACKD-Full.

In short, BIP-Full finds better quality upper and lower bounds than ACKD-Full, and in contrast to ACKD-Full (which has large optimality gaps), is able to solve large real-world graph instances to near optimality. Providing additional time to ACKD-Full does not change the conclusions regarding the superior performance of BIP-Full.

4.4. Evaluating the Performance of Heuristics

The third set of experiments uses the Watts-Strogatz simulated networks with 200 nodes, where we evaluate the weight of the optimal solution against heuristics for the problem. Recall that in these small instances, we

Figure 8. (Color online) Improvement (%) in BIP-Full Lower Bound over ACKD-Full Lower Bound on Real-World Graphs

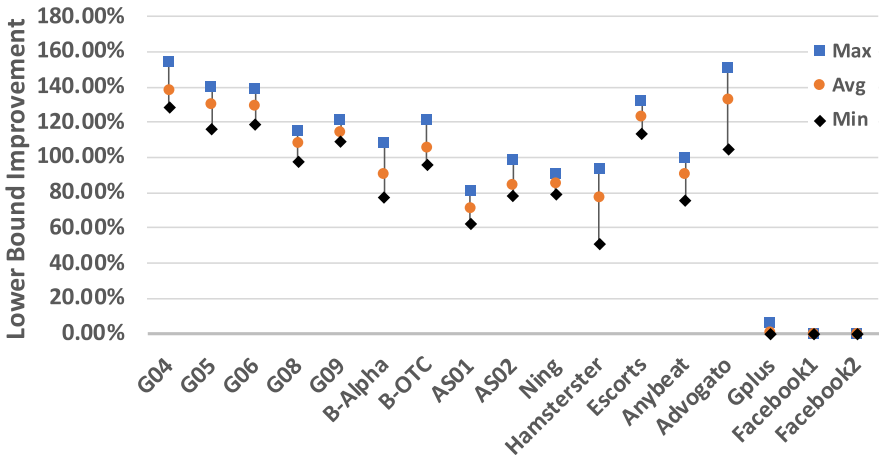


Table 4. Comparison of BIP-Full and ACKD-Full on the Douban Graph

	Optimality gap (%)			Upper bound improvement (%)		Lower bound improvement (%)	
	BIP-Full	ACKD-Full	ACKD-Full (20 hr)	ACKD-Full	ACKD-Full (20 hr)	ACKD-Full	ACKD-Full (20 hr)
Avg	0.13	54.32	48.92	13.93	5.51	88.14	84.77
Min	0.06	48.60	47.13	4.59	3.83	84.92	81.63
Max	0.18	90.71	51.60	82.54	9.91	89.70	86.54

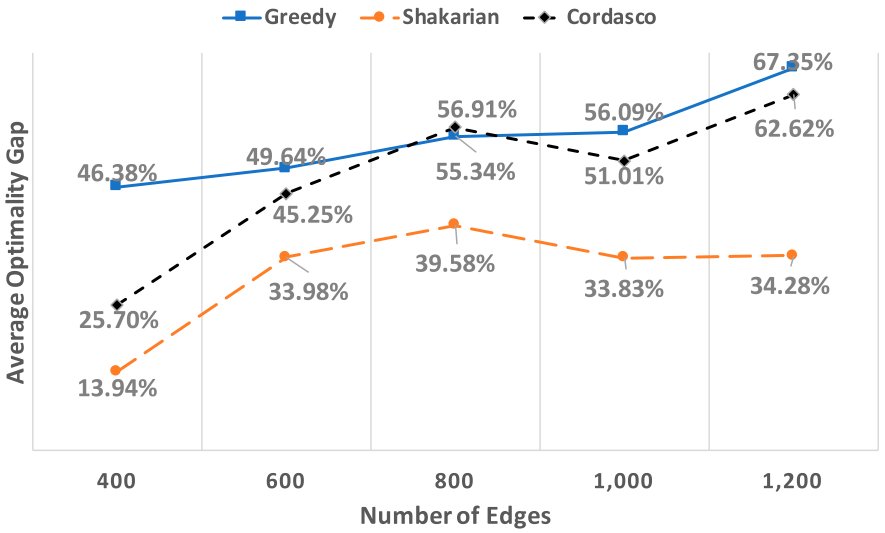
varied the graph density and created 50 graphs, where the numbers of edges take the values of 400, 600, 800, 1,000, and 1,200. These graph instances are solved to optimality using BIP-Full. We consider three heuristics to compare against optimal targeting. The first is our initial heuristic, which is a greedy algorithm. The second is based on a heuristic proposed by Shakarian et al. (2013) for the TSS problem. In this heuristic, a measure *dist* equal to $\deg(i) - g_i$ is computed for each node i in V. At each iteration, the heuristic picks the node with the smallest nonnegative *dist* and removes it. Then, the graph and *dist* are updated accordingly (the degree of nodes adjacent to node i is reduced by one). Once all nodes have a negative *dist*, the remaining nodes in the network are used as the target set. Because the heuristic does not consider weights, to adapt it to the WTSS problem, we use the node weights to break ties (by choosing the node with the higher weight) when picking the node with the smallest nonnegative *dist*. The third heuristic, proposed by Cordasco et al. (2015), is similar to that of Shakarian et al., but with two minor differences. First, it selects a node for removal from the graph by looking for the node with the largest value of $\frac{b_i g_i}{\deg(i)(\deg(i)+1)}$ (instead of the smallest nonnegative *dist*). Second, it immediately adds a node to the target set (instead of waiting until the end) when its degree is strictly smaller than its threshold value ($\deg(i) < g_i$) and updates the graph by propagating influence from the nodes added to the target set.

