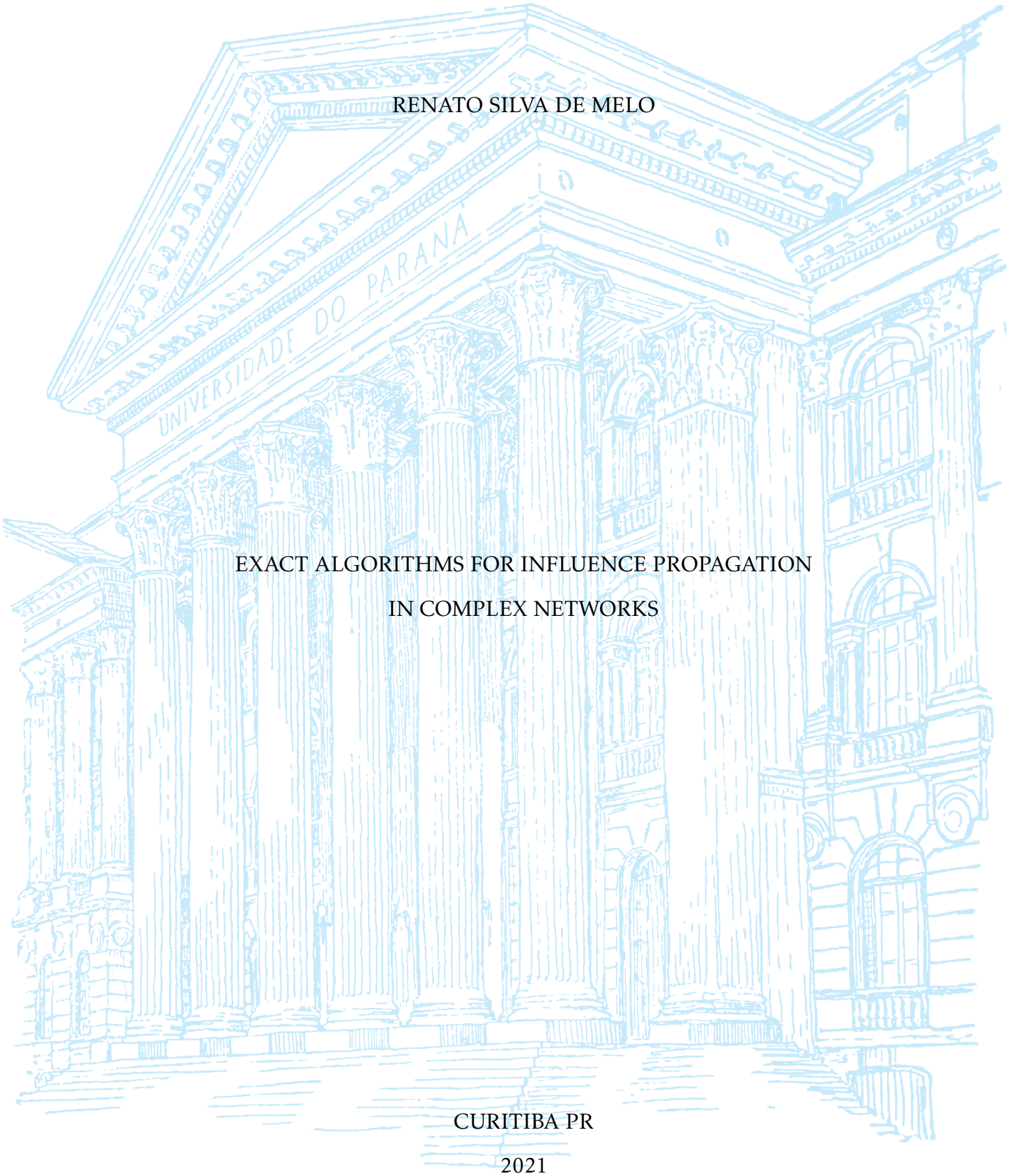
UNIVERSIDADE FEDERAL DO PARANÁ

RENATO SILVA DE MELO



EXACT ALGORITHMS FOR INFLUENCE PROPAGATION IN COMPLEX NETWORKS

CURITIBA PR

2021

RENATO SILVA DE MELO

EXACT ALGORITHMS FOR INFLUENCE PROPAGATION IN COMPLEX NETWORKS

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: André Luís Vignatti.

CURITIBA PR

2021

CATALOGAÇÃO NA FONTE – SIBI/UFPR

M528e Melo, Renato Silva de

Exact algorithms for influence propagation in complex networks [recurso eletrônico]/ Renato Silva de Melo - Curitiba, 2021.

Tese apresentada no Programa de Pós-Graduação Graduação em Informática, setor de Ciências Exatas da Universidade Federal do Paraná.
Orientador: Prof. Dr. André Luís Vignatti.

1. Redes. 3. Informática. I. Vignatti, André Luís. II. Título. III. Universidade Federal do Paraná.

CDD 658.4038

Bibliotecária: Vilma Machado CRB9/1563

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **RENATO SILVA DE MELO** intitulada: **Exact algorithms for Influence Propagation in Complex Networks**, sob orientação do Prof. Dr. ANDRÉ LUÍS VIGNATTI, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 23 de Março de 2021.

Assinatura Eletrônica
26/03/2021 10:35:54.0
ANDRÉ LUÍS VIGNATTI
Presidente da Banca Examinadora

Assinatura Eletrônica
25/03/2021 17:18:50.0
CASSIUS TADEU SCARPIN
Avaliador Externo (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
25/03/2021 17:38:42.0
ANDRÉ LUIZ PIRES GUEDES
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
25/03/2021 18:06:32.0
GUILHERME ALEX DERENIEVICZ
Avaliador Externo (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
26/03/2021 12:00:05.0
PEDRO HENRIQUE DEL BIANCO HOKAMA
Avaliador Externo (UNIVERSIDADE FEDERAL DE ITAJUBÁ)

Dedico esse trabalho a minha mãe Maria dos Santos, que já se foi, mas continua sendo minha maior fonte de força e inspiração.

ACKNOWLEDGEMENTS

Sou grato a Universidade Federal do Paraná e ao Programa de Pós-graduação em Informática pela grande oportunidade que tive de imergir no doutorado em Ciência da Computação. Desde 2016, o Departamento de Informática foi praticamente minha morada. Enquanto estudante de doutorado, segui aprendendo dia após dia coisas que, para mim, são valiosíssimas e que só foram possíveis graças aos professores e a estrutura da universidade. Agradeço também ao meu orientador, professor André Vignatti, pela confiança e por todo o suporte tanto pessoal como profissional.

Sou grato aos meus amigos do grupo TEORIA (Grupo de Pesquisa em Teoria da Computação, Otimização e Combinatória) pelos quais tenho muito carinho. Como membro do grupo, eu aprendi a enxergar Algoritmos e Computação de outra maneira. Além disso, a vivência no laboratório me proporcionou aprendizados que vão muito além do escopo da tese. Também tive a felicidade de fazer amigos de vários outros laboratórios e colecionar momentos que levarei no coração para sempre. Infelizmente, devido a pandemia de COVID-19, tivemos que nos afastar fisicamente na reta final no meu doutorado, mas guardo todos na memória.

Além de todos os amigos que conquistei no Paraná, também tive o apoio de amigos de longa data que me acompanham desde o Mato Grosso. Me deram suporte e me incentivaram como se fosse família. Não poderia deixar de mencionar as pessoas que conheci no LOCo - (Laboratório de Otimização Combinatória) da Universidade Estadual de Campinas, onde estive durante um ano. Lá tive a oportunidade de conhecer pessoas admiráveis que também nunca esquecerei.

Por último, mas não menos importante, agradeço a minha pequena família, Sangela, Sonia e Henrique, por todo carinho e amor, e também por me acompanharem na busca desse meu sonho.

Obrigado a todos os envolvidos.

RESUMO

Uma rede complexa é um tipo de grafo com características topológicas que normalmente aparecem em grafos que modelam dados do mundo real. Um fato importante sobre tais redes é que existem indicações práticas da tratabilidade de alguns problemas de otimização combinatória em grafos cuja distribuição de graus segue uma lei de potência. Um problema combinatório decorrente da investigação sobre propagação de influência em redes sociais é o problema de Maximização de Influência, que surge no contexto de adoção em cadeia de novos comportamentos pelos indivíduos de uma rede social. Embora o problema Maximização de Influência tenha sido estudado intensivamente, a literatura a partir da perspectiva de métodos exatos é limitada e nenhum estudo explora a distribuição de graus desta perspectiva. Os resultados recentes em programação matemática para este problema são relevantes, mas ainda existe espaço para melhorias. Portanto, para resolver o problema de Maximização de Influência usando algoritmos exatos, propomos tirar proveito de algumas propriedades dos grafos lei de potência. As contribuições desta tese são três. Duas destas são técnicas de pré-processamento para variantes do problema de maximização de influência. Ambos os algoritmos são especializados na distribuição de grau de redes complexas. A terceira contribuição é um algoritmo combinatório específico do problema para calcular um limitante dual "apertado" para o problema de Influência de Custo Mínimo. Além disso, apresentamos análises teóricas para nossas descobertas e fornecemos experimentos empíricos para avaliar nossas propostas em grafos sintéticos e dados do mundo real.

Palavras-chave: Propagação de influência. Otimização combinatória. Redes complexas.

ABSTRACT

A complex network is a type of graph with topological characteristics that usually appear in graphs that model data observed in real-word. An important fact about these networks is that there are practical indications of the tractability of some combinatorial optimization problems in graphs with power-law degree distribution. A combinatorial problem arising from the investigation on the spread of influence in social networks is the Influence Maximization, which appears in the context of a chain adoption of new behaviors by the individuals of a social network. Although the Influence Maximization problem has been studied intensively, only limited literature is from the exact methods perspective, and none of them exploits the degree distribution. Recent results in mathematical programming for this problem and its variants are relevant. However, there is room for improvements in the exact solutions of literature. Therefore, to address the Influence Maximization problem using exact algorithms, we propose to take advantage of some properties of complex networks. The contributions of this thesis are threefold. Two of them are presolving techniques for variants of the influence maximization problem. Both algorithms are specialized in the power-law degree distribution of complex networks. The third contribution is a problem-specific combinatorial algorithm to compute a tight dual bound for the Least Cost Influence Problem. Moreover, we present theoretical analysis for our findings and provide empirical experiments to evaluate our proposals in synthetic graphs and real-world data.

Keywords: Influence propagation. Combinatorial optimization. Complex networks.

LIST OF FIGURES

2.1 A branch-and-bound tree in which the branching decisions are made on a variable x_j. Each node represents a subproblem of the original one P_0. Nodes P_i' and P_i'' represent the sub-problems obtained by lower and upper bounding, respectively, the variable x_j. The factional solution of the continuous relaxation for the variable x_j is represented by $\hat{x}_j$. 22

3.1 Activating w in the LT model. Vertices u and v are active, and vertex w is about to decide whether will be activated or not. 28

3.2 Activating w in the IC model. Vertices u and v are active, and vertex w is about to decide whether will be activated or not. 29

3.3 Suppose that the three points inside the bigger cycle are the initial adopters and the gray cycles represent the reach of its influence. The part of the smaller cycle not intersecting the bigger is the marginal gain of the new element.. 31

3.4 Activation process in the threshold model starting from a target set $S = \{a\}$. The number attached to each vertex represents its threshold. Sets A_t contains the active vertices at time t, for $t = 0, ..., 4$. Highlighted arcs indicate who influences whom in the process.. 33

3.5 Activation process in the threshold model considering incentives, starting from a target set $S = \{a\}$. The number attached to each vertex represents its threshold. Sets A_t contains the active vertices at time t, for $t = 0, ..., 4$. Labels in each arc (u, v) denote the weight of influence d_{uv}. Highlighted arcs indicate who influences whom in the process. The vector **y** indexed by the vertices contains the amount of incentive offered to each vertex. . . 35

3.6 Random graphs.. 36

3.7 Degree histogram of random graphs.. 37

3.8 Histogram and log-log scale plot of the degree distribution of a synthetic power-law graph with 91789 vertices and 221053 edges with $\beta = 2.3$.. . . 38

4.1 Influencing-set $U \subseteq N(v)$ of a vertex v, where $N(v) = \{a, b, c\}$ and $U = \{a, b\}$. 45

5.1 Coverage set $D \subseteq V$, candidate set $C \subseteq D$ and a vertex v of a graph.. . . . 51

5.2 One step of the preselection process. 52

5.3 In the preselection, v can be discarded by two different ways, (a) and (b) are the two ways in which the neighbors of v can be covered by the set of candidates C. 54

5.4 A vertex v that have out-neighbors covered. That is, it shares all the out-neighbors with the set of candidates, but such neighbors can be both in C and $N^+(C)$.. 55

5.5 The set of candidates can directly reach all the out-neighbors of a vertex $v \notin C$. It means that the set of candidates exert at least the same influence of v over the out-neighbors of v.. 57
5.6 The simulations in NetHEP, NetPHY and Amazon have propagation probability $p = 0.025$, and the simulations in DBLP, Enron and Epinions have $p = 0.0025$. 61
5.7 Simulations in synthetic graphs with 64 thousand vertices and propagation probability $p \in [0, \frac{1}{4}]$. We plot only one size of graph due to similar results in all tested sizes.. 64
5.8 Average running time (a) and number of calls to the σ function (b) of both algorithms to find 50 seeds on synthetic graphs. 65

6.1 Example of the algorithm execution. 68
6.2 Degree distribution in logarithmic scale of real world networks. 73
6.3 Order of growth of the running time of the PREPROCESSING algorithm.. . . 75

7.1 Searching for short paths from a vertex k to its incoming neighbors N_k.. . 78
7.2 The decision tree where the branching decisions are made on binary variables. The black node represents the sub-problem obtained by fixing variables $z_{ae} = 1$ and $z_{dc} = 0$.. 80
7.3 The directed graph in Figure (b) is obtained by removing the arc (d, c) . . 81
7.4 Graph G', in (a), has three different strongly connected component. The components are C_u, C_v and C_w with set of vertices $\{a, b, c\}, \{d, e, f\}$ and $\{g, h\}$, respectively. Graph H, in (b), has three vertices, u, v and w, associated to C_u, C_v and C_w, respectively. In both figures, the vertices are labeled with name and threshold. For example, vertex $v \in V(H)$ has threshold $t_v = 2$, which is the smallest threshold in the component C_v. . . 82
7.5 The graph in Figure (a) contains a instance of the LCIP with $\alpha = 1$. The solution is the graph of Figure (b), were the incentives are $y_a = y_c = 1$ and $y_b = y_d = 0$.. 84
7.6 The solution of LCIP in Figure 7.6(a) has value 5, the incentives are: $y_a = 2, y_b = y_g = y_h = 1$ and $y_c = y_d = y_e = y_f = 0$. The solution of WLCIP in Figure 7.6(b) has value 3, the incentives are: $y_u = 2, y_w = 1$ and $y_v = 0$. 90

LIST OF TABLES

5.1 Statistics information of the social networks. The β values are from Liu et al. (2014) and Tang et al. (2008) results.. 60

5.2 Difference between the running time of CELF and PREVALENTSEED, for $k = 50$. 62

5.3 Calls to the σ function in all tested graphs. Columns 4 and 5 shows the total of reorganizations of Q needed to PREVALENTSEED and CELF, respectively. 62

6.1 Reduction in size of real social networks after preprocessing.. 72

7.1 Experiments with synthetic small-world graphs. 95

7.2 Real world social networks. 96

7.3 Experiments on real world based social networks for $\alpha = \{1, 0.5, 0.1\}$. . . . 97

CONTENTS

1 INTRODUCTION 12
1.1 RESEARCH QUESTIONS 13
1.2 CONTRIBUTIONS 13
2 MATHEMATICAL OPTIMIZATION 16
2.1 LINEAR PROGRAMMING 16
2.1.1 Duality theory 17
2.2 INTEGER LINEAR PROGRAMMING 17
2.3 EXACT ALGORITHMS FOR INTEGER PROGRAMMING 19
2.3.1 Branch-and-bound method 19
2.3.2 Cutting plane methods 22
2.3.3 Branch-and-cut 24
3 SOCIAL INFLUENCE AND POWER-LAW GRAPHS 27
3.1 NOTATION 27
3.2 DIFFUSION MODELS 28
3.2.1 Linear threshold model. 28
3.2.2 Independent cascade model 29
3.3 INFLUENCE MAXIMIZATION PROBLEM 30
3.3.1 Properties of the influence function 30
3.3.2 Greedy $(1 - 1/e)$-approximation algorithm 31
3.4 TARGET SET SELECTION PROBLEM 32
3.4.1 Threshold model 32
3.5 LEAST COST INFLUENCE PROBLEM 34
3.5.1 Threshold model with incentives 34
3.6 POWER-LAW GRAPHS 36
4 LITERATURE REVIEW 40
4.1 APPROXIMATION AND HEURISTICS 40
4.2 DISCRETE OPTIMIZATION FOR INFLUENCE PROPAGATION 40
4.2.1 Target set selection 41
4.2.2 Weighted target set selection 42
4.2.3 Target set with partial incentives 43
4.2.4 Generalized lest cost influence problem 44
4.3 COMBINATORIAL OPTIMIZATION ON POWER-LAW GRAPHS 47
4.3.1 On general problems 47
4.3.2 On the influence propagation problems 48

5 PRESELECTION FOR INFLUENCE MAXIMIZATION 50
5.1 PRESELECTION 51
5.1.1 Set of candidates 51
5.1.2 Analysis of the preselection process 52
5.2 THE PREVALENTSEED ALGORITHM. 58
5.3 EXPERIMENTS 60
5.3.1 Real world power law graphs 60
5.3.2 Synthetic graphs. 63
6 PREPROCESSING RULES FOR TARGET SET SELECTION 66
6.1 PROBLEM DEFINITION. 66
6.2 RULES 67
6.2.1 Expected reduction on power-law graphs 70
6.3 COMPUTATIONAL EXPERIMENTS 71
6.3.1 Instance size reduction 71
6.3.2 Scalability 74
7 TIGHTER DUAL BOUNDS FOR LEAST COST INFLUENCE PROBLEM. 76
7.1 PROBLEM DEFINITION. 77
7.2 INTEGER LINEAR PROGRAMMING FORMULATION 77
7.2.1 Separation of generalized cycle elimination 78
7.2.2 Tighter Bounds 79
7.3 LOWER BOUND ALGORITHM. 79
7.3.1 The case of $\alpha < 1$ 85
7.4 BRANCHING RULE 92
7.5 PREPROCESSING FOR $\alpha = 1$ 93
7.6 COMPUTATIONAL EXPERIMENTS 94
7.6.1 Synthetic graphs. 94
7.6.2 Real world networks 96
8 CONCLUSION 99
REFERENCES 102

1 INTRODUCTION

The diffusion of ideas, behaviors, and innovations in social networks have the curious property of always starts from a small group of *early adopters* (Kleinberg, 2007). Over time, more and more people adopt the same behavior from this group by observing what their friends, neighbors, or colleagues have already done so. Thus information spreads like an epidemic. In the study of diffusion of information, the Influence Maximization problem arises naturally under a framework where the goal is to maximize the reach of the given information in a specific social system. Informally, the Influence Maximization problem aims to identify a set of early adopters (or *target set*) of fixed size that maximizes the expected number of achieved individuals at the end of a chain of adoptions.

Initially inspired by viral marketing applications in social networks, Domingos and Richardson (2001) were the first to study this problem algorithmically in probabilistic settings. From Domingos and Richardson's encounters, Kempe et al. (2003) formulate it as a discrete optimization problem where the goal is to find a set S of fixed size, such that the reach of S is the greatest possible. They use two basic probabilistic diffusion models, called linear threshold and independent cascade, to represent the spread of information. They also prove that this problem is NP-hard in both diffusion models and propose an $(1 - \frac{1}{e})$-approximate greedy algorithm (Algorithm 5). With the development of new researches on this area, more approximation results for particular cases and different generalizations of the diffusion models were introduced in the literature. Many works concentrated on approximation algorithms and heuristics methods, especially concerning the estimation of the influence function (Chen et al., 2010a, 2009, 2010b; Dinh et al., 2014; Goyal et al., 2011a,c; Leskovec et al., 2007; Tang et al., 2014). There exist few studies concerning *exact methods* for the influence maximization problem. A few known results using *mathematical programming* methods have been proposed in the last decade (Ackerman et al., 2010; Fischetti et al., 2018; Günneç et al., 2016; Raghavan and Zhang, 2015; Wu and Küçükyavuz, 2018). Although the recent results are very relevant for this research field, there are still many techniques of exact algorithms that were not applied to this problem yet.

Moreover, the problem is well known to be hard to solve and approximate even in restricted classes of graphs, such as for notoriously easy cases as bounded degree and bipartite graphs (Ben-Zwi et al., 2011). Nonetheless, we try to take advantage of knowing a topological property present in most of the large scale social networks, which is a *power-law degree distribution*. In such networks, there are few vertices with a large number of neighbors, called *hubs*, and many with low degree (Clauset et al., 2009; Easley and Kleinberg, 2010; Liu et al., 2014). Liu et al. (2014) observe that only a few out-neighbors of the hubs have considerable influence. At the same time, many of these neighbors contribute little to the spread of information from the target set. These findings suggest that the degree distribution and sparsity of the input graph are closely related to the performance of algorithms in the problem considered here.

It is worth noting that power-law distribution has been observed in a variety of situations in nature and social sciences. Not surprisingly, power-law also occurs in computer science environments. It has long been observed that, in graphs that model social networks, the degree distribution appears to abide by a power-law. That

is, the number of vertices of degree d is proportional to $d^{-\beta}$ for some exponent $\beta > 0$. In terms of probability, if we choose a vertex uniformly at random, a vertex of degree d is selected with probability roughly proportional to $d^{-\beta}$ (Clauset et al., 2009). Networks with a power-law degree distribution are also called scale-free networks. A significant number of works are devoted to studying similarities in power-law graphs and analyzing the structure of such networks to infer knowledge about an individual or group (Mitzenmacher, 2004). However, there has been little work on how to exploit these properties for algorithmic problems. As pointed by Ferrante et al. (2008), there is empirical evidence that solving combinatorial optimization problems in power-law graphs can be tractable compared to the general case. Nevertheless, for specific values of β, many classical NP-hard problems remain NP-hard.

In this thesis, we attempt to determine how the type of the network influences the performance of exact methods when solving the influence maximization problem or variants. We address this problem by optimality with a particular focus on power-law graphs. One important method to solve combinatorial problems by optimality is mathematical programming. The benefit we have by designing exact algorithms employing mathematical optimization is that it often results in a deeper understanding of the structure of the problem. Besides, it provides a rigorous mathematical basis on which to study new algorithmic approaches. Furthermore, the discrete optimization domain has several methods and algorithms that can lead to the optimal solution of a combinatorial problem. We can mention, for example, branch-and-bound, column generation, constraint generation, lagrangian relaxation, and others.

1.1 RESEARCH QUESTIONS

This study aims to contribute to the current knowledge of exact algorithms for influence propagation problems and provide directions in reducing the computational effort to solve this class of problems in social networks with power-law degree distribution.

In summary, some of the questions that guide this research are:

i) What are the implications of the degree distribution and graph sparsity to the design of exact algorithms for the influence maximization problem and its variants?

ii) The power-law exponent β determines the graph density so that a small β implies a more dense graph. Considering this, how can the value of the power-law exponent β influence the formulation of algorithms for these graphs?

iii) How can specific characteristics of these problems contribute to design faster exact algorithms or to strengthen existing formulations?

1.2 CONTRIBUTIONS

We investigate the literature on mathematical programming methods to understand the knowledge frontiers on exact methods for influence propagation. Besides, we examine the algorithmic complexity of combinatorial optimization problems in power-law graphs. Thus we come with at least three useful strategies that can be combined:

- Firstly, we introduce a strategy to select some promising vertices in advance, in order to reduce the running time of the Influence Maximization problem.

The selection process explores some properties of power-law graphs and the relationship between social influence and degree distribution. It prevents, in this way, unnecessary processing by cutting out some vertices from the search. The solutions found using the preselection in the computational experiments preserve the quality of the baseline. Nevertheless, it has the limitation of not having a theoretical guarantee of optimality. However, as it provides gains in running time, it is a beneficial heuristic preselection method to accelerate other techniques. This contribution is entitled "*A Preselection Algorithm for the Influence Maximization Problem on Power Law Graphs*" (Melo and Vignatti, 2018) and published on the 33rd ACM/SIGAPP Symposium On Applied Computing 2018. We discuss it in more detail in Chapter 5.

- In the same sense as the item before, we propose a set of preprocessing rules for the Target Set Selection problem when the input graph is a complex network. In this problem, we want to find a minimum set of individuals to spread information across an entire network. The preprocessing rules are based on the idea of removing some arcs and vertices to construct a partial solution in advance through logical implications, remaining only a small part of the problem to be solved using standard techniques. This strategy has proved to be effective and advantageous on graphs with power-law degree distribution, such as several real-world complex networks. The preprocessing algorithm can be applied with any other technique, being exact or heuristic algorithms, because it guarantees to preserve the optimality. These results were published on a paper entitled "*Preprocessing Rules for Target Set Selection*" (Melo and Vignatti, 2020) on the IX Brazilian Workshop on Social Network Analysis and Mining (BraSNAM 2020) and is explained in Chapter 6.

- Next, we present a new algorithm to compute dual bounds for the Least Cost Influence Problem. Instead of searching for individuals to start the propagation, this problem consists of offering incentives to trigger a cascade that spreads to a given fraction of the network. The dual bound algorithm is based on particular properties of the problem and seeks faster solutions with an optimality guarantee. The principle is to take advantage of the structure of sub-problems and prune the branches in a branch-and-bound scheme. Our algorithm works well for general cases and finds a lower bound tighter than the LP-relaxation in linear time in the size of the graph. The idea is to exploit the connectivity properties of sub-graphs of the input graph associated with each node of the branch-and-bound tree and use it to increase the lower bound of each sub-problem. Computational experiments with synthetic graphs and real-world social networks show the benefit of using our proposed bounds. The benefits are gains in running time or gap reduction for exact solutions to the problem. Preliminary results were published on a paper entitled "*Tighter Dual Bounds on the Least Cost Influence Problem*" (de Melo et al., 2020) on the 52nd Brazilian Operational Research Symposium and is explained in Chapter 7. In this thesis, we present an expanded version of the findings of the conference paper.

Each of the items mentioned above approaches different aspects of the structure of the problem or the topology of the input network. These findings are closely related,

so they can be applied together to compose a general algorithmic framework for these problems. For example, the combinatorial dual bound introduced in Chapter 7 makes use of preprocessing rules of Chapter 6 to solve the problem in real-world social networks.

The rest of this thesis is organized as follows. Chapter 2 contains a brief overview of combinatorial optimization concepts as well as an introduction to the basic exact algorithms for integer linear programming. Then, in Chapter 3, a part is devoted to state the problems and the most basic diffusion models. Also, we give the definitions of power-law distribution and random power-law graphs models. In Chapter 4, we present the literature review about the integer linear programming formulation of the influence maximization problem and its variants. We also present several studies about the complexity of combinatorial problems in power-law graphs. Chapters 5, 6, and 7 are dedicated to addressing the preselection algorithm, the preprocessing rules, and the dual bound algorithm mentioned above, respectively. We conclude, in Chapter 8, with a discussion about the results in general and open questions.

2 MATHEMATICAL OPTIMIZATION

This chapter presents a short review of mathematical optimization and some topics of linear programming. We proceed to cover the linear programming based enumeration algorithms for solving integer linear programs. Further, by making use of the cutting plane concept, we extend the branch-and-bound into the branch-and-cut method for solving difficult combinatorial problems. We assume in this thesis that the reader is somewhat familiar with optimization theory, so the goal is to set up the adopted terminology. We intend to use the more common terminology. So the reader can feel free to skip this chapter and back when encountering unfamiliar concepts in the document. For a more thorough discussion on the subject, textbooks (Boyd and Vandenberghe, 2004; Cornuéjols, 2008; Mitchell, 2002; Papadimitriou and Steiglitz, 1998; Schrijver, 1998, 2003) are our suggestions, but there exists a vast amount of literature on the topic.

As defined by Boyd and Vandenberghe (2004), a *mathematical optimization problem* is a problem of maximization (or minimization) that has the form

$$\begin{aligned} \text{Maximize} \quad & f_0(x), \\ \text{subject to} \quad & f_i(x) \leq b_i, i = 1, ..., m. \end{aligned}$$

where the n-vector x contains the **decision variables** of the problem, the function $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the **objective** function, the functions $f_i : \mathbb{R}^n \to \mathbb{R}$ are the **constraints** function. If for any entry x with $f_i(x^*) \leq b_i$ and $f_i(x) \leq b_i$, we have $f_0(x^*) \geq f_0(x)$ then the vector x^* is said to be the **optimal solution** of the problem. That is, it has the greatest objective value among all solutions abiding the constraints.

Generally, optimization problems are characterized by the form of the objective and constraint functions. This thesis focuses on two important classes of optimization problems called *Linear Program* (LP) and *Integer Linear Program* (ILP), in which the constraints and objective function are linear functions. LP problems contain continuous variables, while ILP problems have discrete variables.

2.1 LINEAR PROGRAMMING

In matrix notation, a linear programming model is an optimization problem that takes as input an $m \times n$ matrix A and column vectors $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^n$. The goal is to find a vector $\mathbf{x} \in \mathbb{R}^n$ that maximizes (or minimizes) $\mathbf{c}^T\mathbf{x}$ satisfying $A\mathbf{x} \leq \mathbf{b}$ (or $A\mathbf{x} \geq \mathbf{b}$). Rewriting this problem, we have the following formulation

$$\begin{aligned} \text{Maximize} \quad & \mathbf{c}^T\mathbf{x}, && (2.1) \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, && (2.2) \\ & \mathbf{x} \geq \mathbf{0}. && (2.3) \end{aligned}$$

Explaining shortly, the components of the problem above are named as follows. The objective function in Equation 2.1 describes the value to be maximized. It contains the decision variables $\mathbf{x} = (x_1, x_2, ..., x_n)$ (the values to be determined) and the corresponding *cost coefficients* $\mathbf{c} = (c_1, c_2, ..., c_n)$. Inequalities 2.2 and 2.3 describe the constraints, which any valid solution to the problem must respect. A

feasible solution (or feasible point) is a combination of variable values. The set $X = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq b \text{ and } \mathbf{x} \geq 0\}$ of all feasible points satisfying the constraints defines the feasible region for the problem. We call **infeasible** a problem in which the set of solutions is empty, and **unbounded** if there is no maximal (or minimal) $\mathbf{x}$, in other words, if occurs $\mathbf{x}$ such that $r < \mathbf{c}^T\mathbf{x}$ for all $r \in \mathbb{R}$. One equivalent form often considered when discussing algorithms for linear programming is $\max\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$.

Linear programming problems are known to be solvable in polynomial time. In practice, such problems are solvable by the *Simplex* method. However, the worst-case time complexity of Simplex is exponential. There are other algorithms with polynomial time complexity, for example, *Interior Points* and *Ellipsoid* methods. The description of the algorithms to these problems is beyond the scope of this text.

2.1.1 Duality theory

There are many interesting theoretical aspects in linear programming, especially concerning to combinatorial problems. All of these aspects are related directly or indirectly to the linear programming dual problem.

Definition 1 (Dual Problem) *Denote as **primal** the linear program (2.1)-(2.3). The **dual** to the this problem is defined as*

$$\begin{aligned} &\textit{Minimize} && \mathbf{b}^T\mathbf{y}, && (2.4)\\ &\textit{subject to} && A^T\mathbf{y} \geq \mathbf{c}, && (2.5)\\ & && \mathbf{y} \geq \mathbf{0}. && (2.6) \end{aligned}$$

If matrix $A \in \mathbb{R}^{m \times n}$ is in the primal space, then we have $\mathbf{y} \in \mathbb{R}^m$ as dual variables. For both primal and dual, there are three possible outcomes: no feasible solution exists, feasible unbounded solutions, and finite optimum. Some important results are the following two theorems, the proofs are skipped here but can be found in (Schrijver, 1998).

Theorem 1 (Weak Duality) *If $\mathbf{x}$ is a feasible solution to the primal problem and $\mathbf{y}$ is a feasible solution to the dual problem, then $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}$.*

Theorem 2 (Strong Duality) *If the primal problem has a feasible solution $\mathbf{x}^*$ with finite optimal objective value, its dual has a feasible solution $\mathbf{y}^*$ with the same objective value, that is, $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$.*

From the weak duality, we have that $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}$. So the dual provides an upper bound for the value of any primal solution. It implies that if the primal is unbounded, then the dual must be infeasible. The strong duality says that it is possible to prove the optimality of a primal solution $\mathbf{x}$ by exhibiting an optimal dual solution $\mathbf{y}$ (Schrijver, 2003).

2.2 INTEGER LINEAR PROGRAMMING

Fractional solutions are undesirable in some applications of linear programming. A common use of linear programs is the case when the decision variables are restricted to

integer values. Integer linear programming (ILP) is the optimization problem stated in the general form as

$$\begin{aligned} \text{Maximize} \quad & \mathbf{c}^T\mathbf{x}, && (2.7)\\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, && (2.8)\\ & \mathbf{x} \in \mathbb{Z}_+^n && (2.9) \end{aligned}$$

where the variables x are restricted to be integral.

The feasible region $X = \{\mathbf{x} \in \mathbb{Z}_+^n : A\mathbf{x} \leq \mathbf{b}\}$ of the program above is a subset of the feasible region $S = \{\mathbf{x} \in \mathbb{R}_+^n : A\mathbf{x} \leq \mathbf{b}\}$ of its corresponding linear program. Thus we refer to the set S as the continuous relaxations of the set X.

