Graph-based maximization algorithms for submodular functions and influence maximization on social networks

PhD Thesis

Eszter Csókás
Supervisor: Tamás Vinkó, PhD

Doctoral School of Computer Science
Department of Computational Optimization
Faculty of Science and Informatics
University of Szeged

Szeged
2025

Contents

Abbreviations

BILP Binary Integer Linear Programming 6

CG Constraint generation algorithm 3, 10, 14, 15, 17, 18, 21, 22, 37, 38, 41, 42, 44–47

COV Weighted coverage 19, 24, 27–29, 43, 44, 46

DLTM Deterministic linear threshold model 10, 52, 63, 72, 73

ECG Improved constraint generation algorithm using enumeration 27–29, 33, 44, 46, 47, 98, 101, 104, 107

ER Erdős-Rényi model 8, 27

GCG Improved constraint generation algorithm using graph structure 25, 28, 29, 33, 46, 98, 101, 104, 107

IAtI The Influenceability and the Ability-to-Influence heuristic 77, 78, 80–85, 100, 106

ICG Improved constraint generation algorithm 18, 21, 23, 25, 28, 29, 33, 97, 98, 103, 104

ICG($k-1$) Improved constraint generation algorithm with reduced k 23, 28, 29, 33, 46, 98, 104

ILP Integer Linear Programming 5, 6, 54, 59, 61, 63, 64, 77, 79, 80, 85

IM Influence maximization 51–55, 63, 72, 73

INF Bipartite influence 19–21, 24, 27, 33, 43, 46, 47

LFR Lancichinetti-Fortunato-Radicchi benchmark 9, 64, 65, 67, 69, 71, 79–82

LOC Facility location 19, 20, 24, 27, 28

LP Linear Programming 4, 5, 55

MILP Mixed Integer Linear Programming 6, 21, 39, 98, 104

MIP Mixed Integer Programming 14, 15, 18, 19, 21–23, 26, 33, 39, 40, 43, 46, 47

NS-CG Constraint generation algorithm using new starting point as initial set 44, 46

NS-ECG Improved constraint generation algorithm using enumeration and new starting point as initial set 44, 46, 47

NS-GCG Improved constraint generation algorithm using graph structure and new starting point as initial set 44, 46, 47

NS-ICG Improved constraint generation algorithm using new starting point as initial set 44, 46, 47

NS-ICG($k-1$) Improved constraint generation algorithm with reduced k using new starting point as initial set 44, 46, 47

WS Watts-Strogatz model 9, 64–66, 68

Chapter 1

Introduction

Discrete optimization and graph-related problems are central topics in applied mathematics and computer science, with numerous practical applications in fields such as network analysis, social networks, and the optimization of complex systems. This dissertation addresses two fundamental problems: the submodular function maximization and the influence maximization, both framed within a graph-theoretic context. Accordingly, the thesis is divided into two parts along these two topics.

The first part of the dissertation focuses on the maximization of submodular functions, which is an attractive optimization model and also a well-studied problem with a variety of algorithms available. Constraint generation (CG) approaches are appealing techniques to tackle the problem with, as the mixed-integer programming formulation of the problem suffers from the exponential size of the number of constraints. Most of the problems in these topics are of combinatorial nature and involve graphs on which the maximization is defined. The first aim is to improve the runtime of existing solution algorithms by exploiting the graph structure of the problem and the inherent submodular properties. Therefore, in Chapter 2 we introduce variants of the CG algorithm which take into account certain properties of the input graph aiming at informed selection of the constraints.

In Chapter 3, we continued this theme, but shifted the focus a little. The greedy strategy quickly finds a feasible solution that guarantees an approximation of $(1-1/e)$ of the optimal solution. However, there are many applications that expect an optimal result within a reasonable computational time. Traditionally, the initial feasible solution of CG is given by the greedy algorithm. Thus, a new centrality metric is presented that proposes an initial starting point based on the structure of the input graph.

The second part of the thesis deals with influence maximization, which is a crucial problem in network analysis, mainly in the context of viral marketing, opinion formation, and information diffusion. It is a challenging combinatorial optimization problem on (social) networks given a diffusion model and limited choice for initial seed nodes. In Chapter 4, a solver algorithm is considered, which proposes a procedure with an integer program formalism. The correctness of the program is proved, and its efficiency is empirically demonstrated.

In addition, in Chapter 5, a new centrality-based metric was developed to identify seed nodes in the network, where two features are a combination: the susceptibility of the nodes to influence and their potential to spread the influence. This metric can be exploited to reduce the number of possible seed nodes, and finally from this smaller set the solver selects initial seed nodes, thus reducing the runtime.

Taken together, each of the 4 main chapters deals with an optimization problem for which a solver algorithm exists and for which the problem instances have a graph representation. These chapters are, naturally, divided into two parts by topic. However, another type of ordering can be observed, which does not come so naturally. Chapter 2 and Chapter 4 are based on recently published work that includes algorithms for solving the problem at issue. These methods and further work on them are presented. Chapter 3 and Chapter 5 are connected by the centrality metric. In both of them, taking into account the structure of the input graph, a new centrality metric is developed to assist the exact solver, thus reducing the running time of the algorithm.

The latter, "non-naturally" obtained entanglement is also reflected in the title, i.e., the focus is on maximizing procedures using graph instances.

However, it was necessary to split the thesis into two main parts because the influence propagation maximization is sub-modular, depending on the propagation model. The bipartite influence shown in the first part of the thesis is submodular, while the influence propagation considered in the second part, with a deterministic threshold model, does not show submodular properties. For this reason, I have found it necessary to separate the thesis into two parts, so that the results can be easily distinguished.

