https://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING

Vol. 00, No. 0, Xxxxx 0000, pp. 000–000

Submitted to *INFORMS Journal on Computing*

ISSN 0899-1499, EISSN 1526-5528

A Hybrid Matheuristic for the Spread of Influence on Social Networks

Felipe de Carvalho Pereira, Pedro Jussieu de Rezende
Institute of Computing, University of Campinas (Unicamp), Campinas, SP, Brazil, pereira_felipe@ic.unicamp.br, pjr@unicamp.br

Tallys Yunes
Miami Herbert Business School, University of Miami, Coral Gables, FL, USA, tallys@miami.edu

Abstract. We introduce a hybrid matheuristic for combinatorial optimization problems involving the spread of information on social networks. The proposed algorithmic framework is divided into four stages (preprocessing, construction, filtering, and replacement) and combines linear and integer programming techniques with a large neighborhood search procedure. We apply this framework to the Weighted Target Set Selection Problem (WTSSP), a generalization of the well-known Target Set Selection Problem (TSSP), to design two novel tailored heuristics for the WTSSP. Additionally, we propose a streamlined variant of the state-of-the-art integer programming formulation for the WTSSP and present a theoretical result on the strength of a relaxation for this new model. Extensive computational experiments on 260 instances from previously available benchmarks, including both synthetic and real-world ones, demonstrate the efficacy and efficiency of the proposed heuristics, showing that they significantly outperform existing heuristics. Additionally, by combining the solutions obtained by our heuristics with the exact methods, we attained provably optimal solutions for 174 (of the 260) instances, including 61 that were previously unsolved, and notably reduced the average optimality gap for the remaining unsolved instances. To demonstrate the potential of the proposed method and its applicability to a broad set of problems, we extended the experiments to a maximization version of the WTSSP, highlighting the algorithm's effectiveness and efficiency. These results further underscore the potential of applying our matheuristic to other TSSP-like problems and more generally to combinatorial optimization problems on social networks.

Funding: Supported in part by grants from: *Santander Bank*, Brazil; *Brazilian National Council for Scientific and Technological Development* (CNPq), Brazil, #313329/2020-6, #314293/2023-0; *São Paulo Research Foundation* (Fapesp), Brazil, #2023/04318-7, #2023/14427-8; *Fund for Support to Teaching, Research and Outreach Activities* (Faepex), Brazil; *Coordination for the Improvement of Higher Education Personnel* (Capes), Brazil – Finance Code 001.

Key words: Spread of information, Weighted Target Set Selection, Matheuristic, Integer Programming, Large Neighborhood Search.

1. Introduction

Imagine a new blockbuster movie currently under development. The film production company is strategizing an advertising campaign aimed at engaging specific members of a social network. To achieve this, the executives have opted for a *word-of-mouth marketing* approach (Brown and Reingen 1987, Dye 2000, Goldenberg et al. 2001). This strategy entails actively stimulating selected individuals to disseminate information to their social circles, thereby initiating a cascading effect across the network. This dissemination process operates person-to-person, fostering consumer influence through interpersonal connections.

In the digital realm of online platforms such as Facebook and Instagram, the propagation of a piece of information takes place via various channels, including posts (both original and shared), comments, and direct messaging. Through these avenues, users engage in discussions, share content, and exchange opinions, contributing to the organic spread of information within their network of contacts.

The primary objective is to identify suitable initial spreaders who can effectively promote the movie in their posts, giving rise to a broad propagation across the social network. Additionally, the executives are mindful of the varying costs associated with influencers' promotional posts (Bakshy et al. 2011), prompting a desire to optimize the expenses and impact of the campaign. These concerns underscore the significance of *influencer marketing*, an increasingly pivotal aspect of the marketing industry, which has managed billions of dollars in recent years (Campbell and Farrell 2020).

Several combinatorial optimization problems address the scenario described above (Banerjee et al. 2020, Li et al. 2018). Most of them can be described as problems on graphs in which vertices represent the individuals of a social network and edges denote influence relationships between individuals. Typically, the goal is to select a subset of vertices as the initial spreaders while optimizing a certain objective function, such as minimizing the cost of selecting the vertices while ensuring that the information reaches a substantial portion of the graph, or maximizing the number of influenced vertices while respecting a budget constraint.

1.1. Our Contributions

We introduce a framework for developing tailored heuristics for combinatorial optimization problems related to the dissemination of influence in social networks. We refer to this framework as a *matheuristic* (Boschetti and Maniezzo 2022), as it is a high-level heuristic algorithm grounded in mathematical programming techniques that offers structural versatility adaptable to various problems with minor modifications.

Our devised matheuristic is a hybrid approach. Initially, it performs a preprocessing approach to reduce the instance size. Then, it constructs a feasible solution by leveraging insights garnered from a linear programming (`LP`) model. Subsequently, the algorithm performs a local search procedure that consists of a *large neighborhood search* (`LNS`), combined with integer programming (`IP`) formulations, to improve the initial solution, thereby converging toward a locally optimal solution. For a comprehensive survey of the literature on matheuristics and `LNS` approaches, we refer the reader to Boschetti et al. (2009), Caserta and Voß (2010), Maniezzo et al. (2009, 2021), Ramos (2018), Ramos et al. (2020) and Ahuja et al. (2002), Pisinger and Ropke (2019), Shaw (1998), respectively.

To demonstrate the effectiveness of our proposed framework, we design two specific heuristics for the Weighted Target Set Selection Problem (`WTSSP`), a prominent optimization problem that belongs to the topic investigated in this paper (Cordasco et al. 2015, Raghavan and Zhang 2019). The motivation comes from the results obtained in a study by Raghavan and Zhang (2019), which showed that for 180 real-world instances, the best-known heuristic for the `WTSSP` yielded solutions approximately 5.47 times worse, on

average, than the best-known solutions. This significant gap underscores the pressing need for further research efforts to refine and develop enhanced heuristics for that problem.

Our contributions include:

- a hybrid matheuristic framework for combinatorial optimization problems stemming from the spread of information on social networks;
- two effective heuristics to solve the WTSSP, adapted from the proposed framework;
- a streamlined variant of the state-of-the-art IP model for the WTSSP by Raghavan and Zhang (2019);
- a theoretical result on the strength of a relaxation of the simplified IP model;
- extensive experimental showcasing of the efficacy of our heuristics for the WTSSP on a previously available set of 260 instances, encompassing both synthetic and real-world ones. Remarkably, our algorithms outperform all the existing heuristics on both benchmarks, irrespective of the IP model employed within the LNS procedure, be it the original IP by Raghavan and Zhang (2019) or a simplified version of it;
- provably optimal solutions for 174 of those instances, including 61 that were previously unsolved, as well as tighter optimality gaps for the remaining unsolved instances;
- Additional experimental results on the maximization version of the WTSSP.

The remaining of this text is organized as follows. In Section 2, we provide a brief literature review on combinatorial optimization problems related to the spread of information in social networks, with particular attention to the WTSSP, including previously proposed heuristics and IP formulations. In Section 3, we introduce a simplified IP formulation for the WTSSP and present a theoretical result concerning the strength of a relaxation for this new model. Section 4 describes the matheuristic, while Section 5 details the tailored heuristics implemented for the WTSSP and elaborates on their correctness and time complexity. We report the computational experiments conducted with our heuristics and the other pertinent algorithms in Section 6. Lastly, Section 7 concludes the paper by summarizing key findings and offering insights into potential avenues for future research. In the Appendix to this paper, we present demonstrations of all propositions introduced throughout the text.

2. Literature Review

The work by Kempe et al. (2003) was among the first to study the identification of influential nodes in social networks from a discrete optimization perspective. They introduced two classical diffusion models: the *independent cascade model* and the *linear threshold model*. In both, when an individual u becomes influenced, it attempts to transmit influence to its neighbors. The difference lies in how this influence takes effect: in the independent cascade model, each attempt succeeds with a given probability; in the linear threshold model, the influence is always transmitted, but each neighbor v requires a cumulative amount of influence from its neighbors (including u) to reach its threshold and become influenced.

In this work, we focus on an algorithmic framework for problems based on the linear threshold model. This family of problems includes the Influence Maximization Problem (IMP) (Kempe et al. 2015) and the

Target Set Selection Problem (TSSP) (Chen 2009), two of the most studied formulations in this topic. In IMP, the goal is to select a subset of individuals within a budget to maximize the spread of influence. Conversely, TSSP seeks the minimum-cost set of initial adopters required to influence the entire network. Both problems have inspired a broad range of combinatorial optimization variants under different settings and constraints (Banerjee et al. 2020, Li et al. 2018). Here, we address the Weighted Target Set Selection Problem (WTSSP), a generalization of TSSP, to demonstrate the effectiveness of our proposed matheuristic.

2.1. The WTSSP Problem

In the WTSSP, a social network is represented by an undirected graph $G = (V, E)$, where V and E are sets of vertices and edges of G, respectively. Throughout this paper, we assume that all graphs are simple, i.e., they do not contain double edges or self loops. Each vertex in V represents an individual, and each edge $\{u, v\} \in E$ indicates that there is reciprocal communication between u and v within the network. The neighborhood of a vertex $v \in V$ is denoted by $N_G(v) = \{u \in V : \{u, v\} \in E\}$, and $deg_G(v) = |N_G(v)|$ represents the degree of vertex v. We omit the subscripts whenever they can be unambiguously determined from the context.

A set of vertices chosen as initial spreaders (*targets*) is called a *target set* and is denoted by S. When the targets share the information with their neighbors, some vertices may become *influenced* and, as a consequence, forward the information, thus initiating a *propagation*. During propagation, each vertex adopts one of two potential states:

- *active* – when the vertex is a target or has been influenced;
- *inactive* – when the vertex remains uninfluenced.

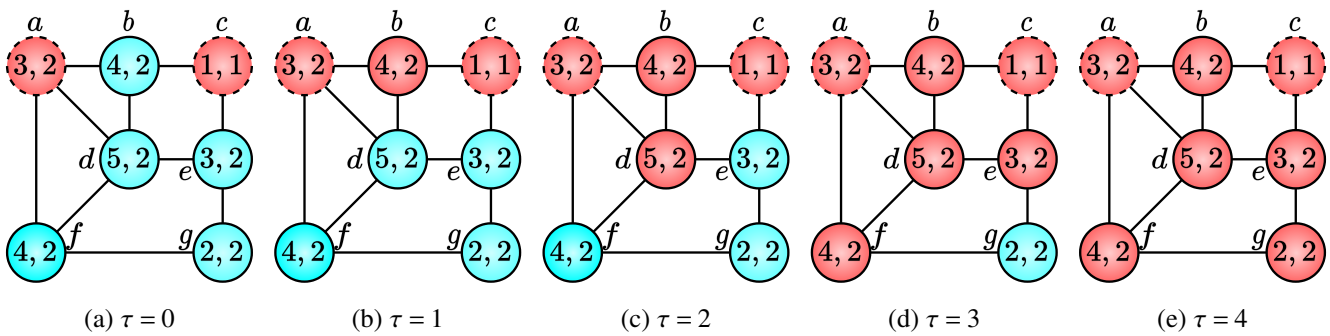
The *cost* of targeting a vertex is determined by a function $c : V \to \mathbb{Z}^+$. In the original paper that introduced the WTSSP (Raghavan and Zhang 2019), the term *weight* is used to describe the cost of a vertex. However, we adopt the term *cost* for clarity and consistency with the broader literature. Additionally, a vertex's resistance to influence is characterized by a *threshold* function $t : V \to \mathbb{Z}^+$. For any vertex v, the threshold $t(v)$ indicates that if v is not a target, then at least $t(v)$ of its neighbors must become active before v does.

Each time interval in a propagation is represented by a *round* $\tau \in \mathbb{N}$ and the subset of V containing the active vertices in round τ is denoted by S_τ. At $\tau = 0$, only the vertices in the target set are active, *i.e.*, $S_0 = S$. For every $\tau \geq 1$, $S_\tau = S_{\tau-1} \cup \{v \in V \setminus S_{\tau-1} : |S_{\tau-1} \cap N(v)| \geq t(v)\}$. It follows that $S_\tau \subseteq S_{\tau+1}$ for all $\tau \geq 0$.

A non-target v is activated in round $\tau \geq 1$ if $v \in S_\tau \setminus S_{\tau-1}$. The propagation terminates in the earliest round ρ in which no vertex becomes activated. If $S_\rho = V$, *i.e.*, all vertices are active at the end of the propagation, S is a *feasible* target set; otherwise, it is *infeasible*. The cost of a target set S is denoted by $c(S) = \sum_{v \in S} c(v)$.

PROBLEM 1 (WTSSP). Given an instance $I = (G, c, t)$, where $G = (V, E)$ is an undirected graph, $c : V \to \mathbb{Z}^+$ assigns costs to the vertices and $t : V \to \mathbb{Z}^+$ defines thresholds, the objective is to find a feasible target set of minimum cost.

Figure 1 provides an illustration of the propagation dynamics in the WTSSP. Within each circle, the cost and threshold of the corresponding vertex are indicated. In this instance, we consider the target set $\{a, c\}$, with a total cost of 4, which constitutes an optimal solution for this scenario. Target vertices are delineated by dashed circles, while active vertices are colored in red, and inactive ones in blue.



(a) $\tau = 0$ (b) $\tau = 1$ (c) $\tau = 2$ (d) $\tau = 3$ (e) $\tau = 4$

Figure 1 Example of a propagation in the WTSSP.

The WTSSP is a more realistic version of the well-studied Target Set Selection Problem (TSSP), since every instance of the latter is simply an instance of the former where every vertex has cost equal to one (Ben-Zwi et al. 2011, Chen 2009). The TSSP is sometimes also described as the minimization version of the Influence Maximization Problem (IMP), in which the primary aim is to maximize the number of active nodes by the end of the propagation, subject to a constraint on the size of the target set (Domingos and Richardson 2001, Kempe et al. 2003, 2005, Richardson and Domingos 2002). Both the IMP and TSSP are known to be NP-hard and they give rise to a collection of optimization problems that have been extensively studied (Banerjee et al. 2020, Chen et al. 2013, Li et al. 2018, Pereira 2021, Pereira et al. 2021, Pereira and de Rezende 2023, Singh et al. 2022, Ravelo and Meneses 2021).

Despite the fact that the WTSSP is in P for trees (Raghavan and Zhang 2021), the problem is NP-hard in the general case. Coping with large instances of such difficult optimization problems often entails resorting to heuristic algorithms to swiftly produce good (though not necessarily optimal) solutions. To date, three heuristics have been proposed for the WTSSP (Cordasco et al. 2015, Raghavan and Zhang 2019, Shakarian et al. 2013). On the other hand, the WTSSP has been mathematically modeled as three different integer programming (IP) formulations (Ackerman et al. 2010, Raghavan and Zhang 2019, Shakarian et al. 2013, Spencer and Howarth 2013). In particular, the state-of-art exact approach to optimally solve the WTSSP is a branch-and-cut algorithm proposed by Raghavan and Zhang (2019). Next, we detail the heuristics and IP formulations previously proposed for the WTSSP.