Definition 2 (Continuous Relaxation) *For an ILP model P, the LP model obtained by dropping the integrality requirements of all decision variables is called the LP relaxation of P.*

Considering that $A, \mathbf{b}$ and $\mathbf{c}$ are the same for both programs (2.1 - 2.3) and (2.7 - 2.9), the first is a linear relaxation of the second one. Let $\hat{\mathbf{x}}$ be the optimal solution of the linear program in (2.1 - 2.3) and $\mathbf{x}^*$ the optimal solution of the integer linear program in (2.7 - 2.9). Therefore, as we consider a maximization problem, we have $\mathbf{c}^T\mathbf{x}^* \leq \mathbf{c}^T\hat{\mathbf{x}}$. In words, the relaxation provides an upper bound on the optimal value of the integer program.

Apart from pure ILP, a special case of these problems is the Binary Integer Programming (BIP) problem. The decision variables are restricted to accept *zero* or *one* values only. The form of a BIP is

$$\begin{aligned} \text{Maximize} \quad & \mathbf{c}^T\mathbf{x}, && (2.10)\\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, && (2.11)\\ & \mathbf{x} \in \{0,1\}^n. && (2.12) \end{aligned}$$

Another common modeling class is Mixed Integer Linear Programming (MILP). In these models, a subset of the decision variables is allowed to take continuous values, while the remainder is enforced to be integer. Mathematically, a general MILP model can be formulated as:

$$\begin{aligned} \text{Maximize} \quad & \mathbf{c}^T\mathbf{x}, && (2.13)\\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, && (2.14)\\ & \mathbf{x} \geq \mathbf{0}, && (2.15)\\ & x_j \in \mathbb{Z} \quad \text{for } j \in J \subseteq \{1, ..., n\}. && (2.16) \end{aligned}$$

Note that every x_i such that $i \in \{1, ..., n\} \setminus J$ is a continuous variable, then we have a LP when $J = \emptyset$ and a pure ILP when $J = \{1, ..., n\}$.

In contrast to the complexity of linear programming problems, ILPs are generally NP-hard, then there exists no known algorithm for solving it in polynomial time. For the next section, we pay attention to methods for obtaining optimal solutions for integer programs that use continuous relaxation.

2.3 EXACT ALGORITHMS FOR INTEGER PROGRAMMING

In many practical situations, describing an ILP model with many variables or constraints and explicitly enumerating them may be computationally intractable. Thus, there exist several different approaches to solve ILP models by optimality. We can distinguish three main categories of algorithms. The *enumerative* methods (branch-and-bound and branch-and-cut) are based on smartly enumerate integer solutions to find one optimal. The *cutting plane* algorithms, which are based on polyhedral combinatorics, and the *decomposition* techniques. These exact algorithms are guaranteed to terminate with an optimal solution but need, in general, an exponential number of iterations.

2.3.1 Branch-and-bound method

The branch-and-bound is not a technique limited to integer programming problems and can be applied to different type of problems. In general combinatorial optimization problems, the branch-and-bound approach exploits the following observation:

Remark 1 *Given a feasible region X of a combinatorial optimization problem $P := \max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in X\}$, and given a partition $X_1, ..., X_k \subseteq X$ such that*

$$\bigcup_{i=1}^{k} X_i = X,$$

define $z^i = \max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in X_i\}$ for $i = 1, ..., k$. The optimal solution of P is $\max_{i=1,...,k} z^i$.

In other words, the principle is to split the feasible space into successively smaller subsets so that certain subsets can be evaluated directly by implicit enumeration until the best solution is found. When applied to integer programs, it is used in conjunction with its underlying relaxation.

As a framework to organize the successive subdivisions of X, the method employs a tree structure consisting of nodes and branches in which each node represents a subproblem. To generate the solutions and efficiently explore the feasible region, two problem-specific routines are required, the *branch* and the *bound*.

- **Branching** is the procedure that splits a parent node into smaller subproblems generating child nodes. Here a set of level h is partitioned into t subsets of level $h + 1$.

- **Bounding** is an optimistic estimation of the objective function for the region represented by the node at hand. The goal is to avoid the complete exploration of all subtrees.

In this way, a tree structure is obtained. Some branches are pruned early, and plenty of work is saved.

The branch-and-bound can be implemented in many ways. To understand the method for solving ILP problems, consider the problem $\max\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \mathbf{x} \in \mathbb{Z}^n\}$ and its relaxation $\max\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \mathbf{x} \in \mathbb{R}^n\}$ in which the latter can be solved by the Simplex method. In the branch-and-bound tree, each node has two assigned bounds, an *upper bound* and a *lower bound*. The bounding operation always finds an upper bound $\bar{z}$ such that $\mathbf{c}^T\mathbf{x} \leq \bar{z}$ which, in this case, is the solution of the LP

relaxation; a lower bound $\underline{z}$ in which $\mathbf{c}^T\mathbf{x} \geq \underline{z}$ called the *incumbent* integer solution, refers to the best known feasible solution; and sometimes finds the optimum solution, that is, $\underline{z} = \mathbf{c}^T\mathbf{x} = \bar{z}$. In this way, the feasible solutions have been narrowed to values between $\bar{z}$ and $\underline{z}$. We must determine the best solution within these bounds. Then we can create subsets from the present relaxed solution and use the branching operation to split a node into smaller subproblems.

Algorithm 1 summarizes the steps for determining an optimal integer solution for a maximization problem. The notation is:

- P_0: original optimization problem.
- $\bar{z}$: upper bound, defined here as the objective value of the continuous relaxation.
- $\mathcal{L}$: represents the list of *active* nodes. Every node contains a problem (P_i), where P_i is the sub problem (or feasible region).
- $\hat{\mathbf{x}}^i$: solution of the continuous relaxation of P_i.
- $\mathbf{x}$: the best feasible integer solution found so far (incumbent solution).
- $\underline{z}$: lower bound (value of $\mathbf{c}^T\mathbf{x}$).

At the beginning of Algorithm 1, we save the original problem in the list of active nodes and set the lower bound to $-\infty$. While there are active nodes in the list, at line 4, we take a node P_i for partitioning the subset into smaller subsets. After that, we solve the relaxed model and bound the optimum for P_i in line 6. If P_i's relaxed solution is integral and any of the new lower bounds are higher than $\underline{z}$, then we set $\underline{z}$ to the maximum of these lower bounds (line 10). The effectiveness of the branch-and-bound algorithm relies on its ability to skip the unnecessary subproblems, so the criteria for pruning subproblems are:

i) **Pruning by optimality**: those nodes in which we found the optimum do not need to be expanded further (line 12).

ii) **Pruning by infeasibility**: when the continuous relaxation of a problem P_i is infeasible (line 14), the problem cannot be partitioned.

iii) **Pruning by bound**: if any node in the list has $\bar{z} \leq \underline{z}$, none of the solution of this branch are interesting. So, this node can be thrown out, as done in line 16.

If a node is not finished yet, the optimum of the linear relaxation did not fulfill the integer constraints, so we must continue branching (lines 17-21). For some $j \in \{1, ..., n\}$, choose a basic variable $w = \hat{x}_j^i$ not integer. Thus, x_j will be the branching variable. In lines 19 and 20, two subproblems are created with new constraints ($x_j \leq \lfloor w \rfloor$ and $x_j \geq \lceil w \rceil$) to eliminate the fractional part of the solution. We add each new constraint to an ILP model, which will then be solved normally. Then we solve the models

$$\begin{aligned}
&\text{Maximize} && \mathbf{c}^T\mathbf{x} && && (2.17)\\
&\text{Subject to} && A\mathbf{x} \geq \mathbf{b} && && (2.18)\\
& && \mathbf{x} \geq \mathbf{0} && && (2.19)\\
& && x_j \leq \lfloor w \rfloor && \text{for some } j \in \{1, 2, ..., n\} && (2.20)\\
& && \mathbf{x} \in \mathbb{Z}^n && && (2.21)
\end{aligned}$$

Algorithm 1: A linear programming based Branch and Bound

Input: An integer linear programming problem
Output: An optimal integer solution

$\mathcal{L} \leftarrow \{P_0\}$; // Initialization
$\underline{z} \leftarrow -\infty$
while $\mathcal{L} \neq \emptyset$ **do**
  $P_i \leftarrow$ choose a node from $\mathcal{L}$ and remove it from the list;
  Solve the relaxed model of P_i to get $\hat{\mathbf{x}}^i$, or determine the infeasibility
  $\bar{z}_i \leftarrow \mathbf{c}^T\hat{\mathbf{x}}^i$; // optimistic estimation
  if $\hat{\mathbf{x}}^i \in \mathbb{Z}^n$ **then**
    if $\mathbf{c}^T\hat{\mathbf{x}} > \underline{z}$ **then**
      $\mathbf{x} \leftarrow \hat{\mathbf{x}}^i$; // current best
      $\underline{z} \leftarrow \mathbf{c}^T\hat{\mathbf{x}}^i$; // lower bound
    else
      Prune P_i by *optimality*;
  if *the relaxation of* P_i *is infeasible* **then**
    Prune P_i by *infeasibility*
  if $\bar{z}^i \leq \underline{z}$ **then**
    // P_i cannot have solutions better than $\mathbf{x}$
    Prune P_i by *bound*
  // If none of the three cases above holds, then branch
  if $\hat{\mathbf{x}} \notin \mathbb{Z}^n$ *and* $\mathbf{c}^T\hat{\mathbf{x}} > \underline{z}$ **then**
    Select $j \in \{1, ..., n\}$ such that $\hat{x}^i_j \notin \mathbb{Z}$ and branch on variable x_j
    $P'_i \leftarrow P_i \cap \{x_j \leq \lfloor \hat{x}^i_j \rfloor\}$; // Generate two children of P_i
    $P''_i \leftarrow P_i \cap \{x_j \geq \lceil \hat{x}^i_j \rceil\}$
    /* Put P'_i and P''_i in the list of active nodes */
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{P'_i, P''_i\}$
return *the incumbent solution* $\mathbf{x}$

and

$$\begin{aligned}
&\text{Maximize} && \mathbf{c}^T\mathbf{x} && && (2.22)\\
&\text{Subject to} && A\mathbf{x} \geq \mathbf{b} && && (2.23)\\
& && \mathbf{x} \geq \mathbf{0} && && (2.24)\\
& && x_j \geq \lceil w \rceil && \text{for some } j \in \{1, 2, ..., n\} && (2.25)\\
& && \mathbf{x} \in \mathbb{Z}^n && && (2.26)
\end{aligned}$$

with the appropriate constraints added. These solutions reflect the partitioning of the original relaxed model into two subsets formed by adding the two constraints. In case of feasible problems, the process stops when we have a solution for the original problem with an objective function value, in the case of maximization problem, greater or equal to all upper bounds of the generated subsets. Figure 2.1, shows an illustration of the branching step, in lines 17-21, where we split a problem P_i in two smaller sub-problems P_i' and P_i''.

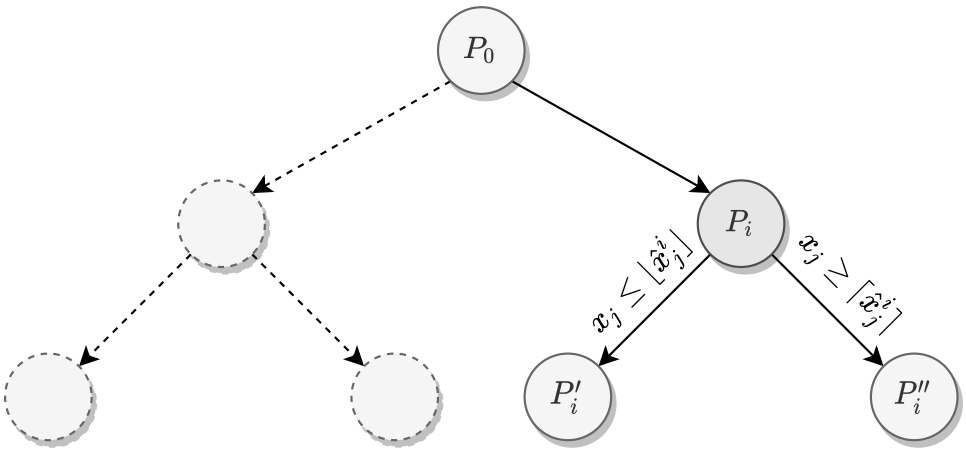


Figure 2.1: A branch-and-bound tree in which the branching decisions are made on a variable x_j. Each node represents a subproblem of the original one P_0. Nodes P_i' and P_i'' represent the sub-problems obtained by lower and upper bounding, respectively, the variable x_j. The factional solution of the continuous relaxation for the variable x_j is represented by $\hat{x}_j$.

The efficiency of the branch-and-bound algorithm depends on how close to the optimal solution are the bounds of the sub problems. In integer linear models, this means how "good" are the relaxed sub problems. To improve the efficiency of this algorithm, we can combine it with a technique called cutting plane to get the branch and cut method.

2.3.2 Cutting plane methods

Consider $X = \{\mathbf{x} \in \mathbb{Z}_+^n : A\mathbf{x} \leq \mathbf{b}\}$ the feasible region of a linear integer program $\max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in X\}$ and let $S = \{\mathbf{x} \in \mathbb{R}_+^n : A\mathbf{x} \leq \mathbf{b}\}$ be the feasible region of its continuous relaxation.

Definition 3 *A **cutting plane** is an inequality $\alpha^T\mathbf{x} \leq \beta$ which is satisfied by all points in X but not by all points in S.*

The principle of cutting plane methods is to add constraints to a (relaxed) linear program until the optimal (relaxed) solution takes on integer values. The Algorithm 2 gives a brief statement of the main points behind cutting plane methods.

Algorithm 2: Generic cutting plane algorithm

Input: An integer linear programming problem P
Output: An optimal integer solution $\mathbf{x}^*$
1 Solve the LP relaxation of P. If the optimum $\hat{\mathbf{x}}$ of the relaxation is an integer solution, stop.
2 Find a cutting plane constraint such that the new constraint removes $\hat{\mathbf{x}}$, but satisfies all integer solutions of P.
3 Insert the cutting plane to the formulation of P, and go to first step.

There are in general many cutting planes that can be chosen to separate $\hat{\mathbf{x}}$ from the integer feasible region at each iteration. The performance of the Algorithm 2 critically depends on the choice of the cutting planes added at each iteration. An algorithm presented by Gomory et al. (1958) was the first finitely terminating cutting plane method for integer programming problems. It can be proved that after adding a finite number of Gomory cuts, we obtain an optimal integer solution. Unfortunately, this method's drawbacks are that it may require many constraints before achieving the optimum integer solution, leading to a slow convergence. The development of the polyhedral theory and the consequent introduction of problem-specific cutting planes led to the resurgence of these approaches. For specialized problems, one can derive cutting planes using polyhedral combinatorics. Thus, the cuts exploit the structure of those particular problems and then cut a larger part of the feasible region.

In the following definitions, we introduce the necessary tools from polyhedral theory and gives a geometric understanding of valid inequalities.

Definition 4 *A **polyhedron** is a set of the form $P := \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq b\}$, where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.*

Definition 5 *A finite collection of vectors $\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^k \in \mathbb{R}^n$ is **affinely independent** if the unique solution to $\sum_{i=1}^{k} \lambda_i \mathbf{x}^i = 0 \wedge \sum_{i=1}^{k} \lambda_i = 0$, is $\lambda_i = 0$ for $i = 1, 2, ..., k$.*

Definition 6 (Dimensions of a Polyhedron) *A polyhedron P has dimension d, denoted by $dim(P) = d$, if the maximum number of affinely independent points in P is $d + 1$.*

Definition 7 (Valid Inequality) *If the inequality $\alpha^T \mathbf{x} \leq \beta$ holds for all $\mathbf{x} \in P$, we call it a valid inequality for P, where P is a polyhedron, $\alpha \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$.*

To improve a given formulation of a particular ILP problem, we can proceed by identifying additional *valid inequalities* for the integer feasible region X, cutting away regions of S that contain no integer solutions, and thus obtaining better formulations for S. As detailed by Cornuéjols (2008), there are several classical families of valid inequalities such as Gomory cuts, rounding cuts, lift-and-project cuts, split cuts, intersection cuts, etc. The difficult relies in constructing inequalities defining X given those defining S.

Definition 8 (Face of a Polyhedron) *If $\alpha^T \mathbf{x} \leq \beta$ is as valid inequality for P and $F := \{\mathbf{x} \in P : \alpha^T \mathbf{x} = \beta\}$, F is said to be a **face** of P.*

Definition 9 *A face F is said to be a **facet** of P if $dim(F) = dim(P) - 1$.*

The inequality corresponding to a *facet* is called a *strong* valid inequality. In order to strengthen formulations of difficult integer programs, these inequalities are dynamically generated using separation procedures.

The search for a cutting plane is called *separation problem*, which means finding a violated valid inequality of the integer linear programming model. Ideally, an efficient exact method to solve the separation problem is desired. Unfortunately, such methods usually are computationally expensive, and heuristic procedures are employed. Formally, the problem of generating a cutting plane can be stated as follows.

Definition 10 (Separation Problem) *Given a polyhedron $P \subseteq \mathbb{R}^n$ and a solution $\hat{\mathbf{x}} \in \mathbb{R}^n$, determine whether $\hat{\mathbf{x}} \in P$ and if not, determine an inequality valid for P such that $\alpha^T \hat{\mathbf{x}} > \beta$.*

The following pseudo code presents a cutting plane algorithm in terms of the separation problem defined above.

Algorithm 3: Cutting plane

```
1  Solve the current linear relaxation max{c^T x : x ∈ S} and let x̂ denote an
     optimal solution.
2  if x̂ ∈ Z^n then
3  |  Stop ;                    // x̂ is also an optimal of the ILP
4  else
5  |  Solve the separation problem for x̂ and X;        // Definition 10
6  |  if α^T x ≤ β is found then
   |  |  // Adding the cutting plane
7  |  |  S ← S ∩ {x ∈ R^n : α^T x ≤ β} Go to first step.
8  |  else
9  |  |  Stop
10 |  end
11 end
```

2.3.3 Branch-and-cut

Now that we know about the branching paradigm and cutting planes, we can state the *Branch-and-Cut* method as a combination of the LP based branch-and-bound with cutting plane techniques. In this hybrid algorithm, at each node of the branch and bound tree, valid inequalities are dynamically generated by solving the associated separation problem. It follows the definition:

Definition 11 *Branch-and-cut is a branch-and-bound scheme in which, at each node of the enumeration tree, cutting planes are generated and used to improve the linear programming relaxation.*

Algorithm 4 presents a simple branch-and-cut algorithm. The separation phase, at lines 7-12, is the central part. Where we try to find violated valid constraints and then adding it to the problem. The rest of the pseudo-code is the branch-and-bound scheme. This procedure aims to get an integer optimal solution from the continuous

Algorithm 4: Branch-and-cut

Input: An integer linear programming problem
Output: An optimal integer solution

$\mathcal{L} \leftarrow \{P_0\}$; // Initialization
$\underline{z} \leftarrow -\infty$
while $\mathcal{L} \neq \emptyset$ **do**
  $P_i \leftarrow$ choose a node from $\mathcal{L}$ and remove it
  Solve the relaxed model of P_i to get $\hat{\mathbf{x}}^i$, or determine the infeasibility
  $\bar{z}_i \leftarrow \mathbf{c}^T\hat{\mathbf{x}}^i$; // optimistic estimation
  Solve the separation problem for $\hat{\mathbf{x}}^i$ and P_i; // Definition 10
  if $\alpha^T\mathbf{x} \leq \beta$ *is found* **then**
    $P_i \leftarrow P_i \cap \{\mathbf{x} : \alpha^T\mathbf{x} \leq \beta\}$; // Adding the cutting plane
    $\hat{\mathbf{x}}^i \leftarrow$ relaxation of P_i
    $\bar{z}_i \leftarrow \mathbf{c}^T\hat{\mathbf{x}}$
    Go to step 7
  if $\hat{\mathbf{x}}^i \in \mathbb{Z}^n$ **then**
    if $\mathbf{c}^T\hat{\mathbf{x}} > \underline{z}$ **then**
      $\mathbf{x} \leftarrow \hat{\mathbf{x}}^i$; // current best
      $\underline{z} \leftarrow \mathbf{c}^T\hat{\mathbf{x}}^i$; // lower bound
    else
      Prune P_i by *optimality*;
  if *the relaxation of* P_i *is infeasible* **then**
    Prune P_i by *infeasibility*
  if $\bar{z}^i \leq \underline{z}$ **then**
    // P_i cannot have solutions better than $\mathbf{x}$
    Prune P_i by *bound*
  // If none of three cases above holds, then branch
  if $\hat{\boldsymbol{x}} \notin \mathbb{Z}^n$ *and* $\mathbf{c}^T\hat{\mathbf{x}} > \underline{z}$ **then**
    Select $j \in \{1, ..., n\}$ such that $\hat{x}^i_j \notin \mathbb{Z}$ and branch on variable x_j
    $P'_i \leftarrow P_i \cap \{x_j \leq \lfloor \hat{x}^i_j \rfloor\}$; // Generate two children of P_i
    $P''_i \leftarrow P_i \cap \{x_j \geq \lceil \hat{x}^i_j \rceil\}$
    /* Put P'_i and P''_i in the list of active nodes */
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{P'_i, P''_i\}$
return *the incumbent solution* $\mathbf{x}$

relaxation by seeking better lower bounds for a more effective pruning. The addition of cuts can yield significant performance improvements because it reduces the number of branches required to solve the integer problem.

3 SOCIAL INFLUENCE AND POWER-LAW GRAPHS

In social network analysis, when the individuals of a social system make decisions based on other individuals' actions rather than on their own knowledge about a topic, we say that this describes a *cascade* (Easley and Kleinberg, 2010). There are models of cascade based in decision-making, normally using game theory and probabilistic models, that look at the individual's susceptibility to being part of the cascade. In this chapter, we present some definitions and properties of diffusion of influence and detailed problem definitions for the problems considered here. Also, in Section 3.6, we provide a formal definition of the power-law distributions and present a theoretical model for random power-law graphs.

3.1 NOTATION

For a directed graph G with vertices $V(G)$ and edges (or arcs) $E(G)$, consider the following notation. For vertices $u, v, w \in V(G)$:

- $N^+(v) = \{w : (v, w) \in E(G)\}$ the *out-neighborhood* of a vertex v;
- $N^-(v) = \{u : (u, v) \in E(G)\}$ is the *in-neighborhood* of v;
- If $A \subseteq V(G)$, then $N^+(A) = \bigcup_{x \in A} N^+(x) \setminus A$ and $N^-(A) = \bigcup_{x \in A} N^-(x) \setminus A$, are respectively the out-neighbors and in-neighbors of the set A.
- The *out-degree* of v is the number $\delta^+(v) = |N^+(v)|$;
- The *in-degree* of v is $\delta^-(v) = |N^-(v)|$;
- $\delta(v) = \delta^+(v) + \delta^-(v)$ is the *degree* of v;
- We say that a vertex v with $\delta^-(v) = 0$ is a *source*;
- When $\delta^+(v) = 0$ we say that v is a *sink*;
- Also, v is a *isolated* vertex when $\delta(v) = 0$;
- A directed graph H is a *subgraph* of G, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$;
- H is an *induced* subgraph of G, if every arc $(u, v) \in E(G)$ for which $u, v \in V(H)$, is also an arc of H;
- A directed graph G is called *acyclic*, or a DAG (Directed Acyclic Graph), if it has no directed cycles.

In this text, we refer to $V(G)$ and $E(G)$ as the vertices and edges of G, mainly when there is more than one graph to consider in the context. When it is explicit what graph we are referring to, we use only V and E to the same sets. Also, the terms *arcs* and *edges* are used as synonym since we are dealing with directed graphs.

3.2 DIFFUSION MODELS

The influence maximization problem depends on theoretical diffusion models for the formal definition, so it follows a brief description of the basic mathematical models used to represent an information spread. To describe these models, we consider a social network defined as a graph $G = (V, E)$, where V is a set of individuals and E is the set of the relationships between these individuals. As is done in (Kempe et al., 2003; Kleinberg, 2007), the behaviors modeled here are *progressive*, that is, each vertex can assume one of two states, *active* or *inactive*, and can change from inactive to active, but not from active to inactive. Informally, a vertex is active if it has been persuaded to adopt a new behavior (for instance, convinced to buy a new product), and inactive otherwise. At time $t = 0$, a subset S of V is chosen to be active (the set of early adopters). In this process, active vertices tend to activate others. Thus, when a vertex v becomes active because of S, we say that v has been influenced by S. From now onward we only consider *directed* graphs, such that for two nodes v and w the influence of v to w is different from the influence of w to v. We state explicitly if we talk about undirected graphs.

3.2.1 Linear threshold model

Abbreviated as LT, in the linear threshold model each vertex w has a threshold $t(w)$ chosen uniformly at random from the interval $[0, 1]$, where $t(w)$ indicates the fraction of w's neighbors which should adopt the behavior before w. The strength of influence on the edges are defined such that each edge (v, w) has a non-negative weight $b_{v,w}$ indicating the influence of v on w, satisfying

$$\sum_{v \in N(w)} b_{v,w} \leq 1,$$

where $N(w)$ denotes the set of w's incoming neighbors. Furthermore, the vertices in S begin the process as active and all the others as inactive. Time progresses in discrete steps $t = 1, 2, ...$ At time t, an inactive vertex w becomes active if their fraction of active neighbors, denoted by $N^a(w)$, exceeds its threshold $t(w)$, that is,

$$\sum_{v \in N^a(w)} b_{v,w} \geq t(w).$$

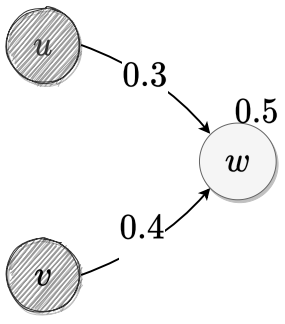


Figure 3.1: Activating w in the LT model. Vertices u and v are active, and vertex w is about to decide whether will be activated or not.

Figure 3.1 illustrates how this process evolves. Supposing that w has threshold $t(w) = 0.5$, since

$$\sum_{x \in N^a(w)} b_{x,w} = 0.3 + 0.4 \geq t(w),$$

then w is activated in the next step. Note that different thresholds in the vertices implies different predisposition to becomes active. That is, a low $t(w)$ means less resistance to w adopts a new behavior, while a high $t(w)$ needs more active neighbors influencing w. The process runs until no more vertex can update from inactive to active.

3.2.2 Independent cascade model

Abbreviated as IC, in the independent cascade model each edge has an activation probability and the influence spreads through active vertices. Each active vertex can activate independently its inactive neighbors based on the probability on the edges (Chen et al., 2009). As well as in the LT model, the adoption process starts from a set S of active nodes and unfolds into discrete time steps. When the vertex v becomes active in step t, it has a chance to activate each inactive neighbor w, with a probability $p_{v,w}$ of success. If v succeeds, then w is activated in step $t + 1$, but if v is not successful it cannot try to activate w in subsequent rounds (Kempe et al., 2003). Again, the activation process ends when there are no more vertices to be activated.

For example, in the Figure 3.2 w is activated by u and v with probabilities $p_{u,w} = 0.3$ and $p_{v,w} = 0.4$, respectively. Suppose that u and v were activated at time t. Thus at time $t + 1$, u and v can independently activate w and, therefore, w is activated with probability

$$p_{u,w} + p_{v,w} - p_{u,w} \cdot p_{v,w},$$

that is,

$$0.3 + 0.4 - (0.3 \cdot 0.4) = 0.58.$$

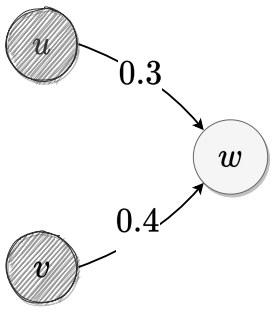


Figure 3.2: Activating w in the IC model. Vertices u and v are active, and vertex w is about to decide whether will be activated or not.

It is worth observing that both the independent cascade and linear threshold models are probabilistic. Thus we can think that the spread in these models is a stochastic process result.

3.3 INFLUENCE MAXIMIZATION PROBLEM

Now we can give a formal definition of the key combinatorial problem of social networks considered in this proposal.

Problem 1 (Influence Maximization) *Given a directed weighted graph $G = (V, E)$, an influence model m and an integer $1 \leq k \leq |V|$. Find a subset $S^* \subseteq V$ such that $\sigma(S^*) = \max_{S \subseteq V}\{\sigma(S)\}$ and $|S| \leq k$.*

The function $\sigma : 2^V \to \mathbb{R}$ to be maximized is called *influence function*, where its value depends on the influence model m, such as IC or LT. Thus, given a set $S \subseteq V$ of early adopters, $\sigma(S)$ denotes the expected number of active vertices at the end of the activation process starting from S (Goyal et al., 2011a; Kempe et al., 2003; Kleinberg, 2007). Expressing this function as a counting problem, it follows the definition of the expected propagation:

Problem 2 (Expected Propagation ($\sigma(S)$)) *Given a directed weighted graph $G = (V, E)$, an influence model m and a set $S \subseteq V$. Find the expected number of vertices activated from S at the end of the activation process.*

The Influence Maximization problem is NP-hard both as IC to LT models (Kempe et al., 2003). To get an approximation guarantee, Kempe et al. (2003) have shown that the σ function is submodular and monotone for IC and LT. Furthermore, computing the exact value of $\sigma(S)$ is a #P-hard problem in both models discussed above (Chen et al., 2010a,b).

3.3.1 Properties of the influence function

Let X be a finite set and $f : 2^X \to \mathbb{R}^+$ an arbitrary function. Intuitively, f is submodular if adding a new element to a set T provides no more gains than adding the same element to a smaller subset S of T, as defined in Definition 12.

Definition 12 *For a finite set X, the function $f : 2^X \to \mathbb{R}^+$ is **submodular** if for any two subsets $S \subseteq T \subseteq X$ we have*

$$f(S \cup \{w\}) - f(S) \geq f(T \cup \{w\}) - f(T)$$

for all $w \in X \setminus T$.

The same function f is monotone when its marginal gain is non-decreasing. Formally, the definition is:

Definition 13 *For a finite set X, the function $f : 2^X \to \mathbb{R}^+$ is **monotone** if $f(S) \leq f(T)$ for any two subsets $S \subseteq T \subseteq X$.*

In terms of the expected propagation function σ, suppose we want to add a new vertex w to a set S of early adopters to increase the value of $\sigma(S)$. The **marginal gain** of w is the difference $\sigma(S \cup \{w\}) - \sigma(S)$, Figure 3.3 illustrates this idea. In this context, the submodularity says that the marginal gain of a new vertex decreases as S grows. In the same sense, the monotone property means that adding w to S do not decreases $\sigma(S)$.

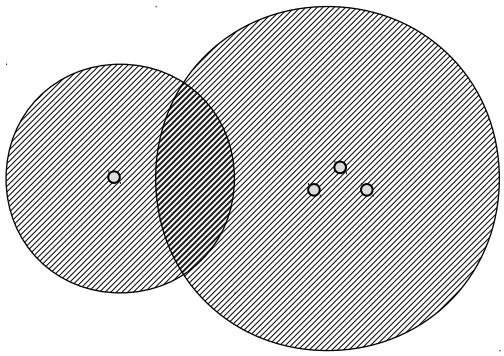


Figure 3.3: Suppose that the three points inside the bigger cycle are the initial adopters and the gray cycles represent the reach of its influence. The part of the smaller cycle not intersecting the bigger is the marginal gain of the new element.

3.3.2 Greedy $(1 - 1/e)$-approximation algorithm

Due to these properties, a greedy algorithm that iteratively chooses the vertex with the greatest marginal gain can performs "well". Algorithm 5 shows the pseudo-code. Let S be the set chosen by the greedy algorithm and let S^* be the set of larger influence among all the possible sets of size k in V. Algorithm 5 ensures that

$$\sigma(S) \geq (1 - 1/e) \cdot \sigma(S^*).$$

Nemhauser et al. (1978) have proved this approximation ratio. Further, this result holds for every submodular and monotone maximization function. However, two major sources of inefficiency affect this algorithm. First, the processing time of the $\sigma(S)$ function is too high since to get the exact value of σ is a #P-hard problem on LT and IC models. In these probabilistic diffusion models, one way to get an estimation of the σ value is by means of Monte Carlo simulations, but this method is very expensive computationally. The second shortcoming is that the algorithm makes many calls to the estimation of the σ value.

Algorithm 5: Greedy

Input: $G = (V, E), k \in \mathbb{N}, \sigma$
Output: Seed set S

```
1 begin
2     S = ∅
3     while |S| ≤ k do
4         u = arg max_{w∈V\S} {σ(S ∪ {w}) − σ(S)}
5         S ← S ∪ {u}
```

After the rise of the Influence Maximization problem, many variations and generalizations of this problem emerged in the literature — each variant with a different application that contributes to understanding several aspects regarding the theme. Below we define two more problems related to the propagation of influence. The problems are the Target Set Selection and the Least Cost Influence Problem.

3.4 TARGET SET SELECTION PROBLEM

For this section and the next one, we consider deterministic settings for the diffusion model. Thus, we present a variation of the linear threshold model. We consider this model because, in this problem, we want to concentrate only on the problem of finding the early adopters, and we do not want to worry about the σ function of the probabilistic models.

3.4.1 Threshold model

To represent how the influence spreads over a network, in this section, we consider the *threshold model*, introduced by Granovetter (1978). Unlike the above mentioned models, LT and IC, this model is deterministic.