1.1 Optimization

1.1.1 Brief historical overview

Many people have asked when and by whom the first linear optimization problem was written. The ancient Greeks were looking for the best solutions to geometric problems. One interesting formulation of the problem leading to the linear programming problem was the so-called money-change problem, which was recorded by money-changers in 12th century in Italy. Its formal mathematical foundation came in the 17th century with the work of Newton and Leibniz. Furthermore, French military engineers in the 18th century also used the problem to address the issue of optimal material transport in the context of fortress construction.

Later, modern linear programming also came about through a series of steps. In 1902, Gyula Farkas published the Farkas lemma, which, although much later, played an important role in the theory of duality and its proof [35]. Significant progress was made when George B. Dantzig introduced the simplex algorithm in 1947 [20], which proved that it could be used to solve systems of linear inequalities. Finally, his method was published only in 1951. Since many practical problems could be formulated as LPs and experience showed that they

could be solved very quickly, the simplex algorithm soon became widespread.

In 1984, N. Karmarkar developed a new method, the so-called interior point method [61]. This algorithm is a theoretically and practically efficient procedure for solving LP in polynomial time. In contrast, the simplex iteration cannot be constrained by any polynomial of the task's dimensions, but nevertheless works reliably in practice.

Numerous Nobel Prizes in public science have been awarded for achievements in the field of operations research. Prominent among these is the 1975 Nobel Prize in Economics of Linear Programming, shared by Kantorovich and Koopmans.

1.1.2 Linear Programming (LP)

The aim of mathematical optimization is to support decision-making using mathematical methods. It is a fundamental topic in operations research whose basic task is, for a system of linear inequalities, to minimize a linear objective function. So we want to choose the best element among the possible configurations based on some criterion. Geometrically, we are interested in the extremal value of a linear function in a convex polyhedron. The standard formulation of a system of linear program is:

$$\begin{aligned} \min \quad & c^T x && (1.1)\\ \text{subject to} \quad & Ax \geq b && (1.2)\\ & x \geq 0, && (1.3) \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $c, x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. The vector x contains the *decision variables*. The *objective function* is described by equation (1.1), which specifies the function to be minimized (or maximized). The associated *constraints* are inequalities (1.2) and (1.3), which determine the bounds of the set of possible solutions. The *feasible solution* is the point that satisfies the constraints and the *optimal solution* is the best available feasible solution.

The set of possible solutions is closed, given by non-strict constraints, and we know that a continuous function on a non-empty, bounded and closed set $X \subset \mathbb{R}^n$ takes absolute minimum and maximum [103]. Thus, LP tasks can be classified into three classes with respect to the solution: either the task has an optimal solution; or it has no possible solution (infeasible); or the objective function is unbounded on the set of feasible solutions.

Optimization models can be classified according to several criteria. Of these, we will consider here the classification in terms of decision variables:

- Continuous, if each decision variable can take any value in a (not necessarily finite) interval, this is the general case.

- Integer Linear Programming (ILP), if the decision variables are integers, so it is satisfied that

$$x \in \mathbb{Z}^n. \tag{1.4}$$

- A restricted version of ILP, where the decision variables can only take a value of zero or one, i.e. they are binary, is called Binary Integer Linear Programming (BILP). That is, for the variable x the following is fulfilled:

$$x \in \{0,1\}^n. \tag{1.5}$$

- Mixed Integer Linear Programming (MILP) is when some of the decision variables are continuous and the others are integers:

$$\begin{aligned} \min \quad & c^T x + d^T y \\ \text{subject to} \quad & Ax + Dy \geq b \\ & x, y \geq 0 \\ & x \in \mathbb{Z}^n. \end{aligned} \tag{1.6}$$

1.1.3 Heuristics and metaheuristics

Heuristic solution methods are used to simplify a problem solution or even to solve problems that do not have an exact solving algorithm. In many cases, exact solution methods, while giving optimal results, may be computationally expensive, may take too long to run, may be prohibitive for large instances, and may require significant storage resources. In contrast, heuristic methods provide an approximate solution, where the goal is not necessarily to find the optimum, but to find a good enough result in a shorter, foreseeable time. It may also be useful to use heuristics to "help" the exact solver, i.e., either to provide a good initial solution or to improve an intermediate step of the algorithm. If the method fails to find a solution, we cannot be sure why: there is no feasible solution, or the algorithm just did not find it. In other words, we do not know how close the resulting solution is to the optimum, and, in general, there is no guarantee that the heuristic will find a "good" solution. Perhaps the best known heuristic is the greedy algorithm [107] and another very famous approach is hill climbing [57].

Similarly, metaheuristics is used when complex optimization problems need to be approximated. A good overview of metaheuristics is given in [6]. It employs two main strategies: exploiting the current search region for the most promising solution and exploring new search regions for efficient solution finding. They are adaptable and thus applicable to various optimization problems, such as in combinatorial optimization, continuous optimization, and multi-objective optimization. Perhaps the best known example of metaheuristic optimization algorithms is the genetic algorithm [38].

The main difference between heuristics and metaheuristics is that while heuristics are designed for specific problems, metaheuristics are used as general-purpose algorithms, regardless of the specifics of the optimization problem.

1.2 Graphs

1.2.1 A historical overview

Graph theory is one of the few research areas where we know exactly when and where it was born. In Königsberg, the capital of East Prussia, a heavy fleet boosted traffic and 7 bridges were built over the river that crossed the city. This gave rise to the riddle of the time: could the seven bridges be traveled over without crossing each one more than once? In 1735, the mathematician Leonhard Euler proved that no such road could be made [34]. To solve this problem, Euler modeled the problem where the different parts of the city were vertices and the bridges were the edges between the vertices. The term *graph* itself was finally introduced in 1878 by J. J. Sylvester. The first book on graph theory was later written by the Hungarian mathematician Dénes Kőnig in 1936.