Table 5 displays the results. First, we wanted to compare the weight of the optimal target set against the total weight of the nodes in the graph. We compare the optimal objective value to the sum of all node weights, $\sum_{i \in V} b_i$. The results are shown in the "Percentage of total weight" column of Table 5. We can see that on average, the weight of the optimal target set for graphs with 400 edges is approximately 11% of the total weight of the nodes in the graph (recall that each row is an average of 10 instances). As the number of edges in the graph increases to 1,200, the weight of the optimal target set is approximately 5% of the total weight of the nodes in the graph. In Table 5, the gaps are calculated as the difference between the heuristic solution and the optimal solution, divided by the value of the optimal solution. The column "Greedy to opt gap" shows that the solutions obtained by the greedy strategy are approximately 50% greater than the weight of the optimal ones. Furthermore, this gap increases with graph density. The column "Shakarian to opt gap" shows that on average, the Shakarian et al. heuristic performs better, although its maximum gap values can be significantly greater than the greedy heuristic. The average gap for the Shakarian et al. heuristic is 14% when the number of edges is 400 and is greater than 30% when the number of edges is larger. The last column, "Cordasco to opt gap," shows that on average, the Cordasco et al. heuristic's performance is worse than that of the Shakarian et al. heuristic. The average gap for the Cordasco et al. heuristic is 26% when the number of edges is 400 and is greater than 62% when the number of edges is 1,200. Figure 9 compares the average optimality gaps of the three heuristics. On the basis of our experiments, we concluded that the Shakarian heuristic is the best of the three.

Table 5. Optimality Gaps (%) for the Greedy, Shakarian, and Cordasco Heuristics on Small (200 Node) Simulated Graphs

	Percentage of total weight		Greedy to opt gap (%)		Shakarian to opt gap (%)		Cordasco to opt gap (%)	
Edges	Avg	Max	Avg	Max	Avg	Max	Avg	Max
400	10.71	11.67	46.38	52.74	13.94	27.94	25.70	45.03
600	7.83	9.95	49.64	60.33	33.98	105.03	45.26	129.42
800	5.23	7.07	55.34	100.83	39.58	73.31	56.91	121.25
1,000	5.12	6.85	56.09	80.35	33.83	57.24	51.01	90.86
1,200	4.99	7.57	67.35	92.02	34.28	125.85	62.62	154.76

Figure 9. (Color online) Average Optimality Gaps (%) for the Greedy, Shakarian, and Cordasco Heuristic on Small (200 Node) Simulated Graphs