2.2. Heuristics for the WTSSP

We begin by describing the three existing heuristics for the WTSSP, denoted as RZ, CGRV, and SEP. Each of these algorithms follows a greedy and iterative approach. However, while RZ directly assembles a

target set by selecting a new target in each iteration, CGRV and SEP focus on identifying vertices that will not be included in the target set, thereby compelling other vertices to become targets. As a result, in CGRV and SEP, the target set is constructed indirectly.

The RZ heuristic, introduced by Raghavan and Zhang (2019), was designed mainly as a greedy algorithm to obtain an initial feasible solution within a branch-and-cut procedure. Starting from an empty target set, each step involves adding a vertex v satisfying the following condition: v has the smallest cost among the vertices that would remain inactive in a propagation initiated by the target set constructed thus far. The algorithm terminates upon achieving feasibility. RZ can be implemented to run in $O(|V| \log |V| + |E|)$ time.

The CGRV heuristic was introduced by Cordasco et al. (2015) and works as follows. Initially, the target set is empty. In each iteration, three cases are considered:

1. if there exists a vertex v in the graph that would be activated in a propagation started from the target set built so far, then the thresholds of v's neighbors are decremented by 1, and v is removed from the graph;
2. if no such vertex exists, but there is a vertex v in the graph such that $deg(v) < t(v)$, then v is added to the target set, the thresholds of v's neighbors are decremented by 1, and v is removed from the graph;
3. otherwise, a vertex v in the graph that maximizes $\frac{c(v)t(v)}{deg(v)(deg(v)+1)}$ is removed from the graph.

Observe that, in CGRV, the target set is only incremented in the second case, with the addition of a vertex from the remaining graph. So, by removing a vertex v from the graph in the third case, the algorithm prevents v from becoming a target and, consequently, v will rely on its neighbors to become active. The algorithm halts upon the removal of the last vertex from the graph in any of the three cases. CGRV can be implemented to run in $O(|E| \log |V|)$ time.

The last heuristic, SEP, initially devised for the TSSP by Shakarian et al. (2013) and later adapted for the WTSSP by Raghavan and Zhang (2019), operates as follows: in each iteration, a vertex v such that $deg(v) - t(v)$ is minimum and nonnegative is removed from the graph. In the case of a tie, the vertex with the highest cost is removed. This is repeated until $deg(v) < t(v)$ for every vertex v in the remaining graph, and the residual vertices are selected as targets. SEP can be implemented to run in $O(|E| \log |V|)$ time.

2.3. Integer Programming Formulations for the WTSSP

We now present three existing IP formulations for the WTSSP. The most intuitive one, denoted by TimeIndexed, was proposed by both Shakarian et al. (2013) and Spencer and Howarth (2013). It employs binary variables $x_{v,\tau}$, associating each vertex v with each round $\tau \in \{0, 1, \ldots, |V|\}$ such that $x_{v,\tau}$ equals 1 if and only if v is active in round τ. The target set comprises every vertex v for which $x_{v,0} = 1$. It has been shown, however, that the linear relaxation of TimeIndexed can be arbitrarily weak (Raghavan and Zhang 2019). Another IP model for the WTSSP, denoted by ACK (Ackerman et al. 2010), features a binary variable x_v for each vertex v, determining whether v is a target or not. Additionally, the model associates binary variables $b_{u,v}$ for each pair of distinct vertices u and v, where $b_{u,v}$ equals 1 iff u becomes active before v in a propagation. Notably, ACK includes constraints ensuring transitivity among the b variables, i.e., if u,

v, and c are distinct vertices and $b_{u,v} = b_{v,c} = 1$, then $b_{u,c} = 1$. In practice, `ACK` establishes stronger linear relaxations compared to `TimeIndexed` (Raghavan and Zhang 2019).

The last model, denoted by `RAG`, was introduced by Raghavan and Zhang (2019) and exhibits the property that its linear relaxation yields integral solutions for instances based on graphs that are trees. Moreover, `RAG` boasts the strongest linear relaxation among the three models in practical scenarios (Raghavan and Zhang 2019). Since we employ `RAG` later in this paper, we provide a detailed description of the formulation now.

Given an instance $I = (G, c, t)$ of the `WTSSP` with $G = (V, E)$, we construct an undirected bipartite graph G' by replacing every edge $\{u, v\}$ in G with two edges $\{d, u\}$ and $\{d, v\}$, where d is a new *dummy* vertex. Formally, $G' = (V \cup D, E')$, where $D = \{d^e : e \in E\}$ represents the set of dummy vertices and $E' = \bigcup_{e=\{u,v\}\in E}\{\{d^e, u\}, \{d^e, v\}\}$. Dummy vertices have threshold 1 and are ineligible for selection as targets. `RAG` encompasses the following sets of binary variables:

- $\{x_v : v \in V\} : x_v = 1$ iff v is a target;
- $\bigcup_{\{u,v\}\in E}\{h_{u,v}, h_{v,u}\} : h_{u,v} = 1$ ($h_{v,u} = 1$) iff u influences v (v influences u) on G;
- $\bigcup_{\{d,v\}\in E'}\{y_{d,v}, y_{v,d}\} : y_{d,v} = 1$ ($y_{v,d} = 1$) iff d influences v (v influences d) on G'.

Denote by Ξ the collection of all directed cycles of length at least 3 that are orientations of undirected cycles of G. The model reads:

$$\min \sum_{v\in V} c(v)x_v \tag{1}$$

$$x_v \le y_{v,d} \qquad \forall v \in V, d \in N_{G'}(v) \tag{2}$$

$$y_{v,d} + y_{d,v} = 1 \qquad \forall \{d, v\} \in E' \tag{3}$$

$$h_{u,v} + h_{v,u} = 1 \qquad \forall \{u, v\} \in E \tag{4}$$

$$h_{u,v} \le y_{u,\, d^{\{u,v\}}} \qquad \forall u \in V, v \in N_G(u) \tag{5}$$

$$\sum_{d\in N_{G'}(v)} y_{d,v} + t(v)x_v \ge t(v) \qquad \forall v \in V \tag{6}$$

$$\sum_{(u,v)\in\xi} h_{u,v} \le |\xi| - 1 \qquad \forall \xi \in \Xi \tag{7}$$

Given an attribution of values to the variables of `RAG` that satisfies all constraints, the target set $\{v : x_v = 1\}$ forms a feasible solution for I. The objective function (1) aims to minimize the cost of the target set. Constraints (2) enforce that if v is designated as a target, then v must exert influence over all dummy vertices connected to it in G'. Constraints (3) ensure that, for every pair of adjacent vertices v and d in G', influence flows in only one direction between them. Similarly, constraints (4) enforce this same directional influence relationship, but for adjacent vertices in G. Constraints (5) specify that if u influences v in G, then u must also influence the dummy node positioned between u and v in G', which we denote by $d^{\{u,v\}}$. Constraints (6) mandate that every vertex $v \in V$ become active during a propagation on G', either by being a target or by receiving influence from at least $t(v)$ dummy neighbors. Finally, constraints (7) prevent the h variables from inducing a directed cycle of length at least 3. `RAG` has a total of $|V| + 6|E|$ binary variables and $|V| + 5|E| + |\Xi|$ constraints.

3. A Simplified IP Formulation for the WTSSP

In this section, we present a new IP formulation for WTSSP, denoted by PRY, which simplifies the RAG model described in Section 2.3. Let $I = (G, c, t)$ be an instance of WTSSP, where $G = (V, E)$. Similar to RAG, the central idea of PRY is to represent the influence flow during a propagation via a set of binary variables on arcs that are orientations of the edges in E. However, unlike RAG, we do not define variables and constraints using an extended graph with dummy vertices. Instead, all variables and constraints of PRY are defined solely on I. PRY includes the following sets of binary variables:

- $\{s_v : v \in V\}$: $s_v = 1$ iff v is a target;
- $\bigcup_{\{u,v\}\in E}\{f_{u,v}, f_{v,u}\}$: $f_{u,v} = 1$ iff u influences v in a propagation.

Denote by Ξ the collection of all directed cycles of length at least 3 that are orientations of undirected cycles of G. The model reads:

$$\min \sum_{v\in V} c(v)s_v \tag{8}$$

$$s_v + f_{u,v} \leq 1 \qquad \forall v \in V, u \in N(v) \tag{9}$$

$$f_{u,v} + f_{v,u} \leq 1 \qquad \forall \{u, v\} \in E \tag{10}$$

$$\sum_{u\in N(v)} f_{u,v} + t(v)s_v \geq t(v) \qquad \forall v \in V \tag{11}$$

$$\sum_{(u,v)\in\xi} f_{u,v} \leq |\xi| - 1 \qquad \forall \xi \in \Xi \tag{12}$$

The objective function (8) minimizes the cost of the target set. Constraints (9) prevent the targets from being influenced by any of their neighbors. Constraints (10) ensure that, for every pair of adjacent vertices u and v, the influence goes in at most one direction. Constraints (11) force every vertex v to become active during the propagation, either by being a target or by receiving influence from at least $t(v)$ neighbors. Lastly, constraints (12) prevent the f variables from inducing a directed cycle of length at least 3.

The PRY model comprises $|V| + 2|E|$ binary variables and $|V| + 2|E| + |\Xi|$ constraints, which is fewer than the number of variables and constraints in RAG (see Section 2.1). We now demonstrate PRY's correctness.

PROPOSITION 1. *If S is a feasible solution for I, then there exists an attribution of values to the variables of PRY such that $s_v = 1$ iff $v \in S$, and all of PRY's constraints are satisfied.*

PROPOSITION 2. *Given an attribution of values to the variables of PRY that satisfies all of its constraints, the set $S = \{v : s_v = 1\}$ is a feasible solution for I.*

PROPOSITION 3. *An optimal solution for PRY provides an optimal solution for I.*

Now, let R^{PRY} denote the linear relaxation of PRY, excluding the anti-dicycle constraints of type (12). Similarly, let R^{RAG} denote the linear relaxation of RAG, excluding the anti-dicycle constraints of type (7). We claim that optimal solutions for R^{PRY} and R^{RAG} provide the same lower bound for an optimal integer solution for I.

PROPOSITION 4. *The value of an optimal solution for R^{RAG} coincides with the value of an optimal solution for R^{PRY}.*

In the Appendix to this paper, we provide proofs for Propositions (1), (2), (3), and (4). In the next section, we present an algorithmic framework for TSSP-like problems.

4. A Matheuristic for Influence Propagation Problems

In this section, we introduce a hybrid matheuristic, denoted as HMF, designed to address combinatorial optimization problems related to the spread of influence in social networks. This algorithm is particularly useful for various versions of the TSSP, including the WTSSP. We first describe HMF and later, in Section 5, adapt it to develop two heuristics specifically tailored for the WTSSP utilizing the RAG and PRY models, as well as the $R^{\mathtt{RAG}}$ and $R^{\mathtt{PRY}}$ relaxations.

Let P be an influence propagation problem with a minimization objective (e.g., minimizing the size or the cost of the target set) and let I be an instance of P that contains a graph $G = (V, E)$. Moreover, consider an IP formulation for P that contains a set of binary variables $X = \{x_v : v \in V\}$ meaning that $x_v = 1$ iff v is a target in a feasible solution for I. Also, let R be a relaxed formulation obtained by removing a subset of the IP model's constraints (e.g., the linear relaxation). Here, we assume that R is solvable in polynomial time. HMF is divided into four stages and is described by Algorithm 1.

First stage (preprocessing) In line 1, the original instance I undergoes preprocessing using a problem-specific method. This step may involve identifying and removing trivial elements from the solution space, thereby reducing the instance size and search space. Although optional, the preprocessing stage can significantly enhance the algorithm's performance by simplifying the problem before the main heuristic stages.

Second stage (relaxation-oriented construction) Lines 2 to 14 comprise the second stage, where the algorithm solves a relaxation R of an IP formulation associated with I to construct a feasible solution S. From an optimal solution to R (or a good suboptimal feasible solution), we obtain a function $X^* : X \to \mathbb{Q} \cap [0, 1]$ that assigns values to the variables in X. Typically, X^* is not integral, but we aim to obtain an integral solution. To achieve this, while X^* remains fractional we perform an iterative procedure that progressively forces some variables $x_v \in X$ to take the value 1. This is done by adding constraints of the form $x_v = 1$ to R and resolving it until integrality is reached. To guide this process, we partition X into tiers according to the values in X^*: the larger the value of a variable, the higher the tier it belongs to. Then, at each iteration, the unforced variables in the highest non-empty tier are fixed to 1. This mechanism is precisely what is implemented in the loop of lines 4 to 9. Next, a priority queue Q is constructed to contain all vertices from V, with priorities based on X^*. Specifically, the priority of each $v \in V$ is set to $X^*(x_v)$, with a tie-breaker rule favoring variables with smaller reduced costs, since in the LP relaxation reduced costs measure the deterioration in the objective function when a variable is forced to take value 1. Hence, variables with smaller reduced costs are more likely to belong to a good integral solution. Starting with an empty target

set S, the vertex from Q with the highest priority is dequeued and added to S as a new target. This process repeats until S becomes feasible.

Algorithm 1: `HMF`

Input : Instance I
Output : Feasible solution S

```
/* 1st stage (preprocessing) */
1  Preprocess(I)
/* 2nd stage (relaxation-oriented construction) */
2  R ← a relaxation of an IP formulation associated with I
3  X* ← Solve(R)
4  while X* is not integral do
5      Partition X into tiers according to X*
6      foreach x_v in the highest non-empty tier do
7          if x_v is not forced then
8              Add constraint x_v = 1 to R /* Force x_v to 1 */
9      X* ← Solve(R)
10 Q ← BuildPriorityQueue(V, X*)
11 S ← ∅
12 while S is not feasible do
13     v ← Dequeue(Q)
14     S ← S ∪ {v}
/* 3rd stage (filtering) */
15 foreach v ∈ S do
16     if S \ {v} is feasible then
17         S ← S \ {v}
/* 4th stage (replacement) */
18 Q ← BuildPriorityQueue(S)
19 B ← ∅
20 while Q is not empty do
21     v ← Dequeue(Q)
22     Find a locally optimal solution S' ⊆ V \ B such that (S \ {v}) ⊆ S'
23     if v ∉ S' then
24         B ← B ∪ {v}
25     foreach u ∈ S' \ S do
26         Enqueue(Q, u)
27     S ← S'
28 return S
```

Third stage (filtering) Lines 15 to 17 comprise the third stage, where the target set S is refined by iteratively removing expendable targets. After the second stage, S may contain targets that can be removed without compromising its feasibility. For example, some targets chosen earlier might become unnecessary due to the influence of targets chosen later. During the loop starting in line 15, each target $v \in S$ is checked to see whether $S \setminus \{v\}$ remains feasible. If so, v is removed from S. By the end of this stage, S is both feasible and minimal. The order in which targets are inspected can be adjusted based on the specific problem requirements.