In the first half of the 20th century, remarkable results were achieved using graphs in certain specific contexts. Mathematicians started to study graphs in terms of their structure: following the work of Pál Erdős and Alfréd Rényi, they began to use probability theory to study graphs. Their efforts gave rise to the theory of random graphs and introduced a model for generating random graphs, the Erdős-Rényi model [26, 27]. Another direction was also developing at this time, based on the results of László Lovász [74, 75] and Tibor Gallai [42]: perfect graphs were introduced.

Later, however, it turned out that the structures of real networks are very different from the random graph defined by Erdős and Rényi. With the technological development of the 1990s, it became possible to analyze more real data. Stanley Milgram started small-world experiments that led to a further analysis of real-world networks. It was then that the study of complex networks evolved and Watts and Strogatz's network model was born in 1998 [109], which attempts to describe small-world networks. Furthermore, a year later, in 1999, Barabási and Albert published a paper describing a preferential attachment algorithm [3]. This model can generate scale-free meshes characterized by a power-law degree distribution.

These models form the basis of the network science we know today. In the last few decades, mainly subfields related to real networks have emerged. Important to highlight are community detection, core/periphery network structures, social network analysis, diffusion models and controllability.

1.2.2 Basic definitions

Formally, a pair $G = (V, E)$ is called a *graph* if V is a finite set and $E \subseteq \binom{V}{2}$, where G is the graph, V is the set of vertices, and E is the set of edges of the graph. The vertices are connected by edges, i.e., they fit on vertices such that each edge has at least one and at most two vertices. If a graph has no parallel edges and no loop edges, then it is called a *simple graph*.

A directed graph G can be described by the quadruple (V, E, K, B). V is the set of

vertices, E is the set of edges; K and B are relations between V and E. Let $v \in V$, then every $e \in E$ is an edge: if vKe, then we say that v is the starting point of edge e. If vBe, then we say that v is the ending point of edge e. So, if we distinguish the starting and ending points of the edges of the graph, we speak of a *directed graph*, otherwise the graph is *undirected*. An edge *loops* if its starting and ending points are the same.

A graph G is called a *weighted graph* if its edges are assigned to real numbers. The real numbers assigned to the vertices are called the *threshold* of the vertex.

A graph G is called *connected* if any two points in G can be connected by a path. A (sub)graph whose any two points are connected by a path is called a *component*.

A graph $G = (V, E)$ is called *bipartite* if its vertices can be classified into disjoint sets A and F such that each edge E has one endpoint in A and one endpoint in F. A graph G is a *complete bipartite graph* or *biclique* if it is a special kind of bipartite graph where every vertex of the first set is connected to every vertex of the second set.

A *lattice graph* is a graph built over an n-dimensional grid that induces regular tiling of the space.

A *walk* describes a sequence of vertices and edges that traverse the graph, permitting revisits to both vertices and edges. A walk is called a *trace* if it does not contain repeated edges. A *path* is a walk where there are no repeated vertices or edges. The length of a path is determined by the number of edges it includes.

1.2.3 Centrality in general

A centrality measure shows the nodes' importance in a graph which is based on the location of the nodes within the graph. Accurately, if given a weighted graph $G = (V, E, W)$, a centrality measure is a function $C : V \to \mathbb{R}_+$, i.e., it assigns a non-negative centrality value to every node. The order of the nodes formed by the centrality values is usually more important than the centrality value itself [23, 99].

There are two main categories of centralities. One is based on shortest path, which includes closeness [94] and betweenness [41], among others. The other one is based on neighborhoods, where the centrality metrics, just to mention the most frequently used ones, are degree [100], eigenvector [7], and PageRank [8].

1.2.4 Random graphs

Random graph models are essential for graph generation. In my thesis, I use some of these models, which are briefly summarized below.

Erdős-Rényi model (ER) The model is the result of remarkable publications by Erdős and Rényi [26, 27, 28, 29, 30, 31, 32, 33]. From a modeling aspect, networks are made up of nodes and connections. If we want to reproduce the complexity of real networks, the important question is where should we make the connections. The idea of random networks

is to randomly create connections between nodes. A random network has N vertices and between each pair of nodes there is an edge with probability p.

Watts-Strogatz model (WS) Watts and Strogatz extended the random network model in [109]. It is also called the small-world model, since it is characterized by the small-world property and high clustering. Thus, it is typical for WS graphs to fall into the category of random networks and regular lattices. This is because random graphs have the small-world property but low clustering, while regular lattices have no small-world phenomenon but high clustering [87].

Lancichinetti-Fortunato-Radicchi benchmark (LFR) The LFR method creates networks in which both the degree number distribution and the size distribution of the clusters in the networks follow a power law distribution [71]. Networks, benchmarks in which the community structure is predefined can be used to test the accuracy of a community search algorithm [87]. In this work we exploit the community nature of the generated benchmark.

1.3 Combinatorial Optimization

Combinatorial optimization is a well-known discipline in operations research and computer science. It is the corresponding subfield of discrete optimization with many real-world applications. Its goal is to find an optimal object from a finite set of objects [24].

Graphs are central objects of study in combinatorial optimization due to the discrete nature of most problems and the ubiquity of network data in the real world [9]. Accordingly, combinatorial optimization problems over graphs are encountered in many application areas, such as scheduling, transportation, and social networks. Over the years, they have attracted significant attention from theoretical and algorithmic researchers, as they are mostly NP-hard problems. In fact, 10 of the 21 problems in [62] by Karp are decision versions of graph optimization problems, while most of the other problems can naturally be formulated on graphs. For the solution methods of such NP-hard graph optimization problems, there are traditionally 3 main directions: exact solution methods, approximate algorithms, and heuristics. The exact algorithms are mostly based on branch-and-bound or enumeration with an integer programming formulation. These are not efficient for large instances, i.e., the solution time can be prohibitively long. Approximate algorithms are polynomial-time, which may be desirable, but the optimality guarantee is not always satisfactory, or the approximation method for the problem does not exist. Heuristics have no theoretical guarantees, but are mostly fast solving procedures. Achieving adequate efficiency usually requires a problem-specific approach in the algorithm, which requires in-depth knowledge and often even trial-and-error [66].