We are given a directed graph G representing a social network. Let $t : V(G) \to \mathbb{N}$ be a threshold function that models the "resistance" of an individual to become influenced. As before, each vertex is in one of two states, active or inactive. For a while, we assume that each vertex exerts the same influence over each neighbor. So, a vertex v gets active if at least $t(v)$ of its in-neighbors are active at the previous step. More precisely, let A_t be the set of active vertices at the time step t with $S = A_0$. We have $v \in A_t$ if

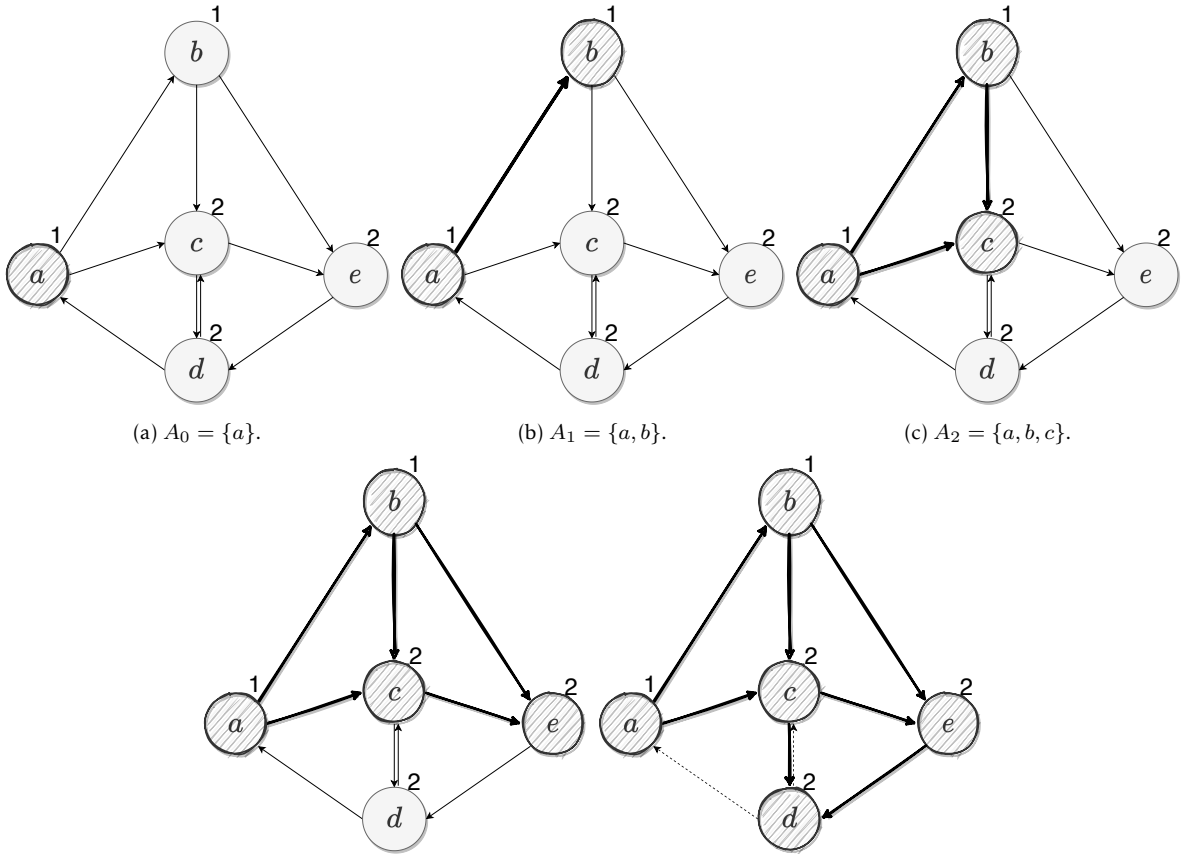
$$|N^-(v) \cap A_{t-1}| \geq t(v)$$

or $v \in S$. The process runs until there are no more vertices to be activated. When a vertex v becomes active at time t because of its neighbors that are in A_{t-1}, we say that u exerts influence over v, for every $u \in N^-(v) \cap A_{t-1}$.

Definition 14 (Propagation graph) *A propagation graph G^* is the subgraph of an input graph G with $V(G^*) = V(G)$, induced by a subset $E(G^*) \subseteq E(G)$ of arcs in which the influence is exerted, that is, for every $(u, v) \in E(G^*)$ where u exerts influence over v, at the end of the activation process.*

Figure 6.2 illustrates the activation process in the threshold model step by step. In this example, we are given a directed graph with thresholds on the vertices. The process starts by setting vertex a as active and all the other as inactive. Note that, in Figure 3.4(a), vertex c cannot be activated by a because of its threshold, which is 2, meaning that c needs at least two active incoming neighbors to be activated but, at this moment, it has only one. On the other hand, vertex b has threshold 1, and it will be activated by a in the next step. Next, in Figure 3.4(c), vertex c has two active neighbors, achieving its threshold now, so c is activated. The process continues as a cascade of activation. At the end of the process (in Figure 3.4(e)), the highlighted arcs are the arcs of the propagation graph. Observe that this graph is acyclic due to the temporal aspect of this process, where a vertex cannot activate another vertex that is already active. For example, vertex d cannot activate vertex a, so the arc (d, a) will not be chosen.

The second problem studied in this thesis is a minimization related version of the Influence Maximization problem, which seeks a target set of minimum size ensuring that all the vertices, or at least a given fraction of the network, will be activated. Let $A(S)$ be the set of active vertices at the end of an activation process. Observe that $A(S)$ is different of the influence function $\sigma(S)$, because we have defined $\sigma(S)$ as the expected propagation 2 in probabilistic scenarios. This problem is defined as follows.



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

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

Figure 3.4: Activation process in the threshold model starting from a target set $S = \{a\}$. The number attached to each vertex represents its threshold. Sets A_t contains the active vertices at time t, for $t = 0, ..., 4$. Highlighted arcs indicate who influences whom in the process.

Problem 3 (Target Set Selection (TSS)) *Given an integer ℓ and a directed graph $G = (V(G), E(G))$ with thresholds $t(v)$ on each vertex $v \in V(G)$. Find a target set $S \subseteq V(G)$ of minimum size, such that the propagation graph G^* is acyclic with $\delta^-(v) \geq t(v)$ for every $v \in V(G^*) \setminus S$ and $|A(S)| \geq \ell$.*

3.5 LEAST COST INFLUENCE PROBLEM

In this section, we consider a variation of the threshold model in which we consider external influence through incentive to the individuals of the social network. Furthermore, this influence model considers distinct weights of influence on the edges.

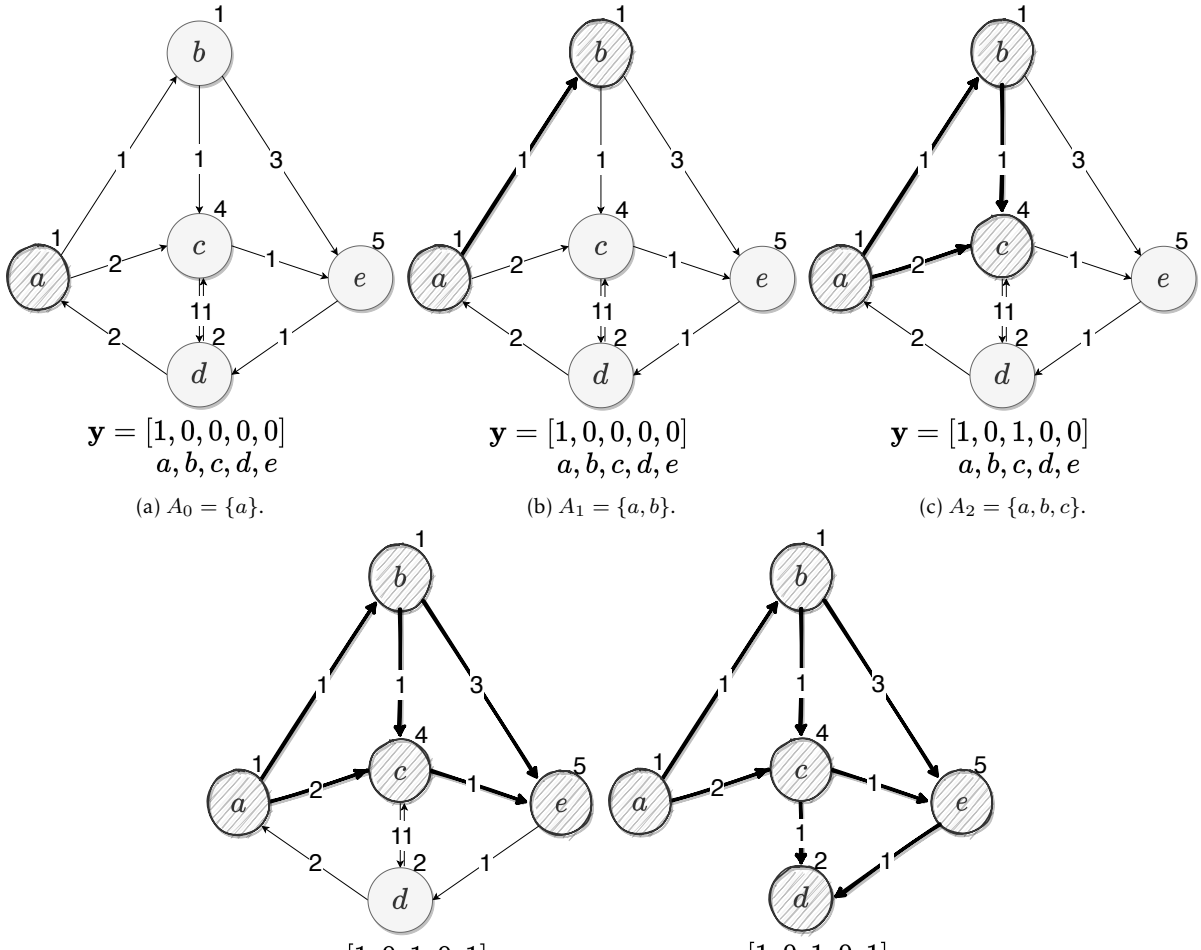
3.5.1 Threshold model with incentives

Let G be a directed graph that models a social network. Each arc $(u, v) \in E$ has an associated weight $d_{uv} > 0$ that models the strength of the influence of u over v. Every $v \in V$ has a threshold $t(v) > 0$ which indicates the amount of influence needed to activate v, coming from v's neighbors. Also, we consider the offer of external influences. These influences, which we call incentives, aim to break the resistance of an individual in becoming influenced in the activation process. The incentives are represented by a vector $\mathbf{y} \in \mathbb{Z}^{|V|}$, where each coordinate $y_v \in \mathbb{N}_0$ denotes the amount of incentive given to a vertex $v \in V$. Applying the incentive y_v on a vertex v decrease its threshold $t(v)$ and make it more susceptible to be activated. This incentive is added with the influence coming from the other vertices. More formally, the initial set of active vertices is given by $A_0 = \{v \in V : y_v \geq t(v)\}$. The vertices in A_0 begin the process as active and all the others as inactive. Time progresses in discrete steps $t = 0, 1, \ldots, k$, an inactive vertex i becomes active at time t if the total influence of its active in-neighbors plus its incentive exceeds the threshold $t(v)$, i.e., if

$$\sum_{u \in N(v) \cap A_{t-1}} d_{uv} \geq t(v) - y_v.$$

Figure 3.5 illustrates the activation process in threshold model with incentives. Again, we have a directed graph with thresholds on the vertices and weights of influence on the arcs. Suppose that we want to activate 100% of the vertices. The process starts by setting the vertex a as active and all the other as inactive. We are paying enough incentive to achieve a's threshold without the influence of the neighbors, so vertex a receives 1 of incentive. We will try to activate the remaining vertices using incentives to decrease its threshold. Vertex b has threshold 1, and it will be activated only by a in the next step, so no incentive is necessary. Next, in Figure 3.5(c), vertex c has two active neighbors in which the weigh of incoming arcs sums to 3. To achieving its threshold, we need to pay an incentive of 1, so c is activated. The process continues as a cascade until all vertices are activated. Therefore we needed to pay a total of 3 of incentives to activate the whole network, i.e. $y_a = y_c = y_e = 1$ and $y_b = y_d = 0$.

The problem consists in offering incentives to the vertices in a way that it will trigger a cascade of influence that spreads to a given fraction α of the network. The goal is to minimize the total of incentives given to the individuals of the social network. The definition for this problem follows.



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

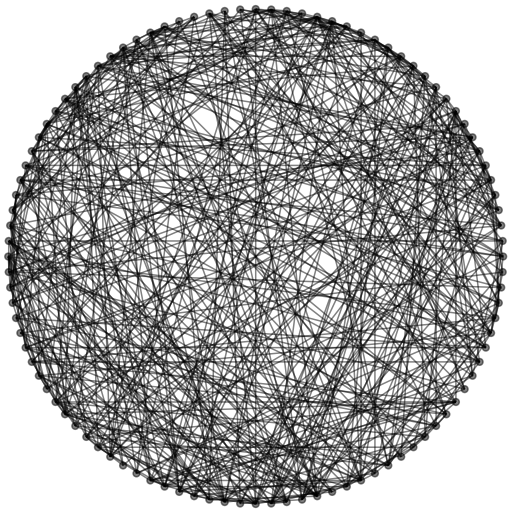
(d) $A_3 = \{a, b, c, e\}$. (e) $A_4 = \{a, b, c, d, e\}$.

Figure 3.5: Activation process in the threshold model considering incentives, starting from a target set $S = \{a\}$. The number attached to each vertex represents its threshold. Sets A_t contains the active vertices at time t, for $t = 0, ..., 4$. Labels in each arc (u, v) denote the weight of influence d_{uv}. Highlighted arcs indicate who influences whom in the process. The vector $\mathbf{y}$ indexed by the vertices contains the amount of incentive offered to each vertex.

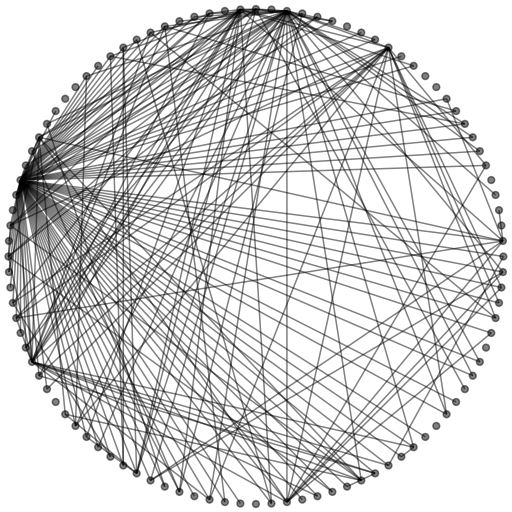
Problem 4 (Least Cost Influence Problem (LCIP)) *We are given a real number $\alpha \in [0, 1]$, a directed graph G with weights d_{uv} on each arc $(u, v) \in E$, and a threshold $t(v)$, for each vertex $v \in V$. The goal is to find a vector* $\mathbf{y}$ *of incentives which minimizes the sum $\sum_{v \in V} y_v$, ensuring that at least $\lceil \alpha |V| \rceil$ vertices are activated by the end of the activation process.*

3.6 POWER-LAW GRAPHS

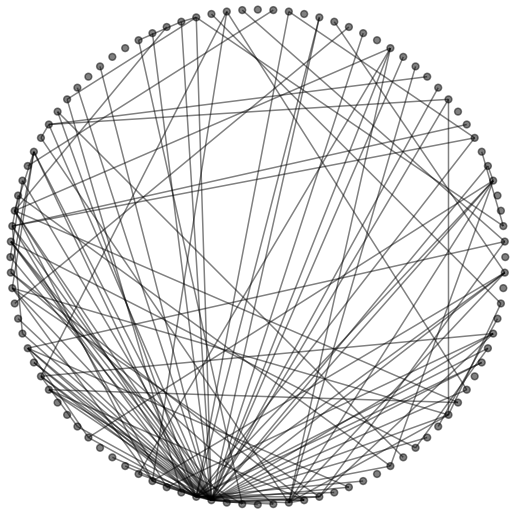
In graph theory, when vertices degree of a graph follow a power-law distribution, we call it *power-law graph*. In these special kind of graphs, the number y of vertices with a given degree d is proportional to $d^{-\beta}$ where the exponent $\beta > 0$ is called **scale parameter** (or power-law exponent). For the majority of large scale real-world power-law graphs, the scale parameter is typically in the range $2 < \beta < 3$ (Chung and Lu, 2002; Clauset et al., 2009).



(a) $G(n, p)$ random graph with 100 vertices and probability $p = 0.1$.



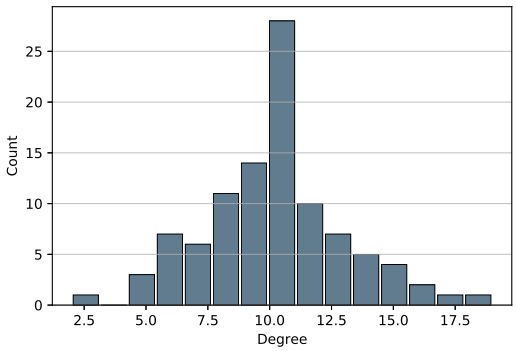
(b) Power-law random graph with 100 vertices and $\beta = 2$



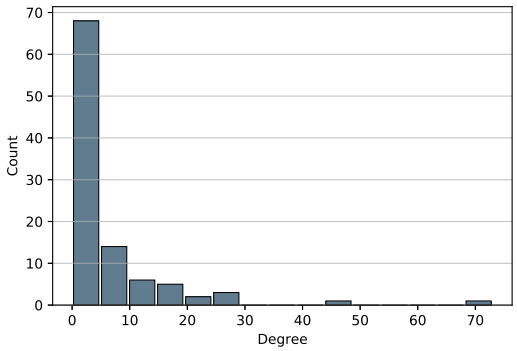
(c) Power-law random graph with 100 vertices and $\beta = 3$

Figure 3.6: Random graphs.

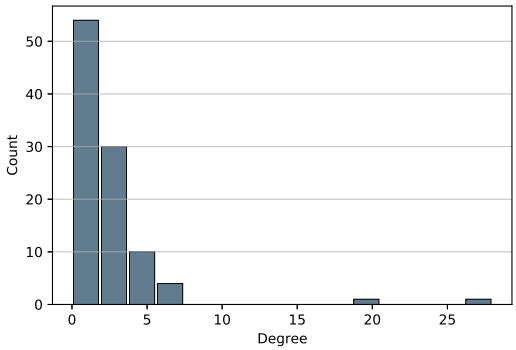
For comparative purposes, Figure 3.6 exhibits three different random graphs. First, in Figure 3.6(a), we have the classical Erdös-Rényi random graph model $G(n, p)$, where n is the number of vertices to be generated, and p is the probability of each edge be included in the graph. Since the degree of each vertex is the sum of $n-1$ independent random variables, the degree distribution is a *binomial*. In Figure 3.6(b), we have a power-law random graph with exponent $\beta = 2$. Observe that this graph is sparse and has the characteristic that few vertices concentrate the vast majority of the edges. At the same time, there are many vertices of low degree. Figure 3.6(c) also is a power-law graph, but with the difference that the exponent β is 3. This graph is sparser than the second one. However, as the degree distribution is the same, few vertices have many edges incident to it, while most vertices have a low degree.



(a) Degree distribution of the $G(n, p)$ random graph in Figure 3.6(a).



(b) Degree distribution of the power-law random graph in Figure 3.6(b)

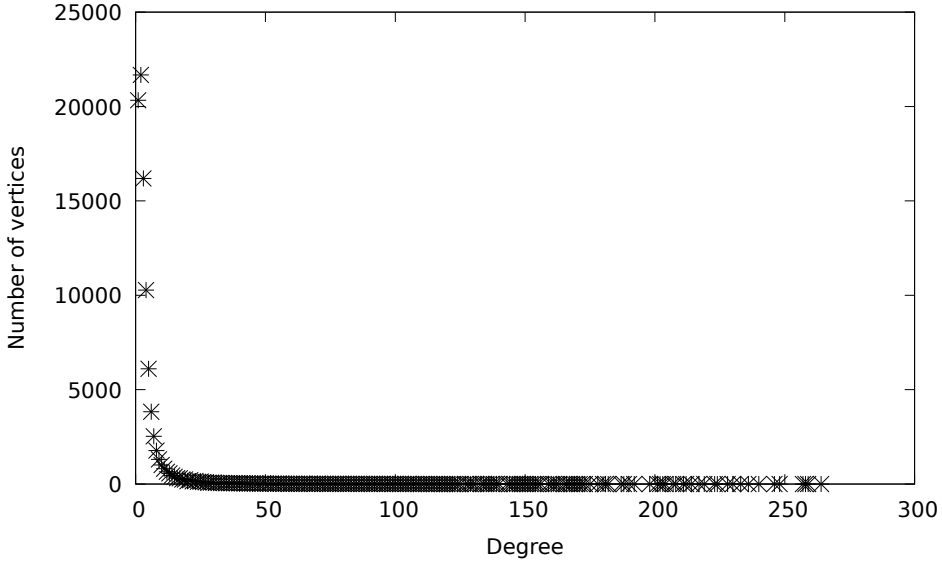


(c) Degree distribution of the power-law random graph in Figure 3.6(c).

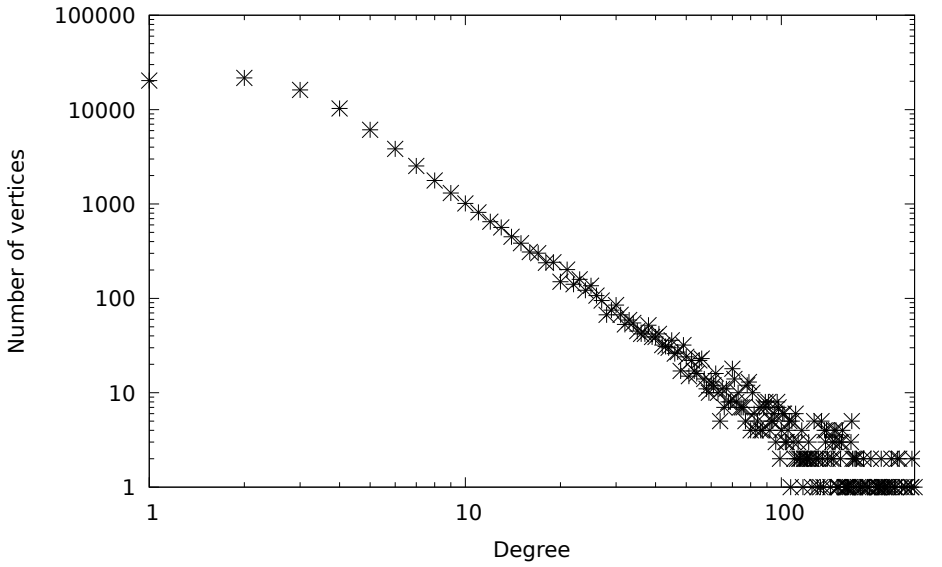
Figure 3.7: Degree histogram of random graphs.

The histograms in Figure 6.2 show the number of vertices of a given degree. In these figures, the differences between the distributions are evident. It is worth remembering that the distribution is best represented when it comes to large-scale graphs. Nevertheless, in these figures, it is possible to observe that Figures 3.7(b) and 3.7(c) show the typical long-tail form of the power-law distribution, having many occurrences far from the "head" of the distribution. On the right side of the histogram is the long-tail, while on the left side are the few vertices of a higher degree.

Given a sample, an empirical test to know whether the degree distribution follows a power-law, is when the behavior of a log-log plot of the function will be asymptotically a strait line. But just because many other distributions produce nearly



(a) Degree histogram



(b) Plot on log-log scale

Figure 3.8: Histogram and log-log scale plot of the degree distribution of a synthetic power-law graph with 91789 vertices and 221053 edges with $\beta = 2.3$.

linear outcomes on a log-log plot, a straight line does not guarantee a power-law distribution. However, when the results are far from a straight line, a power-law is unlikely (Mitzenmacher and Upfal, 2005). Figure 3.8 shows the degree distribution of a power-law graph with scale parameter $\beta = 2.3$. In the first plot (a) we can see the long tail of the degree distribution and that there are few vertices with high degree and many with low degree. The second plot (b), in logarithmic scale, reveals the straight line mentioned above.

Many power-law random graph models have been proposed in the literature. Such models are designed to capture and mimic the empirically observed topology of real-world networks. To have an underlying power-law model for the studies in this thesis, we consider a generative model denoted as $P(\alpha, \beta)$, introduced by Aiello et al. (2001) for random power-law graphs. This model ensures a power-law degree

distribution by fixing the degree sequence. Roughly speaking, the parameters α and β determines the size and density of the resulting graph. Let $\mathcal{G}_{\alpha,\beta}$ denotes the family of all random power-law graphs corresponding to the $P(\alpha,\beta)$ model. To get a random element of this space, the model $P(\alpha,\beta)$ assigns a uniform probability for all graph $G \in \mathcal{G}_{\alpha,\beta}$. Definition 15 describes such graphs (Ferrante et al., 2008).

Definition 15 *Given α and β values, a graph is a $G = (V, E)$ power-law graph of the model $P(\alpha,\beta)$ if its degree sequence is such that the number y_i of vertices with degree i is*

$$|\{v \in V : \delta(v) = i\}| = y_i = \begin{cases} \left\lfloor \dfrac{e^\alpha}{i^\beta} \right\rfloor, & \text{if } i > 0 \text{ or } \sum\limits_{i=1}^{e^{\frac{\alpha}{\beta}}} \left\lfloor \dfrac{e^\alpha}{i^\beta} \right\rfloor \text{ is even} \\ \lfloor e^\alpha \rfloor + 1, & \text{otherwise} \end{cases} \tag{3.1}$$

for all $i = 1, ..., e^{\frac{\alpha}{\beta}}$, where $\delta(v)$ denotes the degree of a vertex v and $e^{\frac{\alpha}{\beta}}$ is the maximum degree.

Some important facts of this model are that it allows us to derive a number of structural properties. For example, the size of the connected components in the generated graphs and the number of vertices can be computed by counting all the vertices with degree i for $i = 1, ..., e^{\frac{\alpha}{\beta}}$ (Aiello et al., 2001), as given by the following equation

$$|V| = \sum_{i=1}^{e^{\frac{\alpha}{\beta}}} y_i \approx \begin{cases} \zeta(\beta)e^\alpha & \text{if } \beta > 1 \\ \alpha e^\alpha & \text{if } \beta = 1 \\ \dfrac{e^{\frac{\alpha}{\beta}}}{1-\beta} & \text{if } 0 < \beta < 1 \end{cases} \tag{3.2}$$

and the edges can be computed as

$$|E| = \frac{1}{2} \sum_{i=1}^{e^{\frac{\alpha}{\beta}}} \left\lfloor \frac{e^\alpha}{i^{\beta-1}} \right\rfloor \approx \begin{cases} \zeta(\beta-1)e^\alpha & \text{if } \beta > 2 \\ \dfrac{1}{4}\alpha e^\alpha & \text{if } \beta = 2 \\ \dfrac{e^{\frac{2\alpha}{\beta}}}{4-2\beta} & \text{if } 0 < \beta < 2 \end{cases} \tag{3.3}$$

where $\zeta(\beta) = \sum\limits_{i=1}^{\infty} \dfrac{1}{i^\beta}$ is the Riemann's Zeta function. In this way, we can use these facts to generate a graph in which the degree sequence respects a power-law distribution. In the rest of this document, we focus on simple graphs with this particular kind of degree sequence.

4 LITERATURE REVIEW

This chapter presents several studies on the influence maximization problem and its variants, emphasizing Integer Linear Programming and the polyhedral structure aspects for exact methods. None of such exact algorithms is known for the influence maximization problem regarding the power-law degree of social networks. Also, we talk about the tractability and hardness of many combinatorial problems in power-law graphs.

4.1 APPROXIMATION AND HEURISTICS

There is a great number of studies in the literature on the various heuristics for influence propagation problems. A proper review of them would require a long detailed written study, so only the most significant published heuristics are mentioned here.

To overcome the inefficiency of Algorithm 5, several works propose improvements and reduction of the computational cost. Two algorithms, CELF (Leskovec et al., 2007) and CELF++ (Goyal et al., 2011b), stand out for providing good results using Monte Carlo simulations. The main idea of the CELF algorithm is that the marginal gain of a vertex at a given iteration cannot be greater than its gain in the previous iterations. The algorithm maintains a list of vertices sorted by the marginal gain in non-increasing order. CELF++ proposes new settings to CELF. The central idea is that if the last selected vertex is still the first on the sorted list, then the marginal gain of such vertex does not need to be recomputed. Arora et al. (2017) explains that besides the Monte Carlo based methods, there are well-known heuristics that use a method called *Score Estimation* to deal with the influence function, for instance SIMPATH (Goyal et al., 2011c) and LDAG (Chen et al., 2010b). Moreover, recent studies show good results using a technique known as *Reverse Reachable* sets (Arora et al., 2017), which has provided algorithms as efficient as the heuristics, but with the plus of having approximation factor guarantee, for example TIM+ (Tang et al., 2014) and IMM (Tang et al., 2015). We can think of algorithms for the influence maximization problem as having two phases, (i) the influence function estimation and (ii) the target set selection, where Monte Carlo simulations, Score Estimation and Reverse Reachable techniques address the first one. In this work, we are interested in approaches to the second phase.

4.2 DISCRETE OPTIMIZATION FOR INFLUENCE PROPAGATION

Over the past decade, the application of mathematical programming methods for solving optimization problems related to influence maximization has proved very useful. This section aims to provide an overview of the recent works on discrete optimization techniques to solve influence propagation problems. We focus mainly on the TARGET SET SELECTION problem and its specific variations: WEIGHTED TARGET SET SELECTION; LEAST COST INFLUENCE PROBLEM and generalizations. We devote a subsection to each of these. Hereafter, we discuss their shortcomings and related open problems.

4.2.1 Target set selection

Unlike Kempe et al. (2003), the diffusion model considered in (Ackerman et al., 2010; Ben-Zwi et al., 2011; Chen, 2009; Raghavan and Zhang, 2015) to study the Target Set Selection problem, is deterministic and each vertex exerts the same influence over each neighbor. This simplest model is the threshold model (explained in Chapter 3).

By considering different types of threshold models, Chen (2009) has the following hardness results. An important model is the *majority threshold*, where a vertex will be activated if at least half of its neighbors are active, that is, $t(v) = \left\lceil \frac{\delta(v)}{2} \right\rceil$, for all $v \in V$. For this setting, Chen (2009) shows that the TSS problem does not admit approximation algorithm of ratio within $O(2^{\log^{1-\epsilon} n})$, for any constant $\epsilon > 0$. The *small threshold* model is another interesting special case, where all the threshold are equal. For $t(v) = 1$ the problem can be solved trivially, but for $t(v) \geq 2$, TSS is NP-hard and approximating in this setting is as hard as in general setting. The model in which the individuals are more resistant to be influenced is the *unanimous threshold*, in this setting the threshold of each vertex is equal to its degree, i.e. $t(v) = \delta(v)$ for any $v \in V$. In this case, the problem is equivalent to the Minimum Vertex Cover, which implies that it has a 2-approximation algorithm. Further, the TSS problem is NP-hard to approximate within a polylogarithmic factor. However, an exact polynomial-time algorithm is given for the case when the input graph is a tree. This algorithm uses dynamic programming to find an optimal solution.

Given that the problem is easy to solve on trees and considering that the *treewidth* parameters of graphs play an essential role in producing exact and approximation algorithms, Ben-Zwi et al. (2011) proved that the treewidth parameter ω governs the complexity of TSS in a strict sense. Roughly speaking, the treewidth measures the similarity between a given graph and a tree, comparing its extension in a structural sense. Moreover, they also proposes an exact algorithm for Target Set Selection with running time of $|V|^{O(\omega)}$ on graphs with treewidth bounded by ω. Therefore, there are exact efficient algorithms for the TSS problem when the given social network is a tree (Chen, 2009) or if it has bounded treewidth (Ben-Zwi et al., 2011).

Ackerman et al. (2010) are among the first to develop a combinatorial model for the TSS problem and use integer linear programming models to represent it. They integer programming model seeks a subgraph of an acyclic tournament. Let U be an undirected graph, we say that a directed graph D is a *orientation* of U if D contains a directed edge (u, v) or (v, u) but not both whether $\{u, v\}$ is an edge of U. A *tournament* is an orientation of a complete undirected graph. The model follows.

4.2.1.1 Sets

Let $E' = \{(u, v) : (u, v) \notin E, u, v \in V\}$ be the set of non edges of a given directed graph G.

4.2.1.2 Variables

Consider two binary decision variables x_v and $y_{u,v}$ such that

$$x_v = \begin{cases} 1 & \text{if } v \in S \\ 0 & \text{otherwise} \end{cases}$$

representing the target set variables and

$$y_{u,v} = \begin{cases} 1 & \text{if } u \text{ exerts influence over } v \\ 0 & \text{otherwise} \end{cases}$$

denoting whether the directed edge $(u, v) \in E(G)$ belongs to the propagation graph G^*.