Fourth stage (replacement) Lines 18 to 27 encompass the fourth stage, which consists of a local search procedure that iteratively explores the search space of neighboring solutions of the incumbent target set S. For each target v, the algorithm seeks the best feasible solution that includes all vertices from S except for v. The idea is to replace v with a better combination of non-targets, if it exists, ultimately leading to a locally optimal solution. We first construct a priority queue Q and try to substitute the vertex from Q with the highest priority in each replacement attempt. Initially, Q contains all vertices from S, prioritized in a way that suits problem P. We also create a set B, initially empty, to contain the replaced vertices. The algorithm continues iterating until Q is empty. In each iteration, it finds a locally optimal solution $S' \subseteq V \setminus B$ that includes all vertices from $S \setminus \{v\}$. If S is already locally optimal, it is possible that $S' = S$. If $v \notin S'$, then v is replaced by adding v to B (to prevent v from being reintroduced into the solution), enqueuing each vertex from $S' \setminus S$ into Q (these are the new targets that replace v), and updating S to S'. Note that the fourth stage employs an `LNS` strategy, as the search space explored in line 22 can be exponential.

We note that `HMF` can also be adapted to problems with maximization objectives (e.g., maximizing the number of activated vertices or the revenue associated with their activation). For these problems, a target set is typically considered feasible if its size (or cost) respects a budget constraint. To modify the second stage accordingly, we can change the loop of line 12 so that it is executed while there exists a vertex u in the priority queue Q such that $S \cup \{u\}$ respects the budget. Additionally, we can subject the step of line 14 to the condition that $S \cup \{v\}$ be feasible and improves the objective value. The third stage can be adjusted to remove a target v from S only if $S \setminus \{v\}$ has the same objective value as S. Lastly, the fourth stage can be modified in line 22 so that the resulting locally optimal solution $S' \subseteq V \setminus B$ corresponds to the best possible solution containing all vertices from $S \setminus \{v\}$ that respects the budget constraint.

5. New Heuristics for the WTSSP

We now specialize `HMF` to design two new heuristics for the `WTSSP`, denoted by `HMF-RAG` and `HMF-PRY`, based on the `RAG` and `PRY` models, respectively. Although both heuristics include a linear model in the second stage and an integer model in the fourth stage, the specific models used differ between `HMF-RAG` and `HMF-PRY`. Before proceeding, we introduce some preliminary concepts and results that will be useful later in this section.

5.1. Preliminaries

Let $I = (G, c, t)$ be an instance of the `WTSSP`, where $G = (V, E)$, and denote by S_{final} the collection of vertices that are active at the end of the propagation on G started from a target set $S \subseteq V$.

Proposition 5 asserts that if S and S' are subsets of V, then the final set of active vertices in the propagation on G initiated from $S \cup S'$ is identical to the final set of active vertices in the propagation on G initiated from $S_{\text{final}} \cup S'$. This result is particularly useful when seeking a feasible solution of the form $S \cup S'$, where S is predetermined. Consequently, we can concentrate on finding S' restricted to $V \setminus S_{\text{final}}$.

PROPOSITION 5. *Let $S, S' \subseteq V$. Then $(S \cup S')_{\text{final}} = (S_{\text{final}} \cup S')_{\text{final}}$.*

Now, we show that if part of a target set for I is already fixed, say $S \subseteq V$, then we can formulate a subinstance of the original instance such that a feasible solution for the subinstance can be used as a complement to S to form a feasible target set for the original instance.

Let $inf(v)$ denote the amount of influence received by vertex v during a propagation on G started from S, *i.e.*, the number of vertices from $N_G(v)$ that are active before v becomes active. Also, let G' be the subgraph of G induced by the vertex set $V' = V \setminus S_{\text{final}}$. Lastly, let $I' = (G', c', t')$ be a new instance of the `WTSSP`, where $c'(v) = c(v)$ and $t'(v) = t(v) - inf(v)$ for all $v \in V'$.

PROPOSITION 6. *If $S' \subseteq V'$ is a feasible solution for I', then $S \cup S'$ is a feasible solution for I.*

PROPOSITION 7. *If $S' \subseteq V'$ is such that $S \cup S'$ is a feasible solution for I, S' is feasible solution for I'.*

PROPOSITION 8. *Let $S' \subseteq V'$. Then, S' is a feasible solution for I' iff $S \cup S'$ is a feasible solution for I.*

In the Appendix to this paper, we provide proofs for Propositions (5), (6), (7), and (8). In the following sections, we detail the `HMF-RAG` and `HMF-PRY` heuristics referencing Algorithm 1. Both heuristics follow the same steps, differing only in the models employed during the second and fourth stages. From now on, the descriptions apply to both heuristics unless explicitly stated otherwise.

5.2. Feasibility Checking

In this section, we discuss how to determine whether a target set is feasible. This procedure is applied in both the second and third stages, specifically in lines 12 and 16 of Algorithm 1.

Let $I = (G, c, t)$ be an instance of the `WTSSP`, where $G = (V, E)$. Recall that $S \subseteq V$ is feasible for I iff the number of active vertices at the end of the propagation started from S equals $|V|$. Next, we describe how we algoritimically simulate a propagation from S to test its feasibility.

To simulate a propagation in the `WTSSP`, we store two attributes for each $v \in V$: $state(v)$ keeps track of the current state of v along the propagation (active or inactive) and $inf(v)$ indicates the amount of influence received by v so far. To continue a propagation in progress, the vertices that have become active but have not yet influenced their neighbors are kept in a queue Q. While Q is not empty, we iteratively take the next vertex u from Q and, for each neighbor v of u that is still inactive, we increment $inf(v)$ by 1. If $inf(v)$ reaches $t(v)$, we change the state of v to active and enqueue v into Q. Algorithm 2 formalizes these steps.

To simulate a complete propagation on G started from a target set S, we use Algorithm 3 in which all vertices are set as inactive, except the targets, and Algorithm 2 is called with Q containing the vertices in S. Also, we maintain a counter on the number of active vertices during a propagation. The feasibility of S is determined by comparing $|V|$ with the counter on the number of active vertices at the end of Algorithm 3.

Whenever we simulate a whole propagation using Algorithm 3, we visit all vertices to initialize their states and each edge is visited at most twice, when one of its endpoints becomes active. Thus, a single propagation runs is $O(|V| + |E|)$ time.

In the following sections, we delve into each stage of the heuristics, providing detailed discussions on their correctness and time complexity.

Algorithm 2: `ContinuePropagation`

Input : Instance I, Queue Q
while Q *is not empty* **do**
  $u \leftarrow$ `Dequeue` (Q)
  foreach $v \in N(u)$ *s.t.* $state(v) = inactive$ **do**
    $inf(v) \leftarrow inf(v) + 1$
    if $inf(v) \geq t(v)$ **then**
      $state(v) \leftarrow active$
      `Enqueue` (Q, v)

Algorithm 3: `CompletePropagation`

Input : Instance I, target set S
$Q \leftarrow \emptyset$
foreach $v \in V$ **do**
  $inf(v) \leftarrow 0$
  if $v \in S$ **then**
    $state(v) \leftarrow active$
    `Enqueue` (Q, v)
  else
    $state(v) \leftarrow inactive$
`ContinuePropagation` (I, Q)

5.3. First Stage

We now examine the preprocessing step in the first stage, where we identify vertices that are either trivial targets or can be disregarded as targets, which allows us to reduce the instance size and the search space.

Let $I = (G, c, t)$ be an instance of the `WTSSP`, where $G = (V, E)$. Consider a vertex $v \in V$ such that $deg_G(v) = t(v)$ and $\sum_{u \in N(v)} c(u) \leq c(v)$. Since $deg_G(v) = t(v)$, v can only be activated if v is a target or if all its neighbors become active before v. In the first case, we can replace v with its neighbors without compromising the feasibility of the solution or increasing its cost. Therefore, in both cases, v is *inert*, meaning its activation does not contribute to the activation of any other vertex. Thus, whenever we are looking for a feasible solution, we can safely disregard v from being a target and remove v from the instance.

Now, let G' be the graph with vertex set V' obtained from G by iteratively removing all inert vertices. Also, let $I' = (G', c', t')$ be a reduced instance, where $c'(v) = c(v)$ and $t'(v) = t(v)$ for every $v \in V'$. Since some vertices were removed from I to form I', there might be vertices in G' whose degrees are less than their respective thresholds. In this case, the set $T = \{v \in V' : deg_{G'}(v) < t'(v)\}$ is non-empty, and each vertex in T must be a target in any feasible solution for I'. Thus, T is a set of *trivial targets*.

Finally, let G'' be the graph with vertex set V'' obtained from G' by removing every vertex that becomes active during the propagation on G' started from T. Let $inf(v)$ denote the total influence received by v during that propagation. Also, let $I'' = (G'', c'', t'')$ be a further reduced instance, where $c''(v) = c'(v)$ and $t''(v) = t'(v) - inf(v)$ for all $v \in V''$. Proposition 8 ensures that $S \subseteq V''$ is a feasible solution for I'' if and only if $T \cup S$ is a feasible solution for I'.

With this result in hand, the preprocessing stage reduces I to I'' in line 1 of Algorithm 1 and I'' becomes the instance tackled in the subsequent stages. Ultimately, the heuristics return $T \cup S$ as the final solution, where S is the feasible target set for I'' obtained at the conclusion of the fourth stage.

Given that graphs are represented using adjacency lists, identifying inert vertices, trivial targets, and vertices that become active during a propagation initiated from T can be accomplished in $O(|V| + |E|)$ time.

Additionally, deleting a vertex v (to form either G' or G'') can be done in $O(deg(v))$ time. Therefore, the preprocessing stage can be executed in $O(|V|+|E|)$ time in the worst case. As we will see in Section 6, this procedure is very efficient in practice and can significantly reduce the size of the original instance.

5.4. Second Stage

In this section, we describe the relaxation-oriented construction approach employed in the second stage, starting from the relaxation R used in lines 2 and 3 of Algorithm 1.

The linear relaxation of `RAG` would be a natural choice for R due to its superior strength compared to the linear relaxations of other existing formulations like `TimeIndexed` and `ACK` (see Section 2.3 and (Raghavan and Zhang 2019)). However, the number of anti-dicycle constraints in `RAG` can be exponential in the input size and to tackle this issue, we would need to separate such constraints when solving `RAG`'s linear relaxation, which can be very time-consuming. As this could compromise efficiency, which is a crucial feature for heuristics, we propose a different approach.

For the `HMF-RAG` heuristic, we set R as the $R^{\texttt{RAG}}$ relaxation, which is the linear relaxation of `RAG` without the anti-dicycle constraints (of type (7)). Similarly, for the `HMF-PRY` heuristic, we use the $R^{\texttt{PRY}}$ relaxation, which is the linear relaxation of `PRY` without the anti-dicycle constraints (of type (12)). This approach allows us to solve R efficiently and, subsequently, construct a good (and probably suboptimal) feasible solution for the original instance.

We remark that, according to Proposition 4, the optimal values of $R^{\texttt{RAG}}$ and $R^{\texttt{PRY}}$ coincide. Consequently, we can expect the target set constructed in the second stage of `HMF-PRY` to be similar to the one constructed in the second stage of `HMF-RAG`.

For the force-and-resolve procedure in the loop of lines 4 to 9 of Algorithm 1, we employ tiers of the form $\{x_v \in X : i < x_v^* \leq i + 0.1\}$ for $R^{\texttt{RAG}}$ and $\{s_v \in X : i < s_v^* \leq i + 0.1\}$ for $R^{\texttt{PRY}}$, with $i = 0.0, 0.1, 0.2, \ldots, 0.9$. Once the loop halts, we construct the priority queue Q in line 10, setting the priority of each $v \in V$ equal to the value of the variable x_v (s_v) in the optimal solution for $R^{\texttt{RAG}}$ ($R^{\texttt{PRY}}$). In the event of a tie, higher priority is given to the vertex v that minimizes the reduced cost of x_v (s_v). If a tie still persists, higher priority is assigned to the vertex v that maximizes $deg(v)/c(v)$, meaning that this vertex spreads the greatest amount of influence per unit of cost.

We now discuss how to determine the feasibility of the target set S while it is being constructed in the loop of line 12. A straightforward approach would be to invoke Algorithm 3 at every iteration. However, this would require simulating the entire propagation from scratch each time a new target is selected, since Algorithm 3 is designed to simulate propagation from a fully specified (*i.e.*, already constructed) target set. Instead, we apply Algorithm 3 only in the first iteration – used solely to initialize vertex states (as inactive) and influence counters (set to zero), without triggering any propagation – and subsequently employ Algorithm 2 in the remaining iterations, thereby simulating a single propagation process throughout the construction of S.

This way, whenever a new target v is selected, we simply continue the propagation from the current state with v as the newly activated vertex. Specifically, the first time the condition in line 12 of Algorithm 1 needs to be checked, we call Algorithm 3 with $S = \emptyset$, which is obviously an infeasible target set. In subsequent iterations, we call Algorithm 2 with Q containing only the most recently selected target. The feasibility of S is then determined by comparing $|V|$ with the counter of active vertices at the end of Algorithm 2.

To see why this approach works, recall that Proposition 5 ensures that the propagations started from $S \cup \{v\}$ and from $S_{\text{final}} \cup \{v\}$ yield the same final set of active vertices, confirming the procedure's correctness.

Additionally, whenever a vertex v is dequeued in line 13, we add v to the solution in line 14 only if v remains inactive in the propagation initiated from the current target set.

Regarding the time complexity of the second stage, lines 2 and 3 of Algorithm 1 run in polynomial time on the instance size, assuming that a polynomial-time algorithm is used to solve the relaxation. Additionally, the priority queue Q can be constructed in $O(|V|)$ time. Since there are at most $|V|$ iterations of the loop in line 12, there will be at most $|V|$ dequeues from Q, each taking $O(\log |V|)$ time. Also, with the single propagation used to continuously check the feasibility of S, the remainder of this stage runs in $O(|V| + |E|)$ time. Therefore, excluding lines 2 and 3, the second stage runs in $O(|V| \log |V| + |E|)$ time.

5.5. Third Stage

In the third stage, we refine the target set S constructed in the second stage by removing expendable targets. This is done by traversing S in reverse order of its targets' selection priorities from the second stage in the loop of line 15 of Algorithm 1. This reverse traversal ensures that we attempt to remove the least critical targets first, thereby maintaining the feasibility of the solution while potentially reducing its cost.

To determine the feasibility of $S \setminus \{v\}$ in line 16, we call Algorithm 3 to simulate a complete propagation starting from $S \setminus \{v\}$. We then check the counter for the number of active vertices. The set $S \setminus \{v\}$ is deemed feasible if and only if the number of active vertices equals $|V|$ at the end of the propagation.

Since the loop starting in line 15 iterates at most $|V|$ times, and each iteration takes $O(|V| + |E|)$ time due to the feasibility check of $S \setminus \{v\}$, the total runtime for the third stage is $O(|V|^2 + |V| \cdot |E|)$.

5.6. Fourth Stage

In the fourth stage, we apply a `LNS` procedure that attempts to replace the vertices in the target set S, constructed in the third stage, with other vertices in V to reduce the solution cost. We start by building the priority queue Q in line 18 of Algorithm 1, prioritizing each $v \in S$ by $c(v)$. This ensures that we first attempt to replace the most expensive targets.