This includes the two main topics of this thesis: influence maximization and maximization of submodular functions. These topics are described and the necessary concepts are

introduced in the beginning of the parts. The chapters of the thesis also contain different solution concepts, exact solvers and heuristics/heuristic steps for these problems.

1.4 Contributions

The focus of this dissertation is on maximization of submodular functions and maximization of influence propagation using a deterministic linear threshold model. There are two chapters on each topic, containing ideas and results that have been published in scientific papers.

Chapter 2: I present a constraint generation algorithm (CG) to solve the submodular function maximization problem. Three further versions of this solver algorithm are developed, based on the definition of submodular maximization and the structure of the input graph; these were constructed jointly with my supervisor. The effectiveness of the procedures is demonstrated by numerical testing, which I performed. The chapter is based on [17] publications.

Chapter 3: Selection of the initial constraint set is crucial when solving an integer linear programming model. Therefore, this chapter focuses on the selection of initial constraint conditions when using constraint generation type algorithms. A centrality metric was developed in collaboration with my supervisor, which proposes initial constraints given the input graph. I compare this starting point with other strategies: the greedy solution and a recently proposed GRASP heuristic. I conducted numerical tests to demonstrate the validity of the method. The results of this chapter are published in [18].

Chapter 4: I point out the flaws of an integer linear programming model to maximize influence propagation, introduced in a recent paper [65]. After the detection of the problem, the model is corrected and its correctness was demonstrated by theoretical investigations. Proposing and proving the theorems was an inseparable collaborative work with my supervisor, while I did the algorithm implementation and the numerical testing. We published the related results in [16].

Chapter 5: For the influence maximization problem, it is a research question of great interest whether we can find a vertex ranking (i.e., centrality) value that can be used to predict which vertices to choose as the seed set at the initial moment. We constructed two centrality metrics that can provide a ranking that helps to select relevant seed nodes using a deterministic linear threshold model (DLTM). The basic idea and formulation of the centrality metric was developed together with the supervisor. I was responsible for the technical parts: implementation and numerical testing. [19] contains the results of this chapter.

Part I

Submodular function maximization based on graph properties

Basic concepts and definitions

Submodular function maximization

A crucial problem in combinatorial optimization is the submodular function maximization, and in many cases it involves graphs on which the maximization is specified. The problem is well-studied and hence there are several proposed algorithms in the literature.

Let $N = \{1, \ldots, n\}$ be a finite set. The function $f : 2^N \to \mathbb{R}$ is called *submodular* if it fulfills $f(S) + f(T) \geq f(S \cap T) + f(S \cup T)$ for all $S, T \subseteq N$. There are many natural linkage between submodular functions and both convexity and concavity, see, e.g., in [68]. Perhaps more intuitive is the following equivalent definition: $f(\{i\} \mid S) \geq f(\{i\} \mid T)$ holds for every $S \subseteq T \subseteq N$ and $i \in N \setminus S$ where $f(\{i\} \mid S) := f(S \cup \{i\}) - f(S)$. A submodular function f is called *non-decreasing* if $f(S) \leq f(T)$ holds for all $S \subseteq T \subseteq N$. In the rest of the thesis it is assumed that f fulfills this property, i.e., it is a non-decreasing submodular function.

An important property of submodular functions is that the sum of two submodular functions is also submodular. To prove it, let $f_1, f_2 : 2^N \to \mathbb{R}$ be two submodular function. Then $f : 2^N \to \mathbb{R}$ with $f(A) = f_1(A) + f_2(A)$ is submodular:

$$\begin{aligned} f(A) + f(B) &= f_1(A) + f_2(A) + f_1(B) + f_2(B) \\ &\geq f_1(A \cup B) + f_2(A \cup B) + f_1(A \cap B) + f_2(A \cap B) \\ &= f(A \cup B) + f(A \cap B). \end{aligned} \tag{1.7}$$

That is, it holds for each component of f in each member of the inequality.

The submodular maximization problem with a cardinality constant k (where $0 < k \leq n$) is defined in the following way:

$$\begin{aligned} \max \quad & f(S) \\ \text{subject to} \quad & |S| \leq k,\ S \subseteq N. \end{aligned} \tag{1.8}$$

Solvability

Interestingly, the submodular function minimization problem can be solved in polynomial time [49, 55, 98], whereas the non-decreasing submodular function maximization is NP-hard [36, 69]. The greedy strategy is an often applied approach to solve (1.8) as it guarantees the $(1 - 1/e)$ approximation of the optimal solution [83], however, it might be computationally inefficient for large-scale instances. Although this is a simple, easy-to-implement technique for solving many optimization problems and is therefore very commonly used, the global optimum is often more needed in real-world applications, which was the motivation for a new solution method proposed in [82]. This is based on a mixed integer programming (MIP) model:

$$\begin{aligned} \max \quad & z \\ \text{s.t.} \quad & z \leq f(S) + \sum_{i \in N \setminus S} f(\{i\} \mid S) \cdot y_i, \quad S \in F, \\ & \sum_{i \in N} y_i \leq k, \\ & y_i \in \{0,1\}, \;\; i \in N, \end{aligned} \tag{1.9}$$

where $f(T \mid S) := f(S \cup T) - f(S)$ for all $S, T \subseteq N$ and F denotes the set of all feasible solutions satisfying the cardinality constraint $|S| \leq k$. For the proof for problem (1.9) being a reformulation of problem (1.8) the reader is referred to [82].