4.2.1.3 Model

(P.1)

$$\begin{aligned}
\text{Minimize} \quad & \sum_{v \in V} x_v & & \\
\text{subject to} \quad & \sum_{(u,v) \in E} y_{u,v} + t(v)x_v \geq t(v) & \forall v \in V & \quad (4.1) \\
& y_{u,v} + y_{v,u} = 1 & \forall u \neq v : u, v \in V & \quad (4.2) \\
& y_{u,v} + y_{v,w} + y_{w,u} \leq 2 & \forall u \neq v \neq w : u, v, w \in V & \quad (4.3) \\
& y_{u,v} \in \{0, 1\} & \forall (u, v) \in E \cup E' & \quad (4.4) \\
& x_v \in \{0, 1\} & \forall v \in V & \quad (4.5)
\end{aligned}$$

The objective function of the model P.1 aims to minimize the number of vertices in the target set. The first constraint (4.1) models the activation criterion for every vertex. Constraints 4.2 and 4.3 determine an acyclic tournament induced by the edges and non-edges. This formulation for the TSS problem uses technique designs typically applied in ordering problems (e.g., schedule based problems). The triangle elimination constraints ensure a cycle free propagation graph. They focus mainly on a version of the problem that asks for the minimum initial set of vertices whose activation will activate the entire network. Furthermore, Ackerman et al. (2010) use this mathematical model to show lower and upper bounds on the size of the minimum perfect target set under majority thresholds for directed and undirected graphs. When the input graph is a tree, we can ignore the tournament cycle elimination constraints since a tree has no cycles.

As we have seen, this combinatorial formulation admits many generalizations and variants. For example, adding weights in every vertex of the network and seeking a target set of minimum total weight. A possible variant is considering a budget k and asking for a target set of size k, which maximizes the size of the active set. Note that if we model the case in which each vertex exerts a different influence on each other, representing influence as the weights on edges of the network, we fall back on the original influence maximization formulation from Section 3.

4.2.2 Weighted target set selection

A study by Raghavan and Zhang (2015) considers a weighted variant of the TSS problem on undirected graphs, which definition follows.

Problem 5 (Weighted Target Set Selection (WTSS)) *Given an undirected graph $G = (V, E)$ with thresholds $t(v)$ and weights $w(v)$ on each $v \in V$. Find a set $S \subseteq V$ of minimum total weight $\sum_{v \in S} w(v)$ such that the entire graph is activated.*

In this formulation, there are weights $w(v)$ on each vertex $v \in V$ representing the costs for initially activating a target set. Raghavan and Zhang (2015) identifies that the linear programming relaxation of P.1, for the TSS problem, produces fractional extreme points even on trees, meaning that the formulation is weak. Then, they introduce a polynomial-time algorithm and a tight and compact extended integer linear programming formulation for the WTSS problem on trees. In addition to vertex and edge variables from model P.1, the central idea consists of splitting every edge of the input graph G by introducing a new "dummy" vertex and get four variables for each edge of G that indicate exerted influence. The new model can be solved in polynomial time on trees and cycles with integer solutions. It is possible because the constraints matrix of the presented program is *totally unimodular*. It is important to remember that it was considered a simplified version of the problem, where the influence weight is the same for all the edges. Subsequently, they extend this new formulation for general graphs by adding cycle prevention constraints. Then they arrive at a branch-and-cut algorithm by applying the extended formulation derived for trees.

Based on the observation that the activation process can be modeled as a directed acyclic graph (propagation graph), the extended formulation for WTSS can be applied to general graphs by introducing a new set of constraints as decision variables. Unlike the program P.1, now the cycle elimination constraints have an exponential number of inequalities in the size of G. Here, Raghavan and Zhang (2015) apply the branch-and-cut method to generate these inequalities dynamically. Solving its mixed-integer linear programming model requires solving the separation problem for such constraints. They use a procedure introduced by Grötschel et al. (1985) for the separation problem. Grötschel et al. (1985) show that violated directed cycle inequalities can be recognized in polynomial time. Given a directed graph G, we can find a shortest directed cycle in polynomial time by making modifications on any polynomial shortest path algorithm (e.g., Dijkstra's shortest path algorithm). In terms of branching, an useful information in (Raghavan and Zhang, 2015) is that the integrality of variables $\mathbf{x}$ and $\mathbf{y}$ can be relaxed and this suffices to keep the integrality of the vector $\mathbf{h}$ of variables $h_{u,v}$ for every $\{u, v\} \in E$.

4.2.3 Target set with partial incentives

Motivated by practical applications (primarily on viral marketing), another extension of the TSS problem brings the idea of offering additional incentives (or payments) to activate vertices. Instead of selecting a subset S of vertices to activate, the problem consists of offering incentives to the vertices of a graph so that it will trigger a cascade of influence that spreads to a given fraction of the network. Nowadays, this problem is named as Least Cost Influence Problem and is defined in Problem 4.

Both TSS and WTSS problems are special cases of the LCIP since selecting a target set S corresponds to choosing $\mathbf{y} \in \{0, 1\}^{|V|}$ (Demaine et al., 2014; Cordasco et al., 2015). Demaine et al. (2014) observe that LCIP is a fractional version of TSS. Despite this fact, they proved that LCIP maintains the same computational complexity of TSS from a theoretical point of view. However, the authors of (Demaine et al., 2014) argue that, in practical settings, the problem should require less computational effort to ensure the activation of individuals influenced by the target set.

Cordasco et al. (2015) presented a reduction from TSS problem to LCIP and WTSS problems, implying that LCIP has the same inapproximability ratio of TSS, which cannot be approximated within a poly-logarithmic factor of $O(2^{\log^{1-\epsilon} n})$, for every

$\epsilon > 0$. They studied a specialized version of the LCIP problem where all the weights of influence between the individuals are fixed to one. This variant introduces a polynomial-time algorithm that always finds optimal incentives on trees and complete graphs. This algorithm finds a vector of incentives $\mathbf{y}$ with a bounded cost on general graphs.

Günneç et al. (2016) focused on mathematical programming approaches for this problem. They showed that the problem is NP-complete even for restricted cases as bipartite graphs, equal weights of influence on the edges, and the whole network influenced. They focused on the version of the problem with equal weights of influence on arcs and required to achieve all the individuals of the networks (100% of adoption). They studied the LCIP on trees and proposed polynomial-time algorithms and a totally unimodular formulation for the LCIP on these type of graphs. Furthermore, they embedded such formulation on trees into a formulation on general graphs with exponential cycle elimination constraints. In the sequence, they arrived at a general branch-and-cut approach.

4.2.4 Generalized lest cost influence problem

Fischetti et al. (2018) introduces a generalization of the LCIP problem and propose a new ILP formulation for this problem. The new variant is called Generalized Least Cost Influence Propagation (GLCIP) and the definition follows.

Problem 6 (Generalized Least Cost Influence Propagation (GLCIP)) *Given a real number $\alpha \in [0, 1]$, a directed graph $G = (V, E)$ where each $v \in V$ has threshold $t(v) > 0$, a set $P_v \subset [0, \infty)$ of potential incentives with a cost w_{vp} associated to every $p \in P_v$ and $v \in V$, and an weight $d_{u,v} > 0$ on each edge $(u, v) \in E$ representing the strength of influence that u exerts on v. Find an assignment of incentives p_v^* for every $v \in V$ that minimizes the total cost*

$$\sum_{v \in V} \sum_{p \in P_v} w_{p,v}$$

paid such that $|A| \geq \alpha |V|$, where A is the set of active vertices.

According to Fischetti et al. (2018), the new problem GLCPI aims to overcome limitations of previous models (e.g., TSS (Chen, 2009), WTSS (Raghavan and Zhang, 2015), LCIP (Günneç et al., 2016)) that might be prohibitive for application in realistic scenarios. This new problem contains the previous problems (which are variants of TSS) as special cases. Further, they introduce the concept of *activation function*, which is an extension of the commonly used *threshold functions*. Such activation functions are used to decide whether an individual gets active or not. Based on the observation that the previous formulations are proposed for a particular class of the activation function, the alternative integer linear programming model presented generalizes the previous formulations. Moreover, they develop solutions based on constraint and column generation.

Let $N(v)$ be the set of in-neighbors of a vertex v, the decision making of v to be activated based on its active neighbors is modeled according to the *activation function* $f_v : 2^{N(v)} \times P_v \to \mathbb{R}_{\geq 0}$ satisfying $f_v(\emptyset, 0) = 0$, where for a set of active neighbors $U \subseteq N(v)$ and an incentive $p \in P_v$, v is activated if $f_v(U, p) \geq t(v)$. When v is activated by U we call U as the *influencing-set* of v.

With the objective of preserve the same notation used in (Fischetti et al., 2018), we define a *minimal influencing-set* as follows. For a given set $U \subseteq N(v)$ and an

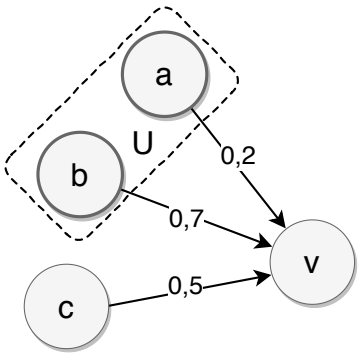


Figure 4.1: Influencing-set $U \subseteq N(v)$ of a vertex v, where $N(v) = \{a, b, c\}$ and $U = \{a, b\}$.

incentive $p \in P_v$ such that $f_v(U, p) \geq t(v)$, set p' as the minimum incentive necessary to activate v using U, i.e. $p' = \min\{p \in P_v : f_v(U, p) \geq t(v)\}$. We say that a set U is minimal if and only if there is no $U' \subseteq U$ such that $f_v(U', p') \geq t(v)$. Also, define $\Lambda_v \subseteq 2^{N(v)}$ as the set of all minimal influencing-sets of a vertex v. Lastly, let w_v^U be the cost of the cheapest incentive possible to activate v for a fixed influencing-set U, i.e. $w_v^U = w_{vp'} : p' = \min\{p \in P_v : f_v(U, p) \geq t(v)\}$. Recall that, $w_v^U = 0$ if the set U activates v with no incentives ($f_v(U, 0) \geq t(v)$).

The column generation model for the GLCIP follows.

4.2.4.1 Variables

For each vertex $v \in V$ we set the binary decision variable x_v which is 1 if v is active and 0 otherwise. In every arc $(u, v) \in E$ define a binary variable z_{uv} to indicate whether (u, v) is in the solution, i.e. vertex u exerts influence over v. The binary variable λ_v^U associated to each minimal influence set U of a vertex v is 1 if set U, together with a incentive, is the influencing set that activates v and 0 otherwise.

4.2.4.2 Model

$$
\begin{aligned}
\text{minimize} \quad & \sum_{v \in V} \sum_{U \in \Lambda_v} w_v^U \lambda_v^U & & \\
\text{s. t.} \quad & \sum_{U \in \Lambda_v} \lambda_v^U = x_v & \forall v \in V & (4.6)\\
& \sum_{U \in \Lambda_v : u \in U} \lambda_v^U = z_{uv} & \forall (u,v) \in E & (4.7)\\
& \sum_{j \in X} \sum_{U \in \Lambda_j : U \cap X = \emptyset} \lambda_j^U \geq x_k & \forall k \in X, \forall X \subseteq V & (4.8)\\
& z_{u,v} \leq x_u & \forall (u,v) \in E \text{ s.t. } (v,u) \notin E & (4.9)\\
& \sum_{v \in V} x_v \geq \lceil \alpha |V| \rceil & & (4.10)\\
& z_{uv} \in \{0,1\} & \forall (u,v) \in E & (4.11)\\
& x_v \in \{0,1\} & \forall v \in V & (4.12)\\
& \lambda_v^U \geq 0 & \forall v \in V, \forall U \in \Lambda_v & (4.13)
\end{aligned}
\tag{P.5}
$$

The objective is to minimize the total cost of necessary incentives. Constraints in (4.6) determines that exactly one influencing-set will activate v and (4.7) says that an arc (u, v) is in solution only if the source u belongs to the chosen influencing-set. The notable contribution in this model is the set of constraints in (4.8) named *Generalized Propagation Constraints*, here particular properties of the problem are exploited to strength the model. In short, it ensures that there will be no cycles in a solution by forcing that for every subset $X \subseteq V$ if k is active, some element of X needs to receive external influence. Constraints (4.9) means that an arc z_{uv} is active only if the vertex u is activated. Constraint (4.10) ensures that at least the given fraction of vertices will be activated. Due to the constraints (4.6) and (4.7) the variable λ_v^U can be relaxed to continuous variable.

Chen et al. (2020) studied the polyhedral structure of the LCIP formulated as a mixed-integer programming problem. Considering the threshold model and deterministic parameters, they arrive at a cutting plane method by introducing a class of new facet-defined inequalities derived from the substructure of the model for the propagation constraint in a single-node relaxation. Also, they provide an exact polynomial-time algorithm for the separation problem of these inequalities. Another study of the polyhedral structure is the work of Soltani and Moazzez (2019). In this work, they approach the TSS problem and present new families of facet-defining valid inequalities for the polytope of the TSS problem (referred to as Dynamic Monopoly). Also, they present a new compact integer programming formulation for the TSS problem, which provides stronger LP-relaxations compared to the formulation of Ackerman et al. (2010).

Nannicini et al. (2020) propose an exact branch-and-cut algorithm for the robust influence maximization problem. In their robust version of the problem, they discuss the uncertainty on the vertex activation threshold. They present a new formulation to solve a deterministic version of the influence maximization problem. This new formulation exploits the duality of subproblems. Next, they extend this formulation to

the robust case by adding an exponential set of cuts that are separated in a branch-and-cut algorithm.

It is worth noting that the TSS, WTSS, LCIP, and GLCIP problems are defined in a fixed propagation model, which is the threshold model. The formulations would inevitably be different if other diffusion models, like independent cascade and linear threshold, would be considered.

4.3 COMBINATORIAL OPTIMIZATION ON POWER-LAW GRAPHS

In order to classify the NP-hard optimization problems in terms of their approximability, some studies have investigated these problems in the context of power-law networks. Studies in these families of graphs are motivated by recent empirical indications that combinatorial optimization problems may be easier to solve in such networks than general graphs (Brach et al., 2016; Dinh et al., 2014; Ferrante et al., 2008; Friedrich and Krohmer, 2015; Liu et al., 2014; Zhang et al., 2012). It is, therefore, natural to investigate a real-world inspired problem on power-law graphs. Such discovery on real-world networks brings a systematic theory of optimization in power-law graphs, as is discussed next.

4.3.1 On general problems

Although some problems seem to be easy in complex networks, Ferrante et al. (2008) give theoretical results that, on power-law graphs, many classical NP-hard problems remain intractable for certain values of the power-law exponent β. The problems studied in (Ferrante et al., 2008) were:

i) Maximum Clique (Max-Clique);

ii) Minimum Coloring (Min-Coloring);

iii) Minimum Vertex Cover (Min-VC);

iv) Minimum Dominating Set (Min-DS);

v) Maximum Independent Set (Max-IS).

Problems i) and ii) are NP-hard for $\beta > 1$, while the last three are NP-hard for $\beta > 0$. The results for Min-VC, Min-DS and Min-IS holds due to an *optimal substructure* property that they have in common. Shortly, a problem P is said to have an optimal substructure if we can use the optimal solutions of its subproblems to produce a globally optimal solution. Thus, if P satisfy this property and is NP-hard in general graphs then it is also NP-hard in power-law graphs for all $\beta > 0$.

On the other hand, about the tractability of some combinatorial optimization problems, Friedrich and Krohmer (2015) proved that the problem of finding cliques (k-Clique) is efficiently solvable in power-law graphs with $\beta \geq 3$ (it is well known that the k-Clique problem is NP-complete on general inputs). Here the specific structure of the inhomogeneous random graph model was algorithmically exploited. For $2 < \beta < 3$ cliques of size k can be found in $O(ne^{k^4})$ time and in expected linear time for $\beta \geq 3$. Shen et al. (2010) claimed that the Max-IS, Min-VC and Min-DS problems are APX-hard even for the case of power-law graphs, here the model used was the $P(\alpha, \beta)$ model

mentioned in the Section 3.6 (recall that the MAX-IS problem is NP-hard and does not admit a constant factor approximation for general graphs).

Motivated by the shortcoming of random graph models designed to capture properties of power-law graphs (e.g., $P(\alpha, \beta)$), the model introduced by Brach et al. (2016) defines deterministic conditions for checking whether a graph is power-law without use the randomness assumption about the graph models. This model is called *Power-Law Bounded* (PLB) and its definition admits to derive new properties of power-law graphs that can be exploited to give efficient algorithms for a number of classical graph problems. Furthermore, Brach et al. (2016) provides a theoretical information why there are algorithms that run faster in real world than predicted by worst case analysis. Such properties hold for many real world networks, which implies that the mentioned problem on power-law graphs can be solved faster than the worst case for general graphs.

4.3.2 On the influence propagation problems

There are also studies about the spread of information considering the power-law degree distribution of social networks. Zhang et al. (2012) concentrate on working with the POSITIVE INFLUENCE DOMINATING SET problem on power-law graphs and proved that a greedy algorithm presented by Wang et al. (2011) admits a constant approximation ratio in such networks. The study in (Zhang et al., 2012) shows that the power-law degree distribution can improve the performance of greedy algorithms for a class of problems known as *submodular cover* problems. They show that the PIDS problem belongs to the class of submodular cover problems and has a constant factor approximation of

$$1 + \left(\frac{1}{\beta - 1} + \ln \gamma\right)\left(1 - \frac{1}{\gamma + 1}\right)^{1-\beta}$$

where $\gamma = \sqrt[\beta-2]{\frac{\beta-1}{\beta-2}}$. The same approximation guarantee holds for the more general MIN-DS problem in power-law graphs (Zhang et al., 2012). Related to these results about the approximability, Dinh et al. (2014) stated that, for every graph of the $P(\alpha, \beta)$ model, the size of the optimal solution D^* of PIDS is

$$|D^*| = \begin{cases} \Omega(n^\beta) & \text{if } \beta < 1 \\ \Omega\left(\frac{n}{\log n}\right) & \text{if } \beta = 2 \\ \Omega(n) & \text{if } \beta > 2 \end{cases}$$

where n is the number of vertices in the graph. Moreover, if a social network has an optimal solution D^* such that $|D^*| = \Omega(n)$, then any algorithm that produces a valid solution yields a constant approximation ratio (Dinh et al., 2014).

Regarding the influence maximization problem, Liu et al. (2014) presented a Monte Carlo based method for estimating the spread of influence in a social network that follows a power-law degree distribution. The estimation method proposed by Liu et al. (2014) aims to use a supervised sampling to predict the number of vertices needed to be sampled according to the power-law exponent β of the given social network. This strategy avoids computing the exact value of the influence function and efficiently estimates the influence spread at a small cost of precision.

As summarized above, there is a reasonable amount of works seeking to understand the implications of power-law degree distribution for combinatorial problems. Most of them in recent years, but none of them use the mathematical programming approach.

5 PRESELECTION FOR INFLUENCE MAXIMIZATION

An important technique for obtaining efficient algorithms is to use preprocessing. The most common mathematical programming techniques are tightening the bounds (when we use cutting planes, for example), eliminating redundant variables, and variable fixing. In this way, before starting to directly address the problem, a useful strategy could be preprocessing to reduce its size. To this end, our preselection methodology presented here can be suitable to restrict the search for a reduced feasible region of the Influence Maximization problem (Problem 1).

In short, the preselection strategy works as follows. To highlight the most promising vertices, the selection process explores some properties of power-law graphs and the relationship between social influence and degree distribution. It prevents, in this way, unnecessary processing by cutting out some vertices from the search. The preselection adapts a greedy algorithm for the classical Dominating Set problem and is biased in favor of high degree nodes since it consists of discarding nodes in which higher-degree vertices cover all of its out-neighbors. The resulting subset is called *set of candidates*, and this strategy is presented in Algorithm 6. We have verified by experimental analysis that this preselection reduces the running time while preserving the quality of solution.

The greedy algorithm (Algorithm 5) presented by Kempe et al. (2003) has two major sources of inefficiency. First, the processing time of $\sigma_m(S)$ function is too high, since to get the exact value of σ is a #P-hard problem on LT and IC models (Chen et al., 2010a,b; Kempe et al., 2003). Second, the algorithm makes many calls to σ.

While most of the studies focus on proposing algorithm to find the set of early adopters or to estimate the influence function, for instance (Chen et al., 2010a, 2009; Goyal et al., 2011b,a,c; Leskovec et al., 2007; Tang et al., 2015, 2014), we try to take advantage of knowing the topology of most of the large scale social network. Liu et al. (2014) shows that only a few out-neighbors of the hubs have considerable influence, while many of these neighbors contribute little to the marginal gain. These findings suggest that there is a relation between the degree distribution and the reach of an information that spreads along the network. We explore these findings to recognize and rule out the less probable influencer nodes.

In summary, the contribution discussed in this chapter is twofold. An efficient heuristic to select the more promising vertices and, as an application of such strategy, we present an algorithm to select the early adopters in power-law graphs. The main contribution is the Preselector algorithm which chooses a subset of the vertices based on its degree, where the objective is to decrease the number of evaluated vertices by the greedy algorithm. Such strategy allied with the Celf optimization lead us to the second contribution, the PrevalentSeed, that makes less calls to the σ function. Experimentally, this approach reduces up to 57% the Celf's run time.

The rest of the chapter is organized as follows. Initially, we introduce the algorithm to select the most promising vertices to become early adopters. Then we show some theoretical analysis that have been carried out on running time and quality results. Next, we present the algorithm that chooses the early adopters using the preselection combined to a lazy forward update scheme. Lastly, in the experiments section, some

observations are described about the empirical results achieved on real-world power-law graphs.

5.1 PRESELECTION

Instead of computing the marginal gain over the whole set of vertices at each iteration to select those with the highest marginal gain, we only travel a subset of nodes. This subset will be called *set of candidates*, and it is selected by a heuristic (Algorithm 6) in advance. In order to reduce the number of calls to σ, we discard the nodes that may have small marginal gain before processing them in fact. Thus, we compute the marginal gain only for the more promising nodes, so we avoid multiple calls to the σ's estimation by choosing correctly such nodes. Our strategy to select the candidates is fundamentally based on the following criteria. We assume that a vertex will not have high marginal gain if their out-neighbors already can be influenced by a node of higher degree. So the search chooses nodes that can cover the greatest number of non-covered nodes. A vertex is covered when it has at least one in-neighbor that already was chosen as candidate in the iterative process. The procedure stops when there are no more uncovered vertices.

5.1.1 Set of candidates

The set of candidates, selected by Algorithm 6, is defined as $C \subseteq V$. Initially, $C = \emptyset$, and nodes are added iteratively during the execution of the algorithm. For every $v \in V$, the set of out-neighbors of v is denoted by $N^+(v) = \{w \in V : (v, w) \in E\}$. Similarly, the out-neighborhood of the set C is denoted as $N^+(C) = \{v \in V : (u, v) \in E, u \in C$ and $v \notin C\}$. To simplify the pseudo-code, we denote by $D = C \cup N^+(C)$ the set of vertices covered by C. Figure 5.1 illustrates the set of candidate C (inner cycle) and de set D of vertices covered by C.

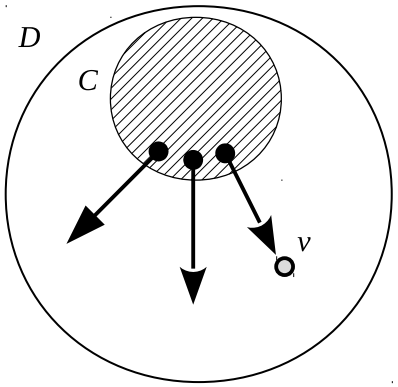


Figure 5.1: Coverage set $D \subseteq V$, candidate set $C \subseteq D$ and a vertex v of a graph.

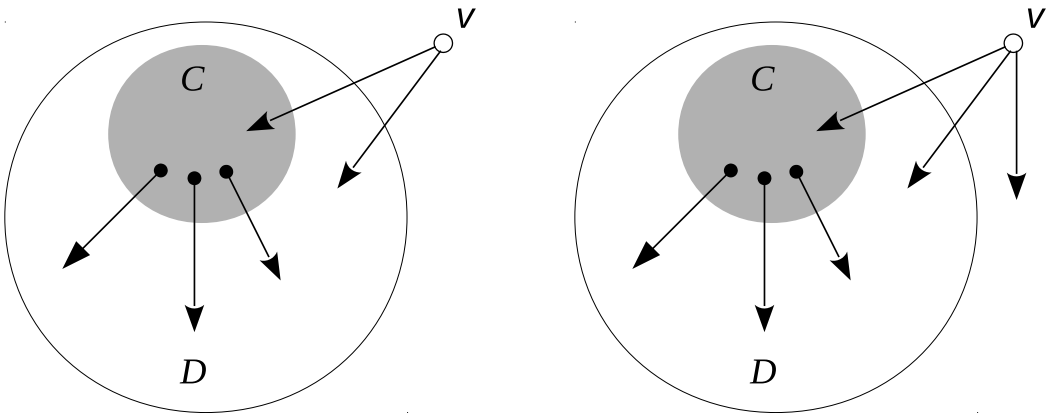
Algorithm 6 is the pseudo-code of the preselection. In this algorithm, each node v_i is selected as candidate when it has at least one out-neighbor that is still uncovered. Notice that the set of candidates C is a *dominating set* in directed graphs by definitions, so the PRESELECTOR is an greedy algorithm to find a dominating set in directed graphs. Theorem 3 determines an upper bound for the PRESELECTOR's running time.

Figure 5.2 shows an example of one step of the Algorithm 6. In the case of 5.2(a) the vertex v is not selected as a candidate because all its neighbors are covered,

Algorithm 6: PRESELECTOR

Input: $G = (V, E)$
Output: Set C of candidates

```
1 begin
2     Sort the vertices v_1, v_2, ..., v_|V| in decreasing order by out-degree
3     C ← ∅; D ← ∅
4     for i ← 1 to |V| do
5         if N+(v_i) ⊈ D and N+(v_i) ≠ ∅ then
6             C ← C ∪ {v_i}
7             D ← D ∪ {v_i} ∪ N+(v_i)
```



(a) Vertex v is not selected to the candidate set.

(b) Vertex v is selected to the candidate set.

Figure 5.2: One step of the preselection process.

while in case 5.2(b), the vertex v will become a candidate because there is one neighbor of v not covered by C yet.

Theorem 3 *Let G be a directed graph with n vertices and m edges. Algorithm 6 ends in $O(n \log n + m)$ steps.*

Proof: At the first line, if we use an efficient algorithm like the *merge sort* to sort the vertices, this task will be performed in $O(n \log n)$ time on the number of vertices. After, in the loop of the lines 4-7, the trickiest operation is the condition that depends on whether the set D contains $N^+(v_i)$, for all $v_i \in V$. Thus, let $\delta^+(v_i)$ be the out-degree of v_i. No more than $\delta^+(v_i)$ steps are needed to check each v_i. Thus, directly by the "*Handshake*" lemma, the loop demands $\sum_{v_i \in V} \delta^+(v_i) = O(m)$ comparisons.

Finally, the total time for preselection is $O(n \log n) + O(m) = O(n + m)$, where $O(n \log n)$ is the time to sort the list of vertices. □

5.1.2 Analysis of the preselection process

The preselection process is biased in favor of nodes of high degree, since it consists of discarding nodes in which all its out-neighbors are covered by higher-degree vertices. So we want to show that when a vertex v does not belong to C after preselection, v tends

to have a low marginal gain compared to the selected nodes. In this way, we can avoid unnecessary computation, for the marginal gain of v, during the greedy search for the k nodes of the highest marginal gain.

For this purpose, we use three results to argue about it. First, for each vertex v in which all its out-neighbors are already candidates, we can activate v plus $N^+(v)$ to improve the spread of active nodes. However, in the number of activated nodes, activate v has the same effect of activating only $N^+(v)$. For this reason, we do not need to activate v since only its neighbors are sufficient. Lemma 4 provides a demonstration of this statement. Second, if all the out-neighbors of a vertex v are covered by the set of candidates, putting v together with C as active nodes can increase the probabilities of such neighbors being activated, but such increase is low and limited. Thereby, as shown in Lemma 5, v can be left aside. The last result obtained from this analysis is the Theorem 6, which says that for each $v \notin C$, the set C has the possibility of activate all the nodes that v would activate.

To analyze the quality of the PRESELECTOR's output, we have to assume some simplifications. First, we consider the IC model with activation probabilities p equals on each edge. Second, to make some calculus, we use a more simple influence function called *direct influence* instead of σ itself. Not making these simplifications implies in compute the exact value of σ, which is not the goal of this analysis, since to get this value is a #P-hard problem. The Definitions 1 and 2 describe the concept of direct influence that we consider here.

Definition 16 (Direct influence of a vertex) *Let v be an active vertex. We call direct influence of v the number inf(v) of vertices in $N^+(v)$ activated by v.*

Definition 17 (Direct influence of a set) *Let A be a set of active nodes. We denote inf(A) the direct influence of A, the number of nodes in $N^+(A)$ activated by vertices in A.*

In this way, note that for all $v \in V$, each $w \in N^+(v)$ becomes active with probability p so inf(v) is a random variable. Thus, at every execution of the activation process, inf(v) can assume a different value between zero and $|N^+(v)|$. The same happens with inf(A). Therefore, we can use expectation on the following results.

In Algorithm 6, notice that a vertex v will not become a candidate if all its out-neighbors are covered by the set C, at line 5. This can happen in two different ways: (i) either $N^+(v) \subseteq C$, or (ii) the out-neighbors of v are covered but not all $w \in N^+(v)$ belongs to C (see Figure 5.3). Lemmas 4 and 5 address these cases respectively.

Lemma 4 *Let v_i be a vertex at the i-th iteration of the Algorithm 6. If $N^+(v_i) \subseteq C$, then the additional influence that v_i could yield for the set C is either null or negative, that is,* $\mathrm{E}[inf(C \cup \{v_i\})] - \mathrm{E}[inf(C)] \leq 0$.

Proof: To validate the inequality of the lemma, we need to know the value of inf(C). Knowing that the set C has at least one edge to each $w \in N^+(C)$, the reasoning is as follows. To find the number of vertices that can be directly activated by C, consider a random variable Y_w which is 1 if w was activated by a vertex in C, and 0 otherwise. Thus,

$$\text{inf}(C) = \sum_{w \in N^+(C)} Y_w.$$

By the linearity of expectation and the definition of Y_w as binary variable, the expected value of $\text{inf}(C)$ is

$$\begin{aligned} \text{E}[\text{inf}(C)] &= \sum_{w \in N^+(C)} \text{E}[Y_w] \\ &= \sum_{w \in N^+(C)} \Pr(Y_w = 1). \end{aligned} \tag{5.1}$$

Suppose now that we have added v_i to C in order to increase $\text{inf}(C)$. To determine the effect of this change in the activation probabilities, we need to consider whether v_i was in the neighborhood of C before becoming a candidate. In the negative case, that is, if $v_i \notin N^+(C)$, it is simple to visualize that including v_i in C does not increase the value of $\text{inf}(C)$, once v_i has no neighbor outside of C, no edge will be added to the sum of the probabilities on the Equation 5.1, that is,

$$\text{E}[\text{inf}(C \cup \{v_i\})] = \text{E}[\text{inf}(C)].$$

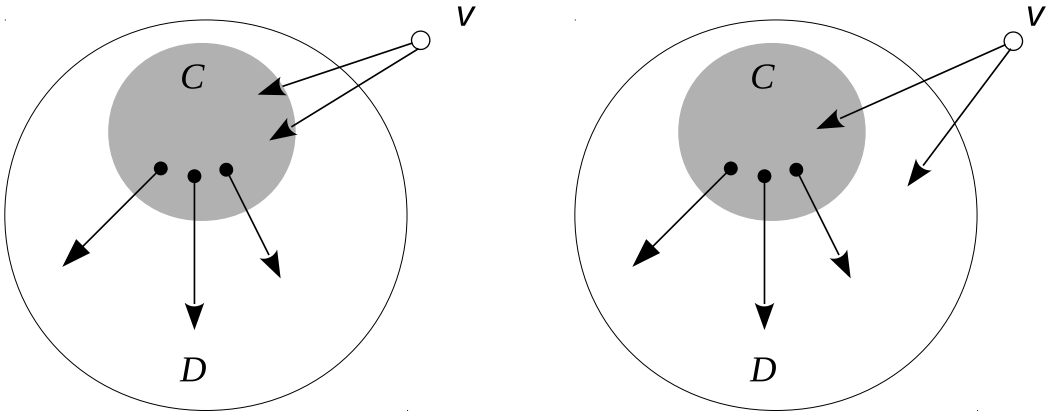
In an activation process in which the set C is active, all the out-neighbors of v_i would already be activated, and then v_i would not activate another vertex. However, if $v_i \in N^+(C)$, things can be different because adding v_i to C reduces one element of $N^+(C)$, then the value of $\text{inf}(C)$ cannot be larger than $|N^+(C)| - 1$. Hence, at least one edge is removed from the sum of the Equation 5.1. Then we have

$$\begin{aligned} \text{E}[\text{inf}(C \cup \{v_i\})] &= \sum_{w \in N^+(C) \setminus \{v_i\}} \Pr(Y_w = 1) \\ &< \text{E}[\text{inf}(C)]. \end{aligned}$$

Given these two possibilities, we finally have

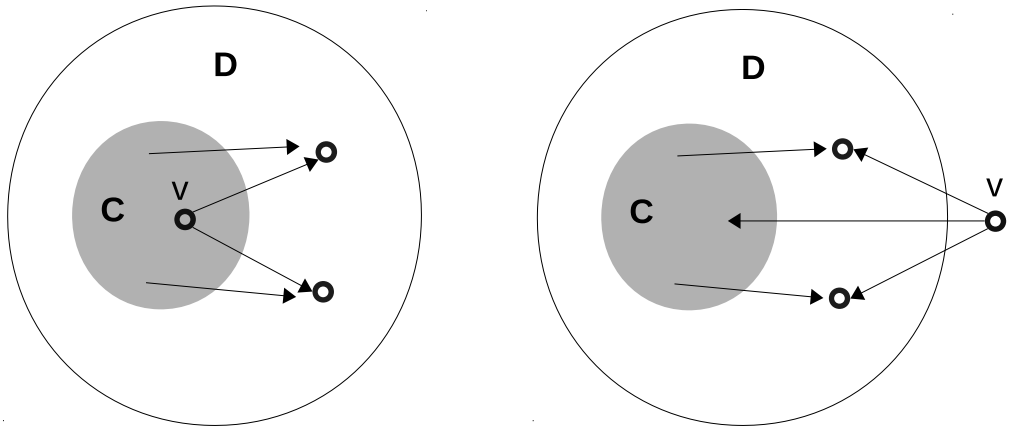
$$\text{E}[\text{inf}(C \cup \{v_i\})] \leq \text{E}[\text{inf}(C)].$$

□



(a) $N^+(v) \subseteq C$. (b) $N^+(v) \nsubseteq C$.