We decided to run the Shakarian heuristic (because it was the best one) on the 180 real-world instances and compare its performance with the upper bound obtained by BIP-Full. Let z^{Shak} denote the objective value of the Shakarian heuristic. Table 6 compares the Shakarian heuristic with the upper and lower bounds of BIP-Full on the 180 real-world instances. The $z^{\text{Shak}}/z^{UB}_{\text{BIP-Full}}$ columns provide an "upper bound ratio," with an average upper bound ratio of 5.47 over the 180 instances. The Shakarian heuristic performs somewhat better on the Gnutella and Escorts networks. The upper bound ratio is between 1.17 and 1.37, with an average of approximately 1.24. It is worth noting that Gplus, Facebook1, and Facebook2 have very big upper bound ratios. The average ratios are more than 22 for all three graphs, and the maximum ratio could be as high as 67. However, these instances are actually easier ones for BIP-Full. Even excluding these three graphs, the average ratio is still more than 1.57 across all other graphs. This clearly demonstrates that BIP-Full outperforms the best heuristic for the problem. The $z^{\text{Shak}}/z^{LB}_{\text{BIP-Full}}$ columns compare the Shakarian heuristic with the lower bound of BIP-Full. We observe a pattern similar to that of the upper bound ratio. The average ratio is 5.49, which means that these solutions are far from optimality and shows that even the best heuristic performs poorly for the problem.

Table 6. Comparing the Shakarian Heuristic with the Upper Bound and Lower Bound of BIP-Full on Real-World Graphs

	$z^{\text{Shak}}/z^{UB}_{\text{BIP-Full}}$			$z^{\text{Shak}}/z^{LB}_{\text{BIP-Full}}$		
	Avg	Min	Max	Avg	Min	Max
G04	1.21	1.19	1.24	1.21	1.19	1.26
G05	1.22	1.18	1.26	1.23	1.18	1.27
G06	1.22	1.17	1.28	1.22	1.17	1.28
G08	1.25	1.20	1.33	1.25	1.20	1.33
G09	1.27	1.21	1.33	1.27	1.21	1.33
B-Alpha	1.52	1.32	1.78	1.52	1.32	1.78
B-OTC	1.69	1.29	2.29	1.70	1.31	2.29
AS01	1.94	1.47	2.85	1.94	1.47	2.85
AS02	1.92	1.56	2.27	1.95	1.58	2.31
Ning	1.29	1.20	1.51	1.32	1.25	1.57
Hamsterster	1.33	1.15	1.58	1.36	1.20	1.60
Escorts	1.27	1.18	1.37	1.27	1.18	1.37
Anybeat	2.23	1.45	4.49	2.27	1.50	4.61
Advogato	1.37	1.22	1.55	1.43	1.28	1.63
Gplus	23.14	14.94	39.11	23.31	14.94	39.48
Facebook1	29.06	17.42	34.69	29.06	17.42	34.69
Facebook2	22.61	3.38	67.96	22.61	3.38	67.96
Douban	2.92	2.85	3.01	2.92	2.85	3.01

4.5. Studying the Effect of Graph Density

The last set of experiments investigates the role that graph density plays in the difficulty of solving the problem. For this, we use large simulated networks. Recall that we generated 10 graphs with 10,000 nodes and an average degree of 4, 10 graphs with 5,000 nodes and an average degree of 8, and 10 graphs with 2,500 nodes and an average degree of 16. We use the BIP-Full setting, but add 3-dicycle inequalities a priori (3-dicycles have $|C| = 3$ and are easy to add at the outset). As will be evident, graph density plays a huge role in terms of the difficulty in solving the problem (and thus, the running time). We give 5-minute, 1-hour, and 2-hour time limits to the 10,000-node, 5,000-node, and 2,500-node instances, respectively.