Since the number of constraints increases exponentially in (1.9), this motivated a new procedure, the so-called constraint generation (CG) algorithm, proposed in [82]. CG is an iterative algorithm that starts with solving a reduced problem. The reduced problem consists of a set of constraints generated from the initial set, which is extended on demand at each iteration by the addition of a feasible solution. So in essence it solves many reduced MIP problems, which are not always sufficiently efficient in applications. For this reason, the branch-and-bound algorithmic approach is often used, which exploits the relaxation of MIP.

Constraint generation algorithm (CG)

A constraint generation algorithm was proposed in [82], which starts from a reduced MIP problem with a few constraints to solve. It is an iterative algorithm, and in every iteration it solves the reduced MIP problem while adding a new constraint. Let us define the reduced problem MIP(Q) of (1.9), where $Q \subseteq F$ is a set of feasible solutions:

$$\begin{aligned} \max \quad & z \\ \text{s.t.} \quad & z \leq f(S) + \sum_{i \in N \setminus S} f(\{i\} \mid S) \cdot y_i, \quad S \in Q, \\ & \sum_{i \in N} y_i \leq k, \\ & y_i \in \{0,1\}, \;\; i \in N. \end{aligned} \tag{1.10}$$

Algorithm 1 CG($S^{(0)}$)

Input The initial feasible solution $S^{(0)}$.

Output The optimal solution S^* of problem (1.9).

Step 1: Set $Q \leftarrow \{S^{(0)}_{[0]}, \ldots, S^{(0)}_{[k]}\}$, $S^* \leftarrow S^{(0)}$ and $t \leftarrow 1$.

Step 2: Solve MIP(Q). Let $z^{(t)}$ be the optimal value of MIP(Q) and $S^{(t)}$ is the set corresponding to the optimal solution $\mathbf{y}^{(t)}$ of MIP(Q).

Step 3: If $f(S^{(t)}) > f(S^*)$, then let $S^* \leftarrow S^{(t)}$.

Step 4: If $z^{(t)} = f(S^*)$ holds, then output the solution S^* and exit.

Step 5: Set $Q \leftarrow Q \cup \{S^{(t)}\}$, $t \leftarrow t + 1$ and return to Step 2.

The pseudo code of CG is shown in Algorithm 1. The starting point of the algorithm is a set $Q = \{S^{(0)}_{[0]}, \ldots, S^{(0)}_{[k]}\}$, where $S_{[i]}$ is the first i elements of a feasible solution $S^{(0)}$ which comes from the order of the greedy algorithm's solution. In the t-th iteration we solve the problem MIP(Q), $Q = \{S^{(0)}_{[0]}, \ldots, S^{(0)}_{[k]}, \ldots, S^{(t-1)}\}$ to obtain the optimal solution $\mathbf{y}^{(t)} = (y^{(t)}_1, \ldots, y^{(t)}_n)$ and $z^{(t)}$ the optimal value which gives an upper bound of the problem (1.9). Let S^* be the best feasible solution of problem (1.9) up to this point and $S^{(t)} \in F$ be the set which generated the optimal solution $\mathbf{y}^{(t)}$ of MIP(Q), i.e., $\mathbf{y}^{(t)}$ is the characteristic vector of $S^{(t)}$. When $f(S^{(t)}) > f(S^*)$ holds, then update S^* with $S^{(t)}$. If $z^{(t)} > f(S^*) \geq f(S^{(t)})$ holds, then we have that $S^{(t)} \notin Q$, so (in Step 5) add $S^{(t)}$ to Q. This effectively adds the following constraint to MIP(Q):

$$z \leq f(S^{(t)}) + \sum_{i \in N \setminus S^{(t)}} f(\{i\} \mid S^{(t)}) \cdot y_i. \tag{1.11}$$

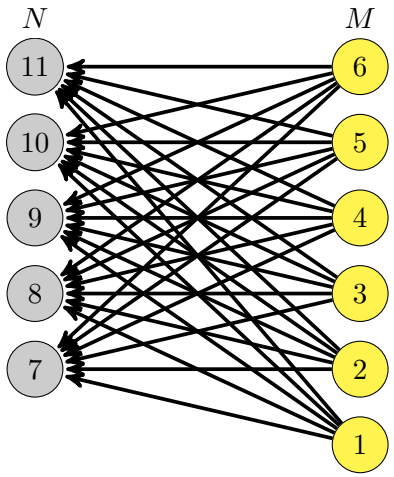
The algorithm stops when $z^{(t)} = f(S^*)$ is satisfied which means that it is proven that the optimal solution is found.

Small illustrative example For an easier understanding of the formalisms introduced and the CG procedure, a small example exercise is used to illustrate the CG procedure. The example graph is shown in Figure 1.1, and the table with the weights of the edges is shown in Table 1.1. The test benchmark is the example of a facility location, a typical submodular function, which will be discussed later. Let $k = 2$ be the cardinality constant. The CG algorithm solves the problem in 2 iterations, both iterations and the constraints obtained by condition (1.11) are included in the description below.

Table 1.1: *Edge weights of the graph used to illustrate the CG algorithm*

	7	8	9	10	11
1	0.32	0.59	0.02	0.14	0.02
2	0.46	0.41	0.36	0.51	0.06
3	0.09	0.12	0.05	0.12	0.23
4	0.38	0.05	0.58	0.67	0.14
5	0.58	0.23	0.46	0.17	0.09
6	0.28	0.45	0.60	0.73	0.59

Figure 1.1: *Example graph to demonstrate the CG algorithm*



Input The solution of Greedy: $S^{(0)} = 7,\ 10$.