Figure 5.3: In the preselection, v can be discarded by two different ways, (a) and (b) are the two ways in which the neighbors of v can be covered by the set of candidates C.



(a) $N^+(v) \subseteq C$. (b) $N^+(v) \nsubseteq C$.

Figure 5.4: A vertex v that have out-neighbors covered. That is, it shares all the out-neighbors with the set of candidates, but such neighbors can be both in C and $N^+(C)$.

The vertices in which all its out-neighbors are candidates do not improve the direct influence of C. Now we can think about the vertices that have out-neighbors covered. It shares all the out-neighbors with the set of candidates, but such neighbors can be both in C and $N^+(C)$ (see Figure 5.4). In this case, Lemma 5 gives us an upper bound to the additional direct influence that this type of vertex can provide to the set of candidates.

Lemma 5 *Let v_i be a vertex at the i-th iteration of the Algorithm 6. Denoting as $\mathcal{G}(C, v_i)$ the additional influence provided by adding v_i in C. If $N^+(v_i) \subseteq D$, but $N^+(v_i)$ is not fully in C, then $\mathcal{G}(C, v_i) \leq \frac{1}{4}|N^+(v_i) \setminus C|$.*

Proof: Whereas the set $N^+(v_i)$ is not fully contained in C but belongs to D, v_i should stay in $V \setminus C$. However, if we add v_i to C in order to increase $\inf(C)$, as in Lemma 4, we have to consider two possible situations: (i) $v_i \in N^+(C)$, and (ii) $v_i \notin N^+(C)$. Unlike the Lemma 4, now in both cases (i) and (ii) the probabilities will change and increase the direct influence of C. This happens because the number of edges incident to $N^+(C)$ increases, and so the probability of such vertices becoming active increases. We want to find an upper bound for this probability growth.

Given a vertex $w \in N^+(v_i) \setminus C$, this vertex can be directly activated by the vertices in C, and we can obtain the probability of C activate w directly as follows. Let A be the event in which w is activated by C. Remembering that p is the activation probability in the independent cascade model, we have $\Pr(A) \geq p$, that is, the set C has at least one edge to w. Including v_i in C, w could be activated by C and by v_i. Thus, let B be the event in which w is activated by v_i, then $\Pr(B) = p$ and the probability of the set $C \cup \{v_i\}$ activate w can be obtained with the equation

$$\begin{aligned}\Pr(A \cup B) &= \Pr(A) + \Pr(B) - \Pr(A) \cdot \Pr(B) \\ &= \Pr(A) + p - \Pr(A) \cdot p.\end{aligned}$$

Here we apply the principle of inclusion and exclusion in the sum of the activation probability. The first equality occurs due to the independence between the events A and B.

The increase supplied by v_i on the probability of C activate w is the difference between $\Pr(A \cup B)$ and $\Pr(A)$. Note that $\Pr(A)$ is equivalent to $\Pr(Y_w = 1)$ in the Equation 5.1. Thus, let $\mathrm{E}[\mathrm{inf}_w(C)]$ be the expected direct influence of C on w. By the Equation 5.1, we have

$$\mathrm{E}[\mathrm{inf}_w(C)] = \Pr(Y_w = 1) = \Pr(A),$$

similarly,

$$\Pr(A \cup B) = \mathrm{E}[\mathrm{inf}_w(C \cup \{v_i\})].$$

Let $\mathcal{G}_w(C, v_i)$ be the additional influence provided by v_i on the probability of C activate w. The value of $\mathcal{G}_w(C, v_i)$ is given by the following equation

$$\begin{aligned}
\mathcal{G}_w(C, v_i) &= \mathrm{E}[\mathrm{inf}_w(C \cup \{v_i\})] - \mathrm{E}[\mathrm{inf}_w(C)] \\
&= \Pr(A \cup B) - \Pr(A) \\
&= \Pr(A) + p - \Pr(A) \cdot p - \Pr(A) \\
&= p - \Pr(A) \cdot p \\
&= p(1 - \Pr(A)) \\
&\leq p(1 - p),
\end{aligned}$$

where the inequality holds because $\Pr(A) \geq p$. This holds for any $w \in N^+(v_i) \setminus C$. Consequently, we need to apply the same difference for all vertices in $N^+(v_i) \setminus C$. The increase of direct influence led us to a quadratic function that represents a parabola upside-down, such that the max value is $\frac{1}{4}$, when $p = \frac{1}{2}$. In consequence,

$$\begin{aligned}
\mathcal{G}(C, v_i) &= \prod_{w \in N^+(v_i) \setminus C} \mathcal{G}_w(C, v_i) \\
&\leq \prod_{w \in N^+(v_i) \setminus C} p(1 - p) \\
&\leq \prod_{w \in N^+(v_i) \setminus C} \frac{1}{4} \\
&= \frac{1}{4} \cdot |N^+(v_i) \setminus C|.
\end{aligned}$$

As the final point, the value of $\mathcal{G}_w(C, v_i)$ is mutually independent for each $w \in N^+(v_i) \setminus C$, then we can take the product over all results. □

As shown above, selecting as candidate a vertex v_i such that $N^+(v_i) \subseteq D$, provides a negligible additional influence. Additionally, the Theorem 6 brings up that at the end of preselection any vertex ruled out has less influence on its neighbors than the set C. Figure 5.5 illustrates this situation, where all the outgoing neighbors of a vertex v can be reached by the set of candidates.

Theorem 6 *Let $inf_{N^+(v)}(C)$ be the direct influence of C on $N^+(v)$, where $v \in V$. At the end of the Algorithm 6, we have $\mathrm{E}[inf(v)] \leq \mathrm{E}[inf_{N^+(v)}(C)]$, for all $v \in V \setminus C$.*

Proof: By the Algorithm 6, if $v \in V \setminus C$ then all $w \in N^+(v)$ should be in D. We want to compare the v's influence with C's influence on $N^+(v)$. To this end, we need to consider just the vertices in $N^+(v) \setminus C$. Thus, when $w \in N^+(C)$, besides receiving an edge of v,

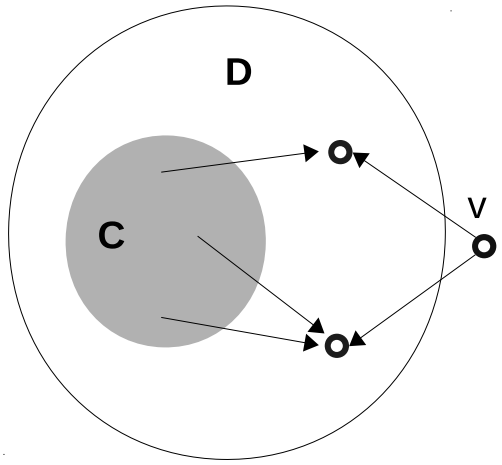


Figure 5.5: The set of candidates can directly reach all the out-neighbors of a vertex $v \notin C$. It means that the set of candidates exert at least the same influence of v over the out-neighbors of v.

w also receives at least one edge from C. Let $X_{v,w}$ be a binary random variable, which is 1 if v activates w and 0 otherwise. We can get the value of inf(v) by the equation

$$\text{inf}(v) = \sum_{w \in N^+(v) \setminus C} X_{v,w}.$$

Using the linearity of expectation, we have

$$\begin{aligned} \text{E}[\text{inf}(v)] &= \sum_{w \in N^+(v) \setminus C} \text{E}[X_{v,w}] \\ &= \sum_{w \in N^+(v) \setminus C} \Pr(X_{v,w} = 1) \\ &= \sum_{w \in N^+(v) \setminus C} p. \end{aligned} \tag{5.2}$$

Since some vertices in C have edges to $N^+(v)$, each $w \in N^+(v)$ have at least one edge from C. Let $B_w = \{u \in C \text{ such that } (u,w) \in E\}$ be the set of vertices in C with edges to w, for all $w \in N^+(v) \setminus C$. Note that, $|B_w| \geq 1$, otherwise w would not be covered by C. Now consider the events $E_1, E_2, ..., E_{|B_w|}$ such that E_i is the event in which w is activated by $u_i \in B_w$. Once we have defined that all edges have activation probability p, then

$$\Pr(E_1) = \Pr(E_2) = ... = \Pr(E_{|B_w|}) = p.$$

As a result, for each $w \in N^+(v) \setminus C$, we have

$$\begin{aligned} \Pr(X_{v,w} = 1) = p &\leq \Pr(E_1 \cup E_2 \cup ... \cup E_{|B_w|}) \\ &= \Pr(B_w \text{ activate } w). \end{aligned} \tag{5.3}$$

To know the number of nodes in $N^+(v)$ activated by C, consider a random variable Y_w which is 1 if w is activated by a vertex in C and 0 otherwise. Thus,

$$\text{inf}_{N^+(v)}(C) = \sum_{w \in N^+(v) \setminus C} Y_w.$$

Again, by the linearity of expectation,

$$\begin{aligned}
\mathrm{E}[\mathrm{inf}_{N^+(v)}(C)] &= \sum_{w \in N^+(v) \setminus C} \mathrm{E}[Y_w] \\
&= \sum_{w \in N^+(v) \setminus C} \Pr(Y_w = 1) \\
&= \sum_{w \in N^+(v) \setminus C} \Pr(B_w \text{ activate } w) \\
&\geq \sum_{w \in N^+(v) \setminus C} p \\
&= \mathrm{E}[\mathrm{inf}(v)],
\end{aligned}$$

where the inequality follows from Equation 5.3 and the last equality came from Equation 5.2. □

Due to the criteria to choose C, all the vertices that have no out-neighbors are in $V \setminus C$ while many of the higher degree vertices belong to C. Hence, we suppose that the marginal gain of the vertices in $V \setminus C$ is always low and would be discarded in any way. In view of power-law graphs, where most of the vertices have low degree and a small number have high degree, there are two general aspects of the excluded vertices. First, a large number of nodes has degree equal to one, many of which has no out-neighbors. Furthermore, some vertices which degrees are greater than one also have no out-neighbors. Consequently, these nodes would not activate other vertices. Second, for all $v \in V \setminus C$ in which the out degree is greater than one, v is subject to Lemmas 4, 5 and Theorem 6, that is, the nodes in C are enough to achieve the vertices in $N^+(v)$.

It is worthwhile noticing that, based on the number of nodes without out edges, it is possible to quantify the number of vertices that cannot activate anyone. But it requires a very thorough analysis on random power-law graphs, and this is not the scope of this work. Given this, we suppose that we can select the k early adopters of set S within the set C without losing quality of spread, then we do not need to consider every vertex of the graph using the greedy algorithm. Therefore, although the analysis was simplified, there are strong indications that the results would be positive in more complex models, with distinct propagation probabilities.

5.2 THE PREVALENTSEED ALGORITHM

We now present an algorithm that chooses the early adopters in power-law graphs, called PrevalentSeed. We combine the Preselector with the Celf's "*Lazy Forward*" update scheme. The idea is to show how the preselection can be used in a seed set selection algorithm.

Algorithm 7 shows the pseudo-code. Initially, we divide the vertex set into two disjoint subsets, C and $V \setminus C$, such that C is the set of candidates (line 3). Here, we chose to use the Celf optimization as a sub routine instead of the greedy algorithm of Kempe et al. (2003), since it is faster. Therefore, the code snippet between lines 4-15 is a modification of Celf, in which the difference is the loop at lines 4-7, where we inserted in the list Q only the vertices of set C. From this moment on, the marginal gain of the vertices in $V \setminus C$ is not estimated anymore. Next, the marginal gain of each $v \in C$ is estimated and v is added to Q in a non increasing order of marginal gain. The search

follows the CELF's idea, at lines 8-15, to make a greedy search and select the k vertices of higher marginal gain from the vertices belonging to the set C.

Algorithm 7: PREVALENTSEED

```
   Input: G, k, σ
   Output: Seed set S
1  begin
2      S ← ∅, Q ← ∅
3      C ← PRESELECTOR(G)
4      foreach u ∈ C do
5          δ_u ← σ({u})
6          u.it ← 0
7          Add u to Q in a non increasing order by δ_u
8      while |S| ≤ k do
9          Dequeue u from Q
10         if u.it = |S| then
11             S ← S ∪ {u}
12         else
13             δ_u ← σ(S ∪ {u}) − σ(S)
14             u.it ← |S|
15             Enqueue u in Q and sort
```

As well as in the CELF algorithm, the element of Q corresponding to v stores a table of the form $\langle \delta_v, v.it \rangle$, where $\delta_v = \sigma(S \cup \{v\}) - \sigma(S)$ is the marginal gain of v compared to S, and $v.it$ marks at which iteration the value of δ_v was last updated. In each of the k iterations of 'while' loop, v is removed from the queue and checked if the marginal gain already was computed at the current iteration, using the *it* attribute. If yes, v is the vertex of the greatest marginal gain at the current iteration, so it will be selected as a seed (lines 10-11). Otherwise, the lines 12-15 recompute the v's marginal gain and insert it again in Q such that the order is maintained.

Theorem 7 determines the running time of the PREVALENTSEED and shows that it is asymptotically equal to CELF, even making the preselection.

Theorem 7 *Let G be a directed graph with n vertices and m edges. Algorithm 7 executes in $O(knrm \log n)$ time, where k is the number of early adopters and r is the number of Monte Carlo simulations.*

Proof: The PREVALENTSEED's running time is given as follows.

i) By Theorem 3, the call to PRESELECTOR at line 3 uses $O(n \log n + m)$ steps to find the set C.

ii) In the loop of lines 4-7, $O(|C|rm)$ operations are made. This loop computes the value of $\sigma(v)$ for all $v \in C$. The $\sigma(v)$ is estimated with $r = 10.000$ simulations of spread process (Monte Carlo method). Every call to $\sigma(v)$ spends $O(rm)$ time. Moreover, each insertion in Q has time $O(1)$. Thus, this loop needs $O(|C|rm)$ operations.

iii) To choose k nodes $O(knrm)$ steps are needed at the "while" loop. This loop is an adaptation of CELF optimization in with the difference that the set of vertices V is replaced by the set C of candidates. As explained by Leskovec et al. (2007), this algorithm has time $O(knrm)$. Since $|C| \leq n$, then we have the same bound.

Therefore, the total running time is the sum of items (i), (ii) and (iii), as follows. $O(n \log n + m) + O(|C|rm) + O(knrm) = O(knrm \log n)$. □

5.3 EXPERIMENTS

We conducted the experiments in two types of datasets, real social networks, and synthetically generated graphs. We compare the PREVALENTSEED and the CELF algorithm by taking into account two metrics: the size of the set of vertices achieved by the spread of influence (i.e., quality of seed set) and running time. In the experiments, the proposed algorithm got significant gains in performance compared to CELF besides preserving the expected spread at a competitive level. Although the CELF++ is faster than CELF in the experiments reported by Goyal et al. (2011b) we chose CELF as baseline because in some empirical evaluations the CELF remains more robust on different types of graph.

The algorithm was implemented in Java using the JGraphT library (jgrapht.org), and all experiments were performed on a machine with GNU/Linux (Linux Mint 17) which hardware configurations were: (i) Processor: Intel(R) Core(TM) i5-3210M CPU, 2.50GHz, x86_64 architecture, and 4 CPU's. (ii) Cache memory: 128KiB L1 cache; 512KiB L2 cache; 3MiB L3 cache. (iii) RAM: 6GiB SODIMM DDR3 Synchronous 1600 MHz (0.6 ns).

5.3.1 Real world power law graphs

To use networks that exhibited structural features of large scale social networks and power-law degree distribution, we consider six graphs to exemplify the results. Table 5.1 summarizes some data about these graphs.

Table 5.1: Statistics information of the social networks. The β values are from Liu et al. (2014) and Tang et al. (2008) results.

Social Network	Vertices	Edges	Exponent (β)
NetHEPT	15,233	32,213	2.651
NetPHY	37,154	180,826	2.843
Enron	36,692	367,662	2.357
Epinions	75,879	508,837	2.383
Amazon	262,111	1,234,877	2.432
DBLP	654,628	1,990,259	3.361

We carried out all the tests in the independent cascade model. To evaluate graphs relatively large, we split the experiments into two categories: the smallest graphs and the largest ones. Due to the long time required to make Monte Carlo simulations, we set the propagation probabilities to $p = 0.025$ on smaller graphs and $p = 0.0025$ on larger graphs. It was necessary because higher probabilities would make the experiments infeasible, as far as the running time is concerned. We simulated the propagation process by 10,000 times for each selected set, as in the literature (Leskovec et al., 2007; Kempe et al., 2003; Goyal et al., 2011b).

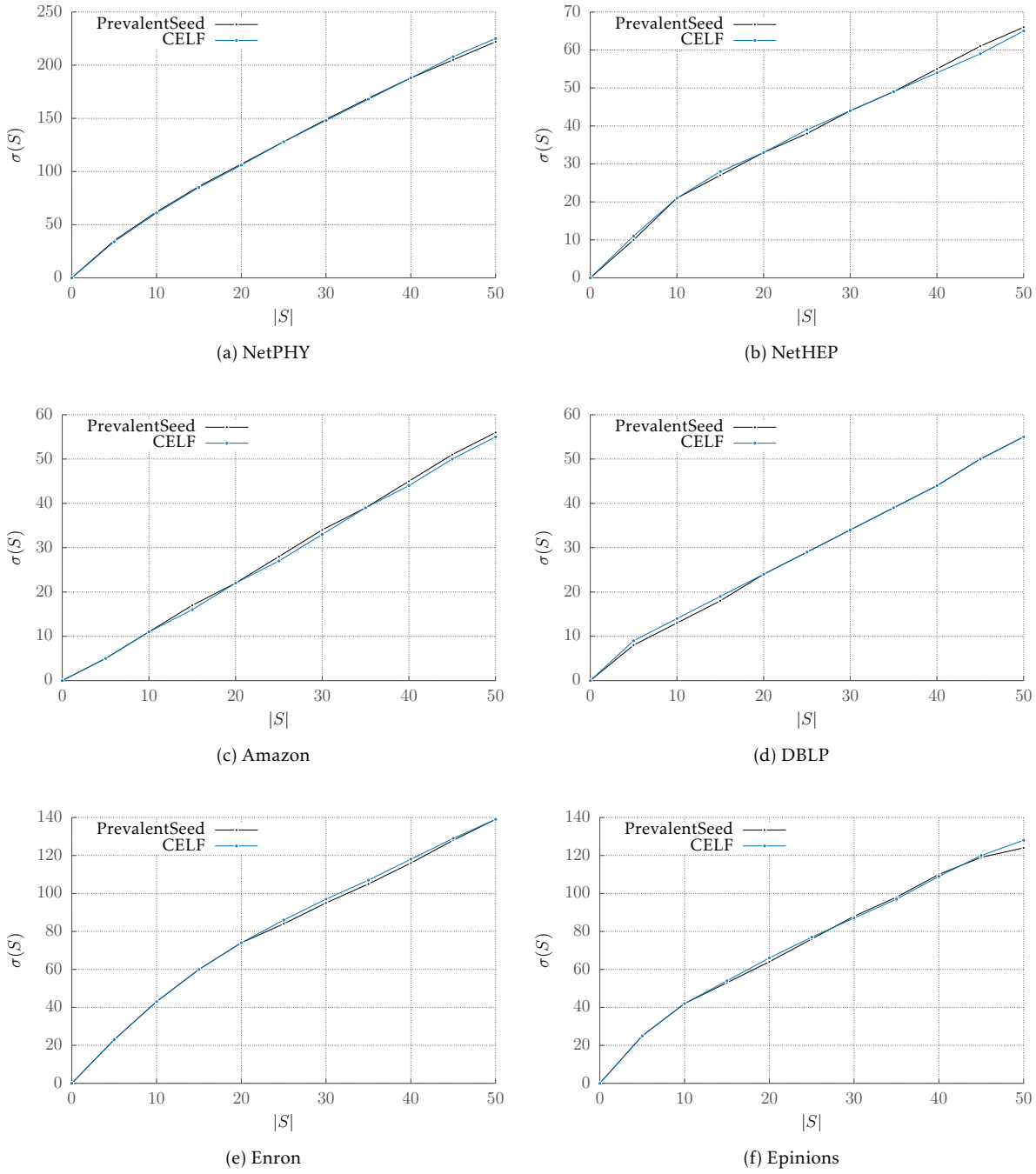


Figure 5.6: The simulations in NetHEP, NetPHY and Amazon have propagation probability $p = 0.025$, and the simulations in DBLP, Enron and Epinions have $p = 0.0025$.

5.3.1.1 *Results and discussion*

The early adopters selected by the algorithms were evaluated based on the number of activated vertices – the higher the spread, the better the quality. In the graphics of Figure 5.6 the algorithms have similar results on the influence propagation, that is, the number of activated vertices are almost the same. Note that despite the difference of probabilities, the results are similar and both algorithms have produced good seed sets.

Concerning the running time, Table 5.2 summarizes the effectiveness of the PrevalentSeed compared to Celf's time when $k = 50$. For simplicity purposes, we kept

Table 5.2: Difference between the running time of CELF and PREVALENTSEED, for $k = 50$.

Network	PREVALENTSEED(s)	CELF(s)	Gain
NetHEP	220.84	315.61	30.02%
NetPHY	3,195.05	5,366.88	40.46%
Amazon	2,438.71	5,679.99	57,06%
DBLP	5,541.06	10,876.31	49,06%
Enron	212,964.45	192,744.20	-10.49%
Epinions	123,392.18	112,237.96	-9,93%

Table 5.3: Calls to the σ function in all tested graphs. Columns 4 and 5 shows the total of reorganizations of Q needed to PREVALENTSEED and CELF, respectively.

Graph	Calls to σ		Reordering of Q	
	PrevalentSeed	Celf	PrevalentSeed	Celf
NetHEP	4389	15370	125	137
NetPHY	8713	37495	213	341
Amazon	107058	262205	93	94
DBLP	117537	654726	96	98
Enron	3617	36861	194	169
Epinions	13367	76055	195	181

only two decimal places of precision. The last column shows how much PREVALENTSEED was faster than CELF. In this case, we achieved a reduction up to 57%, but unfortunately we also got negative results. It is important to note that the result of the two last graphs on Table 5.2 was negative mainly due to a very important fact, which is the size of set C. Table 5.3 explains the reasons behind these results.

In the graphs which the performance of PREVALENTSEED was worse than CELF (Enron and Epinions), the density is higher than in the other graphs. This feature implies that a smaller number of vertices are needed to cover all the graph, that is, the size of C decreases when the number of edges increases. With few candidates, the CELF's priority queue needs to be reorganized more times. To reorganize the priority queue, it is necessary to estimate the marginal gain again, making new calls to the σ function. Since, in this step, the set S is not empty, such computation should be more time consuming because the spread tends to be larger when S grows. Thus, each new call to σ can negatively affect the algorithm's run time. That is why PREVALENTSEED can be worse than CELF.

Table 5.3 presents how many reorganizations were needed to each graph in both algorithms. It is easy to note that PREVALENTSEED behaves badly only in the cases where the number of reordering was greater than in CELF. Even with the notable difference between the total number of calls of the two algorithms, what really impacts the running time are the calls required to reorder the queue. Note that the total of calls is proportional to the number of vertices placed into the queue Q. As the goal of the preselection optimization is to reduce this number of vertex, the PREVALENTSEED makes less calls to σ than CELF at the proportionality of $|C|$.

Fortunately, even with a worst time in some cases, the quality of seed set remained competitive (see on Figure 5.6). Since our preselection heuristic aims to solve

the problem in power-law graphs, we believe that this is not a prohibitive trouble. Based on these findings, we recommend that is enough to pay attention to scale coefficient β. The experiments show that the gain in time reduction is better when $\beta \geq 2.4$. Thus, when the density of the graph is higher, it is more appropriate to use only CELF optimization without PRESELECTOR.

5.3.2 Synthetic graphs

In order to artificially represent realistic social networks, the random graph model used in this study is based on the model of Aiello et al. (2001), denoted as $P(\alpha, \beta)$. The generated networks combine the topology introduced by $P(\alpha, \beta)$ model together with the *Generalized Random Graph* (GRG) model (Van Der Hofstad, 2016). The purpose of this methodology is to generate power law random graphs with an adjustable scale exponent β. Roughly speaking, in the GRG model, a graph starts with a set of n vertices with no edges between them. Each vertex has a weight which determines the probability of having edges. The algorithm adds edges between pairs of vertices according to its weights. Hence, the graph topology depends on the chosen weights, and it can be handled such that the resulting graph has an expected degree according to a desired distribution. So, the $P(\alpha, \beta)$ model provides a well-defined sequence of weights, used as the input of the GRG model, that follows a power-law distribution.

In our experiments, we generated 60 synthetic graphs. The network generation parameters are size and density, in which we vary the number of vertices such that $n = \{2, 4, 8, 16, 32, 64\}$ and the scale exponent was fixed in $\beta = 2.5$. For each n, we perform the experiments on 10 networks, and report the average results to both expected propagation and running time. In all the experiments, we preprocessed the graphs by eliminating the isolated nodes and small components in order to only use the connected graph of the giant component. The propagation probabilities on the edges are draw uniformly at random from the interval $[0, \frac{1}{4}]$. Such settings allow us to perform experiments in feasible time on the IC model with random probabilities.

5.3.2.1 Results and discussion

We plot the performance of PREVALENTSEED and CELF in both metrics, expected propagation and running time. Figure 5.7 presents the average expected propagation of the networks with 64 thousand vertices, where the spread of PREVALENTSEED match almost perfectly with the CELF's spread. This confirms that our heuristic is able to reach the same quality. This matches the intuition from the end of the analysis of the preselection process, which says that without losing the quality of spread, we can select the early adopters within the set C. For the running time, we see that our algorithm does better than CELF. Figure 5.8(a) shows the amount of time required to find a seed set of 50 vertices on the random graphs. The gain in running time for each of the sizes 2k, 4k, 8k, 16k, 32k and 64k of the graphs are 17.2%, 22.42%, 36.4%, 26.61%, 24.4% and 29.67%, respectively.

As Figure 5.8 shows, we observe some interesting points regarding the performance of our algorithm on different size of networks. Figure 5.8(b) presents the average number of calls to σ function. In these settings, the difference between the algorithms is directly related to the size of the set of candidates. Thus, the preselection decreases the quantity of influence estimation along the greedy search. Also, Figure 5.8(b) shows that even with a noteworthy decrease in the number of calls to σ function, the running time

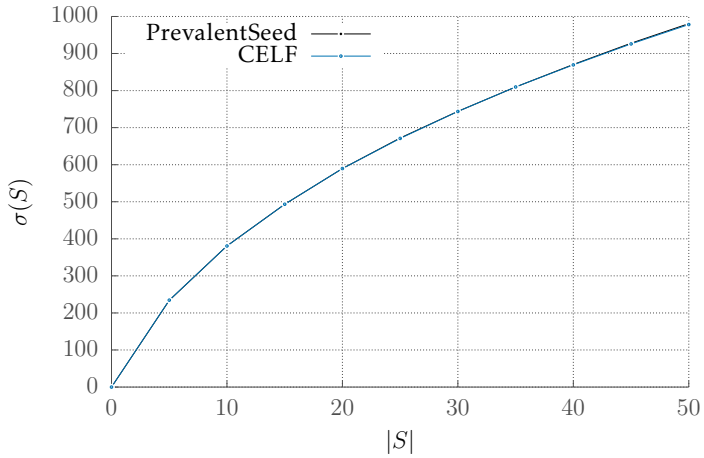


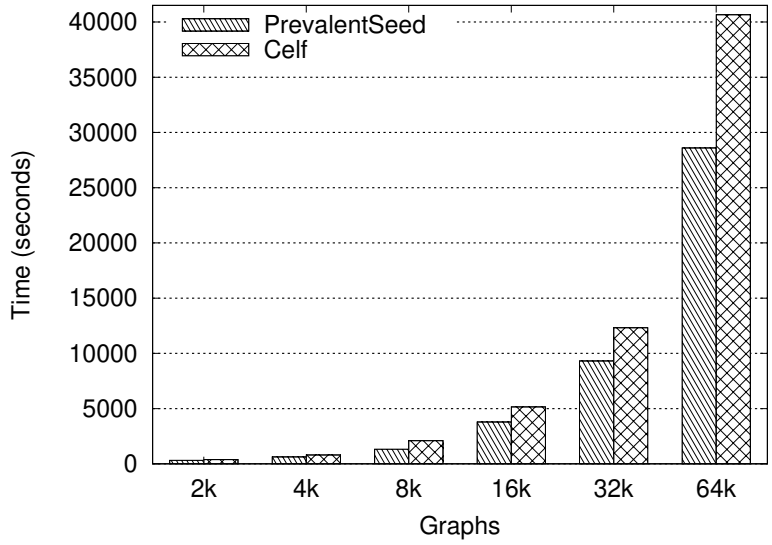
Figure 5.7: Simulations in synthetic graphs with 64 thousand vertices and propagation probability $p \in [0, \frac{1}{4}]$. We plot only one size of graph due to similar results in all tested sizes.

(Figure 5.8(a)) does not decreases proportionally. Again, the comparison between the running time and the number of calls to σ reinforce the idea that the more expensive calls to σ are those performed to reorder the priority queue, as already reported in the real-word network evaluations in Section 5.3.1.

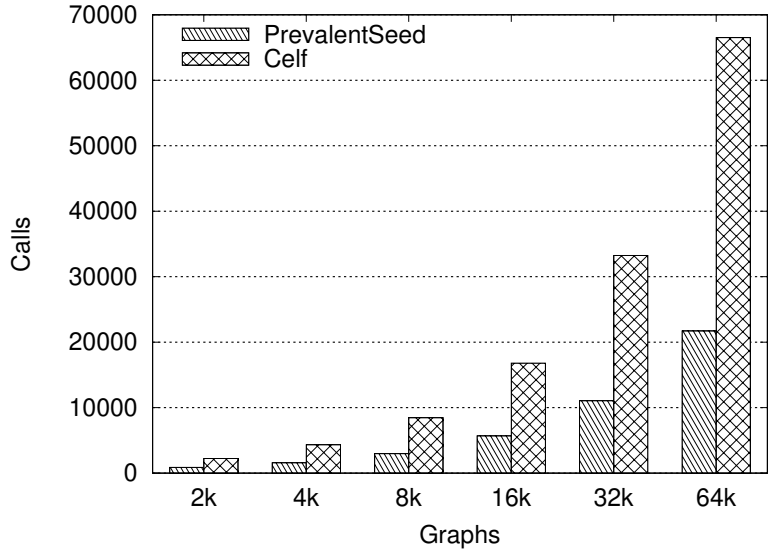
We are aware of new algorithms and heuristics that outperforms our baseline and the PrevalentSeed into this field, for instance IMM (Tang et al., 2015) and TIM+ (Tang et al., 2014), and by the time the thesis is published, some details of our comparison method will be outdated. Nonetheless, our main contribution remains valid because the focus of our work is presenting a preselection methodology specialized in power-law graphs, that can be applied or combined with any greedy algorithm. The purpose, therefore, is not to compete with new algorithms in selecting the seed set, but to offer a way to improve the performance in power-law graphs. The PrevalentSeed algorithm is an example of how the preselection can be applied in a given seed selection algorithm. For this reason, we kept the original Celf as baseline, since our goal is to compare the results with and without the preselection.

The empirical results exposed in this chapter suggests that we can select the k early adopters of the Influence Maximization problem within the set C without losing quality of the spread. This means, in theory, that to solve the problem, we need only to consider a relatively small subset $\{x_u : \forall u \in C\}$ of the variables. Considering, thereby, a integer linear programming model in which we substitute V by C in the branch-and-bound tree, we have a restricted version of the problem. Thus for a problem formulation P_0 it is possible to fix some variables $x_v = 0$ for all vertex $v \in V \setminus C$. So we are constructing a solution that is feasible in the original problem by considering only a subset of the variables.

With some variables fixed, we can possibly apply other preprocessing rules besides the preselection to set the values of more variables. As for example, fixing the variables by logical implications which is commonly used with branch-and-bound method. Therefore, if the resulting integer program (or problem) is stronger (smaller), then it will typically be solved more quickly. An idea in this approach is presented in Chapter 6.



(a) Running time



(b) Calls to σ function

Figure 5.8: Average running time (a) and number of calls to the σ function (b) of both algorithms to find 50 seeds on synthetic graphs.