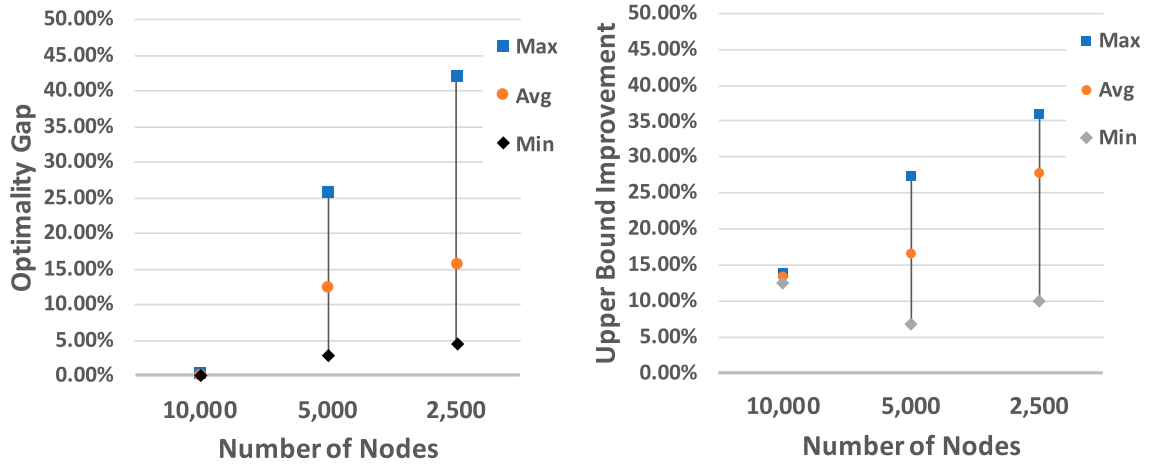
Figure 10 summarizes the results. On the left, it displays the optimality gaps. We observe that the average gap increases from 0.08% to 12.35% when the average degree increases from 4 to 8 (i.e., when the number of nodes goes from 10,000 to 5,000). Then, it increases to 15.78% when the average degree becomes 16. We should note that the larger gaps with 2,500 nodes are due to two particularly difficult instances for the branch-and-cut procedure. As indicated by the increased time limit for instances with greater graph density, it should be evident that the denser the graph, the harder a problem is to solve. Although the WTSS problem instances become more difficult to solve as the graph density increases, we note that our branch-and-cut approach still provides far superior solutions than the Shakarian heuristic. The right side of Figure 10 displays this improvement. On average, BIP-Full improves the Shakarian heuristic solution by approximately 19%. Furthermore, although the average optimality gap increases with the average degree, the average improvement increases from 13% to approximately 28%.

5. Conclusions

With the widespread use of online social networks, there is significant interest in solving problems that deal with viral marketing and influence maximization on social networks. Indeed, there are a growing number of software products, such as Buzzsumo (https://buzzsumo.com/), klear (https://klear.com/), Traackr (http://www.traackr.com/), webfluential (https://webfluential.com/), and zoomph (https://zoomph.com/), that aim to provide information on who influences whom to help advertisers/marketers determine whom to target. The next natural step (in this industry) would be to create software products that build upon this information to optimally target users. In this realm, our models and optimization-based solution methods have the most potential in playing a role.

In this paper, we proposed a branch-and-cut approach for the weighted version of the well-studied TSS problem. By observing that the influence propagation network forms a DAG, we show how the extended formulation for trees in Raghavan and Zhang (2018b) can be embedded into a formulation on arbitrary graphs, where an additional exponentially sized set of k-dicycle constraints are added. We design and implement a branch-and-cut approach for the WTSS problem on arbitrary graphs. Our computational results on a test-bed of 180 real-world graph instances are a testament to the quality of the formulation and the branch-and-cut approach. Over these 180 instances, the branch-and-cut approach finds solutions that are on average 0.9% from optimality and solves 60 of the 180 instances to optimality. Compared with the heuristic solutions, our branch-and-cut procedure provides significantly better upper bounds. Indeed, the performance of the

Figure 10. (Color online) Average Optimality Gaps (%) and Improvement (%) over the Shakarian Heuristic for BIP-Full on Large Simulated Graphs



heuristics on the WTSS problem suggests that there remains significant work to be done by the research community in developing improved heuristic procedures for the WTSS problem.

One generalization to the WTSS problem has unequal influence from neighbors. Unfortunately, this problem is somewhat harder to solve. We show in the Appendix that the WTSS problem with unequal influence is even NP-complete on stars (i.e., a tree where all nodes are leaves other than a single central node). Deriving strong formulations for this and other closely related influence maximization problems is an avenue of our current and future research.

In the WTSS problem, the diffusion process continues until it is complete (i.e., we are allowed as many steps or time periods as needed for the influence to propagate through the entire network). However, if we restrict the number of steps or time periods to one, the entire network must be influenced after one time step. We obtain a weighted version of a problem called the Positive Influence Dominating Set (PIDS) problem (proposed by Wang et al. 2011). This models the situation in which the marketer would like the diffusion process to take place rapidly. In the PIDS problem, either a node is selected at a weight of b_i, or it requires g_i of its neighbors to be selected (notice that when both b_i and g_i are 1, we obtain the dominating set problem). In a related paper (Raghavan and Zhang 2018a), we study the PIDS problem.