Iteration 1:

set $Q[1] \leftarrow 10$
set $Q[2] \leftarrow 7,\ 10$

$S^* \leftarrow 7,\ 10 \quad f(S^*) = 2.93$

constraints[1]: $-0.59 \cdot y[7] - 0.51 \cdot y[8] - 0.29 \cdot y[9] - 0.11 \cdot y[11] + z \leq 2.34$
constraints[2]: $-0.27 \cdot y[8] - 0.11 \cdot y[11] + z \leq 2.93$

$S^{(1)} \leftarrow 7,\ 8 \quad f(S^{(1)}) = 2.58$

$z^{(1)} = 3.20$

Iteration 2:

set $Q[1] \leftarrow 10$
set $Q[2] \leftarrow 7,\ 10$
set $Q[3] \leftarrow 7,\ 8$

$f(S^*) = 2.93$

constraints[1]: $-0.59 \cdot y[7] - 0.51 \cdot y[8] - 0.29 \cdot y[9] - 0.11 \cdot y[11] + z \leq 2.34$
constraints[2]: $-0.27 \cdot y[8] - 0.11 \cdot y[11] + z \leq 2.93$
constraints[3]: $-0.35 \cdot y[9] - 0.62 \cdot y[10] - 0.25 \cdot y[11] + z \leq 2.58$

$S^{(2)} := 7,\ 9 \quad f(S^{(2)}) = 2.63$

$z^{(2)} = 2.93$ which is equivalent to $f(S^*)$

Chapter 2

Constraint generation approaches

2.1 Introduction

In recent years, the theory of submodular maximization has been improved and it has played a key role in extraordinarily varied application areas [45]. Examples include several classes of important combinatorial optimization problems [45], namely, the Simple or "uncapacitated" Plant (facility) Location Problem (SPLP) and its competitive version [4], the Quadratic Cost Partition Problem (QCP) with non-negative edge weights, and its special case – the Max-Cut Problem [46], the generalized transportation problem. Many different problems of data mining and knowledge discovery in biomedical and bioinformatics research (e.g., diagnostic hypothesis generation, logical methods of data analysis, conceptual clustering, and protein functional annotations) as well as applied to statistics, machine learning and experimental design [47, 70], multiobject tracking [101], sparse reconstruction [22], influence spread [64], and also combining multiple heuristics online [68]. There are models in mathematics [45], including the rank function of elementary linear algebra, which is a special case of matroid rank functions [25, 40] that require the solution of a submodular maximization problem.

Solving submodular optimization problems on graphs are also a popular line of study as set functions can easily be defined on graphs. The objectives in these graph based problems vary from simultaneous localization and mapping problem for robots [12], route planning, such as mobile robot sensing and door-to-door marketing [115], and investigation of more general classes involving, e.g., s-t path constraint [96].

This chapter contributes to the research effort invested into submodular function maximization with cardinality constraints by introducing efficient variations of a recently proposed constraint generation algorithm [106]. More precisely, we introduce variants of the CG algorithm which take into account certain properties of the input graph aiming at informed selection of the constraints.

Roadmap: The rest of the chapter is organized as follows. First, we discuss the relevant literature for the non-decreasing submodular function maximization in Section 2.2. Then,

Section 2.3 is about the problem instances that we use for testing the algorithms. In Section 2.4 we describe the algorithms. The first one is the improved constraint generation (ICG) algorithm proposed in [106]. This is followed by the introduction of our ICG modifications, where we motivate and describe three variants. The numerical experiments are presented in Section 2.5, including the description of the computational environments, the properties of the test graphs and finally the details and discussion of the benchmarking results. The chapter is concluded in Section 2.6.

2.2 Related works

Optimization of submodular functions has been actively studied. In this section we discuss the relevant contributions for the non-decreasing submodular function maximization.

The paper [68] provides analysis on the theoretical approximation guarantees of the solver algorithms (greedy as well as more complex methods). The authors also considered the different types of extensions of submodular optimization such as the online settings and adaptive optimization problems. In contrast to our work, the entire paper is dedicated to approximation (incomplete) algorithms.

An A^* search algorithm was proposed in [11]. More precisely, it is a framework for solving non-decreasing submodular optimization problems. The approximate guarantee based on submodularity property is also provided for non-submodular functions. In principle, the A^* algorithm is a heuristic, whereas our approach solves the problem to optimality.

A new implementation of the CG approach was proposed in [97] to solve the problem using the formalism (1.9). The implementation uses lazy constraints, and heuristics are used to select the starting point. This work is discussed in more detail in Section 3.3, furthermore numerical tests for starting point selection are also presented in Chapter 3.

The paper [85] formulated the submodular maximization under submodular cover problem and proposed an approximation framework to solve it. The algorithm provably produces nearly optimal solutions. As this paper aims at solving a specified version of (1.8) its applicability differs from those of our approaches.

A deterministic algorithm based on a new greedy strategy for solving problem (1.8) was proposed by [76]. It is shown by a mathematical proof that this new strategy outperforms the traditional greedy algorithm provided that the function f fulfills certain assumptions.

Finally, [106] presented an improved constraint generation (ICG) algorithm. Being an iterative method, it starts from a small subset of constraints and repeatedly solves relaxed problems while adding a promising set of constraints at each iteration. The ICG method was included into a branch-and-cut algorithm to attain good upper bounds while solving a smaller number of reduced MIP problems. Computational results were obtained for well-known benchmark instances. In Section 2.4 we present this ICG algorithm because our work is based on it. In fact, we created three variants of it with the aim of even better computational efficiency. Note that we did not use the branch-and-cut algorithm.

2.3 Benchmark sets

We use 3 types of well-known and frequently used examples which have the non-decreasing submodular property, termed by facility location (LOC), weighted coverage (COV) and bipartite influence (INF), see [63, 68, 95]. These are the standard benchmarks that are commonly used for testing in the literature. It is important to mention that LOC and COV examples have straightforward MIP formulations, which can be solved by standard MIP solvers quite efficiently, see [106]. On the other hand, the INF problem, to be introduced below cannot be formulated as straightforward MIP and this gives reasons for the attempt to make the universal submodular maximization framework more efficient [97].