6 PREPROCESSING RULES FOR TARGET SET SELECTION

In the Target Set Selection (TSS) problem, we want to find a minimum set of individuals to spread information across an entire network. This problem is NP-hard, so finding good strategies to deal with it is something of interest even for a particular case. We introduce preprocessing rules that reduce the input size without losing the optimality of the solution when the input graph is a complex network. Such a type of network has a set of topological properties that commonly occurs in graphs that model real systems. Our strategies do particularly well on graphs with power-law degree distribution, such as several real-world complex networks. Such rules provide a notable reduction in the size of the problem and, consequently, gains in scalability.

Although combining the spread of influence with complex networks is a natural way of studying the problem, we are not aware of other works dealing with the TSS problem in complex networks. In this work, we present preprocessing rules to obtain a partial optimal solution to the TSS problem. In the partial optimal solution, we efficiently (in linear time) identify some vertices that certainly belong to an optimal solution. Having a fraction of the graph solved optimally, obtaining a complete solution depends only on solving the remaining part, i.e., the part of the graph where the partial optimal solution was not obtained. The preprocessing has linear time complexity, indicating that it is worth performing the preprocessing before any strategy chosen later. In particular, due to the nature of the problem, it is of interest to evaluate the behavior of these preprocessing rules in real-world graphs. In this way, through experiments on power-law graphs, we show that our preprocessing rules obtain an optimal partial solution in a significant fraction of the graph, in some cases the preprocessing is sufficient to completely solve the problem. As the problem is NP-hard, an efficient way to solve it is not known. So the main advantage of our new approach is to accelerate the solving process in general graphs whose structure takes advantage of the use of the preprocessing rules, such as complex networks or, more specifically, power-law graphs.

The rest of the chapter is organized as follows. Section 6.1 is devoted to define the notation and the statement of the problem. In Section 6.2, we introduce the preprocessing rules to reduce the instance of the problem. In Section 6.3, we present an experimental evaluation using real-world complex networks and synthetic random graphs and discussion of our results.

6.1 PROBLEM DEFINITION

To represent diffusion of influence over a social network, in this chapter, we consider the threshold model, defined in Section 3.4. Recall that, in this model, a vertex v gets active if at least $t(v)$ of its in-neighbors are active at the previous step, where $t(v)$ is the threshold of v.

A general definition for this problem is in Problem 3. In this chapter, we consider a particular version of that problem in which we want to activate all the vertices from the target set. but we keep the same name to preserve the nomenclature used in the literature. The problem definition is as follows.

Problem 7 (Target Set Selection (TSS)) *Given a directed graph $G = (V(G), E(G))$ with thresholds $t(v)$ on each $v \in V(G)$, find a set $S \subseteq V(G)$ of minimum size, such that the propagation graph G^* is acyclic with $\delta^-(v) \geq t(v)$ for every $v \in V(G^*) \setminus S$.*

6.2 RULES

In this section, we present preprocessing rules to find an optimal partial solution to the TSS problem. As we intend to solve the problem on complex networks, the rules are designed with the properties of these networks in mind. Observe that, in the TSS problem, every vertex with no incoming edges certainly belongs to the optimal solution because no other vertex can activate it. So, we can include these vertices in the partial optimal solution. By the same reasoning, we can exclude from the optimal target set those vertices with no outgoing edges. It is worth noting that in graphs with power law degree distribution, these two types of vertices are the majority. Thus, we can exploit this graph topology by processing the low degree vertices first.

Consider a directed graph G as an instance of TSS problem. Let $S \subseteq V(G)$ be the optimal target set and let $A(S)$ be the set of active vertices at the end of the activation process. We can set the partial optimal solution according to the following propositions.

Proposition 8 *If $\delta^-(v) = 0$ and v not an isolated vertex, then $v \in S$ for all $v \in V(G)$.*

Proof: Suppose, by contradiction, that $v \notin S$. By the activation process in the threshold model, no vertex in $A(S)$ activates v because $N^-(v) = \emptyset$. Therefore, $A(S) = V(G) \setminus \{v\}$ and S is not a solution of the problem, contradicting the hypothesis. □

Similarly to the reasoning of Proposition 8, vertices with $t(v) > \delta^-(v)$ must be in the target set S. So for Proposition 9, we consider that $t(v) \leq \delta^-(v)$.

Proposition 9 *If $\delta^+(v) = 0$ and v not an isolated vertex, then $v \notin S$ for all $v \in V(G)$.*

Proof: Suppose, by contradiction, that $v \in S$ and let $S' = S \setminus \{v\}$. By the problem definition, $A(S) = V(G)$. Since $\delta^+(v) = 0$, v cannot activate any other vertex of G, then the set of vertices activated by S is not greater than the set of vertices activated by S', i.e., $A(S) \subseteq A(S')$. Furthermore, at the end of the activation process, every in-neighbor of v is active, i.e., $N^-(v) \subseteq A(S')$. Thus, v has to be activated unanimously by $N^-(v)$. Consequently, $v \in A(S')$ and $A(S') = V(G)$. This is a contradiction because S is a minimum target set. □

For disconnected graphs, isolated vertex must be selected as target set elements in advance and each connected component can be treated as a separate graph in the formulation.

From Propositions 8 e 9, we derive Algorithm 8. We start by creating a copying graph G' of the input graph G. The set $S \subseteq V(G')$ starts empty and will contain the optimal partial target set (all vertices satisfying Proposition 8). The set $D \subseteq V(G')$ stores the vertices to be removed from G' during the algorithm execution. The propagation graph G^* starts with no arcs but containing all the vertices of G. In this algorithm, we want to remove as many vertices as possible from G', aiming to decrease the size of the

returned problem instance. The first part of the algorithm (lines 4-18) removes every source, sink, and isolated vertices. The condition in lines 5-7 set the isolated vertices to be in the partial solution S, and add them to the set D of vertices to be removed from G'. Based on Proposition 8, lines 8-13 adds every v with no incoming arcs to the partial optimal solution S, and further inserts the outgoing arcs of v to the propagation graph G^*. This can be done because such arcs do not belong to any directed cycle since v is a source vertex. As a consequence, we can decrease by one the threshold of each v's out-neighbor w in G' meaning that v exerts influence over w. Lines 14-17 deals with the case of Proposition 9, indicating that v is not in S if it has no outgoing arc. By the same reasoning, the incoming arcs are set to be in partial optimal solution G^* without risk of generating cycles. In line 18, the vertices in D will be removed from G' as well as the incident arcs to each $v \in D$. More precisely, the graph obtained by deleting D from G' is the subgraph induced by $V(G') \setminus D$, denoted as $G[V(G') \setminus D]$.

The iteration loop in lines 19-34 of Algorithm 8 proceeds by removing more vertices of G' according to the updated threshold $t(v)$ of each $v \in V(G')$. The first conditional considers that if $t(v) \leq 0$ (due steps 12 and 27), then the incoming arcs that were removed (and inserted in the propagation graph) in the early steps (or early iterations) are enough to activate v, so we can exclude the remaining incoming arcs of v of the reduced instance G'. This helps us to avoid cycles. Further, we also force the outgoing arcs of v to be in the solution and decrease the threshold of out-neighbors of v. Again, in lines 29-32, we repeat steps 14-17. We repeatedly remove vertices from G' that enter the conditionals until there are no more vertices to be removed.

Figure 6.1 presents an example of the algorithm execution in a small graph. The labels in each vertex denote the name and the threshold. For instance, the vertex in the upper left corner has the name a and threshold $t(a) = 1$. In the leftmost graph, the dashed vertices and arcs are the sources and sinks to be removed in the first part of the algorithm, lines 4-17. In the second graph, the sources and sinks were removed. Note that, due the line 12, vertex f has threshold 0. So f is inserted in set D to be removed later. Also, the arc (c, f) is excluded from the solution. The arc (c, f) can be excluded because the threshold of f is 0 at this moment, meaning that the arc (g, f), which is already in the solution, is enough to achieve the threshold of f and activate it. Furthermore, by removing the arc (c, f), we are eliminating the cycle (c, f, d) of the solution. Vertex h will be removed because of the condition in lines 29-32. That is, g can activate f, and as a consequence, h can be activated by f, mimicking the diffusion process of the threshold model. In the rightmost graph, the search stops because there is no vertex to be removed. So, the reduced graph G' is the remaining 2-cycle composed by vertices c and d.

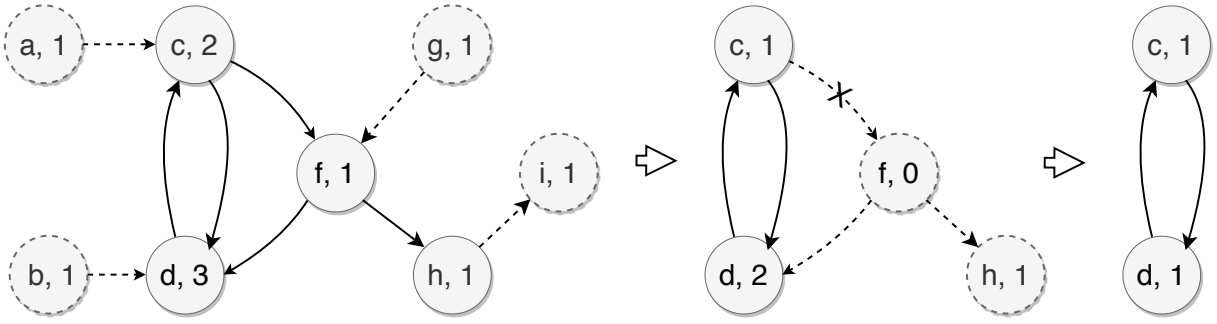


Figure 6.1: Example of the algorithm execution.

Algorithm 8: PREPROCESSING

Input: Graph G, threshold function $t : V(G) \to \mathbb{N}$
Output: Reduced graph G', partial propagation graph G^*, and partial target set $S \subseteq V(G)$

begin
  $G' \leftarrow G$; $S \leftarrow \emptyset$; $D \leftarrow \emptyset$
  Create the graph G^* such that $V(G^*) = V(G)$ and $E(G^*) = \emptyset$
  foreach $v \in V(G')$ **do**
    if $\delta(v) = 0$ ***or*** $t(v) > \delta^-(v)$ **then** `// isolated vertex`
      $S \leftarrow S \cup \{v\}$
      $D \leftarrow D \cup \{v\}$
    else if $\delta^-(v) = 0$ **then** `// source vertex`
      $S \leftarrow S \cup \{v\}$
      foreach $w \in N^+(v)$ **do**
        add arc (v, w) to propagation graph G^*
        $t(w) \leftarrow t(w) - 1$
      $D \leftarrow D \cup \{v\}$
    else if $\delta^+(v) = 0$ **then** `// sink vertex`
      foreach $u \in N^-(v)$ **do**
        add arc (u, v) to propagation graph G^*
      $D \leftarrow D \cup \{v\}$
  $G' \leftarrow G'[V(G') \setminus D]$; `// remove vertices from` G'
  repeat
    $D \leftarrow \emptyset$
    foreach $v \in V(G')$ **do**
      if $t(v) \leq 0$ **then**
        foreach $u \in N^-(v)$ **do**
          remove arc (u, v) from G'
        foreach $w \in N^+(v)$ **do**
          add arc (v, w) to propagation graph G^*
          $t(w) \leftarrow t(w) - 1$
        $D \leftarrow D \cup \{v\}$
      else if $\delta^+(v) = 0$ **then** `// sink vertex`
        foreach $u \in N^-(v)$ **do**
          add arc (v, w) to propagation graph G^*
        $D \leftarrow D \cup \{v\}$
    $G' \leftarrow G'[V(G') \setminus D]$
  until $D = \emptyset$;`// there are no vertices to be removed`

Theorem 10 bounds the time complexity of Algorithm 8.

Theorem 10 *For a directed graph G with $|V(G)| = n$ and $|E(G)| = m$, Algorithm 8 takes $O(m)$ time in the worst case.*

Proof: At line 2, the time complexity for cloning the graph is $O(m)$. Next, creating G^* demands $O(n)$ to copy the vertices of G. The loop in lines 4-17 visits every vertex once, and for each vertex, it iterates through its set of neighbors, overall, this takes $O(m)$. At line 18, given a set of vertices marked for removal, each vertex must be removed along with its incident edges. In the worst case, it takes $O(m)$ time. Next, we argue that the block in lines 19-34 takes $O(m)$ time. Each vertex in $V(G')$ meets the conditionals of lines 22 or 29 at most once because if it meets any of the conditionals, then it is added to D, and immediately afterward it is removed from $V(G')$. For each vertex that meets the conditionals, it iterates through its set of neighbors. Thus, overall, the time spent is proportional to the sum of the degrees, i.e., $O(m)$. However, the loop at line 21 must be carefully implemented, as if in every iteration of line 21, we go through all the vertices, then we could end up with a quadratic time algorithm. We can overcome this issue with a careful implementation, where we keep track of only those vertices that meet the conditionals so that we do not iterate over unnecessary vertices. Finally, the analysis of line 33 is similar to line 18, where the accumulated time over all iterations in the worst case is $O(m)$. □

It is noteworthy to mention the fact stated in Theorem 11.

Theorem 11 *If $V(G') = \emptyset$ when the Algorithm 8 ends, then the propagation graph defined by the arcs chosen in the algorithm is an optimal solution for the TSS problem.*

Proof: As argued before, the vertices in S belong to an optimal target set. Besides, the removed vertices are related to the activated vertices in the diffusion process, so removing all the vertices of G' means that the whole graph is activated. Therefore, we have an optimal partial solution that activates all the vertices in the graph, i.e., the partial solution is optimal, and S is an optimal target set. □

6.2.1 Expected reduction on power-law graphs

The success of Algorithm 8 depends on the topology of the input graph. For example, it has no effect on strongly connected graphs and in graphs with no sources or sinks. On the other hand, the algorithm can perform well in power-law graphs, because in this case, the majority of the vertices has low degree, with many having only one incoming arc or one outgoing arc. Moreover, given a power-law graph model, for example, $P(\alpha, \beta)$ presented by Aiello et al. (2001), it is easy to quantify the expected number of vertices belonging to this category of low degree vertices. In real-world networks, the power-law exponent β is typically in the range $2 < \beta < 3$ (Choromanski et al., 2013). Thus, we can quantify the expected decrease in the instance size after the preprocessing. On the $P(\alpha, \beta)$ model, the number of vertices with degree 1 is e^α, and $|V|$ is (asymptotically) $e^\alpha \zeta(\beta)$, where $\zeta(\beta)$ is the Riemann zeta function. Therefore, the vertices of degree 1 correspond to $\frac{1}{\zeta(\beta)}$ of the total number of vertices. Assuming $2 < \beta < 3$, from 60.7% to 83.1% of the vertices of the graph have degree 1. A vertex of degree 1 has either

incoming degree or outgoing degree equal to zero, therefore, regardless of the case, it is always removed from the original graph by Algorithm 8. In addition to the number of vertices with degree 1, there are also vertices with degree greater than 1 which has no in-edges (or out-edges). Therefore Algorithm 8 sets a huge fraction of vertices as belonging or not to the optimal partial solution, significantly reducing the size of the instance. These properties have also been empirically confirmed, as we show in Section 6.3.

6.3 COMPUTATIONAL EXPERIMENTS

In this section, we proceed with the experimental evaluation. The experiments were launched in an Intel(R) Core(TM) i5-3210M CPU @ 2.5GHz and 4 GB RAM. The algorithms were implemented in Java 11 language. For graph manipulations, we use the JGraphT 1.4 library (Michail et al., 2019).

6.3.1 Instance size reduction

To demonstrate the behavior of Algorithm 8 in reducing the size of the input graph, we consider some real-world complex networks obtained from the Stanford Large Network Dataset Collection (Leskovec and Krevl, 2014). Below, we give a brief description of each considered network.

- Bitcoin-alpha: Who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin Alpha.
- Bitcoin-OTC: Who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin OTC.
- Wiki-vote: Contains all the Wikipedia voting data from the inception of Wikipedia until January 2008. Vertices represent users and a directed edge from u to v represents that user u voted on user v.
- DBLP: Citation network of DBLP, a database of scientific publications such as papers and books.
- Reddit: Network of subreddit-to-subreddit hyperlinks extracted from hyperlinks in the body of the post.
- Epinions: A who-trust-whom online social network of a general consumer review site Epinions.com.
- Slashdot09: Slashdot social network from February 2009.
- Email-EuAll: Email network of a large European Research Institution (between October 2003 and March 2005).

To show the results in different scenarios, we consider three distinct threshold functions, in line with the threshold scenarios considered in (Chen et al., 2009; Ackerman et al., 2010) – recall that here we do not consider the probabilistic models. The threshold functions are the majority threshold,the low threshold and the high thresholds.

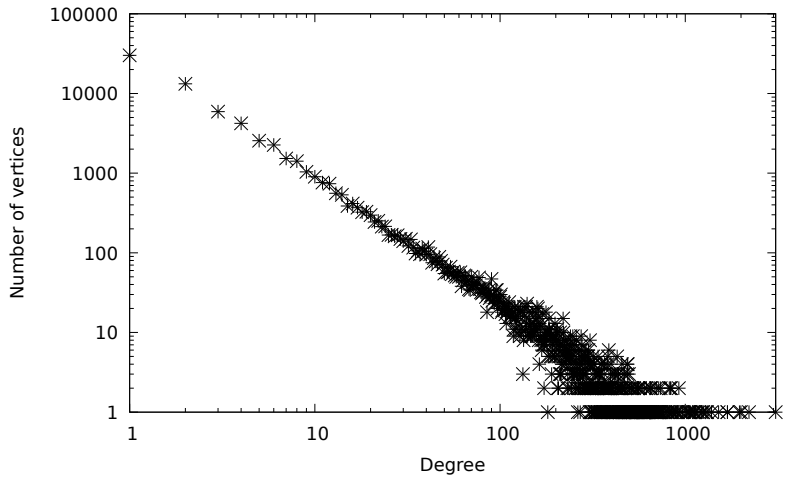
1. The majority threshold is defined as $t(v) = \left\lceil \frac{\delta^-(v)}{2} \right\rceil$, for every $v \in V$;
2. The low threshold is $t(v) = \left\lceil \frac{\delta^-(v)}{4} \right\rceil$, for every $v \in V$;
3. The high thresholds are defined as $t(v) = \left\lceil \frac{3}{4}\delta^-(v) \right\rceil$, for every $v \in V$.

Table 6.1: Reduction in size of real social networks after preprocessing.

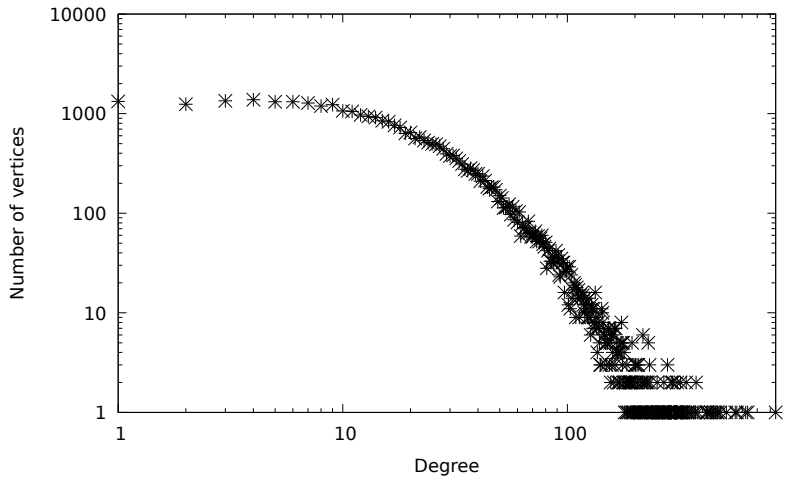
Network		Original	Reduction		
			Majority Threshold	Low Threshold	High Threshold
Bitcoin-alpha	$\|V\|$	3,783	3,251 (14.1%)	-	-
	$\|E\|$	24,186	23,333 (3.5%)	-	-
Bitcoin-OTC	$\|V\|$	5,881	4,763 (19.0%)	-	-
	$\|E\|$	35,592	33,640 (5.5%)	-	-
Wiki-vote	$\|V\|$	7,115	0 (100%)	-	81.9%
	$\|E\|$	103,689	0 (100%)	-	62.1%
DBLP	$\|V\|$	12,591	2 (99.9%)	-	99.5%
	$\|E\|$	49,728	2 (99.9%)	-	99.7%
Reddit	$\|V\|$	35,776	9,854 (72.5%)	99.7%	67.0%
	$\|E\|$	137,821	88,343 (35.9%)	99.9%	28.5%
Epinions	$\|V\|$	75,888	32,088 (57.7%)	98.4%	54.3%
	$\|E\|$	508,837	438,163 (13.9%)	99.7%	11.7%
Slashdot09	$\|V\|$	82,168	71,862 (12.5%)	-	12.5%
	$\|E\|$	870,161	842,217 (3.2%)	-	3.2%
Email-EuAll	$\|V\|$	265,214	56 (99.9%)	-	88.5%
	$\|E\|$	418,956	104 (99.9%)	-	69.7%

Table 6.1 shows the reduction of the instance size obtained after the preprocessing in graphs representing real-world networks. For each graph, we present the number of vertices ($|V|$) and arcs ($|E|$) in the original graph and the reduction after the preprocessing on each considered scenario. Column "majority threshold" shows the absolute size (number of vertices and edges) of the remaining graph together with the percentage of the reduction after applying the preprocessing rules. For example, in the Wiki-vote network, we get a reduction of 100% in the size of both vertex and edge set and thus 0 remaining vertices and edges. This means that we have the best possible case, i.e., in this example, the problem is completely solved by our preprocessing strategy. The last two columns contain the results for the low and high threshold scenarios, and we show the percentage of the reduction. Dashed cells mean that the reduction is exactly the same as the first scenario (majority threshold), this similarity happens because in most cases only the sources and sinks were eliminated by the preprocessing rules. Note that, with low thresholds, the gains are more expressive and, on the contrary, when the thresholds are high the reduction is less noticeable.

Figure 6.2 shows the degree distribution, in logarithmic scale, of two different real-world networks, Epinions, in Figure 6.2(a), and HEP-PH (citation graph of High Energy Physics Phenomenology (Leskovec and Krevl, 2014)), in Figure 6.2(b). The straight-line shape in Figure 6.2(a) indicates that the network is a power law graph



(a) Straight shape.



(b) Curved shape.

Figure 6.2: Degree distribution in logarithmic scale of real world networks.

(Mitzenmacher and Upfal, 2005; Clauset et al., 2009), while a curved shape says that it is not a power-law. In our experiments of Table 6.1 we only use networks that are power-law. However, we note that many real networks present a "curved" degree distribution as in Figure 6.2(b). For such cases, we also experimentally observed a substantial reduction in the size of the graph. We believe that this good behavior is because there are even more vertices of a low degree than in the case with a straight degree distribution.

6.3.2 Scalability

Motivated by the large sizes of real-world networks, it is necessary to understand the execution time behavior of the algorithm as the size of the network increases, i.e., the scalability of the algorithm. To experiment with networks of varying sizes, we use synthetic graph models for power-law graphs. More specifically, among the various models of random power-law graphs, we choose the Bollobás model (Bollobás et al., 2003), as it is tailor-made to generate power-law directed graphs. In this model, the graph grows one edge per step, according to the probabilities a, b and c, where $a + b + c = 1$. Here, we set these values as $a = b = 0.33$, and $c = 0.34$. Were generated ten graphs of each size and considered the average running time.

Figure 6.3 illustrates how is the growth curve of the running time of Algorithm 8. The size of graphs ranges from 1,000 up to 50,000 vertices. The running time is the average between 10 generated graphs for each size. The line in the plot is not smoothed or straight in shape due to the precision of milliseconds, but it has a pattern that suggests linear growth. So Figure 6.3 empirically confirms the result stated theoretically in Theorem 10, that is, the increasing of the running time of Preprocessing is linear in the size of the input graph.

Taking advantage of the fact that we did experiments on synthetic graphs, it is worth mentioning the performance of our algorithm concerning the reduction in the size of the instance. In such a case, for 500 generated graphs, more than 99% were solved by the Preprocessing. Only 0.02% resulted in a not empty reduced graph G', even in this case, the resulting graph G' is a cycle of size 2 or 3, which is trivial to solve. This happens mainly because our strategy naturally behaves better in sparse graphs, and the random graphs generated by the model of Bollobás et al. (2003) are usually very sparse. However, it is worth analyzing whether other models of power-law graphs maintain this behavior.

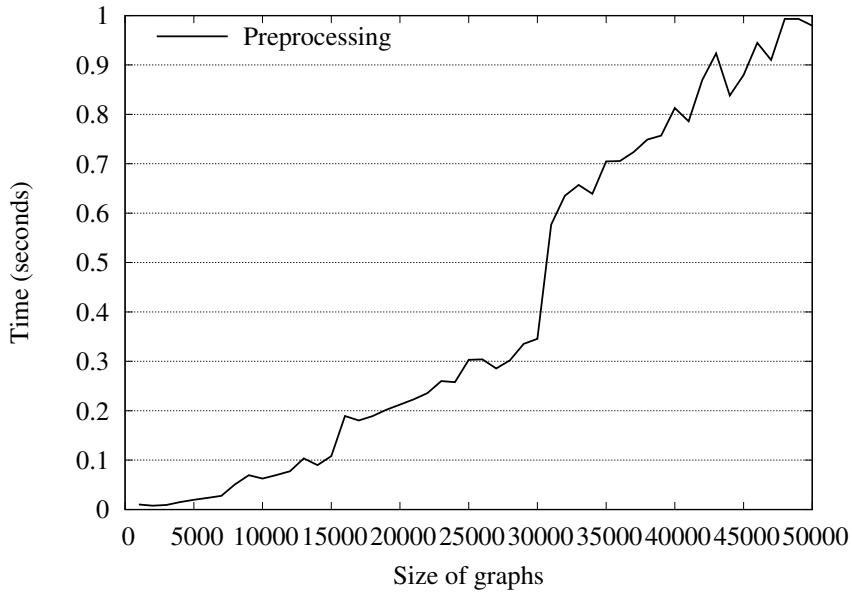


Figure 6.3: Order of growth of the running time of the Preprocessing algorithm.

7 TIGHTER DUAL BOUNDS FOR LEAST COST INFLUENCE PROBLEM

The problems discussed in the previous chapters consist of identifying a good set of individuals to target, the early adopters, and hope that the chosen individuals can persuade their friends into adopting a new behavior, who, in turn, also influence friends of friends by generating a chain of adoption. However, in applications like viral marketing, for example, the early adopters can be influenced by receiving products for free and getting discounts for buying a new product. This chapter investigates an extension of the TSS problem called Least Cost Influence Problem (LCIP). Instead of searching for individuals to start the propagation, this problem consists of offering incentives to trigger a cascade that spreads to a given fraction of the network.

While previous works provided relevant exact solutions (Ackerman et al., 2010; Fischetti et al., 2018; Günneç et al., 2016) and heuristic algorithms that can be used as upper bounds for this problem (Chen et al., 2009; Cordasco et al., 2015; Demaine et al., 2014; Kempe et al., 2003), nothing beyond the standard linear programming relaxation was proposed to compute lower bounds on the LCIP.

We derive a problem-dependent relaxation algorithm based on the observation that the influence propagation network is a directed acyclic graph (DAG). The proposed algorithm exploits connectivity properties of graphs to obtain a lower bound for the problem. Furthermore, we prove that the algorithm is correct and show experimentally that our lower bounds are tighter than the linear programming relaxation, providing smaller optimality gaps. To the best of our knowledge, there are no works on combinatorial lower bounds for this problem. The main objective of our relaxation is for fathoming in a branch-and-bound algorithm and helping reduce the computational effort to obtain exact solutions.

We also provide a theoretical analysis of the complexity of our algorithm when dealing with a particular case of the problem, where the diffusion needs to reach only a fraction of the network instead of the whole network. In this case, our analysis leads to a related problem in which its optimal solution is a dual bound for the original problem. This related problem is NP-hard, but it paves the way for approximate our heuristic approaches to improve the dual bound.

To make our dual bound stronger, we also propose a branching rule that prioritizes the exploration of some branches of the decision tree that improves our algorithm. These improvements are obtained by exploring branches that generate disconnected sub-graphs associated with sub-problems of the decision tree. Furthermore, we make use of preprocessing strategies designed to handle real-world social networks. In such cases, our experiments show that these strategies have proved to be helpful.

The rest of this chapter is organized as follows. Section 7.1 contains the problem definition and a brief overview about the diffusion process considering payment of incentives. Section 7.2 is devoted to describe the mathematical programming formulation of the problem and the exact method to solve it. In Section 7.3, we propose an algorithm to find lower bounds on the problem. We introduce a branching rule designed to strengthen our dual bound algorithm in Section 7.4. In Section 7.5, we describe how to adapt a set of preprocessing rules for the problem. Computational experiments are presented in Section 7.6.

7.1 PROBLEM DEFINITION

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

The diffusion model considered in this chapter is the threshold model with incentives. Both the diffusion model and the LCIP problem are defined in Section 3.5. So, every $i \in V$ has a threshold $t_i > 0$ which indicates the amount of influence needed to activate i, coming from i's neighbors. The number $y_i \in \mathbb{N}_0$ denotes the amount of incentive given to a vertex $i \in V$. Applying the incentive y_i on a vertex i decrease its threshold t_i and make it more susceptible to be activated.

In the remainder of this text we will refer explicitly to the graph associated with a solution $\mathbf{y}$. We say that the propagation graph G^* (Definition 14) is the graph induced by the solution $\mathbf{y}$. The graph in Figure 3.5(e) is an example of propagation graph. Recall that the propagation graph must be acyclic.

7.2 INTEGER LINEAR PROGRAMMING FORMULATION

There are different integer linear programming (ILP) formulations that model the propagation using variables on the arcs (Ackerman et al., 2010; Günneç et al., 2016; Raghavan and Zhang, 2015). The following formulation is a special case of the model proposed by Fischetti et al. (2018).

For the model bellow, we consider three sets of decision variables, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$:

- For each vertex $i \in V$, let x_i be a binary variable that indicates whether i is active at the end of the diffusion process;
- Similarly, for each arc $(i, j) \in E$, let z_{ij} be a binary variable that indicates whether i exerts influence over j.
- As mentioned previously, the integer variable y_i is the amount of incentive to be paid to a vertex $i \in V$.

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

$$\text{s.t.} \sum_{i \in N_j} z_{ij} d_{ij} \geq t_j x_j - y_j \qquad \forall j \in V, \tag{7.1}$$

$$\sum_{(i,j) \in C} z_{ij} \leq \sum_{i \in V(C) \setminus \{k\}} x_i \qquad \forall k \in V(C), \text{cycle } C \subseteq E, \tag{7.2}$$

$$z_{ij} \leq x_i \qquad \forall (i, j) \in E, (j, i) \notin E, \tag{7.3}$$

$$\sum_{i \in V} x_i \geq \lceil \alpha |V| \rceil, \tag{7.4}$$

$$x_i \in \{0, 1\} \qquad \forall i \in V, \tag{7.5}$$

$$y_i \in \mathbb{N}_0 \qquad \forall i \in V, \tag{7.6}$$

$$z_{ij} \in \{0, 1\} \qquad \forall (i, j) \in E. \tag{7.7}$$

The objective function minimizes the total amount of incentives offered to influence a given portion of the network. Constraints (7.1) models the condition that a vertex $i \in V$ gets active only when the total of influence received from its active neighbors plus its incentive is greater than or equal to its threshold. The cycle elimination constraints in (7.2) generalizes the classic cycle elimination from (Grötschel et al., 1985), and impose that the propagation graph of a solution must be acyclic. Meaning that the number of chosen arcs in a cycle C cannot be greater than the number of active vertices in $V(C) \setminus \{k\}$, where $V(C)$ is the set of vertices in the cycle. Constraints (7.3) ensures that an arc (i, j) can be chosen only if vertex i is activated. Note that if there is an arc (j, i) the cycle of size two is eliminated by constraints of type (7.2). Therefore, these constraints are needed only when (i, j) is not in a 2-cycle. Finally, constraints (7.4) impose that, by the end of the diffusion process, the number of active vertices is at least $\lceil \alpha |V| \rceil$.

Due to the number of possible cycles in the graph G, the number of constraints in (7.2) grows exponentially. The standard exact procedure for solving integer linear programs with an exponential number of constraints is the branch-and-cut method, which is a combination of LP-based branch-and-bound and constraint generation techniques. To generate the cycle elimination constraints, we need to solve the separation problem for the inequalities in (7.2). In our approach, we implement the separation procedure proposed by Grötschel et al. (1985). In short, the procedure adapts a shortest path algorithm to find a cycle that violates the cycle elimination constraints.

7.2.1 Separation of generalized cycle elimination

A solution to the problem should represent the influence propagation in the network, the set of arcs indicating who influenced whom. Due to the temporal aspects of the propagation model, the graph representing a solution cannot have directed cycles. To ensure this property, we should avoid any solution with cycles. This is done in constraint (7.2). Since the model have a constraint for every possible cycle in the input graph, the number of constraints can increase exponentially. It can be prohibitive to put explicitly all the constraints in the model, and we need to use a branch-and-cut approach. In order to use the branch-and-cut, we need to solve the separation problem to dynamically add the violated constraints in the model – for a formal definition of separation problem, see Definition 10.

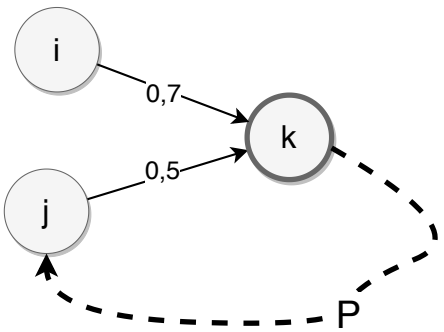


Figure 7.1: Searching for short paths from a vertex k to its incoming neighbors N_k.