Another generalization is a prize-collecting variant of the WTSS problem that does not require 100% adoption. In addition to the weight, each node also has a profit. Instead of minimizing the cost of a target set activating all nodes, the goal is to capture the tradeoff between the total weight and the ultimate profit resulting from the influence propagation process of a target set. All of these are avenues for future research.

Appendix

A. WTSS Problem with Unequal Influence

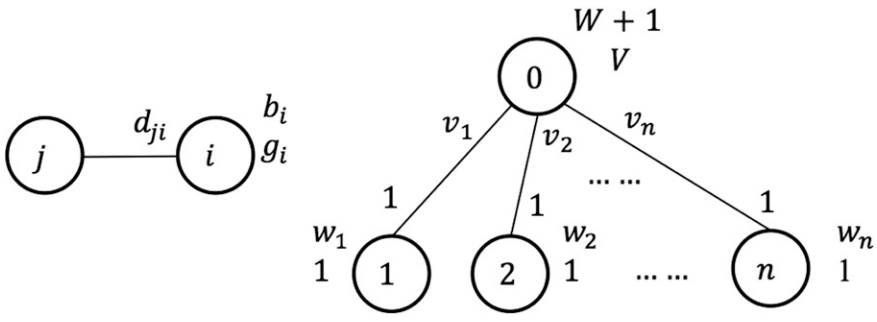
We prove that when a node i receives unequal influence from its neighbors, the WTSS problem is NP-complete on stars. In the propagation progress in the unequal influence case, the incoming influence of node i is calculated as $\sum_{j\in s(i)} d_{ji}$, where d_{ji} is the amount of influence from node j, and $s(i)$ is the set of node i's active neighbors. Furthermore, each node's threshold value g_i can take a value between 1 and $\sum_{j\in n(i)} d_{ji}$.

Theorem A.1. *The WTSS problem with unequal influence is NP-complete on stars.*

Proof. The decision version of the 0-1 knapsack problem is NP-complete (Kellerer et al. 2004) and is defined as follows: Given a set of n items numbered from 1 up to n, each item i with a weight w_i and a value v_i, along with a maximum weight capacity W, can we select a subset of these items such that a value of at least V will be achieved without exceeding the weight W?

We construct a star network from this 0-1 knapsack problem. For each item i, we put a node i in the graph as a leaf node. After that, we add one extra node and label it as node 0, which is the central node. All leaf nodes connect to the central node but do not connect to each other. Thus, there are $n + 1$ nodes numbered from 0 up to n in the graph. Then, for each leaf node i, we have $g_i = 1$, $d_{0i} = 1$, and $b_i = w_i$. For the central node 0, we have $g_0 = V$, $d_{i0} = v_i$ for all $i \in a(0)$ and $b_0 = W + 1$. The constructed star is shown in Figure A.1. The decision question is: Can we select a subset of nodes without the total weight exceeding W to activate the whole network? It is easy to see that we should never select the central node 0. Thus, it is equivalent to ask: Can we select a subset of leaf nodes such that the incoming influence of the central node is at least V and the total weight of those selected leaf nodes does not exceed W? Therefore, if the answer is "Yes," those selected leaf nodes also solve the 0-1 knapsack problem. □

Figure A.1. Transforming a 0-1 Knapsack Problem to the WTSS Problem with Unequal Influence on Stars



B. Optimality of the Shakarian et al. Heuristic on Complete Graphs

Given a complete graph with the assumption that $b_i \geq b_j$ whenever $g_i > g_j$ for any two nodes i and j, we show that our adaptation of the Shakarian et al. heuristic returns the optimal solution. Because we are dealing with a complete graph, the node selected for removal in the heuristic is the node with the highest threshold value among those with a nonnegative *dist*. Without loss of generality, assume that node i is the first node with a negative *dist*. In other words, node i is added first to the target set. Denote the set of nodes selected for removal before node i's *dist* value becomes negative as N_i. Observe that for all $j \in N_i$, we have $g_j \geq g_i$ and $b_j \geq b_i$. Consequently, replacing node i in the target set by any node $j \in N_i$ results in a feasible target set with a no better objective value. Because node i's *dist* value is negative, at least one of the nodes in $\{i \cup N_i\}$ must be in the optimal target set. Repeating this argument iteratively for each of the nodes added to the target set in the order that they are added completes the proof.