In each iteration of the loop of line 20, we search for a locally optimal solution for I in line 22. Specifically, we seek a new feasible solution S^{new} such that $S^{\text{new}} \subseteq V \setminus B$, $(S \setminus \{v\}) \subseteq S^{\text{new}}$, and $c(S^{\text{new}})$ is minimized, where v is the vertex dequeued in the current iteration, and B is the set containing the former targets that were already replaced. Here, the `RAG` and `PRY` models are employed in the `HMF-RAG` and `HMF-PRY` heuristics, respectively, to obtain S^{new}. Next, we outline this process.

A naive approach would be to obtain S^{new} by solving the RAG model for I while simply fixing $x_u = 0$ for each $u \in B$ and $x_u = 1$ for every $u \in S \setminus \{v\}$. The same reasoning applies to the PRY model with regard to the s variables. However, instead of solving an IP model associated with I, we employ a more efficient method that focuses on solving an IP for a subinstance of I. This subinstance contains only the vertices that would not become active in a propagation started from $S \setminus \{v\}$.

First, we use Algorithm 3 to simulate a propagation starting from $S \setminus \{v\}$ and obtain a subgraph G' of G by removing every vertex that becomes active during that propagation. Let V' be the vertex set of G'. Next, we take a subinstance $I' = (G', c', t')$, where $c'(u) = c(u)$ and $t'(u) = t(u) - inf(u)$ for all $u \in V'$, with $inf(u)$ representing the amount of influence received by u during the simulated propagation.

Recall that according to Proposition 8, $S' \subseteq V'$ is a feasible solution for I' iff $(S \setminus \{v\}) \cup S'$ is a feasible solution for I. That result allows us to solve I', thereby obtaining S', and setting $S^{\text{new}} = (S \setminus \{v\}) \cup S'$ while ensuring that S^{new} is a locally optimal solution for I.

We acknowledge that solving an IP model in line 22 can be a bottleneck since IP is NP-hard in general. For that reason, we set a time limit for executing an IP solver in this step. If the limit is reached, we take S' as the best solution found so far.

Concerning the time complexity of the fourth stage, the priority queue Q can be constructed in $O(|V|)$ time, and the loop at line 20 iterates at most $|V|$ times. Each iteration can be implemented to run in $O(|V| + |E|)$ time, except for the step in which we search for a locally optimal solution. Therefore, the fourth stage runs in $O(|V|^2 + |V| \cdot |E|)$ time, excluding the time for finding the locally optimal solutions (line 22).

6. Computational Experiments

In this section, we describe the experiments conducted to empirically evaluate the HMF−RAG and HMF−PRY heuristics. All experiments were carried out on a machine equipped with an Intel® Xeon® E5-2630 v4 processor, 64 GB of RAM, running Ubuntu 22.04.1 LTS, with Gurobi v12.0.3 employed as the LP and IP solver.

To the best of our knowledge, the original implementations of RZ, CGRV, and SEP are not publicly available. Accordingly, we reimplemented these algorithms and conducted the corresponding experiments under the same computational setup described above. For consistency, the stopping criterion for these heuristics followed the procedure outlined in Section 2.2, whereby each algorithm halts once a feasible solution is obtained.

The benchmark of instances used in this study is the same as that employed in the computational experiments reported by Raghavan and Zhang (2019). These instances are categorized into two groups based on their graph types: synthetic and real-world. Table 1 lists the properties of the synthetic graphs, which were generated using the Watts-Strogatz model (Watts and Strogatz 1998) to simulate social networks. Table 2 provides the properties of the real-world graphs, which are snapshots of various online systems, primarily

Table 1 Quantifying topological characteristics of the Watts-Strogatz graphs.

Graph	$\|V\|$	$\|E\|$	Density
200-k4	200	400	0.02010
200-k6	200	600	0.03015
200-k8	200	800	0.04020
200-k10	200	1000	0.05025
200-k12	200	1200	0.06030
2500-k16	2500	20000	0.00640
5000-k8	5000	20000	0.00160
10000-k4	10000	20000	0.00040

Table 2 Quantifying topological characteristics of the real-world graphs.

Graph	$\|V\|$	$\|E\|$	Density
Hamsterster	1788	12476	0.00781
Facebook-NIPS-Ego	2888	2981	0.00072
Bitcoin Alpha	3775	14120	0.00198
Advogato	5042	39227	0.00309
Bitcoin OTC	5875	21489	0.00125
P2P-Gnutella-08	6299	20776	0.00105
P2P-Gnutella-09	8104	26008	0.00079
P2P-Gnutella-06	8717	31525	0.00083
P2P-Gnutella-05	8842	31837	0.00081
Ning	9727	40570	0.00086
Escorts	10106	39016	0.00076
Oregon-01	10670	22002	0.00039
P2P-Gnutella-04	10876	39994	0.00068
Oregon-02	10900	31180	0.00052
Anybeat	12645	49132	0.00061
Google+	23613	39182	0.00014
Facebook-LetsDoIt	39439	50222	0.00006
Douban	154908	327162	0.00003

social networks, sourced from public repositories (Kunegis 2013, Leskovec and Krevl 2014, Lesser et al. 2013, Rossi and Ahmed 2015).

Each graph generated 10 instances, with threshold and cost of each vertex v sampled from uniform distributions within the intervals $[1, deg(v)]$ and $[1, 100]$, respectively. In total, the benchmark consists of 80 synthetic and 180 real-world instances. For both sets of instances, we proceeded as follows.

First, we executed the `RZ`, `CGRV`, `SEP`, `HMF-PRY`, and `HMF-RAG` heuristics for each instance. For `HMF-PRY` and `HMF-RAG`, in each iteration of the fourth stage, we set a time limit of 10 seconds for the `IP` solver to find a locally optimal solution.

It is important to note that, although preprocessing procedures have been discussed for our proposed heuristics, `RZ`, `CGRV`, and `SEP` do not incorporate any preprocessing. Therefore, all executions were performed on the original (unpreprocessed) instances to ensure a fair comparison across heuristics. The preprocessing steps are considered a feature of our heuristics, and their running time is included in the total execution time.

Next, with the primary goal of obtaining lower bounds for the optimal solution values, enabling us to assess the quality of the heuristic solutions, we ran the `IP` solver with the `RAG` and `PRY` models for each instance, setting a time limit of 1 hour per execution. We used the best target sets found by the heuristics as warm-start solutions. For some instances, the `IP` solver managed to find better solutions than the initial incumbent. Additionally, the `IP` solver was able to prove optimality for many instances.

To handle the exponential number of anti-dicycle constraints in both `RAG` and `PRY` formulations (see Sections 2.3 and 3), we employed a lazy constraint strategy. Specifically, whenever the solver found an

integer solution, we performed a depth-first search on the directed graph induced by the integer solution. For each cycle detected, we added the corresponding violated inequality to the formulation. Additionally, for the exact runs with the IP solver, we preloaded all anti-dicycle constraints for cycles of length 3.

We refer the reader to a publicly available repository by Pereira et al. (2024) that accompanies this paper and includes the source code, instances, and computed results, including solutions, bounds, and runtimes.

6.1. Results for the Synthetic Instances

In this section, we present the results obtained for the synthetic instances, grouped by each subset of 10 instances generated from the same graph.

Table 3 shows the average optimality gaps of the solutions obtained by the IP solver, as well as the number of instances solved to proven optimality. The optimality gap of a solution S for an instance I is calculated as $(c(S) - \mathrm{LB}(I))/c(S)$, where $\mathrm{LB}(I)$ is a lower bound for the value of an optimal solution for I. The last two columns provide the combined results, in which we consider the best lower bound and upper bounds obtained by either RAG or PRY.

Table 3 Optimality gaps (%) for the synthetic instances via exact methods.

Graph	PRY		RAG		Combined	
	Opt gap (%)	# Solved	Opt gap (%)	# Solved	Opt gap (%)	# Solved
200-k4	0.00 ± 0.00	10	0.00 ± 0.00	10	0.00 ± 0.00	10
200-k6	0.00 ± 0.00	10	0.00 ± 0.00	10	0.00 ± 0.00	10
200-k8	0.00 ± 0.00	10	0.00 ± 0.00	10	0.00 ± 0.00	10
200-k10	0.00 ± 0.00	10	0.00 ± 0.00	10	0.00 ± 0.00	10
200-k12	0.24 ± 0.76	9	0.00 ± 0.00	10	0.00 ± 0.00	10
2500-k16	7.50 ± 2.42	0	7.09 ± 2.25	0	6.89 ± 2.25	0
5000-k8	3.85 ± 1.18	0	3.84 ± 1.01	0	3.61 ± 2.37	0
10000-k4	0.00 ± 0.00	10	0.00 ± 0.00	10	0.00 ± 0.00	10

Considering the 80 synthetic instances, PRY solved 59 to proven optimality, while RAG solved the same 59 plus 1 additional instance from the 200-k12 group that PRY did not solve. For the remaining 20 unsolved instances, corresponding to all instances in groups 2500-k16 and 5000-k8, PRY (RAG) achieved a better optimality gap than RAG (PRY) in 8 (12) instances. Among the 59 instances solved by both models, PRY had an average runtime of 95.34 ± 342.95 seconds, whereas RAG had an average runtime of 47.22 ± 94.28 seconds.

We highlight that the 10 instances from the 10000-k4 group had no known optimal solutions until now. Also, by combining the best lower and upper bounds obtained from the formulations, we remarkably reduced the average optimality gap reported by Raghavan and Zhang (2019) for the 20 instances in groups 2500-k16 and 5000-k8 from 14.05% ± 10.30% to 5.25% ± 2.42%.

We now turn our attention to the heuristic algorithms. Figure 2 shows the average optimality gaps of the solutions obtained by the heuristics. For comparison, the horizontal bars labeled “Best” in Figure 2 reproduce the penultimate column of Table 3, representing the best-known optimality gaps.

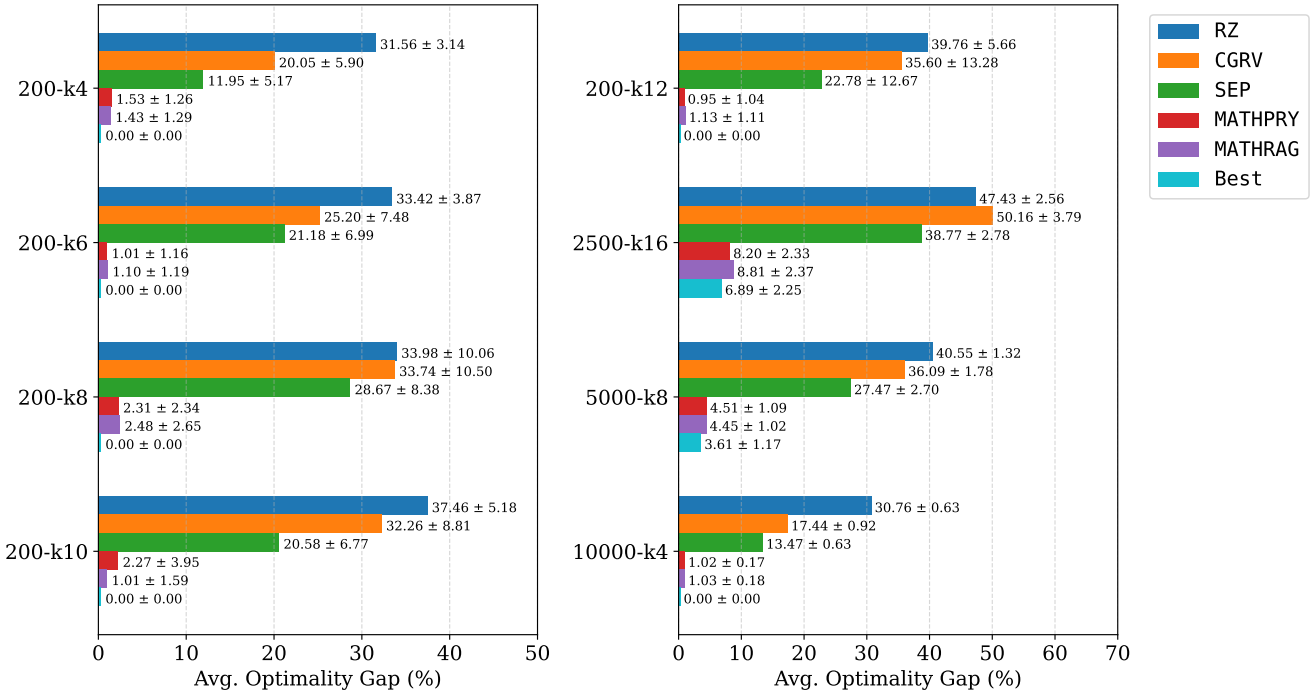


Figure 2 **Optimality gaps (%) for the synthetic instances via heuristics.**

From Figure 2, it is evident that HMF−PRY and HMF−RAG significantly outperformed the other heuristics across all groups of synthetic instances, with HMF−PRY showing a slight overall advantage. For the 60 instances (out of 80) with known optimal solutions (covering all instances from the 200-k4, 200-k6, 200-k8, 200-k10, 200-k12, and 10000-k4 groups), HMF−PRY and HMF−RAG found optimal solutions for the same 14 instances, while HMF−RAG solved 2 additional instances; the other heuristics found none. For the remaining 44 instances, the optimality gaps of the solutions obtained by HMF−PRY and HMF−RAG were very close to the best-known values, as the averages reported in Figure 2 indicate. Next, we present a more detailed analysis of the results achieved by HMF−PRY and HMF−RAG.

Table 4 shows the average percentage reduction in the number of vertices and edges of the graphs after the preprocessing stage. The results indicate that the method was particularly effective for the 200-k4 and 200-k6 instances, which have the sparsest graphs among the instances with 200 vertices, and for the 5000-k8 and 10000-k4 instances, which contain the sparsest graphs amongst the instances with 20000 edges. This suggests that the preprocessing stage tends to be more effective for instances with sparser graphs.

Table 5 shows the average percentage of trivial targets identified during the preprocessing stage for the synthetic instances, along with their corresponding costs. The results indicate that the occurrence of trivial targets was generally low, representing a small percentage of the total cost of the solutions found by the HMF−PRY and HMF−RAG heuristics.

Next, we detail the quality of the solutions obtained by the HMF−PRY and HMF−RAG heuristics at each stage. Table 6 shows the average optimality gaps per stage for the synthetic instances. First, we observe no

Table 4 Reduction in graph size due to the preprocessing stage for the synthetic instances.

Graph	$\|V\|$	$\|E\|$	$\|V\|$ reduction (%)	$\|E\|$ reduction (%)
200-k4	200	400	3.300 ± 2.541	4.600 ± 3.762
200-k6	200	600	0.050 ± 0.158	0.050 ± 0.158
200-k8	200	800	0.000 ± 0.000	0.000 ± 0.000
200-k10	200	1000	0.000 ± 0.000	0.000 ± 0.000
200-k12	200	1200	0.000 ± 0.000	0.000 ± 0.000
2500-k16	2500	20000	0.000 ± 0.000	0.000 ± 0.000
5000-k8	5000	20000	0.004 ± 0.013	0.008 ± 0.024
10000-k4	10000	20000	2.690 ± 0.443	3.713 ± 0.622

Table 5 Quantifying trivial targets for the synthetic instances.