Let $\bar{x}$ and $\bar{z}_{ij}$ be a fractional solution of model (7.1 - 7.7). For easy of clarification we can rewrite the constraints (7.2) in the following way.

$$\sum_{(i,j)\in C} z_{ij} - \sum_{i\in V(C)\setminus\{k\}} x_i \leq 0 \quad \forall k \in V(C), \forall \text{ cycle } C \subseteq E,$$

$$\sum_{(i,j)\in C} z_{ij} - \sum_{i\in V(C)} x_i \leq -x_k \quad \forall k \in V(C), \forall \text{ cycle } C \subseteq E,$$

$$\sum_{i\in V(C)} x_i - \sum_{(i,j)\in C} z_{ij} \geq x_k \quad \forall k \in V(C), \forall \text{ cycle } C \subseteq E. \tag{7.8}$$

The procedure consists in adapting a shortest path algorithm as follows. First, we define new function of weights $w : E \to \mathbb{R}$ for the arcs of G in the current solution as $w_{ij} = \bar{x}_i - \bar{z}_{ij}$ for every arc $(i, j) \in E$. Note that due to the constraint (7.3) we do not need to worry about negative weights on the arcs because $w_{ij} \geq 0$. Then, consider $P = (k, ..., j')$ as a shortest path from k to j' and $j' \in N_k$ for every $k \in V$. In this way, if

$$\sum_{(i,j)\in P} w_{ij} + w_{j'k} < \bar{x}_k, \tag{7.9}$$

then we found a cycle $C = P \cup (j', k)$ that violates the generalized cycle elimination constraint. Observe that the left hand side from Equation 7.9 is equivalent to the left hand side of Equation 7.8. As we have no negative cycles in the graph with weights w on arcs, an algorithm like Floyd-Warshall, can be used to find the shortest paths and generate the cutting planes.

7.2.2 Tighter Bounds

Since we are looking for the optimality conditions that will provide stopping criteria, an important method is to find lower (dual) bound $\underline{z} \leq z$ and an upper (primal) bound $\bar{z} \geq z$ such that $\underline{z} = z = \bar{z}$, where z is the optimum value for the objective function of our problem. Every feasible solution provides an upper bound, while for dual bounds the most common approach is by relaxing the integrality constraints of the original problem (see Section 2.3.1 for a detailed explanation about both the branch-and-bound and the branch-and-cut algorithms). Our combinatorial relaxation can be used at each node of the branch-and-bound tree to obtain lower bounds.

7.3 LOWER BOUND ALGORITHM

Consider the following aspects of the LCIP.

i) For any solution, at least one vertex needs to be paid the whole threshold value. This follows from the fact that for any solution $\mathbf{y}$, the associated propagation graph is a DAG, which means that it has at least one vertex with no incoming arcs (source), i.e. $N_v = \emptyset$.

ii) In the best possible case, the vertices chosen to receive the total incentive have the minimum threshold, meaning that they need less incentive. Here, we are interested in the incentive value, not the vertex that receives the cheapest incentive.

We introduce a combinatorial algorithm to obtain a dual (lower) bound for this problem. The idea consists in using connectivity properties of a sub-graph of the

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

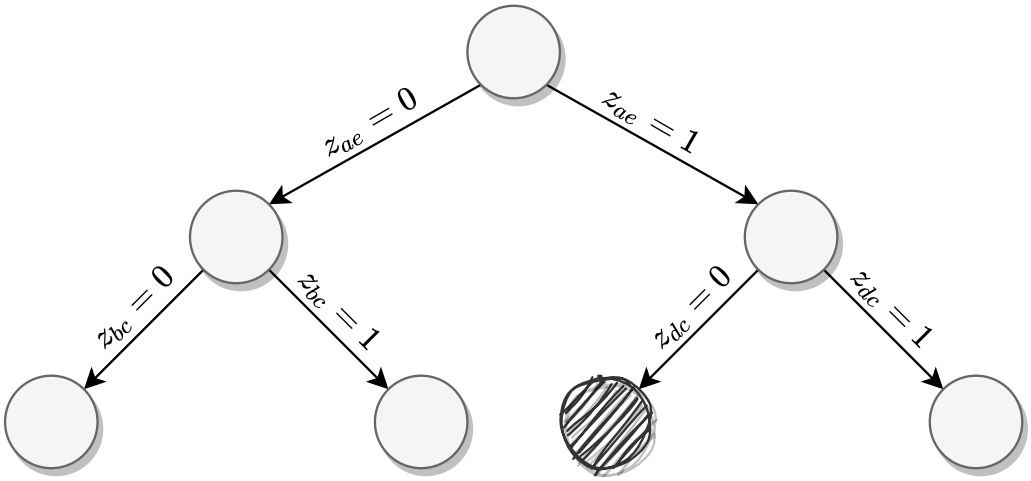


Figure 7.2: The decision tree where the branching decisions are made on binary variables. The black node represents the sub-problem obtained by fixing variables $z_{ae} = 1$ and $z_{dc} = 0$.

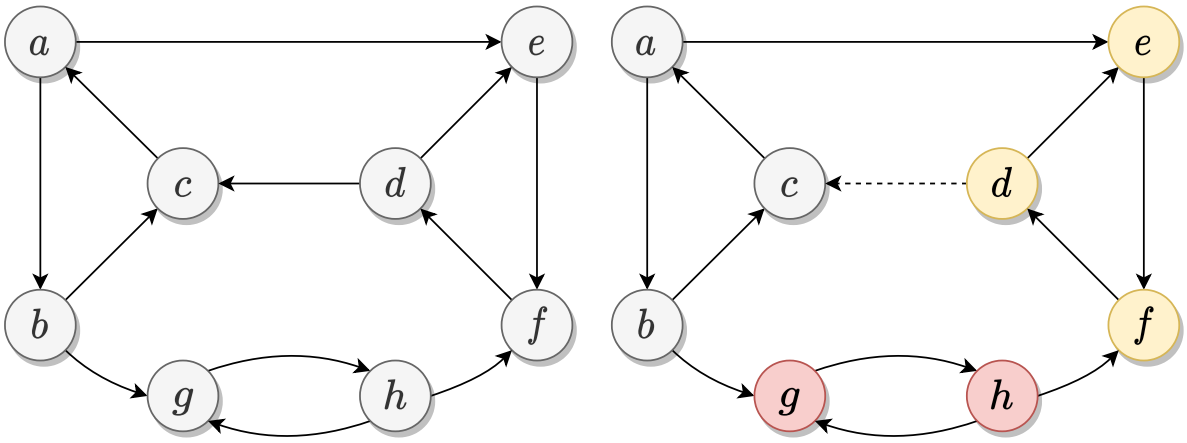
To illustrate the idea behind this strategy, consider the following example. Let the directed graph in Figure 7.3(a) be the input graph of LCIP. We start the branch-and-bound tree by fixing some arc variables. Figure 7.2 shows the first levels of such structure, where the black node represents the subproblem obtained by fixing the arc variables in $z_{ae} = 1$ and $z_{dc} = 0$. We can represent the subgraph associated with this node by removing the arc (d, c) from the original graph, because this arc cannot be chosen in a solution of this subproblem. In this way, we arrive at the subgraph in Figure 7.3(b), which is no more strongly connected and decomposed into three different strongly connected components.

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

Definition 18 (Condensed Component Graph) *A condensed component graph H of a directed graph G is obtained by contracting the strongly connected components (s.c.c.) of G. More formally, each $v \in V(H)$ is associated with a distinct s.c.c. C_v of G and there is an arc $(u, v) \in E(H)$ if and only if there exists an arc from a vertex $i \in V(C_u)$ to a vertex $j \in V(C_v)$, where C_u and C_v are the s.c.c.'s associated with u and v, respectively.*

At each node of the branch-and-bound tree, a different sub-graph G' of the input graph G is considered. The graph G' is obtained from G by removing the arcs and vertices which were fixed at zero by the decision tree. That is,

$$V(G') = V(G) \setminus \{i \in V(G) : x_i \text{ is fixed in zero}\}$$



(a) strongly connected graph

(b) not strongly connected graph

Figure 7.3: The directed graph in Figure (b) is obtained by removing the arc (d, c) of the graph in Figure (a).

and

$$E(G') = E(G) \setminus \{(i,j) \in E(G) : z_{ij} \text{ is fixed in zero}\},$$

at the current node of the branch-and-bound tree.

Let H be the condensed component graph of G'. From now on, the graph H we consider has the arc weights and vertex thresholds defined as follows. Consider C_u and C_v be the s.c.c.'s associated with $u, v \in V(H)$ and

$$E_{uv} = \{(i,j) \in E(G') : i \in V(C_u) \text{ and } j \in V(C_v)\}.$$

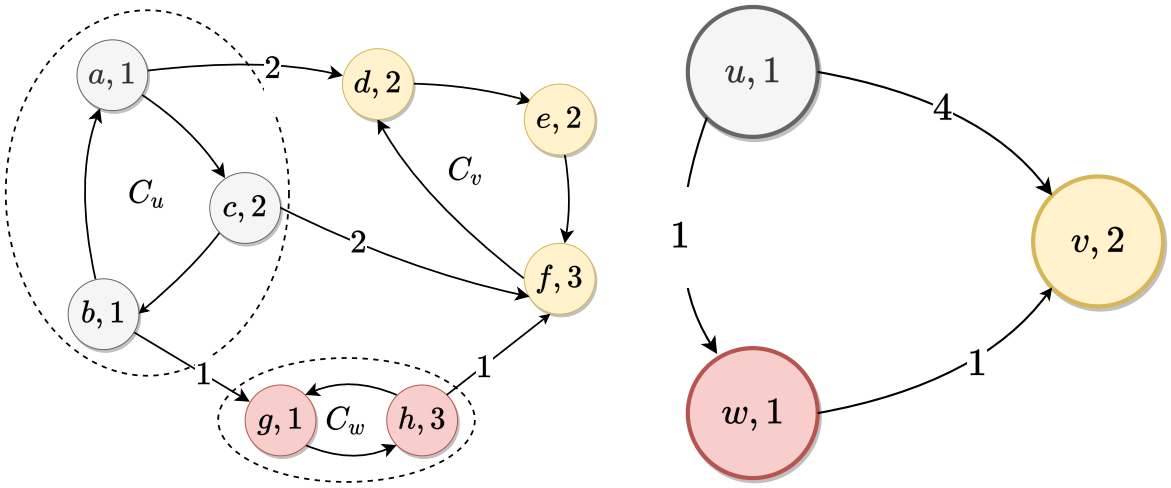
We set the weight of each arc $(u, v) \in E(H)$ to be the sum of all arc weights that go from C_u to C_v, that is,

$$d_{uv} = \sum_{(i,j) \in E_{uv}} d_{ij}.$$

Furthermore, for each vertex $v \in V(H)$ we set $t_v = \min_{i \in V(C_v)} \{t_i\}$.

Figure 7.4 presents an example for the condensed component graph of a small graph. The labels in each vertex of the figure denote the name and the threshold respectively. For instance, the vertex in upper left corner has name a and threshold $t_a = 1$. In the leftmost graph, there are three strongly connected components C_u, C_w and C_v. For the sake of simplicity, we only show the arcs weights between different components in the leftmost graph. In the second graph, we have the condensed component graph with the new thresholds and weights on arcs. For instance, the arc (u, v) has weight $d_{uv} = d_{ad} + d_{cf} = 4$ and the vertex v has threshold $t_v = \min\{2, 2, 3\}$.

The algorithm to find a dual bound for the LCIP follows:



(a) Weighted graph G' not strongly connected.

(b) Condensed component graph H of G'.

Figure 7.4: Graph G', in (a), has three different strongly connected component. The components are C_u, C_v and C_w with set of vertices $\{a, b, c\}$, $\{d, e, f\}$ and $\{g, h\}$, respectively. Graph H, in (b), has three vertices, u, v and w, associated to C_u, C_v and C_w, respectively. In both figures, the vertices are labeled with name and threshold. For example, vertex $v \in V(H)$ has threshold $t_v = 2$, which is the smallest threshold in the component C_v.

Algorithm 9: COMBINATORIAL LOWER BOUND

Input: G', influence d on arcs, threshold t on vertices, and α
Output: A lower bound l for LCIP

1 **begin**
2 **if** *G' is strongly connected* ***or*** $\alpha < 1$ **then**
3 **return** $\min_{i \in V(G')} \{t_i\}$.
4 **else**
5 Let H be the condensed component graph of G'.
  `// Return the optimum value of LCIP on H`
6 **return** $\sum_{v \in V(H)} \max\{0, t_v - \sum_{u \in N_v} d_{uv}\}$; `// Theorem 13`

Let l^{LP} be the lower bound obtained at the current node of the branch-and-bound tree by standard LP-relaxation, and let l be the lower bound obtained by the procedure described in Algorithm 9. When updating the lower bound $\underline{l}$ at the current node, we simply do $\underline{l} = \max\{l^{LP}, l\}$.

Theorem 12 *Given a directed graph G with n vertices and m arcs, Algorithm 9 finds a lower bound for LCIP in $O(n + m)$ time.*

Proof: To check if G is strongly connected, in line 2, we can use Kosaraju's algorithm to compute all the strongly connected components in $O(n + m)$ time. If G has only one strongly connected component, G is strongly connected. In line 3, finding the minimum threshold can be done in $O(n)$ time.

In line 5, to construct the condensed component graph H, we need to compute the strongly connected components of G. Here, we use the Kosaraju's algorithm again. Finally, step 6 of the algorithm can be computed in $O(n + m)$. □

Algorithm 9 is divided into two cases. In the first case, the sub-graph G' is strongly connected, and the lower bound l is trivial by the observations (i) and (ii) at the beginning of this section. When G' is not strongly connected, things become more complicated and require a more detailed examination of the problem elements and structure in the condensed component graph H. To keep the algorithm efficient, we return the trivial lower bound if $\alpha < 1$, even if G' is not strongly connected. Finding a tighter lower bound in this case may be computationally costly. Consequently, the algorithm would no longer be scalable. In Section 7.3.1, we explain this situation in detail.

When $\alpha = 1$ and G' is not strongly connected, we need to compute the total cost for activating all the vertices in the graph H (step (6) of Algorithm 9), as a sub-problem. As H is a condensed component graph of G', then H is directed and acyclic. Therefore, we generalized the algorithm proposed by Günneç et al. (2016) to get the exact solution of LCIP in DAGs in linear time, as Theorem 13 states.

Theorem 13 *If $\alpha = 1$ and the input graph H is a DAG, step (6) of Algorithm 9 gives an exact solution for the LCIP on H.*

Proof: As $\alpha = 1$, every $v \in V(H)$ is active in an exact solution, in particular, each $u \in N_v$ is active. Thus, every incoming arc $(u, v) \in E(H)$ of v can be selected, because there are no cycles in H restricting such selection. Therefore, v receives $\sum_{u \in N_v} d_{uv}$ of influence. If

$$\sum_{u \in N_v} d_{uv} > t_v$$

then the influence coming from neighbors is enough to activate v and y_v is kept at the minimum, i.e., $y_v = 0$. Otherwise, the minimum influence needed to activate v is

$$y_v = t_v - \sum_{u \in N_v} d_{uv}.$$

□

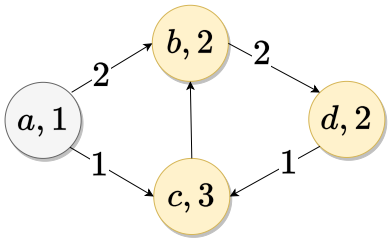
Due to the construction of the auxiliary structures for the relaxation, we claim that the solution of the LCIP in the condensed graph is a lower bound for the problem in G'.

Lemma 14 *When $\alpha = 1$, for a not strongly connected sub-graph G' of G and a condensed component graph H of G', the value of an optimal solution of the LCIP on H is a lower bound for the value of an optimal solution of the problem in G'.*

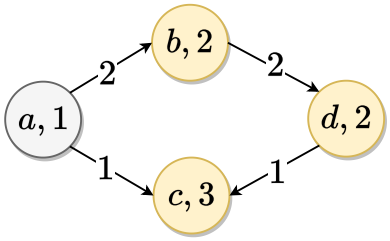
Proof: Each vertex $v \in V(H)$ is associated with a s.c.c C_v of G'. By the construction of H, we have

$$t_v = \min_{i \in V(C_v)} \{t_i\},$$

for every $v \in V(H)$. We start by claiming that the incentive paid to a vertex of H is a lower bound for the necessary incentive to influence vertices of the s.c.c C_v of G'. Let $\underline{y_v}$ and y_i^* be the optimal incentive given to each $v \in V(H)$ and $i \in V(G')$, respectively.



(a) Input graph with two strongly connected components.



(b) Propagation graph.

Figure 7.5: The graph in Figure (a) contains a instance of the LCIP with $\alpha = 1$. The solution is the graph of Figure (b), were the incentives are $y_a = y_c = 1$ and $y_b = y_d = 0$.

When v is a source in H, then it has no incoming neighbors. We are obligated to pay the total threshold to activate it. In an optimal solution, we have that

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

independent of how many vertices in C_v we want to activate. It is even more clear when we are activating every $i \in V(C_v)$, which is the case.

If v is not a source in H, we need to consider the incoming arcs of v. This can decrease the necessary incentives to be offered to v. Recall that, $\alpha = 1$, all vertices must be active in a solution. We can express the incentive offered to $v \in V(H)$ as

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

We want to prove that $\underline{y_v}$ is also a lower bound for the associated s.c.c. C_v in G'. At this point, it is important to remember that the propagation graph of G' must be acyclic. Therefore, some vertices of C_v receive *external influence*. The external influence can be exerted by:

(i) influence coming only from active vertices out of C_v, i.e. $\sum_{j \in N_i \setminus V(C_v)} d_{ji} \geq t_i$ for $i \in V(C_v)$; or

(ii) the payment of incentive enough to cover the threshold of such vertices, that is $y_i \geq t_i$ for $i \in V(C_v)$; or

(iii) the combination of incentives and influence from incoming arcs.

To illustrate, consider the example in Figure 7.5. The input graph (Figure 7.5(a)) has two strongly connected components. Vertex b is of type (i) because the influence coming from a is enough to achieve its threshold. Vertex a is of type (ii), while vertex c is of type (iii) since it receives incentive and influence from neighbors.

In case (i), we have that $y_i^* = 0$. If all the external influence achieving C_v is of this type, consequently, we have that $\underline{y_v} = 0$. For the cases (ii) and (iii), denote as

$S(C_v) = \{i \in V(C_v) : y_i^* > 0\}$ the set of vertices of C_v that receives a positive incentive. The cost for activating $v \in V(H)$ is

$$\underline{y_v} = \max\left\{0, t_v - \sum_{u \in N_v} d_{uv}\right\} \tag{7.10}$$

$$= \max\left\{0, \min_{i \in C_v}\{t_i\} - \sum_{i \in S(C_v)} \sum_{j \in N_i \setminus V(C_v)} d_{ji}\right\} \tag{7.11}$$

$$\leq \max\left\{0, \sum_{i \in S(C_v)} t_i - \sum_{i \in S(C_v)} \sum_{j \in N_i \setminus V(C_v)} d_{ji}\right\} \tag{7.12}$$

$$= \max\left\{0, \sum_{i \in S(C_v)} \left(t_i - \sum_{j \in N_i} d_{ji}\right)\right\} \tag{7.13}$$

$$= \sum_{i \in V(C_v)} \left(\max\left\{0, t_i - \sum_{j \in N_i} d_{ji}\right\}\right) \tag{7.14}$$

$$= \sum_{i \in V(C_v)} y_i^*. \tag{7.15}$$

Equation (7.11) makes the linking of a condensed vertex v with its associated s.c.c C_v of G'. Inequality (7.12) holds because we are increasing only the positive summation. The rest of the equations are just rearrangements of the previous one.

Therefore, activating a vertex $v \in V(H)$ requires an incentive y_v^* which is a lower bound for the cost of activate any vertex $i \in V(C_v)$. It holds for every $v \in V(H)$ and every s.c.c C_v of G'. So doing the sum, we have

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

□

Theorem 15 *The solution obtained by the Algorithm 9 is a lower bound for the* Least Cost Influence Problem *in the sub-graph G'.*

Proof: For the first case, the result is direct by the observations at the beginning of this section. The case in which G' is not strongly connected and $\alpha = 1$ (step 6 of the algorithm) holds by Lemma 14. □

7.3.1 The case of $\alpha < 1$

Algorithm 9 is general and finds a dual bound for every case of LCIP. However, there is room for improvement in the case where $\alpha < 1$ and the sub-graph G' is not strongly connected. The idea is to do something similar to the case of $\alpha = 1$ and solve the problem on the condensed component graph. This leads to a new combinatorial problem, as we explain next.

To increase the lower bound for the case in which we are not interested in achieving 100% adoption, we propose assigning a weight to each vertex of the con-

densed component graph H and use the total weight of the activated vertices to satisfy the portion of achieved vertices on the original problem. In this way, we arrive at a formulation for a new variant of the LCIP, defined in Problem 8.

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

Problem 8 (Weighted Least Cost Influence Problem (WLCIP)) *Given a directed graph H with weight $w(v)$ on each vertex $v \in V(H)$, weight of influence $d_{uv} > 0$ on the arcs $(u, v) \in E(H)$, and an integer κ. Find a vector* $\mathbf{y}$ *of incentives which minimize the sum $\sum_{v \in V(H)} y_v$, ensuring that the total weight of active vertices is at least κ by the end of the activation process.*

A natural integer linear programming formulation for this problem follows.

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

$$\text{s.t.} \sum_{v \in V(H)} w(v) x_v \geq \kappa, \qquad \forall v \in V(H), \tag{7.16}$$

$$\sum_{u \in N_v} d_{uv} x_u + y_v \geq t_v x_v, \qquad \forall v \in V(H), \tag{7.17}$$

$$x_v \in \{0, 1\}, \qquad \forall v \in V(H), \tag{7.18}$$

$$y_v \in \mathbb{N}_0, \qquad \forall v \in V(H), \tag{7.19}$$

where the binary decision variables x_v indicates that v is active and the integer variable $y_v \geq 0$ is the amount of incentive assigned to v.

We do not need to eliminate cycles in this problem because H is acyclic. In this model, the objective function aims to minimize the incentives paid to the vertices of graph H. The cover constraints in (7.16) ensure that active vertices' total weight cover the parameter κ. Note that the value of κ relates to the original graph G because we still want to achieve the portion α of the original network. If we use $\kappa = \lceil \alpha |V(G')| \rceil$ instead, we have a valid lower bound, but with $\lceil \alpha |V(G)| \rceil$, the lower bound can be higher (which is better). If, for some reason, during the branch-and-bound search, we have $\kappa > \sum_{v \in V(H)} w(v)$, the problem (WLCIP) is infeasible, and we must cut off the current node. Constraints in (7.17) respect the thresholds, the total of influence coming from active neighbors of v together with an incentive need to be at least the threshold of v if it is active. Constraints (7.18) and (7.19) ensure the integrality of the variables.

Note that when we add $w(v)$ for every active $v \in V(H)$, we are implicitly saying that all the vertices of C_v are active in G'. Indeed, if $\alpha < 1$, not all vertices of C_v could be active, so we are majoring the number of activated vertices in G' by using $w(v) = |V(C_v)|$.

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

Theorem 16 states that an exact solution of WLCIP on H provides a valid lower bound for the LCIP on G'.

Theorem 16 *Given a sub-graph G' of G and a condensed component graph H of G', the value of an optimal solution of WLCIP on H is a lower bound for the value of an optimal solution of LCIP on G'.*

Proof: Let $\underline{y_v}$ and y_i^* be the optimal incentive given to each $v \in V(H)$ and $i \in V(G')$ for WLCIP and LCIP, respectively. We want to prove that

$$\sum_{v \in V(H)} \underline{y_v} \leq \sum_{i \in V(G')} y_i^*. \tag{7.20}$$

Nevertheless, different from the proof of Lemma 14, we cannot claim that the inequality

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

is always true for every $v \in V(H)$ and its associated $C_v \in G'$. In fact, given any feasible solution **y** of WLCIP, we can divide the set of vertices of H in two disjoint subsets:

(i) $V' = \{v \in V(H) : y_v \leq y(C_v)\}$;

(ii) $V'' = \{v \in V(H) : y_v > y(C_v)\}$,

where C_v is the s.c.c. associated with each $v \in V(H)$ and

$$y(C_v) = \sum_{i \in V(C_v)} y_i^*.$$

If all vertices of H are of type (i) in a solution of WLCIP, Inequality 7.20 holds and the proof is done. By the other hand, a solution in which every $v \in V(H)$ is of type (ii) is impossible because at least one vertex needs to be a source. Let s be a source in a solution, from the definition of the thresholds of the condensed component graph H, we have that

$$y_s = t_s = \min_{i \in V(C_s)} \{t_i\} \leq y(C_s).$$

So, at least one vertex is of type (i).

Figure 7.6 shows a solution with vertices of type (i) and (ii). Vertices u and v of H are of type (i), and w is of type (ii). Vertex w receives incentive $y_w = 1$ in that solution, while in the solution of graph G', component C_w has cost $y_d + y_e + y_f = 0$. It happens because the external influence that arrives in C_w comes from the component C_v of G' in which associated $v \in V(H)$ is not active. Note that the solution of LCIP in Figure 7.6(a) has active vertices in all the strongly connected components. In contrast, in Figure 7.6(b), an optimal solution of WLCIP on H needs to activate only two vertices.

It remains to prove that, even with vertices of type (ii), Inequality 7.20 still holds.

It is important to observe that, for every $v \in V''$, as $y_v > y(C_v)$, then the amount of influence arriving in C_v is greater than the influence that arrives in v. Thus, there are inactive vertices in N_v (for example, in Figure 7.6(b), vertex w is of type (ii) and v is an inactive neighbor of w). As every vertex v of type (ii) has inactive incoming

neighbors, we can activate these vertices with the objective of decrease the value of y_v until v becomes a vertex of type (i). Sufficiently decreasing the incentive of v will make it become a vertex of type (i) in some moment. We can guarantee this because the maximum weight of influence that can arrive at v and its associated C_v is the same due to the construction of H. In addition to that, the threshold of v is the minimum threshold in C_v. So, in an extreme case, if we activate all the incoming neighbors of v, we are considering the maximum weight of influence that can achieve v and its associated component C_v. Doing this does not generate cycles because H is a DAG.

In this way, we can construct a new solution $\mathbf{y}'$ with the proper incentives modified. Algorithm 10 describes the procedure to construct $\mathbf{y}'$.

Algorithm 10: NEW SOLUTION $\mathbf{y}'$

Input: H, a set A_H of active vertices and an optimal solution $\underline{\mathbf{y}}$ for WLCIP.
Output: A new solution $\mathbf{y}'$ of WLCIP

1 **begin**
2 $Q \leftarrow V''$
3 $A'_H \leftarrow A_H$
4 **while** $Q \neq \emptyset$ **do**
5 Take a vertex $v \in Q$ and remove it from Q
6 $y'_v \leftarrow \max\{0, t_v - \sum_{u \in N_v \cap A'_H} d_{uv}\}$
7 **while** $y'_v > y(C_v)$ **do**
  `// take an inactive incoming neighbor`
8 $u \leftarrow$ arbitrary vertex from $N_v \setminus A'_H$
9 $A'_H \leftarrow A'_H \cup \{u\}$ `// activate` u
10 $y'_v \leftarrow \max\{0, y'_v - d_{uv}\}$ `// decrease the incentive of` v
11 $y'_u \leftarrow \max\{0, t_u - \sum_{w \in N_u \cap A'_H} d_{wu}\}$
12 **if** $y'_u > y(C_u)$ **then**
13 $Q \leftarrow Q \cup \{u\}$

Algorithm 10 generates a new feasible solution for WLCIP from an optimal solution. The condition in line 7 ensures that, at the end of this procedure, all the vertices that previously are of type (ii) will be of type (i). To achieve this, we activate some inactive incoming neighbors of vertices in V'' and pay the price of this activation (line 11). Note that if there are no inactive incoming neighbors of v, then it does not enter the condition of line 7. Consequently, at the end, each vertex in V'' has a new incentive of value

$$y'_v = \max\left\{0, \underline{y_v} - \sum_{u \in N_v \cap U} d_{uv}\right\},$$

where the subset $U \subset V(H)$ is the set of vertices activated in this procedure (line 9).

Let us analyze the new solution $\mathbf{y}'$ obtained by Algorithm 10 in function of V' and V'' defined above. We can write the total cost of $\mathbf{y}'$ as

$$\sum_{v \in V(H)} y'_v = \sum_{v \in V'} y'_v + \sum_{v \in V''} y'_v. \tag{7.21}$$

Recall that $V' \cup V'' = V(H)$, because inactive vertices receive an incentive equal to zero, then they belong to V'. Taking each summation of the right-hand side of Equation 7.21 separately, we have

$$\sum_{v \in V'} y'_v = \sum_{v \in V' \setminus U} \underline{y_v} + \sum_{u \in U} y'_u \geq \sum_{v \in V'} \underline{y_v} \tag{7.22}$$

and

$$\sum_{v \in V''} y'_v = \sum_{v \in V''} \max \left\{ 0, \underline{y_v} - \sum_{u \in N_v \cap U} d_{uv} \right\} \leq \sum_{v \in V''} \underline{y_v}. \tag{7.23}$$

In the equality of (7.22), we increased the incentives given to vertices in U because there is a non-negative cost for activating each incoming neighbor of a vertex $v \in V''$, i.e., $y'_u \geq 0, \forall u \in N_v$. The rest of the vertices in V' remain with the same incentives as in $\underline{\mathbf{y}}$. As a result of this increment, we have the inequality in (7.22). In contrast, the cost of activating vertices of V'' decreases (see the inequality of (7.23)), which is the objective of Algorithm 10.

Therefore, to compare $\underline{\mathbf{y}}$ with $\mathbf{y}'$, it is sufficient to focus on the vertices "affected" by the new solution, that is, vertices in U and V''. In the transition from $\underline{\mathbf{y}}$ to $\mathbf{y}'$, vertices in U have their incentive increased. At the same time, vertices in V'' have incentive decreased. The difference are $y'_u - \underline{y_u}$, for every $u \in U$ and $\underline{y_v} - y'_v$, for each $v \in V''$. Note that $\underline{y_u} = 0$ because u is inactive in $\underline{\mathbf{y}}$. It generates two possible outcomes that depend on the increment in U be greater or less than the reduction in V'' on the value of solution $\underline{\mathbf{y}}$. First, the cost to activate the vertices in U is less than the reduction we obtain in incentives given to the vertices of V'', that is,

$$\sum_{u \in U} y'_u < \sum_{v \in V''} (\underline{y_v} - y'_v),$$

it implies that we found a solution of cost smaller than the optimum, but this is a contradiction with the optimality. So there is no way this procedure produces a solution better than the optimum. On the other hand, we have

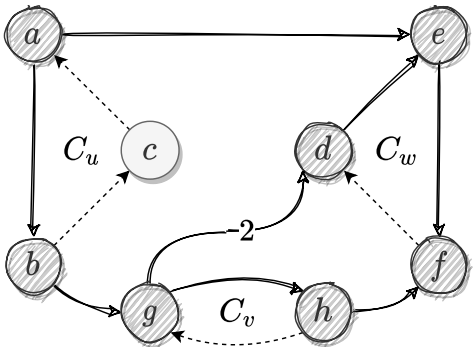
$$\sum_{u \in U} y'_u \geq \sum_{v \in V''} (\underline{y_v} - y'_v).$$

In words, in the new feasible solution $\mathbf{y}'$, the required incentive to activate vertices of U is greater or equal to the decrement obtained to transform the vertices of V'' in vertices of type (i). As a result, the total cost of $\mathbf{y}'$ is at least the same as the optimum (left inequality of (7.24)). Besides that, in this new solution, all vertices of H are of type (i), i.e., $y'_v \leq y(C_v)$ for every vertex of H and its associated s.c.c C_v of G', which is, visibly, a lower bound for the solution of LCIP on G'. This leads to the right inequality of (7.24).

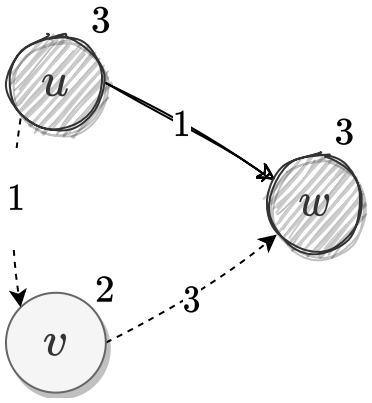
$$\sum_{v \in V(H)} \underline{y_v} \leq \sum_{v \in V(H)} y'_v \leq \sum_{i \in V(G')} y^*_i. \tag{7.24}$$

Consequently, Inequality 7.20 holds for this new solution. □

In this work, we are interested in solving the WLCIP in condensed component graphs. Despite that, the problem is general and is defined for any directed graphs. It



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



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

Figure 7.6: The solution of LCIP in Figure 7.6(a) has value 5, the incentives are: $y_a = 2, y_b = y_g = y_h = 1$ and $y_c = y_d = y_e = y_f = 0$. The solution of WLCIP in Figure 7.6(b) has value 3, the incentives are: $y_u = 2, y_w = 1$ and $y_v = 0$.

is a generalization of LCIP where, for each vertex i of a given input graph, there is a weight $w(i)$ attached to it. We can see LCIP as a special case where all vertices have the same weight. So, as LCIP is NP-hard, WLCIP is NP-hard as well. Besides being difficult to solve in general cases, unfortunately, this problem is difficult to solve in DAGs, as stated in the following theorem.