References

Ackerman E, Ben-Zwi O, Wolfovitz G (2010) Combinatorial model and bounds for target set selection. *Theoret. Comput. Sci.* 411(44):4017–4022.

Barnhart C, Johnson EL, Nemhauser GL, Sigismondi G, Vance P (1993) Formulating a mixed integer programming problem to improve solvability. *Oper. Res.* 41(6):1013–1019.

Ben-Zwi O, Hermelin D, Lokshtanov D, Newman I (2011) Treewidth governs the complexity of target set selection. *Discrete Optim.* 8(1):87–96.

Chen N (2009) On the approximability of influence in social networks. *SIAM J. Discrete Math.* 23(3):1400–1415.

Chen W, Castillo C, Lakshmanan LV (2013) *Information and Influence Propagation in Social Networks* (Morgan & Claypool Publishers, San Rafael, CA).

Chiang CY, Huang LH, Li BJ, Wu J, Yeh HG (2013) Some results on the target set selection problem. *J. Combin. Optim.* 25(4):702–715.

Conforti M, Cornuéjols G, Zambelli G (2014) *Integer Programming* (Springer, New York).

Cordasco G, Gargano L, Rescigno AA, Vaccaro U (2015) Optimizing spread of influence in social networks via partial incentives. Scheideler C, ed. *Structural Information and Communication Complexity* (Springer, New York), 119–134.

Granovetter M (1978) Threshold models of collective behavior. *Amer. J. Sociol.* 83(6):1420–1443.

Grötschel M, Jünger M, Reinelt G (1985) On the acyclic subgraph polytope. *Math. Programming* 33(1):28–42.

Hagberg AA, Schult DA, Swart PJ (2008) Exploring network structure, dynamics, and function using networkX. Varoquaux G, Vaught T, Millman J, eds. *Proc. 7th Python Sci. Conf., Pasadena, CA*, 11–15.

Kellerer H, Pferschy U, Pisinger D (2004) *Knapsack Problems* (Springer, New York).

Kempe D, Kleinberg J, Tardos É (2003) Maximizing the spread of influence through a social network. *Proc. 9th ACM SIGKDD Internat. Conf. Knowledge Discovery Data Mining* (ACM, New York), 137–146.

Kunegis J (2017) Konect network dataset – KONECT. Accessed October 15, 2018, http://konect.uni-koblenz.de/networks/konect.

Leskovec J, Krevl A (2014) SNAP datasets: Stanford large network dataset collection. Accessed October 15, 2018, http://snap.stanford.edu/data.

Lesser O, Tenenboim-Chekina L, Rokach L, Elovici Y (2013) Intruder or welcome friend: Inferring group membership in online social networks. *Social Computing, Behavioral-Cultural Modeling and Prediction* (Springer, Heidelberg, Germany), 368–376.

Nemhauser G, Wolsey L (1988) *Integer and Combinatorial Optimization* (Wiley, New York).

Newman A, Vempala S (2001) Fences are futile: On relaxations for the linear ordering problem. Aardal K, Gerards B, eds. *Integer Programming and Combinatorial Optimization—IPCO 2001*, Lecture Notes in Computer Science, vol. 2081 (Springer, Berlin, Heidelberg), 333–347.

Raghavan S, Zhang R (2018a) Rapid influence maximization on social networks: The positive influence dominating set problem. Working paper, University of Maryland, College Park.

Raghavan S, Zhang R (2018b) Weighted target set selection on tress and cycles. Working paper, University of Maryland, College Park.

Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. *Proc. 29th AAAI Conf. Artificial Intelligence* (AAAI Press, Menlo Park, CA), 4292–4293.

Shakarian P, Eyre S, Paulo D (2013) A scalable heuristic for viral marketing under the tipping model. *Soc. Network Anal. Mining* 3(4):1225–1248.

Spencer G, Howarth R (2013) Maximizing the spread of stable influence: Leveraging norm-driven moral-motivation for green behavior change in networks. CoRR, abs/1309.6455. Accessed October 15, 2018, http://arxiv.org/abs/1309.6455.

Wang F, Du H, Camacho E, Xu K, Lee W, Shi Y, Shan S (2011) On positive influence dominating sets in social networks. *Theoret. Comput. Sci.* 412(3):265–269.

Watts DJ, Strogatz SH (1998) Collective dynamics of 'small-world' networks. *Nature* 393(6684):440–442.