Facility location (LOC) Let n be the number of locations and m be the number of clients. The set of locations $N = \{1, \ldots, n\}$ and the set of clients $M = \{1, \ldots, m\}$ are given. Define $g_{i,j} > 0$ as a non-negative profit when client $i \in M$ is served by location $j \in N$, for all possible pairs. We select a set of k locations $S \subseteq N$ to build the facilities. Each client $i \in M$ gets the profit from the best opened facility, and we want to maximize the overall profit, so the total benefit is defined as:

$$f(S) = \sum_{i \in M} \max_{j \in S} g_{i,j}. \tag{2.1}$$

The submodular property of the problem can be proved in the following way: $f(S)$ can be written as $f(S) = \sum_{i \in M} f_i(S)$, i.e., according to (1.7), if $f_i(S) = max_{j \in S} g_{i,j}$ (the maximum of the i-th vector) is submodular then f can be written as the sum of submodular functions, so f is submodular. For the proof we will now use only the i-th vector of the matrix g. Consider

$$\max_{j \in A} g_j + \max_{j \in B} g_j \geq \max_{j \in A \cup B} g_j + \max_{j \in A \cap B} g_j$$

which follows since we have that

$$\max(\max_{j \in A} g_j + \max_{j \in B} g_j) = \max_{j \in A \cup B} g_j \quad \text{and} \quad \min(\max_{j \in A} g_j + \max_{j \in B} g_j) \geq \max_{j \in A \cap B} g_j.$$

Weighted coverage (COV) Let n be the number of sensors and m be the number of items. The set of sensors $N = \{1, \ldots, n\}$ and the set of items $M = \{1, \ldots, m\}$ are given. Each sensor $j \in N$ covers the subset of items $M_j \subseteq M$ and each item $i \in M$ has a non-negative weight w_i. To cover the items we select a set of sensors $S \subseteq N$. We have $a_{i,j} = 1$, if $i \in M_j$ and $a_{i,j} = 0$ otherwise. Then, the total weighted coverage is:

$$f(S) = \sum_{i \in M} w_i \max_{j \in S} a_{i,j}. \tag{2.2}$$

It can also be interpreted as a special case of the LOC problem, where $g_{ij} = w_i a_{ij}$, though note that the LOC problem is constrained to be $g_{ij} \neq 0$. Therefore, the proof of the submodularity of the function is analogous to the LOC problem.

Bipartite influence (INF) Let $M = \{1, \ldots, m\}$ be the set of targets, where m is the number of targets and $N = \{1, \ldots, n\}$ be the set of items, where n is the number of items. Define an influence maximization problem on a bipartite graph $G = (M, N; E)$, where $E \subseteq M \times N$ is a set of directed edges. The activation probability $p_j \in [0, 1]$ of every $j \in N$ item is given. Let $1 - \prod_{j \in S}(1 - q_{ij})$ be the probability that a set of items $S \subseteq N$ activates a target $i \in M$, where $q_{ij} = p_j$ if (i, j) is a directed edge in E, otherwise $q_{ij} = 0$ holds. The definition of the number of targets activated by the element set $S \subseteq N$ is:

$$f(S) = \sum_{i \in M} \Big(1 - \prod_{j \in S} \big(1 - q_{ij}\big)\Big). \tag{2.3}$$

To prove the submodularity of the function, it is sufficient to consider the submodularity of the function $f_i(S) = 1 - \prod_{j \in S} \big(1 - q_{ij}\big)$ due to (1.7). The proof is done with q matrix for a fixed value i, namely the i-th vector. Then, we will prove the following inequality:

$$\big(1 - \prod_{j \in A}(1 - q_j)\big) + \big(1 - \prod_{j \in B}(1 - q_j)\big) \geq \big(1 - \prod_{j \in A \cup B}(1 - q_j)\big) + \big(1 - \prod_{j \in A \cap B}(1 - q_j)\big).$$

Subtracting the constants from both sides and rearranging them:

$$\prod_{j \in A \cup B}(1 - q_j) + \prod_{j \in A \cap B}(1 - q_j) \geq \prod_{j \in A}(1 - q_j) + \prod_{j \in B}(1 - q_j). \tag{2.4}$$

In order to simplify the notation, let us define the followings:

$$r = \prod_{j \in A \setminus B}(1 - q_j), \qquad s = \prod_{j \in B \setminus A}(1 - q_j) \quad \text{and} \quad t = \prod_{j \in A \cap B}(1 - q_j).$$

Then, for A and B:

$$\prod_{j \in A}(1 - q_j) = r \cdot t \quad \text{and} \quad \prod_{j \in B}(1 - q_j) = s \cdot t.$$

For $A \cup B$, since $A \cup B = (A \setminus B) \cup (B \setminus A) \cup (A \cap B)$: $\prod_{j \in A \cup B}(1 - q_j) = r \cdot s \cdot t$. Let's substitute these into the 2.4 inequality:

$$r \cdot s \cdot t + t \geq r \cdot t + s \cdot t.$$

The inequality is divisible by t, since $t \geq 0$ ($(1 - q_{ij}) \in [0, 1]$ and if $t = 0$ the inequality holds trivially):

$$r \cdot s + 1 \geq r + s.$$

Rearrange this we obtain $1 - r - s + r \cdot s \geq 0$ and the final formula is

$$(1 - r)(1 - s) \geq 0.$$

Since r and s are the results of the product of the expressions $1 - q_{i,j}$ where $q_{i,j} \in [0, 1]$, then $r, s \in [0, 1]$. Therefore, both $(1 - r)$ and $(1 - s)$ are nonnegative, which makes their multiplication nonnegative. This proves the inequality.

Note that the INF problem as a submodular function given in formula (2.3) contains a product involving elements from the set S, which, in case of formulating it using binary variables corresponding to set element selections would result in a polynomial type non-linear program. That *might* be possibly converted into MILP using, e.g., McCormick formalism, but it is neither straightforward nor promising with respect to its solvability.