Theorem 17 *WLCIP is NP-hard on directed acyclic graphs.*

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

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

$$\sum_{i \in J} b(i) \geq B.$$

We can assume

$$0 < B \leq \sum_{i \in I} b(i).$$

Create the following instance for WLCIP. We set $V = \{v_i : i \in I\}, E = \emptyset, w(v_i) = b(i)$ and $t_{v_i} = c(i)$ for all $i \in I$. Moreover, set

$$\alpha = \frac{B}{\sum_{i \in I} b(i)}.$$

Note that

$$\alpha \sum_{v_i \in V} w(v_i) = B.$$

Let $A \subseteq V$ be the set of activated vertices in an optimal solution. Observe $J = \{i : v_i \in A\}$ is an optimal solution for min-knapsack. □

Even though WLCIP is NP-hard on DAGs, demanding a somewhat elaborate mathematical approach, it deserves further consideration. If we solve the WLCIP by the optimality to get a lower bound for $\alpha < 1$, the Algorithm 9 loses scalability. Furthermore, we observed in preliminary experiments that the LP-relaxation of the formulation (14)-(17) does not provide a better lower bound than $\min_{i \in V(G')} \{t_i\}$ (line 3 of Algorithm 9). Because of these practical limitations, we do not solve the WLCIP in the experiments of Section 7.6. Attempts should be made to find lower bounds for WLCIP tighter than the LP-relaxation of the formulation (14)-(17). Such attempts could be done by a combinatorial relaxation or by the LP-relaxation of a stronger formulation, for example.

7.4 BRANCHING RULE

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

The idea is to give higher priority to branch on the fractional variables associated with *strong bridges* (Definition 19) or *strong articulation points* (Definition 20) of G'. In this way, when we fix in zero a variable associated with a strong articulation point (or strong bridge) and remove the associated vertex (or arc) of G', the number of components in G' increases. Consequently, the number of vertices of the condensed graph increases too, and, in turn, the lower bound can also increase.

Definition 19 (Strong Bridge) *A strong bridge of a directed graph G is an arc whose removal increases the number of strongly connected components of G.*

Definition 20 (Strong Articulation Point) *A strong articulation point of a directed graph G is a vertex whose removal increases the number of strongly connected components of G.*

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

Algorithm 11 describes a branching rule based on these concepts. To describe it, we define the following notation. Denote S as the set of all *connectivity cuts* of G, that is, S contains all strong bridges and strong articulation points. Also, let $sc(G, s)$ represent the number of strongly connected components generated by removing s from G', where $s \in S$ can be an arc or a vertex of G. For a solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$ of the continuous relaxation of a sub-problem (or feasible region) P, the set

$$F = \{i \in V(G) : \hat{x}_i \notin \mathbb{Z}\} \cup \{(i, j) \in E(G) : \hat{z}_{ij} \notin \mathbb{Z}\}$$

contains the elements of G associated with the fractional variables of an LP-relaxation of the ILP model in Section 7.2. The list $\mathcal{L}$ represents the list of active nodes of the branch-and-bound tree. Every node contains a problem, where P is the sub-problem (or feasible region).

Algorithm 11: BRANCHING RULE

Input: G'

1 **begin**
2 $S \leftarrow$ CONNECTIVITYCUTS(G')
3 $S' \leftarrow S \cap F$; /* fractional articulation cuts */
4 **if** $S' \neq \emptyset$ **then**
5 $s^* = \arg\max_{s \in S'} \{sc(G', s)\}$
/* Branch on variable associated with s^* */
6 **if** $s^* \in V(G)$ **then** /* branch on a vertex variable */
7 $P' \leftarrow P \cap \{\hat{x}_{s^*} \leq 0\}$
8 $P'' \leftarrow P \cap \{\hat{x}_{s^*} \geq 1\}$
9 **else** /* branch on an arc variable */
10 $P' \leftarrow P \cap \{\hat{z}_{s^*} \leq 0\}$
11 $P'' \leftarrow P \cap \{\hat{z}_{s^*} \geq 1\}$
/* Put P' and P'' in the list of active nodes */
12 $\mathcal{L} \leftarrow \mathcal{L} \cup \{P'_i, P''_i\}$
13 **else**
14 Use a generic branching rule of your choice.

In Algorithm 11, the first step is to compute all the strong bridges and strong articulation points. Subroutine CONNECTIVITYCUTS(G') represents this procedure which can be done using the algorithm presented in (Italiano et al., 2012). If there are fractional variables associated with the connectivity cuts, we choose the one that generates more strongly connected components (line 5). So we branch on the chosen variable by fixing it in zero for one child node and in one for the other.

7.5 PREPROCESSING FOR $\alpha = 1$

In preliminary experiments, we have noticed that our lower bound yields no substantial reduction in the branch-and-cut running time when the input graphs are based on real-world networks. Nevertheless, we can apply a preprocessing rule specialized in this type of graph. The preprocessing rules proposed in (Melo and Vignatti, 2020) exploit the power-law distribution of complex networks to reduce the input networks' size and compute a partial solution in advance for the Target Set Selection problem.

In this paper, we adapt the preprocessing algorithm of (Melo and Vignatti, 2020) (presented in Chapter 6) to work on the LCIP problem when we want to activate all the network vertices ($\alpha = 1$). In short, our adaptation consists of fixing to one (or zero) every arc variable z_{ij} that we are sure belong (or not) to an optimal solution, based on the propositions presented in (Melo and Vignatti, 2020). The algorithm repeatedly removes the sources (vertices with no incoming arcs) and sinks (vertices with no outgoing arcs) of the input graph by simulating an activation process in the threshold model. This strategy preserves the optimality and reduces the feasible region's size significantly when the input graphs have a power-law degree distribution. In our adaptation, we compute the required incentive for each vertex removed in the preprocessing. Then the reduced graph is used as the input for the problem in the branch-and-cut. In

this way, we have a partial solution combined with the final solution obtained by the branch-and-cut.

Algorithm 9 alone is sufficient to achieve notable improvements in the running time of the branch-and-cut. However, the combination with Algorithm 11 and the preprocessing rules can make it stronger. Below we discuss the experimental results.

7.6 COMPUTATIONAL EXPERIMENTS

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

7.6.1 Synthetic graphs

As in (Fischetti et al., 2018; Günneç et al., 2016), we use the generative model proposed by Watts and Strogatz (1998) for small-world random graphs. The rewiring probability parameter for the small world graph is limited to assume values $\beta \in \{0.1, 0.3\}$. The influence weights on the arcs are chosen uniformly at random from $\{1, ..., 10\}$. Let $N(\mu, \sigma)$ be a normally-distributed random variable. For every $i \in V$, we set $t_i = \max\{1, \min\{N(\mu, \sigma), d_i\}\}$, where

$$d_i = \sum_{(j,i)\in E} d_{ji},$$

$\mu = 0.7d_i$ and $\sigma = \dfrac{d_i}{|N_i|}$. We restrict the experiments with synthetic graphs for $\alpha = \{1, 0.5, 0.1\}$.

Table 7.1 summarizes the difference in performance when we apply the lower bound in the branch-and-cut. Each value on the table is the average of 5 executions. Every execution generates a new graph of a given size and average degree. The first column contains the name of instances in format n-deg where n is the number of vertices, and deg is the average degree. The second column is the value of β, the rewiring probability of the generative random graph model. The third column, is the value of α, the portion of the network to be activated in the LCIP. In the other columns, BC means the branch-and-cut algorithm using only the LP-relaxation, and BC+ means we are using our combinatorial relaxation to get the lower bound, including the branching rule (Algorithm 11) and the preprocessing rules (Section 7.5, for $\alpha = 1$).

Next, we present the time in seconds for those that finished before the time limit. The time limit for these instances is set to 1800 seconds. Dashed cells in the column “time" means that the running time reached the time limit. We marked in boldface the best results. E.g., in the instance “50-4” with $\alpha = 1$, our method (BC+) required less running time to find the optimum. In column “dual bound”, the higher, the better, while in the “primal bound” is the contrary. Observe that our algorithm finds a better value for the dual bound in the majority of instances. In the last column, we have the corresponding average gap between the dual and primal bounds. The symbol ∞ in the column “gap" means the gap is infinity or very large. The gap is computed as follows: let l be the dual bound and u be the primal bound. We set the gap to ∞ if $l \leq 0$.

Table 7.1: Experiments with synthetic small-world graphs.

Graph	β	α	Time(s)		Dual bound		Primal bound		Gap	
			BC	BC+	BC	BC+	BC	BC+	BC	BC+
50-4	0.1	1	553.47	**400.62**	32.28	**35.70**	37.35	**35.70**	0.66	**0.00**
		0.5	**608.87**	1,036.11	**22.35**	21.65	**52.75**	66.85	**3.20**	4.18
		0.1	**1,288.6**	1,445.11	**12.95**	11.3	**24.7**	31.55	**1.2**	2.2
	0.3	1	**356.66**	525.56	**51.70**	48.40	51.70	51.70	**0.00**	0.10
		0.5	-	-	**15.18**	10.41	**74.80**	124.10	**5.96**	11.73
		0.1	1,081.55	**722.05**	9.45	**11.15**	21.15	**17.75**	∞	**0.8**
50-8	0.1	1	-	-	7.6	**18.4**	401.95	**389.7**	∞	**21.8**
		0.5	-	-	0	**18.4**	**284.15**	301.05	∞	**16.8**
		0.1	-	-	0.4	**18.4**	87.75	87.75	∞	**4**
	0.3	1	-	-	1.04	**16.17**	**339.45**	393.8	∞	**23.73**
		0.5	-	-	0.00	**15.90**	347.65	**325.05**	∞	**19.6**
		0.1	-	-	0	**15.9**	**75.4**	78.1	∞	**4**
75-4	0.1	1	1,468.29	**1,125.34**	29.07	**32.01**	**45.45**	48.45	**1.10**	1.72
		0.5	**1,149.93**	1,306.35	**22.94**	20.95	**36.05**	46.95	**0.92**	2.23
		0.1	-	**1,442.93**	2.35	**10.35**	40.75	**34.3**	∞	**2.6**
	0.3	1	982.07	**839.66**	64.07	**65.77**	95.55	**86.35**	1.18	**0.59**
		0.5	-	-	16.61	**22.68**	131.50	**94.15**	12.22	**4.83**
		0.1	**1,084.40**	1,092.41	11.7	**12**	28.45	**28.15**	1.8	1.8
75-8	0.1	1	-	-	6.2	**14.6**	**538.25**	560.4	∞	**37.2**
		0.5	-	-	0	**14.6**	**416.45**	449.7	∞	**29.8**
		0.1	-	-	0	**14.6**	124.85	**119.75**	∞	**7.2**
	0.3	1	-	-	11.37	**17.42**	**677.95**	685.75	∞	**41.83**
		0.5	-	-	0	**14.6**	**441.1**	444.75	∞	**29.4**
		0.1	-	-	0	**14.6**	**105.35**	108	∞	**6.4**
75-12	0.1	1	-	-	0	**35.7**	781.85	**779.2**	∞	**20.9**
		0.5	-	-	0	**35.7**	683.6	**663.5**	∞	**17.5**
		0.1	-	-	0	**35.7**	238.35	**235.05**	∞	**5.6**
	0.3	1	-	-	0	**32.65**	**818.85**	882.25	∞	**29**
		0.5	-	-	0	**32.65**	839.3	**813.1**	∞	**26.1**
		0.1	-	-	0	**32.65**	235.9	235.9	∞	**6.8**
100-4	0.1	1	-	-	18.35	**18.63**	**197.25**	230.40	**11.38**	13.37
		0.5	-	-	12.40	**12.64**	**121.75**	179.95	∞	**16.72**
		0.1	**678.41**	1,445.69	**12.15**	8.85	**23.55**	35.65	∞	**3**
	0.3	1	-	**1,793.58**	47.57	**50.68**	188.30	**111.95**	2.84	**1.54**
		0.5	-	-	**17.61**	16.27	**211.50**	254.50	**13.82**	17.64
		0.1	**1,442.32**	1,442.58	8.53	**11.55**	**39.1**	41.15	∞	**3**
100-8	0.1	1	-	-	13.27	**18.67**	859.35	**835.80**	∞	**48.04**
		0.5	-	-	0	**15**	**422.3**	466.6	∞	**30.6**
		0.1	-	-	0	**15**	149.7	149.7	∞	**9**
	0.3	1	-	-	3.75	**16.45**	**986.20**	1,010.60	∞	**60.2**
		0.5	-	-	0	**16.45**	687.2	**670.95**	∞	**39.6**
		0.1	-	-	0	**16.45**	147.6	147.6	∞	**8**
100-12	0.1	1	-	-	0	**37.7**	1136.3	**1099**	∞	**28.2**
		0.5	-	-	0	**37.7**	771.5	771.5	∞	**19.4**
		0.1	-	-	0	**37.7**	275.35	275.35	∞	**6.3**
	0.3	1	-	-	0	**25.2**	**1,166.60**	1,178.05	∞	**48.8**
		0.5	-	-	0	**25.2**	**1,075.35**	1,075.55	∞	**44.5**
		0.1	-	-	0	**25.2**	279.75	279.75	∞	**10.8**
100-16	0.1	1	-	-	0	**47.1**	**1,513.45**	1518	∞	**31.4**
		0.5	-	-	0	**47.1**	1281	1281	∞	**26.2**
		0.1	-	-	0	**47.1**	436.55	436.55	∞	**8.3**
	0.3	1	-	-	0	**50.1**	1,725.55	**1,698.25**	∞	**32.9**
		0.5	-	-	0	**50.1**	1,461.25	1,461.25	∞	**28.3**
		0.1	-	-	0	**50.1**	459.9	459.9	∞	**8.2**

Otherwise, the gap is $(u - l)/l$. Recall that the values in Table 7.1 are averages, so the gaps on the table are the average gaps.

The results exhibited in Table 7.1 illustrate our algorithm behavior for graphs of different sizes and densities. When the graphs are small (50-4, for example), our algorithm is not effective. In some cases, the running time is worse than the branch-and-cut using only the LP-relaxation (BC). However, as the density and the number of vertices increases, our algorithm (BC+) achieves better gaps.While the BC has lower bounds equal to zero in many instances, BC+ always provides a lower bound greater than zero, contributing to smaller gaps. In the vast majority of the instances, our algorithm provides smaller gaps.

7.6.2 Real world networks

To demonstrate the effects of applying the lower bound algorithm on real data, we also performed experiments with real-world social networks. The datasets we use are part of the Koblenz Network Collection (Kunegis, 2013), human social network category. Table 7.2 shows a short description of each social network used here. For each network, n is the number of vertices, and m is the number of arcs. The weights on the arcs are the original weights of the networks. On graphs with no arcs weights, we set the weights to 1. Lastly, the threshold t_i, for each vertex i, are defined in the same way as in the synthetic graphs (see Section 7.6.1).

Table 7.2: Real world social networks.

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

Table 7.3 summarizes the results for the real-world social networks. Dashed cells mean that the running time reached the time limit (1800 seconds). Recall that, for the cases of $\alpha = 1$, we added the processing rules of Section 7.5. Also, we enable all the presolving methods of the SCIP framework for both algorithms BC and BC+. These presolving methods provide gains in running time or gap reduction for our algorithm, except in the Residence hall and Advogato networks. In the column entitled "dual bound", the higher the number, the better. This column shows that our algorithm provides higher lower bounds in the majority of the networks for different values of α. It implies gains in the running time or gap reduction.

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

Table 7.3: Experiments on real world based social networks for $\alpha = \{1, 0.5, 0.1\}$.

Network	α	Time(s)		Dual bound		Primal bound		Gap	
		BC	BC+	BC	BC+	BC	BC+	BC	BC+
High School	1.0	1.20	**0.13**	18	18	18	18	0	0
	0.5	-	**173.3**	0	**6**	21	**6**	106.58	**0**
	0.1	**29.23**	-	3	3	**3**	9	**0**	2
Residence hall	1.0	-	-	18	18	**42**	114	**1.33**	5.33
	0.5	-	-	0	**6**	408	408	∞	**67**
	0.1	-	-	0	**6**	60	60	∞	**9**
Physicians	1.0	-	-	**139.5**	138.4	162	**157.5**	0.16	**0.14**
	0.5	-	-	8.4	**15.5**	229.5	**81**	26.26	**4.23**
	0.1	-	**300.19**	4.9	**13.5**	18	**13.5**	2.66	**0**
Wiki-vote	1.0	0.32	**0.1**	3,692	3,692	3,692	3,692.0	0	0
	0.5	-	-	221.3	**213.3**	**273**	286	**0.23**	0.34
	0.1	**336.28**	472.31	52	52	52	52	0	0
Adolescent	1.0	-	**38.91**	656.3	**661.5**	1,800.8	**661.5**	1.74	**0**
	0.5	-	-	0	**5.3**	94.5	94.5	∞	**17**
	0.1	-	-	0	**5.3**	210	210	∞	**39**
Advogato	1.0	-	-	394,380	**395,640**	**997,794**	1,069,488	**1.53**	1.70
	0.5	-	-	0	**126**	801,864	801,864	∞	**6,363**
	0.1	-	-	0	**126**	82,404	82,404	∞	**653**
DBLP	1.0	17.41	**8.68**	40,645.5	40,645.5	40,645.5	40,645.5	0	0
	0.5	-	-	1,268.5	1,268.5	25,122	25,122	18.81	18.81
	0.1	-	-	51.1	51.1	15,128.5	15,128.5	294.77	294.77
Cora Citation	1.0	-	-	624,899	**624,954**	625,416	**625,086**	0	0
	0.5	-	-	80,096	80,096	1,643,268	1,643,268	19.52	19.52
	0.1	-	-	3,365.5	3,365.5	135,696	135,696	39.32	39.32

In this chapter, we proposed and analyzed an algorithm to compute a lower bound for the Least Cost Influence Problem based on particular properties of the problem. In addition to the algorithm itself, some auxiliary strategies showed to be

useful to increase the lower bounds. In order to strengthen the lower bounds, we select branching variables to choose the next branch to explore the decision tree. Besides, if we are restricted to activate the whole network, applying preprocessing rules can earn scalability on solving the problem. To check for the practical applicability, we also provide computational experiments on large social networks, showing that in addition to the theoretical guarantees, our algorithm out-perform the linear programming relaxation.

8 CONCLUSION

The vast majority of studies about the influence maximization problem and its variants concentrate on sub-optimal solutions to solve the problem quickly. These systematic investigations have provided powerful techniques, e.g., heuristic methods, that estimate the Expected Propagation problem efficiently, and approximation algorithms, which have offered a good approximation ratio for the TSS and its extensions. Some studies have proposed investigating the TSS problem from the mathematical programming point of view in recent years. After the first integer linear programming model for the TSS problem be proposed by Ackerman et al. (2010), several sophisticated formulations and generalizations of TSS were introduced. Some of the newest studies claim to overcome difficulties of the early models in real-world instances, e.g., LCIP and GLCIP (Fischetti et al., 2018; Günneç et al., 2016). In another context, in the last years, we note that the interest in combinatorial problems restricted to power-law graphs has increased. Such interest is mainly due to the possible easiness of solving hard problems in real-world settings.

While early studies have experienced difficulties in developing integer programming formulation for the influence propagation problems, other studies presented theoretical arguments for the hardness in approximate many notorious NP-hard problems in power-law graphs. Unlike these previous investigations, in this research, we consider the so called power-law degree distribution of social networks to design faster exact algorithms.

Looking for solutions specialized in power-law graphs, we performed experiments with the Preselector algorithm (of Section 5.1) to preprocessing instances of the influence maximization problem. In this way, we have empirically quantified the reduction in running time with real-world and synthetically generated power-law graphs. Interesting features of the preselection explore the relationship between influence propagation and degree distribution of social networks to highlight the most promising vertices, preventing unnecessary processing by cutting out some elements of the search. Experimentally, the PrevalentSeed is reasonably faster than Celf in most of the evaluated graphs. It happens mainly due to the reduction of the number of estimation of the influence function. Moreover, the set of activated nodes chosen by PrevalentSeed are very competitive with those found by Celf in terms of quality. Besides, the theoretical analysis concerning the propagation produces results that match the empirical analysis.

For the TSS problem, the preprocessing rules using logical implications also were verified experimentally. Based on graph properties that commonly appear in complex networks, we presented a strategy that provides a notable reduction of the TSS problem search space (Chapter 6). Furthermore, through theoretical analysis, we proved that these rules preserve the optimal solution. Despite the simplicity, the experiments indicate that our strategy provides tractability to the problem in power-law graphs. Also, sophisticated techniques and algorithms can apply this preprocessing in order to gain scalability.

Subsequently, we proposed and analyzed an algorithm to compute a lower bound for the Least Cost Influence Problem (Chapter 7). In this solution, instead of exploring the degree distribution of the input graph, the algorithm is based on particular

properties of the problem. The dual bound exerts an important rule in a branch-and-bound algorithm. Because of that, we achieve improvements in the branch-and-cut for this problem. Theoretical analysis was also carried out on this part by showing the correctness of the algorithm and providing a deeper discussion on the substructures of the problem. Furthermore, this investigation suggests that finding tighter bounds usually demands more effort to solve. In these case, we have to choose the better trade-off between the quality of the relaxation and its complexity. The experiments presented achieved better results in solving the problem when applying the new dual bound.

Each contribution mentioned above reveals some intuition to answer the inquiries raised at the beginning of this thesis (Chapter 1). Recall the questions:

i) What are the implications of the degree distribution and graph sparsity to design exact algorithms for the influence maximization problem and its variants?

ii) How can the value of the power-law exponent β influence the formulation of the algorithm for these graphs?

iii) How can specific characteristics of these problems contribute to design faster exact algorithms or to strengthen existing formulations?

To explain item (i), we notice that, in the literature, the TSS problem is solvable in linear time when the input graph is a tree (Chen, 2009; Cordasco et al., 2015; Ferrante et al., 2008; Raghavan and Zhang, 2015). For graphs with treewidth parameter bounded, there are polynomial-time algorithms (Ben-Zwi et al., 2011). Unsurprisingly perhaps, even for some restrictive special cases of input graphs or cascade models, all the optimization variant problems turn out to be extremely hard to approximate. Just to recap, power-law graphs are sparse and a bit similar to trees. This fine line between easy and hard problems is an interesting phenomenon and naturally has attracted our attention. Here resides the lack of information that motivated our studies. Our experience suggests that the prior knowledge about the degree distribution helps a lot in preprocessing the input graph and reducing the size of the problem (as proposed in the preprocessing algorithms). Also, in this kind of problem, the solution is an acyclic graph. Naturally, sparse graphs tend to have fewer cycles than dense graphs. As the cycle elimination constraints are the bottleneck for the integer programming formulations of these problems, it may be less costly to solve the problem in this type of instance.

Regarding item (ii), previous works show that the algorithmic complexity is better explained in function of the power-law parameter β for some NP-hard problems. As showed by Ferrante et al. (2008), for some values of β, some combinatorial problems can change from easy to intractable. In the Positive Influence Dominating Set problem, an approximation ratio better than the factor for general graphs is given in (Dinh et al., 2014). Such a ratio is obtained in function of the exponent β. Further, for some strategies proposed in this thesis, when we use generative random graph models, we can estimate in advance what would be the impact of such a technique for a given value of β. The analysis in Section 6.2.1 is an example in which we use the $P(\alpha, \beta)$ model to estimate how much vertices would be removed in the preprocessing rule.

Toward item (iii), our algorithm to compute a dual bound for the LCIP (Chapter 7) is an example that we can strengthen an existing model based on the characteristics of the problem. Our preprocessing rules with the guarantee of optimality are a practical way to use the characteristics of the problem to speed up the solution. Furthermore,

in the literature, we have the example of (Fischetti et al., 2018), which proposes a new set of constraints for cycle elimination, which is integrally dependent on the problem's structure. These findings reveal that, despite the complexity of the problem, details that are often minute can support significant gains.

Due to recent advances in the study of related problems on influence propagation, as more generalizations are considered, more challenging becomes such problems. So our solutions are specialized for different versions of the problem. To summarize, each contribution in this thesis is for a different version of the problem. It is a drawback if we want a generic solution. However, the experiments suggest that it worth developing algorithms under these restrictive assumptions. Furthermore, although we are pursuing exact algorithms, we can not guarantee optimality in our preselection algorithm (Chapter 5). However, it still is a useful strategy as heuristic preprocessing. Lastly, the dual bound algorithm proposed in Chapter 7 do not need special information about the topology of the input graph to speed up the solutions. Even though it is a solution proposed to general graphs, it works quite well in power-law graphs.

Nevertheless, the limitations cited above suggest directions to follow in order to answer some open questions. There are some directions to follow in order to refine the preprocessing rules to achieve better results. First, combine our preprocessing rules with other techniques with similar purposes, aiming to preprocess more general graphs besides power-law. Next, incorporate these techniques into a linear programming based branch-and-bound framework, together with lower and upper bounds to speed up the solving time of such techniques.

Regarding the dual bound algorithm for the LCIP problem, our results show that the subject should be approached carefully, and we envision some space for improvements. For example, it is possible to improve the experimental results by finding a relaxed solution such that its value corresponds to the lower bound found by our algorithm. Also, in dense graphs, we observe that bounds behave better when $\alpha = 1$ than the case when $\alpha < 1$. However, when we are not interested in influencing all the individuals of a network and the sub-graphs are not strongly connected, we need to give up the efficiency of the algorithm to find higher lower bounds. Unfortunately, in this case, getting higher lower bounds implies solving a new NP-hard problem that we called WLCIP. Because of this, it could be preferable to keep the algorithm simple and efficient. Despite the theoretical conclusions about WLCIP, we do not rule out the possibility of finding other methods for obtaining a better dual bound in polynomial time in the case of $\alpha < 1$. We arrive at this specific problem because we assign weights to the vertices of the condensed component graph. Therefore, seeking new methods to approach this case is an open question in this study. To conclude, we believe that our theoretical findings on WLCIP can open up new research, for instance, deriving a strong formulation for this problem and finding combinatorial algorithms to solve it.

Lastly, although this thesis exhibits important computational experiments, we also dedicate efforts to provide theoretical information about our findings. We do it because we believe that substantial progress in practical applications is tightly linked to deeper mathematical intuitions. We considered power-law because it is a distribution that commonly appears in real-life problems. Thus, reaching improvements in solving the influence propagation problems on power-law graphs can be valuable since they can directly impact realistic applications.

REFERENCES

Ackerman, E., Ben-Zwi, O., and Wolfovitz, G. (2010). Combinatorial model and bounds for target set selection. *Theoretical Computer Science*, 411(44-46):4017–4022.

Aiello, W., Chung, F., and Lu, L. (2001). A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66.

Arora, A., Galhotra, S., and Ranu, S. (2017). Debunking the myths of influence maximization: An in-depth benchmarking study. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 651–666. ACM.

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

Bollobás, B., Borgs, C., Chayes, J., and Riordan, O. (2003). Directed scale-free graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 132–139. Society for Industrial and Applied Mathematics.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Brach, P., Cygan, M., Łącki, J., and Sankowski, P. (2016). Algorithmic complexity of power law networks. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1306–1325. Society for Industrial and Applied Mathematics.

Carnes, T. and Shmoys, D. (2008). Primal-dual schema for capacitated covering problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 288–302. Springer.

Chen, C.-L., Pasiliao, E. L., and Boginski, V. (2020). A cutting plane method for least cost influence maximization. In *International Conference on Computational Data and Social Networks*, pages 499–511. Springer.

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

Chen, W., Wang, C., and Wang, Y. (2010a). Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1029–1038. ACM.

Chen, W., Wang, Y., and Yang, S. (2009). Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 199–208. ACM.

Chen, W., Yuan, Y., and Zhang, L. (2010b). Scalable influence maximization in social networks under the linear threshold model. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 88–97. IEEE.

Choromanski, K., Matuszak, M., and Miekisz, J. (2013). Scale-free graph with preferential attachment and evolving internal vertex structure. *J Stat Phys*, 151:1782–1789.

Chung, F. and Lu, L. (2002). Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145.

Clauset, A., Shalizi, C. R., and Newman, M. E. (2009). Power-law distributions in empirical data. *SIAM review*, 51(4):661–703.

Cordasco, G., Gargano, L., Rescigno, A. A., and Vaccaro, U. (2015). Optimizing spread of influence in social networks via partial incentives. In *International Colloquium on Structural Information and Communication Complexity*, pages 119–134. Springer.

Cornuéjols, G. (2008). Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112(1):3–44.

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

de Melo, R. S., Vignatti, A. L., Miyazawa, F. K., and Ota, M. J. (2020). Tighter dual bounds on the least cost influence problem. In *Proceedings of the 52nd Brazilian Operational Research Symposium*. SBPO.

Demaine, E. D., Hajiaghayi, M., Mahini, H., Malec, D. L., Raghavan, S., Sawant, A., and Zadimoghadam, M. (2014). How to influence people with partial incentives. In *Proceedings of the 23rd international conference on World wide web*, pages 937–948. ACM.

Dinh, T. N., Shen, Y., Nguyen, D. T., and Thai, M. T. (2014). On the approximability of positive influence dominating set in social networks. *Journal of Combinatorial Optimization*, 27(3):487–503.

Domingos, P. and Richardson, M. (2001). Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57–66. ACM.

Easley, D. and Kleinberg, J. (2010). *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press.

Ferrante, A., Pandurangan, G., and Park, K. (2008). On the hardness of optimization in power-law graphs. *Theoretical Computer Science*, 393(1-3):220–230.

Fischetti, M., Kahr, M., Leitner, M., Monaci, M., and Ruthmair, M. (2018). Least cost influence propagation in (social) networks. *Mathematical Programming*, pages 1–33.

Friedrich, T. and Krohmer, A. (2015). Parameterized clique on inhomogeneous random graphs. *Discrete Applied Mathematics*, 184:130–138.

Gomory, R. E. et al. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical society*, 64(5):275–278.

Goyal, A., Bonchi, F., and Lakshmanan, L. V. (2011a). A data-based approach to social influence maximization. *Proceedings of the VLDB Endowment*, 5(1):73–84.

Goyal, A., Lu, W., and Lakshmanan, L. V. (2011b). Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web*, pages 47–48. ACM.

Goyal, A., Lu, W., and Lakshmanan, L. V. (2011c). Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 211–220. IEEE.

Granovetter, M. (1978). Threshold models of collective behavior. *American journal of sociology*, pages 1420–1443.

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

Günneç, D., Raghavan, S., and Zhang, R. (2016). Tailored incentives and least cost influence maximization on social networks. Technical report, Technical report.

Italiano, G. F., Laura, L., and Santaroni, F. (2012). Finding strong bridges and strong articulation points in linear time. *Theoretical Computer Science*, 447:74–84.

Kempe, D., Kleinberg, J., and Tardos, É. (2003). Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM.

Kleinberg, J. (2007). Cascading behavior in networks: Algorithmic and economic issues. *Algorithmic game theory*, 24:613–632.

Kunegis, J. (2013). KONECT – The Koblenz Network Collection. In *Proc. Int. Conf. on World Wide Web Companion*, pages 1343–1350.

Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., and Glance, N. (2007). Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Liu, X., Li, S., Liao, X., Peng, S., Wang, L., and Kong, Z. (2014). Know by a handful the whole sack: efficient sampling for top-k influential user identification in large graphs. *World Wide Web*, 17(4):627.

Melo, R. S. and Vignatti, A. L. (2018). A preselection algorithm for the influence maximization problem in power law graphs. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 1782–1789. ACM.

Melo, R. S. and Vignatti, A. L. (2020). Preprocessing rules for target set selection in complex networks. In *Proceedings of the IX Brazilian Workshop on Social Network Analysis and Mining*, pages 108–119. SBC.

Michail, D., Kinable, J., Naveh, B., and Sichi, J. V. (2019). Jgrapht–a java library for graph data structures and algorithms. *arXiv preprint arXiv:1904.08355*.

Mitchell, J. E. (2002). Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, pages 65–77.

Mitzenmacher, M. (2004). A brief history of generative models for power law and lognormal distributions. *Internet mathematics*, 1(2):226–251.

Mitzenmacher, M. and Upfal, E. (2005). *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press.

Nannicini, G., Sartor, G., Traversi, E., and Wolfler Calvo, R. (2020). An exact algorithm for robust influence maximization. *Mathematical Programming*, 183:419–453.

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. (1978). An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14(1):265–294.

Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Raghavan, S. and Zhang, R. (2015). Weighted target set selection on social networks. Technical report, Working paper, University of Maryland.

Schrijver, A. (1998). *Theory of linear and integer programming*. John Wiley & Sons.

Schrijver, A. (2003). *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media.

Shen, Y., Nguyen, D. T., and Thai, M. T. (2010). On the hardness and inapproximability of optimization problems on power law graphs. In *International Conference on Combinatorial Optimization and Applications*, pages 197–211. Springer.

Soltani, H. and Moazzez, B. (2019). A polyhedral study of dynamic monopolies. *Annals of Operations Research*, 279(1):71–87.

Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., and Su, Z. (2008). Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 990–998. ACM.

Tang, Y., Shi, Y., and Xiao, X. (2015). Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1539–1554. ACM.

Tang, Y., Xiao, X., and Shi, Y. (2014). Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 75–86. ACM.

Van Der Hofstad, R. (2016). Random graphs and complex networks.

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

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, 393(6684):440.

Wu, H.-H. and Küçükyavuz, S. (2018). A two-stage stochastic programming approach for influence maximization in social networks. *Computational Optimization and Applications*, 69(3):563–595.

Zhang, W., Wu, W., Wang, F., and Xu, K. (2012). Positive influence dominating sets in power-law graphs. *Social Network Analysis and Mining*, 2(1):31–37.