Graph	HMF−PRY		HMF−RAG	
	Size (%)	Cost (%)	Size (%)	Cost (%)
200-k4	4.15 ± 3.16	3.35 ± 3.43	4.15 ± 3.16	3.35 ± 3.43
200-k6	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
200-k8	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
200-k10	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
200-k12	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
2500-k16	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
5000-k8	0.02 ± 0.06	0.01 ± 0.04	0.02 ± 0.06	0.01 ± 0.04
10000-k4	2.92 ± 0.61	2.08 ± 0.49	2.92 ± 0.61	2.08 ± 0.49

significant difference between the results obtained by the two heuristics. Second, we note that the optimality gaps consistently decreased as the stages progressed, indicating that each stage effectively improved the solution quality. Additionally, comparing Figure 2 and Table 6, we conclude that the solutions obtained by HMF−PRY and HMF−RAG in the second stage were already significantly superior, on average, to those obtained by the other heuristics.

Table 6 Optimality gaps (%) per stage for the synthetic instances.

Graph	HMF−PRY			HMF−RAG		
	2nd stage	3rd stage	4th stage	2nd stage	3rd stage	4th stage
200-k4	10.04 ± 4.22	4.47 ± 3.47	1.53 ± 1.26	10.74 ± 4.60	4.73 ± 3.65	1.43 ± 1.29
200-k6	12.86 ± 5.57	7.26 ± 4.33	1.01 ± 1.16	12.37 ± 4.06	6.55 ± 3.04	1.10 ± 1.19
200-k8	13.79 ± 8.93	7.74 ± 5.55	2.31 ± 2.34	17.09 ± 9.06	9.77 ± 7.83	2.48 ± 2.65
200-k10	17.98 ± 11.69	9.79 ± 7.13	2.27 ± 3.95	17.81 ± 10.82	9.64 ± 6.59	1.01 ± 1.59
200-k12	23.41 ± 9.60	10.02 ± 7.52	0.95 ± 1.04	25.16 ± 8.02	11.68 ± 9.78	1.13 ± 1.11
2500-k16	36.61 ± 5.28	19.40 ± 3.43	8.20 ± 2.33	38.82 ± 5.90	20.63 ± 4.44	8.81 ± 2.37
5000-k8	23.72 ± 2.09	11.09 ± 1.65	4.51 ± 1.09	24.59 ± 2.37	11.07 ± 2.07	4.45 ± 1.02
10000-k4	10.00 ± 0.99	4.35 ± 0.42	1.02 ± 0.17	10.13 ± 0.99	4.40 ± 0.28	1.03 ± 0.18

We also compare the performance of the HMF−PRY and HMF−RAG heuristics in terms of runtime. Table 7 shows the average time (in seconds) spent per stage for the synthetic instances, excluding the first stage, which was negligible for both heuristics. For each instance, both heuristics spent less than 40 seconds in the second stage and less than 8 minutes in total. The fourth stage was the most time-consuming, as was expected due to the nature of this stage, where multiple IP problems are solved. The results indicate a slight advantage for HMF−PRY regarding the instances from the 2500-k16 and 5000-k8 groups, which are the most challenging among the synthetic instances. According to Figure 2, HMF−PRY also had a small advantage over HMF−RAG in terms of solution quality.

We note that RZ, CGRV, and SEP are extremely fast, requiring less than one second for most instances, including the larger ones. However, this efficiency comes at the expense of solution quality, which is notably poor.

Having detailed the performance of our heuristics, we now discuss their role in the experiments with the exact models. Recall that by providing the solutions obtained by our heuristics to the IP solver, we were

Table 7 Time (in seconds) per stage for the synthetic instances.

Graph	HMF-PRY				HMF-RAG			
	2nd stage	3rd stage	4th stage	Total	2nd stage	3rd stage	4th stage	Total
200-k4	0.04 ± 0.02	0.00 ± 0.00	0.06 ± 0.07	0.11 ± 0.08	0.05 ± 0.01	0.00 ± 0.00	0.05 ± 0.04	0.11 ± 0.04
200-k6	0.07 ± 0.02	0.00 ± 0.00	0.49 ± 1.24	0.56 ± 1.23	0.10 ± 0.03	0.00 ± 0.00	0.10 ± 0.15	0.20 ± 0.16
200-k8	0.07 ± 0.02	0.00 ± 0.00	5.02 ± 8.28	5.09 ± 8.29	0.11 ± 0.03	0.00 ± 0.00	0.86 ± 1.71	0.98 ± 1.73
200-k10	0.11 ± 0.02	0.00 ± 0.00	1.55 ± 3.12	1.65 ± 3.12	0.16 ± 0.05	0.00 ± 0.00	0.25 ± 0.23	0.41 ± 0.24
200-k12	0.14 ± 0.04	0.00 ± 0.00	1.74 ± 3.37	1.88 ± 3.36	0.19 ± 0.05	0.00 ± 0.00	1.90 ± 4.19	2.09 ± 4.20
2500-k16	11.73 ± 4.02	0.09 ± 0.01	161.65 ± 57.81	173.49 ± 59.36	22.26 ± 8.24	0.09 ± 0.01	151.35 ± 58.26	173.72 ± 62.78
5000-k8	11.19 ± 3.26	0.40 ± 0.02	166.30 ± 111.34	177.91 ± 111.99	18.52 ± 5.24	0.40 ± 0.02	125.30 ± 79.33	144.25 ± 80.07
10000-k4	2.78 ± 0.26	1.79 ± 0.04	3.91 ± 0.16	8.51 ± 0.31	4.71 ± 0.42	1.79 ± 0.04	4.14 ± 0.17	10.65 ± 0.43

able to find new optimal solutions and tighten the optimality gaps of the remaining unsolved instances. This is in contrast to the results reported by Raghavan and Zhang (2019), where the warm-starts provided to the IP solver were the solutions obtained by RZ.

To verify whether the improvements were primarily due to our heuristics rather than the evolution of IP solvers over time, we conducted additional runs on the more challenging synthetic instances. Specifically, we focused on the 10 instances from group 2500-k16 (see Table 3). Both PRY and RAG models were tested using the solutions from RZ as initial feasible solutions.

For these instances, the performance of the IP solver was slightly superior when compared to the lower and upper bounds reported by (Raghavan and Zhang 2019). However, the final solutions never outperformed the solutions from our heuristics. In other words, starting from a RZ solution and running the IP solver for one hour resulted in solutions that were consistently worse than the initial heuristic solutions used in our experiments. This provides strong evidence of the importance of supplying high-quality initial solutions when solving the WTSSP via an exact approach.

6.2. Results on Real-World Instances

In this section, we present and analyse the results for the real-world instances, similarly to what we provided for the synthetic ones.

Table 8 shows the average optimality gaps of the solutions obtained by the IP solver, as well as the number of instances solved to proven optimality. The last two columns provide the combined results, in which we consider the best lower bound and upper bounds obtained by either RAG or PRY.

Considering the 180 real-world instances, PRY and RAG solved 99 and 113 instances to proven optimality, respectively, with 98 instances solved by both models. For these 98 instances, PRY required an average runtime of 388.71 ± 595.89 seconds, whereas RAG required 345.64 ± 453.82 seconds. Among the 66 instances not solved by either model, PRY (RAG) obtained a better optimality gap than RAG (PRY) in 39 (21) instances, while both models achieved the same optimality gap in 6 instances.

By combining the best lower and upper bounds obtained from these formulations, we attained provably optimal solutions for 114 instances, including 51 that had no previously known optimal solutions. For the remaining 66 unsolved instances, we reduced the average optimality gap reported in the experiments of Raghavan and Zhang (2019) from 1.87% ± 1.50% to 0.29% ± 0.38%.

Table 8 Optimality gaps (%) for the real-world instances via exact methods.

Graph	PRY		RAG		Combined	
	Opt gap (%)	# Solved	Opt gap (%)	# Solved	Opt gap (%)	# Solved
Hamsterster	0.521 ± 0.487	2	0.546 ± 0.526	2	0.433 ± 0.434	2
Facebook-NIPS-Ego	0.000 ± 0.000	10	0.000 ± 0.000	10	0.000 ± 0.000	10
Bitcoin Alpha	0.062 ± 0.116	6	0.011 ± 0.020	7	0.009 ± 0.017	7
Advogato	0.488 ± 0.503	1	33.135 ± 41.755	2	0.456 ± 0.523	2
Bitcoin OTC	0.114 ± 0.122	2	0.105 ± 0.140	4	0.071 ± 0.118	4
P2P-Gnutella-08	0.001 ± 0.004	9	0.002 ± 0.008	9	0.001 ± 0.004	9
P2P-Gnutella-09	0.000 ± 0.000	10	0.000 ± 0.000	10	0.000 ± 0.000	10
P2P-Gnutella-06	0.024 ± 0.058	8	0.010 ± 0.022	8	0.010 ± 0.022	8
P2P-Gnutella-05	0.021 ± 0.046	8	0.019 ± 0.044	8	0.018 ± 0.043	8
Ning	0.309 ± 0.220	0	0.617 ± 0.758	1	0.224 ± 0.210	1
Escorts	0.021 ± 0.046	6	0.017 ± 0.033	6	0.016 ± 0.031	6
Oregon-01	0.009 ± 0.021	8	0.010 ± 0.031	9	0.006 ± 0.018	9
P2P-Gnutella-04	0.006 ± 0.016	8	0.011 ± 0.027	8	0.006 ± 0.016	8
Oregon-02	0.464 ± 0.393	1	16.292 ± 32.682	1	0.463 ± 0.391	1
Anybeat	0.250 ± 0.479	3	21.161 ± 33.777	3	0.208 ± 0.478	4
Google+	0.361 ± 0.569	5	0.000 ± 0.000	10	0.000 ± 0.000	10
Facebook-LetsDoIt	0.000 ± 0.000	10	0.000 ± 0.000	10	0.000 ± 0.000	10
Douban	0.013 ± 0.019	2	0.006 ± 0.008	5	0.005 ± 0.007	5

We now turn our attention to the heuristic algorithms. Figure 3 shows the average optimality gaps of the solutions obtained by the heuristics for this benchmark. For comparison, the horizontal bars labeled "Best" in Figure 3 reproduce the penultimate column of Table 8, representing the best-known optimality gaps. Note that the horizontal axis is in log scale. Once again, HMF−PRY and HMF−RAG significantly outperformed the other heuristics, alternating in achieving the best average optimality gap across the instance groups. Specifically, of the 114 instances (out of 180) with known optimal solutions, HMF−PRY and HMF−RAG found the same optimal solutions for 32 instances, while HMF−RAG solved 2 additional instances. For the remaining 66 instances, the maximum average optimality gap across the instance groups did not exceed 1.27%, closely matching the best-known gaps.

Table 9 shows the reduction in graph size due to the preprocessing stage. The results indicate that the method was highly effective in reducing the size of the real-world instances. In the most extreme cases, the reductions in the vertex and edge sets were 88.71% and 87.85%, respectively. When comparing the results from Table 9 for instances with a similar number of vertices, we observe that the preprocessing method was more effective for instances with sparser graphs. This trend was also noted for the synthetic instances. This can be explained by the fact that vertices with lower degrees are more likely to be inert (see Section 5.3), and consequently, more likely to be removed during the preprocessing stage.

Table 10 shows the percentage of trivial targets for the real-world instances in terms of quantity and cost when compared to the solutions obtained by HMF−RAG and HMF−PRY. The results were similar for both heuristics and indicate that the percentage of trivial targets was significant for the real-world instances, representing up to 63.64% of the total cost of the solutions found by the heuristics.

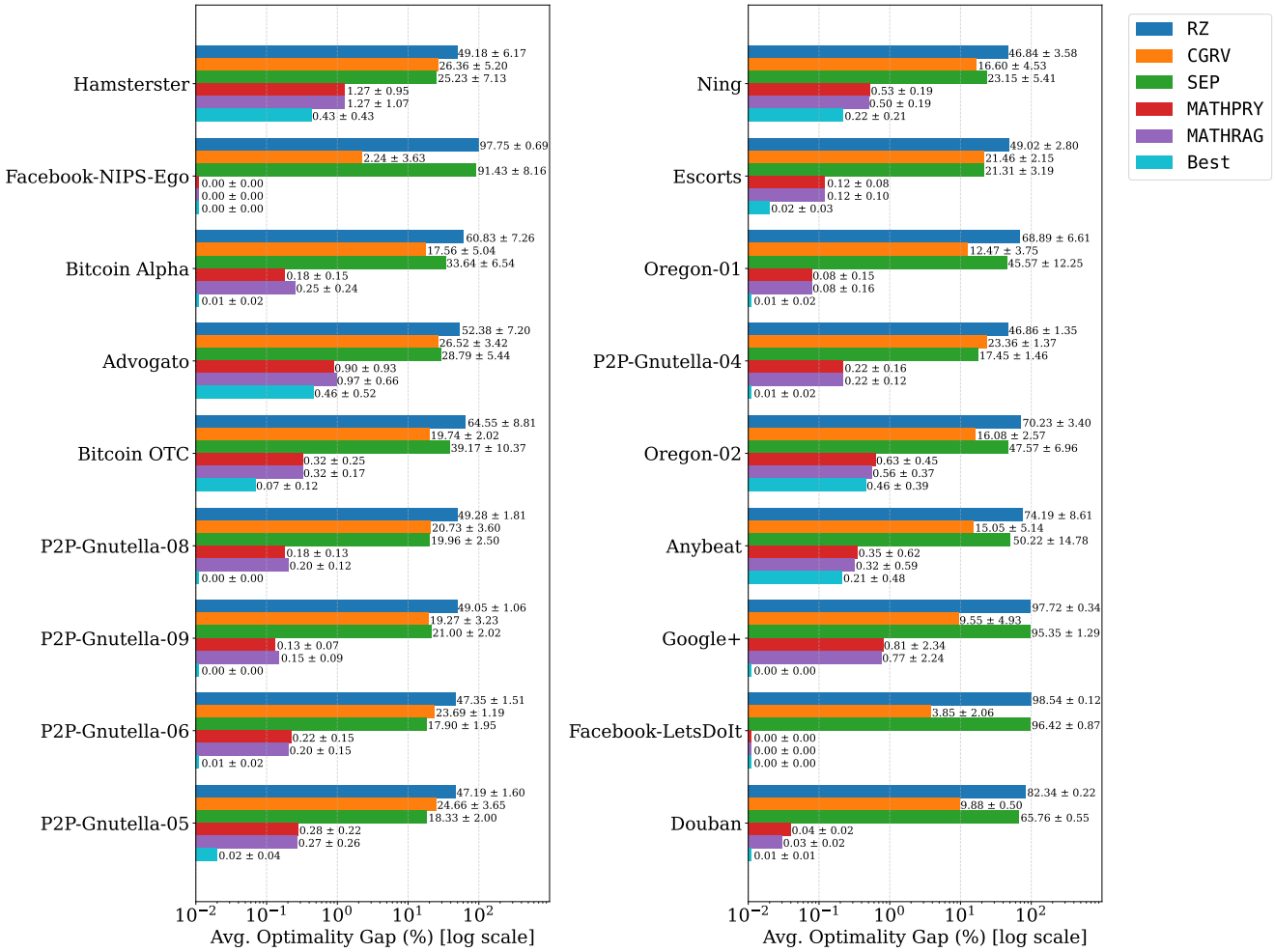


Figure 3 **Optimality gaps (%) for the real-world instances via heuristics.**

Next, Table 11 shows the average optimality gaps per stage for the real-world instances. The results indicate that each stage effectively improved the solution quality. Notably, for the 32 instances where both heuristics found optimal solutions, the first two stages were sufficient for 20 instances, and the third and fourth stages were crucial for the remaining 12 instances.

Table 12 shows the time per stage for the real-world instances, excluding the first stage, which was negligible. The results indicate that the second and fourth stages were the most time-consuming for both heuristics, which is consistent with their asymptotic runtime complexity. From Table 12, we conclude that `HMF-PRY` was faster than `HMF-RAG` in the second stage for most of the instances, which was expected since `HMF-PRY` uses a more compact `LP` formulation (see Section 3).

For the Douban instances, the largest in the real-world benchmark, the heuristics achieved similar solution quality (see Table 11), but `HMF-PRY` had a significantly smaller average total time than `HMF-RAG`. This suggests that `HMF-PRY` scales better than `HMF-RAG` for large instances, which is likely due to the more compact `LP` and `IP` models used. Even for the Douban instances, the average total time spent by `HMF-PRY`

Table 9 Reduction in graph size due to the preprocessing stage for the real-world instances.

Graph	$\|V\|$	$\|E\|$	$\|V\|$ reduction (%)	$\|E\|$ reduction (%)
Hamsterster	1788	12476	17.68 ± 12.45	18.13 ± 20.96
Facebook-NIPS-Ego	2888	2981	72.55 ± 14.82	72.89 ± 14.52
Bitcoin Alpha	3775	14120	37.52 ± 7.22	37.32 ± 13.59
Advogato	5042	39227	24.51 ± 11.28	25.55 ± 21.66
Bitcoin OTC	5875	21489	34.17 ± 6.34	31.45 ± 10.39
P2P-Gnutella-08	6299	20776	30.07 ± 4.65	30.80 ± 9.89
P2P-Gnutella-09	8104	26008	32.25 ± 4.59	33.67 ± 9.41
P2P-Gnutella-06	8717	31525	25.91 ± 3.27	27.92 ± 6.13
P2P-Gnutella-05	8842	31837	23.86 ± 3.33	24.13 ± 6.33
Ning	9727	40570	35.05 ± 9.95	35.12 ± 19.43
Escorts	10106	39016	21.84 ± 3.09	22.18 ± 5.71
Oregon-01	10670	22002	38.19 ± 11.15	37.70 ± 14.31
P2P-Gnutella-04	10876	39994	24.51 ± 1.63	25.48 ± 3.19
Oregon-02	10900	31180	34.58 ± 9.89	29.37 ± 12.86
Anybeat	12645	49132	49.38 ± 18.06	51.19 ± 24.43
Google+	23613	39182	55.17 ± 6.92	55.13 ± 10.23
Facebook-LetsDoIt	39439	50222	67.78 ± 5.51	69.42 ± 5.67
Douban	154908	327162	56.79 ± 1.01	57.88 ± 1.69

Table 10 Quantifying trivial targets for the real-world instances.

Graph	HMF−PRY		HMF−RAG	
	Size (%)	Cost (%)	Size (%)	Cost (%)
Hamsterster	14.07 ± 3.33	13.10 ± 1.90	14.08 ± 3.35	13.10 ± 1.90
Facebook-NIPS-Ego	56.70 ± 17.16	45.47 ± 12.16	56.70 ± 17.16	45.47 ± 12.16
Bitcoin Alpha	25.02 ± 2.84	24.98 ± 4.90	24.97 ± 2.92	24.98 ± 4.90
Advogato	18.18 ± 2.84	18.21 ± 3.33	18.29 ± 2.99	18.22 ± 3.34
Bitcoin OTC	23.93 ± 2.84	23.37 ± 3.07	23.93 ± 2.83	23.37 ± 3.07
P2P-Gnutella-08	17.97 ± 2.01	19.09 ± 2.16	17.99 ± 2.01	19.10 ± 2.16
P2P-Gnutella-09	18.97 ± 2.23	19.70 ± 2.43	18.98 ± 2.24	19.70 ± 2.43
P2P-Gnutella-06	15.44 ± 1.11	16.59 ± 1.46	15.44 ± 1.10	16.58 ± 1.46
P2P-Gnutella-05	14.92 ± 1.22	15.77 ± 1.78	14.92 ± 1.20	15.77 ± 1.78
Ning	29.04 ± 1.16	29.01 ± 1.48	29.08 ± 1.18	29.02 ± 1.48
Escorts	18.12 ± 1.04	17.02 ± 2.01	18.10 ± 1.04	17.02 ± 2.01
Oregon-01	29.01 ± 1.68	27.66 ± 2.13	29.03 ± 1.70	27.66 ± 2.13
P2P-Gnutella-04	15.14 ± 1.06	16.05 ± 1.58	15.14 ± 1.06	16.05 ± 1.58
Oregon-02	23.88 ± 1.40	21.89 ± 1.56	23.97 ± 1.34	21.87 ± 1.56
Anybeat	32.46 ± 2.92	31.73 ± 2.71	32.45 ± 2.93	31.73 ± 2.71
Google+	28.78 ± 5.77	26.57 ± 4.92	28.78 ± 5.77	26.57 ± 4.91
Facebook-LetsDoIt	35.16 ± 6.56	34.36 ± 7.42	35.13 ± 6.58	34.36 ± 7.42
Douban	29.87 ± 1.14	30.03 ± 0.98	29.86 ± 1.14	30.03 ± 0.98

and HMF−RAG was no more than 6 and 8 minutes, respectively, which is practical for heuristics solving such large instances.

As observed for the synthetic instances, RZ, CGRV, and SEP also performed extremely quickly on the real-world dataset, requiring less than one second for most instances, including the larger ones. However, this efficiency comes at the expense of solution quality, which remains notably poor.

Table 11 Optimality gaps (%) per stage for the real-world instances.

Graph	HMF−PRY			HMF−RAG		
	2nd stage	3rd stage	4th stage	2nd stage	3rd stage	4th stage
Hamsterster	16.00 ± 7.59	8.45 ± 4.96	1.27 ± 0.95	16.07 ± 7.40	9.22 ± 4.92	1.27 ± 1.07
Facebook-NIPS-Ego	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Bitcoin Alpha	2.97 ± 1.64	1.42 ± 0.81	0.18 ± 0.15	3.21 ± 1.89	1.65 ± 1.11	0.25 ± 0.24
Advogato	7.78 ± 2.43	4.53 ± 1.65	0.90 ± 0.93	8.40 ± 2.15	4.89 ± 1.67	0.97 ± 0.66
Bitcoin OTC	4.24 ± 1.86	2.19 ± 1.18	0.32 ± 0.25	4.63 ± 1.94	1.93 ± 0.74	0.32 ± 0.17
P2P-Gnutella-08	2.14 ± 1.64	0.71 ± 0.47	0.18 ± 0.13	2.13 ± 1.58	0.76 ± 0.43	0.20 ± 0.12
P2P-Gnutella-09	1.89 ± 1.11	0.76 ± 0.35	0.13 ± 0.07	1.95 ± 1.16	0.80 ± 0.35	0.15 ± 0.09
P2P-Gnutella-06	1.78 ± 1.28	0.68 ± 0.38	0.22 ± 0.15	1.85 ± 1.58	0.70 ± 0.48	0.20 ± 0.15
P2P-Gnutella-05	2.29 ± 2.05	0.80 ± 0.66	0.28 ± 0.22	2.09 ± 1.99	0.76 ± 0.69	0.27 ± 0.26
Ning	4.36 ± 1.42	2.34 ± 0.72	0.53 ± 0.19	4.06 ± 0.78	2.31 ± 0.46	0.50 ± 0.19
Escorts	1.52 ± 1.01	0.51 ± 0.36	0.12 ± 0.08	1.40 ± 0.97	0.55 ± 0.40	0.12 ± 0.10
Oregon-01	1.10 ± 0.72	0.63 ± 0.41	0.08 ± 0.15	1.11 ± 0.65	0.62 ± 0.36	0.08 ± 0.16
P2P-Gnutella-04	1.66 ± 1.35	0.73 ± 0.49	0.22 ± 0.16	1.55 ± 1.00	0.63 ± 0.33	0.22 ± 0.12
Oregon-02	4.86 ± 2.30	2.61 ± 1.16	0.63 ± 0.45	4.76 ± 2.45	2.55 ± 1.11	0.56 ± 0.37
Anybeat	2.84 ± 1.56	1.49 ± 0.81	0.35 ± 0.62	3.03 ± 1.86	1.61 ± 0.88	0.32 ± 0.59
Google+	3.09 ± 4.68	2.12 ± 3.66	0.81 ± 2.34	3.09 ± 4.29	2.19 ± 3.49	0.77 ± 2.24
Facebook-LetsDoIt	0.62 ± 0.95	0.36 ± 0.66	0.00 ± 0.00	0.66 ± 0.94	0.37 ± 0.66	0.00 ± 0.00
Douban	0.50 ± 0.16	0.21 ± 0.08	0.04 ± 0.02	0.48 ± 0.14	0.18 ± 0.06	0.03 ± 0.02

Table 12 Time (in seconds) per stage for the real-world instances.

Graph	HMF−PRY				HMF−RAG			
	2nd stage	3rd stage	4th stage	Total	2nd stage	3rd stage	4th stage	Total
Hamsterster	1.89 ± 0.99	0.02 ± 0.01	25.99 ± 14.83	27.91 ± 15.26	2.88 ± 1.50	0.02 ± 0.01	18.61 ± 17.96	21.52 ± 18.19
Facebook-NIPS-Ego	0.03 ± 0.02	0.00 ± 0.00	0.06 ± 0.03	0.10 ± 0.04	0.08 ± 0.03	0.00 ± 0.00	0.08 ± 0.04	0.17 ± 0.07
Bitcoin Alpha	1.67 ± 0.80	0.04 ± 0.01	24.53 ± 28.44	26.26 ± 28.35	2.13 ± 0.91	0.04 ± 0.01	18.24 ± 23.03	20.43 ± 23.12
Advogato	11.93 ± 6.71	0.14 ± 0.04	47.29 ± 42.21	59.39 ± 45.56	16.99 ± 8.06	0.14 ± 0.03	38.86 ± 34.82	56.01 ± 39.73
Bitcoin OTC	4.10 ± 1.25	0.11 ± 0.01	47.91 ± 35.47	52.14 ± 35.39	5.54 ± 1.46	0.11 ± 0.02	30.38 ± 21.48	36.05 ± 22.00
P2P-Gnutella-08	6.46 ± 3.36	0.24 ± 0.03	5.78 ± 13.14	12.49 ± 13.73	9.03 ± 5.31	0.23 ± 0.03	6.33 ± 12.70	15.61 ± 13.89
P2P-Gnutella-09	9.43 ± 5.63	0.36 ± 0.06	8.67 ± 12.02	18.49 ± 13.83	13.74 ± 7.72	0.36 ± 0.05	7.10 ± 5.41	21.23 ± 10.58
P2P-Gnutella-06	26.35 ± 11.67	0.50 ± 0.05	27.08 ± 32.63	53.95 ± 35.42	41.81 ± 18.42	0.50 ± 0.04	24.41 ± 29.66	66.74 ± 36.23
P2P-Gnutella-05	29.53 ± 10.32	0.54 ± 0.06	23.50 ± 31.72	53.59 ± 37.98	45.80 ± 20.17	0.54 ± 0.06	19.47 ± 26.59	65.83 ± 42.16
Ning	11.86 ± 7.74	0.50 ± 0.12	68.07 ± 49.22	80.47 ± 54.70	17.01 ± 10.53	0.50 ± 0.12	49.51 ± 41.11	67.05 ± 47.31
Escorts	28.02 ± 11.52	0.67 ± 0.06	16.49 ± 15.05	45.22 ± 23.71	36.83 ± 19.18	0.68 ± 0.06	32.43 ± 40.33	69.97 ± 50.92
Oregon-01	3.16 ± 1.81	0.23 ± 0.05	10.46 ± 21.07	13.88 ± 21.35	4.10 ± 2.84	0.23 ± 0.05	9.77 ± 17.15	14.13 ± 17.71
P2P-Gnutella-04	52.21 ± 14.97	0.80 ± 0.05	19.55 ± 18.09	72.60 ± 31.01	82.85 ± 17.98	0.80 ± 0.05	22.61 ± 15.62	106.29 ± 28.41
Oregon-02	7.35 ± 2.58	0.31 ± 0.06	93.26 ± 81.20	100.96 ± 81.72	10.08 ± 3.78	0.31 ± 0.06	65.68 ± 57.18	76.10 ± 57.97
Anybeat	19.03 ± 23.78	0.27 ± 0.14	66.66 ± 166.40	86.04 ± 188.42	22.32 ± 27.65	0.27 ± 0.14	63.18 ± 169.01	85.85 ± 195.03
Google+	2.65 ± 1.69	0.05 ± 0.02	85.47 ± 177.78	88.23 ± 177.67	2.64 ± 1.27	0.05 ± 0.02	92.88 ± 201.94	95.63 ± 201.90
Facebook-LetsDoIt	1.28 ± 0.39	0.06 ± 0.02	2.12 ± 1.96	3.55 ± 2.02	1.62 ± 0.43	0.06 ± 0.02	2.55 ± 2.13	4.32 ± 2.29
Douban	68.47 ± 15.56	22.34 ± 1.28	253.80 ± 112.82	344.94 ± 117.00	102.90 ± 29.63	22.48 ± 1.22	342.37 ± 142.97	468.10 ± 152.85

Based on the data collected from the experiments with both synthetic and real-world instances, we observe that each stage of the framework contributes meaningfully to the overall solution quality. In particular, the second stage – the relaxation-oriented construction – proves crucial for rapidly generating high-quality feasible solutions, which are subsequently refined in later stages. We hypothesize that the key factor is the use of information from a relaxed version of the problem to guide the construction of a feasible integer solution. This approach allows the algorithm to capture implicit structural characteristics of the network that are intrinsic to each instance, enhancing both the robustness and the effectiveness of the subsequent optimization steps.

6.3. Supplementary Results on the Maximization Variant of the WTSSP

In this section, we present supplementary results obtained by testing our matheuristic on the maximization variant of the WTSSP, hereafter denoted MAX-WTSSP. The goal is to show that the proposed algorithm is not only effective for the WTSSP, but also promising for related information propagation problems, including maximization settings. Unlike the more comprehensive experiments reported for the WTSSP, the tests here are exploratory, intended to verify whether a straightforward adaptation of our heuristic can already yield competitive results.

The MAX-WTSSP can be also seen as a natural generalization of the classical Influence Maximization Problem (IMP) Kempe et al. (2015), where vertices are associated with costs. Formally, MAX-WTSSP is defined as follows.

PROBLEM 2 (MAX-WTSSP). Given an instance $I = (G, c, t, b)$, where $G = (V, E)$ is an undirected graph, $c : V \to \mathbb{Z}^+$ assigns costs to the vertices, $t : V \to \mathbb{Z}^+$ defines thresholds, and $b \in \mathbb{Z}^+$ is a budget, the objective is to select a target set whose total cost does not exceed b and that maximizes the number of vertices that become active by the end of the propagation process.

An IP formulation for the MAX-WTSSP can be derived by modifying the PRY model introduced for the WTSSP (see Section 3). First, introduce binary variables $\{a_v : v \in V\}$ such that $a_v = 1$ if and only if v is a target or becomes active during propagation. Then, replace the objective function (8) with

$$\max \sum_{v \in V} a_v, \tag{13}$$

and substitute constraints (11) with the following set:

$$\sum_{u \in N(v)} f_{u,v} \geq t(v) \cdot (a_v - s_v) \qquad \forall v \in V \tag{14}$$

$$f_{u,v} \leq a_u \qquad \forall \{u, v\} \in E \tag{15}$$

$$s_v \leq a_v \qquad \forall v \in V \tag{16}$$

$$\sum_{v \in V} c(v) \leq b \tag{17}$$

Constraints (14) ensure that a non-target vertex becomes active only if the influence it receives meets its threshold. Constraints (15) enforce that a vertex can transmit influence only if it is already active. Constraints (16) guarantee that every target is activated. Finally, (17) imposes the budget constraint.

To adapt HMF to construct a heuristic tailored for the MAX-WTSSP, we proceed as follows. In the first stage (preprocessing), vertices that are never part of any optimal solution are labeled as *forbidden*. These vertices remain in the graph but are never considered as candidate targets in the subsequent stages. Any vertex whose cost exceeds the budget is automatically forbidden. We also label as forbidden any vertex v such that $\deg_G(v) = t(v)$ and $\sum_{u \in N(v)} c(u) \leq c(v)$, which were previously called *inerts* in Section 5.3. These vertices can be replaced in a target set by their neighbors without increasing its total cost, and if they

are not selected as targets, they become active only after all their neighbors, contributing nothing to the activation of other vertices.

In the second stage (relaxation-oriented construction), we follow the approach of the HMF−PRY heuristic described in Section 5.4, with two modifications. First, in the force-and-resolve procedure (lines 4 to 9 of Algorithm 1), variables are forced to 1 only as long as constraint (17) remains satisfied. Second, the loop of line 12 is executed only while there exists a vertex u in the priority queue Q such that adding u to the target set S does not violate the budget.

In the third stage (filtering), targets are removed from S only if their removal does not reduce the objective value. In the fourth stage (replacement), we follow the same procedure as in HMF−PRY, but the IP model used is the one described for MAX−WTSSP in this section.

Regarding the experimental design for the MAX−WTSSP, we first note that any instance of the WTSSP can be extended to an instance of the MAX−WTSSP by assigning a budget. Moreover, if I is an instance of WTSSP and S is a feasible solution for I, then a propagation started from S activates all vertices. Consequently, if we construct an instance I' of the MAX−WTSSP with budget $c(S)$, S is an optimal solution for I' because it respects the budget and activates all vertices, thereby maximizing the objective.

Based on this observation, we reused the WTSSP instances introduced at the beginning of Section 6 to generate 250 MAX−WTSSP instances, comprising 80 synthetic graphs and 170 real-world graphs, by simply assigning the budget as the cost of the best-known solution for each corresponding WTSSP instance. This procedure ensures that the optimal solutions for the benchmark are known a priori, allowing a direct comparison between the heuristic results and the optimal values.

To verify that these instances were not trivial, we preliminarily tested a subset with Gurobi, using either no initial solution or a naive solution. In both cases, the solver frequently failed to find optimal or near-optimal solutions within the 1-hour time limit, often reporting an optimality gap above 25%.

Finally, we applied our heuristic to the MAX−WTSSP for each instance. The resulting optimality gaps and runtimes are reported in Tables 13 and 14.

Table 13 Results for MAX−WTSSP on synthetic instances.

Graph	# Optimal	Optimality gap (%)	Time (s)
200-k4	0	1.63 ± 0.88	0.28 ± 0.18
200-k6	0	1.86 ± 1.86	0.20 ± 0.06
200-k8	1	1.32 ± 0.60	0.18 ± 0.08
200-k10	1	1.43 ± 0.97	0.40 ± 0.65
200-k12	1	1.79 ± 1.04	0.48 ± 0.55
2500-k16	0	0.32 ± 0.13	60.85 ± 29.58
5000-k8	0	0.49 ± 0.14	126.09 ± 55.56
10000-k4	0	0.60 ± 0.09	65.75 ± 22.37

Table 14 Results for MAX−WTSSP on real-world instances.

Graph	# Optimal	Optimality gap (%)	Time (s)
Hamsterster	0	0.16 ± 0.07	11.58 ± 7.34
Facebook-NIPS-Ego	10	0.00 ± 0.00	0.16 ± 0.02
Bitcoin Alpha	0	0.06 ± 0.01	10.58 ± 5.41
Advogato	0	0.08 ± 0.03	48.67 ± 14.80
Bitcoin OTC	0	0.09 ± 0.13	28.07 ± 5.73
P2P-Gnutella-08	0	0.08 ± 0.04	30.36 ± 14.03
P2P-Gnutella-09	0	0.06 ± 0.03	41.17 ± 8.06
P2P-Gnutella-06	0	0.09 ± 0.06	85.78 ± 21.15
P2P-Gnutella-05	0	0.06 ± 0.03	89.32 ± 35.71
Ning	0	0.06 ± 0.02	49.97 ± 7.98
Escorts	0	0.03 ± 0.01	92.25 ± 26.02
Oregon-01	0	0.02 ± 0.01	16.61 ± 5.63
P2P-Gnutella-04	0	0.08 ± 0.06	174.18 ± 42.11
Oregon-02	0	0.03 ± 0.02	33.90 ± 6.98
Anybeat	0	0.03 ± 0.01	55.13 ± 8.33
Google+	3	0.90 ± 2.42	235.77 ± 569.55
Facebook-LetsDoIt	5	0.40 ± 0.44	51.16 ± 48.88

Overall, the results demonstrate that the proposed heuristic for the MAX−WTSSP is highly effective across both synthetic and real-world instances. On synthetic graphs, the method consistently achieved solutions with small optimality gaps (below 2% in most cases), though only a few instances were solved to proven optimality. In contrast, on real-world graphs the heuristic attained extremely small gaps – often close to zero – and was able to identify exact optima for several instances, while maintaining competitive runtimes even on large networks. These findings confirm the robustness and accuracy of the approach in practice.

7. Concluding Remarks and Future Work

In this work, we introduced a hybrid matheuristic for combinatorial optimization problems involving the spread of information on social networks. The HMF framework is divided into four stages, combining different techniques such as preprocessing, linear programming, integer programming, and large neighborhood search. To demonstrate the effectiveness of the proposed framework, we designed two problem-specific heuristics, HMF−PRY and HMF−RAG, for the Weighted Target Set Selection Problem (WTSSP), a generalization of the well-known Target Set Selection Problem (TSSP). Our contributions included a new IP formulation for the WTSSP, denoted as PRY, along with a theoretical result regarding the strength of a relaxation for PRY. Both the IP model and its relaxation were employed in the HMF−PRY heuristic.

To evaluate the proposed heuristics, we conducted extensive computational experiments on 80 synthetic and 180 real-world instances of the WTSSP. The results demonstrated that both HMF−PRY and HMF−RAG consistently found high-quality solutions across both benchmarks, often achieving proven optimal solutions. These heuristics significantly outperformed the previously best-known heuristics for the WTSSP. While there was no clear dominance in solution quality between HMF−PRY and HMF−RAG, HMF−PRY exhibited faster performance on the largest instances, suggesting better scalability. This performance difference is likely attributable to the more compact LP and IP models used in HMF−PRY compared to those in HMF−RAG.

Additionally, by providing the exact methods with the solutions obtained by the heuristics, we were able to find provably optimal solutions for 173 instances (out of 260), including 60 that were previously unsolved, and tighter optimality gaps for the remaining unsolved instances.

Regarding the supplementary experiments on the maximization version of the WTSSP, we were able to employ the proposed framework to find provably optimal solutions for 21 instances (out of 250) and near-optimal solutions for the remaining instances.

For future research, we plan to investigate the efficacy of the proposed framework for other TSSP-like problems, including those with more general settings such as the Generalized Least Cost Influence Problem (Fischetti et al. 2018), as well as additional combinatorial optimization problems on social networks. Additionally, we see potential for enhancing the HMF algorithm, particularly by developing more efficient and effective strategies for the large neighborhood search stage.

Appendix

PROPOSITION 1. *If S is a feasible solution for I, then there exists an attribution of values to the variables of* PRY *such that $s_v = 1$ iff $v \in S$, and all of* PRY*'s constraints are satisfied.*

Proof Let S be a feasible solution for I. For each $v \in V$, set $s_v = 1$ if $v \in S$, and $s_v = 0$ otherwise. Considering a propagation on G starting from S, we now assign values to the f variables. For every $\{u, v\} \in E$, set $f_{u,v} = 1$ if u is a target but not v, or if u becomes active earlier than v during the propagation; otherwise, set $f_{u,v} = 0$. Additionally, assign a value to $f_{v,u}$ using the same rules, inverting the roles of u and v.

It follows directly from the assignment above that constraints (9) and (10) are met. Moreover, for each vertex v, there are two cases. If $v \in S$, then constraint (11) is clearly satisfied for v. Otherwise, since S is feasible for I, v is activated in some round $\tau > 0$, and consequently, there are at least $t(v)$ neighbors of v that are active in round $\tau - 1$. Each of these neighbors is either a target or becomes active earlier than v. Thus, the assignment of values to the f variables ensures that constraint (11) is satisfied for v.

Lastly, assume, for the purpose of obtaining a contradiction, that constraints (12) are violated. Then, there exists a directed cycle $\xi = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)\}$ with $k \geq 3$ such that $f_{u,v} = 1$ for each $(u, v) \in \xi$. Let v_i be a vertex in the cycle such that v_i is either a target or, if there are no targets in ξ, the earliest to be activated. If $i \geq 2$, then v_{i-1} becomes active no earlier than v_i during the propagation and, consequently, $f_{v_{i-1},v_i} = 0$, which contradicts $(v_{i-1}, v_i) \in \xi$. Otherwise, if $i = 1$, then v_k becomes active no earlier than v_1 during the propagation and, consequently, $f_{v_k,v_1} = 0$, contradicting $(v_k, v_1) \in \xi$. □

PROPOSITION 2. *Given an attribution of values to the variables of* PRY *that satisfies all of its constraints, the set $S = \{v : s_v = 1\}$ is a feasible solution for I.*

Proof Consider an assignment of values to the variables of PRY that satisfies all of its constraints. Let $S = \{v : s_v = 1\}$ be a target set. Define $G' = (V, A)$ as a directed graph, where $A = \{(u, v) : f_{u,v} = 1\}$. From constraints (10) and (12), we conclude that G' has no directed cycles, and therefore, any directed path in G' is simple. For each $v \in V$, denote by σ_v the length of the longest directed path in G' that ends at v. We claim that in the propagation on G started from S, every vertex v is active in round σ_v.

Let v be a vertex with $\sigma_v = 0$. Since $t(v) > 0$, constraint (11) implies $s_v = 1$. Consequently, $v \in S$, and therefore, v is active in round 0. Now, as an inductive hypothesis, assume the result holds for length $k \geq 0$. Let v be a vertex with $\sigma_v = k + 1$. Since there exists at least one arc that ends at v in G', it follows from constraint (9) that $s_v = 0$. Thus, constraint (11) implies that there are at least $t(v)$ arcs in G' that end at v. Given $\sigma_v = k + 1$, for each vertex u such that $(u, v) \in A$, we have $\sigma_u = k$. By the inductive hypothesis, these $t(v)$ vertices are active in round k. Therefore, v is active in round $k + 1$. □

PROPOSITION 3. *An optimal solution for* PRY *provides an optimal solution for I.*

Proof Let k be the optimal value for PRY and consider an optimal solution for this formulation. From Proposition 2, there exists a feasible solution S for I such that $v \in S$ iff $s_v = 1$ in the optimal solution for PRY. Thus, $c(S) = k$. We claim that S is optimal for I. Assume, on the contrary, that S is not optimal for I. Then, there exists another feasible solution S' for I such that $c(S') < c(S)$. According to Proposition 1, there exists a feasible solution for PRY such that $s_v = 1$ iff $v \in S'$. Therefore, the objective value of such solution for PRY equals $c(S')$, which is less than the optimal value k, resulting in a contradiction. □

PROPOSITION 4. *The value of an optimal solution for R^{RAG} coincides with the value of an optimal solution for R^{PRY}.*

Proof First, we prove that the value of an optimal solution for R^{RAG} is a lower bound for the value of an optimal solution for R^{PRY}. Consider an optimal solution for R^{PRY}, that is, an attribution of values to the fractional variables s and f that satisfies constraints (9), (10) and (11), and optimizes the objective function (8). We now assign values to the fractional variables of R^{RAG} (x, y, and h) to obtain a feasible solution for R^{RAG}. For each vertex v, set $x_v = s_v$, ensuring that both solutions have the same value for their objective functions (8) and (1). Next, for each edge $\{u, v\}$ and its corresponding dummy vertex $d^{\{u,v\}}$, set $h_{u,v} = y_{u,d} = 1 - f_{v,u}$; $h_{v,u} = y_{d,u} = f_{v,u}$; $y_{d,v} = f_{u,v}$; and $y_{v,d} = 1 - f_{u,v}$.

Since $x_u = s_u$, $x_v = s_v$, $f_{u,v} = 1 - y_{v,d}$ and $f_{v,u} = 1 - y_{u,d}$, the satisfaction of constraints (9) implies that constraints (2) are satisfied. Since $y_{v,d} + y_{d,v} = 1$ and $y_{u,d} + y_{d,u} = 1$, constraints (3) are also met. Moreover, since $h_{u,v} + h_{v,u} = 1$, we have that constraints (4) are satisfied as well. To show that constraints (5) are respected, we first observe that $h_{u,v} \leq y_{u,d}$. Furthermore, since $f_{u,v} + f_{v,u} \leq 1$ due to constraints (10), it follows that $h_{v,u} = f_{v,u} \leq 1 - f_{u,v} = y_{v,d}$. Given that $y_{d,v} = f_{u,v}$ and $y_{d,u} = f_{v,u}$, the satisfaction of constraints (11) ensures constraints (6) are met.

For the second part of this proof, we show that the value of an optimal solution for $R^{\mathtt{PRY}}$ is a lower bound for the value of an optimal solution for $R^{\mathtt{RAG}}$. Consider an optimal solution for $R^{\mathtt{RAG}}$, that is, an attribution of values to the fractional variables x, y and h that satisfies constraints (2), (3), (4), (5) and (6), and optimizes the objective function (1). We now assign values to the fractional variables of $R^{\mathtt{PRY}}$ (s and f) to obtain a feasible solution for $R^{\mathtt{PRY}}$. First, for each vertex v, set $s_v = x_v$. This ensures that both models have the same value for their objective functions (1) and (8). Next, for each edge $\{u, v\} \in E$, set $f_{u,v} = h_{u,v}$ if $x_v = 0$, and $f_{u,v} = 0$ if $x_v = 1$. Similarly, set $f_{v,u} = h_{v,u}$ if $x_u = 0$, and $f_{v,u} = 0$ if $x_u = 1$.

From these assignments it follows that $f_{u,v} \leq 1 - x_v = 1 - s_v$, which guarantees that constraints (9) are met. Moreover, since $f_{u,v} \leq h_{u,v}$ and $f_{v,u} \leq h_{v,u}$, the satisfaction of constraints (4) implies that constraints (10) are also satisfied. It remains to verify constraints (11). If $x_v = 1$, then $s_v = 1$, and the constraint is trivially satisfied. Otherwise, if $x_v = 0$, constraints (6) imply that $\sum_{d \in N_{G'}(v)} y_{d,v} \geq t(v)$, where G' is the extended graph containing dummy vertices. Consider any $d \in N_{G'}(v)$ with $y_{d,v} = 1$. Then, by constraints (3), we have $y_{v,d} = 0$. From constraints (5), $h_{v,u} = 0$, and by constraints (4), $h_{u,v} = 1$. Since $x_v = 0$, this implies $f_{u,v} = 1$. Therefore, $\sum_{u \in N(v)} f_{u,v} \geq t(v)$, and constraints (11) are satisfied. □

PROPOSITION 5. *Let $S, S' \subseteq V$. Then $(S \cup S')_{\text{final}} = (S_{\text{final}} \cup S')_{\text{final}}$.*

Proof First, we prove that $(S \cup S')_{\text{final}} \subseteq (S_{\text{final}} \cup S')_{\text{final}}$. Observe that if T and T' are target sets such that $T \subseteq T'$, then every $v \in V \setminus T$ that becomes active in the propagation on G started from T is either in T' or becomes active in the propagation on G started from T'. Thus, $T_{\text{final}} \subseteq T'_{\text{final}}$. Since $S \subseteq S_{\text{final}}$, $(S \cup S') \subseteq (S_{\text{final}} \cup S')$ and, therefore, $(S \cup S')_{\text{final}} \subseteq (S_{\text{final}} \cup S')_{\text{final}}$.

Now, we show that $(S_{\text{final}} \cup S')_{\text{final}} \subseteq (S \cup S')_{\text{final}}$. Suppose otherwise. Since $S_{\text{final}} \cup S' \subseteq (S \cup S')_{\text{final}}$, there exists $\tau \geq 1$ such that $(S_{\text{final}} \cup S')_{\tau-1} \subseteq (S \cup S')_{\text{final}}$ and $(S_{\text{final}} \cup S')_{\tau} \nsubseteq (S \cup S')_{\text{final}}$. Let $u \in (S_{\text{final}} \cup S')_{\tau} \setminus (S \cup S')_{\text{final}}$. Since $u \in (S_{\text{final}} \cup S')_{\tau}$, we conclude that $|N(u) \cap (S_{\text{final}} \cup S')_{\tau-1}| \geq t(u)$. However, $(S_{\text{final}} \cup S')_{\tau-1} \subseteq (S \cup S')_{\text{final}}$ and, consequently, u also becomes active in the propagation on G started from $(S \cup S')_{\text{final}}$. Thus, $u \in (S \cup S')_{\text{final}}$, contradicting the choice of u. □

PROPOSITION 6. *If $S' \subseteq V'$ is a feasible solution for I', then $S \cup S'$ is a feasible solution for I.*

Proof Let $S' \subseteq V'$ be a feasible solution for I'. Define $H_A = (V, A)$ as a directed graph, where arc $(u, v) \in A$ iff u is active before v becomes active in a propagation on G started from S. Similarly, define $H_{A'} = (V', A')$ as a directed graph, where arc $(u, v) \in A'$ iff u is active before v becomes active in a propagation on G' started from S'. Clearly, both H_A and $H_{A'}$ are acyclic. Moreover, if a vertex $v \in V$ has no incoming arc in H_A, then $\mathit{inf}(v) = 0$. But $t(v) > 0$, which implies that either $v \in S$ or $v \in V'$. On the other hand, since S' is a feasible solution for I', if a vertex $v \in V'$ has no incoming arc in $H_{A'}$, then $v \in S'$.

Now, define $H_{A \cup A'} = (V, A \cup A')$ as a directed graph that is the union of graphs H_A and $H_{A'}$. Since $H_{A'}$ contains arcs only between vertices in V', which are sinks in H_A, $H_{A \cup A'}$ has no repeated arc. Also, since H_A and $H_{A'}$ are acyclic, $H_{A \cup A'}$ is acyclic. For each $v \in V$, denote by σ_v the length of the longest directed path in $H_{A \cup A'}$ that ends at v. We claim that in the propagation on G started from $S \cup S'$, every $v \in V$ is active in round σ_v.

Let $v \in V$ be a vertex with $\sigma_v = 0$. Then, v has no incoming arc in $H_{A \cup A'}$ and, by construction of $H_{A \cup A'}$, we conclude that either $v \in S$ or $v \in S'$. Therefore, v is active in round 0. Now, as an inductive hypothesis, assume that the result holds for length $k \geq 0$. Let $v \in V$ be a vertex with $\sigma_v = k + 1$. Then, v has at least $t(v)$ incoming arcs in $H_{A \cup A'}$, where at least $t(v) - t'(v)$ arcs are from H_A and at least $t'(v)$ arcs are from $H_{A'}$. By the inductive hypothesis, the source vertices of these arcs are active in round k. Hence, v is active in round $k + 1$. □

PROPOSITION 7. *If $S' \subseteq V'$ is such that $S \cup S'$ is a feasible solution for I, S' is a feasible solution for I'.*

Proof Let $S' \subseteq V'$ be a set such that $S \cup S'$ is a feasible solution for I. From Proposition 5, we have that $(S \cup S')_{\text{final}} = (S_{\text{final}} \cup S')_{\text{final}}$, which implies that $S_{\text{final}} \cup S'$ is a feasible solution for I. Define H_A as a directed graph, where arc $(u, v) \in A$ iff u is active before v becomes active in a propagation on G started from $S_{\text{final}} \cup S'$. Clearly, H_A is acyclic. Also, by construction of H_A, each vertex from V is either in $S_{\text{final}} \cup S'$ or has at least $t(v)$ incoming arcs in H_A.

Now, let $H_{A'} = (V', A')$ be the subgraph of H_A induced by V'. Clearly, $H_{A'}$ is acyclic. For each $v \in V$, denote by σ_v the length of the longest directed path in $H_{A'}$ that ends at v. We claim that in the propagation on G' started from S', every $v \in V'$ is active in round σ_v.

Let $v \in V'$ be a vertex with $\sigma_v = 0$. Then, v has no incoming arc in $H_{A'}$ and, by construction of $H_{A'}$, we conclude that $v \in S'$. Therefore, v is active in round 0. Now, as an inductive hypothesis, assume the result holds for length $k \geq 0$. Let $v \in V'$ be a vertex with $\sigma_v = k + 1$. Then, by construction of $H_{A'}$, v has at least $t(v) - \mathit{inf}(v)$ incoming arcs in $H_{A'}$, totaling at least $t'(v)$ arcs. By the inductive hypothesis, the source vertices of these arcs are active in round k. Hence, v is active in round $k + 1$. □

PROPOSITION 8. *Let $S' \subseteq V'$. Then, S' is a feasible solution for I' iff $S \cup S'$ is a feasible solution for I.*

Proof It follows directly from Propositions 6 and 7. □

References

Ackerman E, Ben-Zwi O, Wolfovitz G (2010) Combinatorial model and bounds for target set selection. *Theoretical Computer Science* 411(44):4017–4022, URL http://dx.doi.org/10.1016/j.tcs.2010.08.021.

Ahuja RK, Özlem Ergun, Orlin JB, Punnen AP (2002) A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics* 123(1):75–102, URL http://dx.doi.org/10.1016/S0166-218X(01)00338-9.

Bakshy E, Hofman JM, Mason WA, Watts DJ (2011) Everyone's an influencer: Quantifying influence on twitter. *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, 65–74, WSDM '11.

Banerjee S, Jenamani M, Pratihar DK (2020) A survey on influence maximization in a social network. *Knowledge and Information Systems* 62(9):3417–3455, URL http://dx.doi.org/10.1007/s10115-020-01461-4.

Ben-Zwi O, Hermelin D, Lokshtanov D, Newman I (2011) Treewidth governs the complexity of target set selection. *Discrete Optimization* 8(1):87–96, URL http://dx.doi.org/10.1016/j.disopt.2010.09.007.

Boschetti MA, Maniezzo V (2022) Matheuristics: using mathematics for heuristic design. *4OR* 20(2):173–208.

Boschetti MA, Maniezzo V, Roffilli M, Bolufé Röhler A (2009) Matheuristics: Optimization, simulation and control. *Hybrid Metaheuristics*, 171–177, URL http://dx.doi.org/10.1007/978-3-642-04918-7_13.

Brown JJ, Reingen PH (1987) Social ties and word-of-mouth referral behavior. *Journal of Consumer Research* 14(3):350–362.

Campbell C, Farrell JR (2020) More than meets the eye: The functional components underlying influencer marketing. *Business Horizons* 63(4):469–479, URL http://dx.doi.org/10.1016/j.bushor.2020.03.003.

Caserta M, Voß S (2010) Metaheuristics: Intelligent problem solving. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, 1–38, URL http://dx.doi.org/10.1007/978-1-4419-1306-7_1.

Chen N (2009) On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics* 23(3):1400–1415, URL http://dx.doi.org/10.1137/08073617X.

Chen W, Lakshmanan L, Castillo C (2013) Information and influence propagation in social networks. *Synthesis Lectures on Data Management* 5(4):1–177, URL http://dx.doi.org/10.1007/978-3-031-01850-3.

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

Domingos P, Richardson M (2001) Mining the network value of customers. *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 57–66.

Dye R (2000) The buzz on buzz. *Harvard business review* 78(6):139–146.

Fischetti M, Kahr M, Leitner M, Monaci M, Ruthmair M (2018) Least cost influence propagation in (social) networks. *Mathematical Programming* 170(1):293–325, ISSN 1436-4646, URL `http://dx.doi.org/10.1007/s10107-018-1288-y`.

Goldenberg J, Libai B, Muller E (2001) Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters* 12(3):211–223, URL `http://dx.doi.org/10.1023/A:1011122126881`.

Kempe D, Kleinberg J, Tardos E (2003) Maximizing the spread of influence through a social network. *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 137–146.

Kempe D, Kleinberg J, Tardos É (2005) Influential nodes in a diffusion model for social networks. *Automata, Languages and Programming*, 1127–1138, URL `http://dx.doi.org/10.1007/11523468_91`.

Kempe D, Kleinberg J, Tardos E (2015) Maximizing the spread of influence through a social network. *Theory of Computing* 11(4):105–147, URL `http://dx.doi.org/10.4086/toc.2015.v011a004`.

Kunegis J (2013) KONECT – The Koblenz Network Collection. *Proceedings of the 22nd International Conference on World Wide Web*, 1343–1350, URL `http://dl.acm.org/citation.cfm?id=2488173`.

Leskovec J, Krevl A (2014) SNAP Datasets: Stanford large network dataset collection.

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*, 368–376.

Li Y, Fan J, Wang Y, Tan KL (2018) Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* 30(10):1852–1872, URL `http://dx.doi.org/10.1109/TKDE.2018.2807843`.

Maniezzo V, Boschetti MA, Stützle T (2021) *Matheuristics: Algorithms and Implementations* (Springer).

Maniezzo V, Stützle T, Voß S (2009) *Matheuristics: Hybridizing metaheuristics and mathematical programming* (Springer).

Pereira FC (2021) *A computational study of the Perfect Awareness Problem*. Master's thesis, Institute of Computing, University of Campinas (UNICAMP), Brazil, URL `http://hdl.handle.net/20.500.12733/1641217`.

Pereira FC, de Rezende PJ (2023) The least cost directed perfect awareness problem: complexity, algorithms and computations. *Online Social Networks and Media* 37-38:100255.

Pereira FC, de Rezende PJ, de Souza CC (2021) Effective heuristics for the perfect awareness problem. *Proceedings of the XI Latin and American Algorithms, Graphs and Optimization Symposium*, volume 195, 489–498.

Pereira FC, de Rezende PJ, Yunes T (2024) A hybrid matheuristic for the spread of influence on social networks - complementary data. Mendeley Data, V1, URL `http://dx.doi.org/10.17632/f4kyk7vkst`.

Pisinger D, Ropke S (2019) Large neighborhood search. *Handbook of Metaheuristics*, 99–127.

Raghavan S, Zhang R (2019) A branch-and-cut approach for the weighted target set selection problem on social networks. *INFORMS Journal on Optimization* 1(4):304–322, URL `http://dx.doi.org/10.1287/ijoo.2019.0012`.

Raghavan S, Zhang R (2021) Weighted target set selection on trees and cycles. *Networks* 77(4):587–609.

Ramos N (2018) *A computational study of the Firefighter Problem on graphs*. Master's thesis, Institute of Computing, University of Campinas (UNICAMP), Brazil, URL `http://dx.doi.org/10.47749/T/UNICAMP.2018.1014614`.

Ramos N, de Souza CC, de Rezende PJ (2020) A matheuristic for the firefighter problem on graphs. *International Transactions in Operational Research* 27(2):739–766, URL http://dx.doi.org/10.1111/itor.12638.

Ravelo SV, Meneses CN (2021) Generalizations, formulations and subgradient based heuristic with dynamic programming procedure for target set selection problems. *Computers & Operations Research* 135:105441.

Richardson M, Domingos P (2002) Mining knowledge-sharing sites for viral marketing. *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 61–70.

Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, URL http://networkrepository.com.

Shakarian P, Eyre S, Paulo D (2013) A scalable heuristic for viral marketing under the tipping model. *Social Network Analysis and Mining* 3(4), URL http://dx.doi.org/10.1007/s13278-013-0135-7.

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. *Principles and Practice of Constraint Programming — CP98*, 417–431.

Singh SS, Srivastva D, Verma M, Singh J (2022) Influence maximization frameworks, performance, challenges and directions on social network: A theoretical study. *Journal of King Saud University - Computer and Information Sciences* 34(9):7570–7603, URL http://dx.doi.org/10.1016/j.jksuci.2021.08.009.

Spencer G, Howarth R (2013) Maximizing the spread of stable influence: Leveraging norm-driven moral-motivation for green behavior change in networks. *arXiv* URL http://dx.doi.org/10.48550/arXiv.1309.6455.

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