2.4 Algorithms

In this section, first we describe the so-called improved constraint generation (ICG) solution method which was introduced recently by [106]. We propose three modifications of ICG in which either certain characteristics of the graph describing the problem is used or the submodularity property of the function to be maximized is exploited.

2.4.1 Improved constraint generation (ICG)

An improved constraint generation (ICG) algorithm was proposed in [106] and is shown in Algorithms 2 and 3. This algorithm variant is as appropriate for maximizing non-decreasing submodular functions as the CG algorithm.

What follows is that we give a quick overview and summary of the most important concepts and notations of ICG, the reader is referred to the paper of [106] for the full details. Note that we changed Step 5 in Algorithm 2 to expand the set Q with $S^{(t)}$ only *after* executing Algorithm 3 (SUB-ICG) since for the inputs it is assumed that $S^{(t)} \notin Q$.

The ICG algorithm refers to a reduced problem of (1.9) as $MIP(Q)$, where Q is a set of

feasible solutions, and it is defined as[1]:

$$\begin{array}{ll} \max & z \\ \text{s.t.} & z \leq f(S) + \sum\limits_{i \in N \setminus S} f(\{i\} \mid S) \cdot y_i, \quad S \in Q, \\ & \sum\limits_{i \in N} y_i \leq k, \\ & y_i \in \{0,1\}, \;\; i \in N. \end{array} \tag{2.5}$$

Using the CG algorithm of [82] as baseline, in the t-th iteration let the optimal solution be $\mathbf{y}^{(t)} = (y_1^{(t)}, \ldots, y_n^{(t)})$ and let the optimal value of the problem $MIP(Q)$ be $z^{(t)}$. In this case $z^{(t)}$ is an upper bound for problem (1.9), and that is what we aim to decrease in the subsequent iterations. In order to do so, it is required to add a new feasible solution $S' \in F$ to the set Q to fulfill the inequality:

$$z^{(t)} > f(S') + \sum_{i \in N \setminus S'} f(\{i\} \mid S') y_i^{(t)}. \tag{2.6}$$

By solving $MIP(Q)$ we obtain at least one feasible solution $S^\natural \in Q$ which satisfies the equation

$$z^{(t)} = f(S^\natural) + \sum_{i \in N \setminus S^\natural} f(\{i\} \mid S^\natural) y_i^{(t)}. \tag{2.7}$$

Algorithm 2 ICG$(S^{(0)}, \lambda)$

Input The initial feasible solution $S^{(0)}$ and the number of feasible solutions to be generated at each iteration λ.

Output The optimal solution S^*.

Step 1: Set $Q \leftarrow S^{(0)}$, $Q^+ \leftarrow \{S_{[0]}^{(0)}, \ldots, S_{[k]}^{(0)}\}$, $S^* \leftarrow S^{(0)}$, and $t \leftarrow 1$.

Step 2: Solve $MIP(Q^+)$. Let $S^{(t)}$ and $z^{(t)}$ be, respectively, an optimal solution and the optimal value of $MIP(Q^+)$.

Step 3: If $f(S^{(t)}) > f(S^*)$, then let $S^* \leftarrow S^{(t)}$.

Step 4: If $z^{(t)} = f(S^*)$ holds, then output the solution S^* and exit.

Step 5: Set $Q^+ \leftarrow Q^+ \cup \{S^{(t)}\} \cup$ SUB-ICG$(Q, S^{(t)}, \lambda)$, $Q \leftarrow Q \cup \{S^{(t)}\}$, and $t \leftarrow t + 1$.

Step 6: For each feasible solution $S \in$ SUB-ICG$(Q, S^{(t)}, \lambda)$, if $f(S) > f(S^*)$ holds, then set $S^* \leftarrow S$. Return to Step 2.

[1] Note that problem (2.5) is referred as $BIP(Q)$ in [106]

Algorithm 3 SUB-ICG $(Q, S^{(t)}, \lambda)$

Input A set of feasible solutions $Q \subseteq F$, a feasible solution $S^{(t)} \notin Q$ and the number of feasible solutions to be generated λ.

Output A set of feasible solutions $Q^{'} \subseteq F$.

Step 1: Set $Q' \leftarrow \emptyset$ and $h \leftarrow 1$.

Step 2: Select a feasible solution $S^{\natural} \in Q$ satisfying the equation (2.7) at random, therefore solve $MIP(Q)$. Set a random value r_i $(0 \leq r_i \leq p_i)$ for $i \in S^{\natural} \cup S^{t}$.

Step 3: If $\mid S^{\natural} \mid = k$ holds, then take the k largest elements $i \in S^{\natural} \cup S^{(t)}$ with respect to r_i to generate a feasible solution $S^{'} \in F$. Otherwise, take the largest element $i \in S^{(t)} \setminus S^{\natural}$ with respect to r_i to generate a feasible solution $S^{'} = S^{\natural} \cup \{i\} \in F$.

Step 4: If $S^{'} \notin Q^{'}$, then set $Q^{'} \leftarrow Q^{'} \cup \{S^{'}\}$ and $h \leftarrow h + 1$.

Step 5: If $h = \lambda$ holds, then output $Q^{'}$ and exit. Otherwise, return to Step 2.

For all $i \in N$ let q_i be the number of feasible solutions $S \in Q$ including an element i. Using this quantity the occurrence rate p_i of element i is calculated as:

$$p_i = \frac{q_i}{\sum_{j \in N} q_j} \tag{2.8}$$

For the heuristic part of the algorithm, meaning selecting the added nodes, we use r_i that is generated uniformly at random from 0 and p_i, i.e., $0 \leq r_i \leq p_i$. We choose one solution at random from $S^{\natural} \in Q$ in that case when multiple feasible solutions exist which are satisfying equation (2.7).

2.4.2 ICG with reduced k (ICG($k-1$))

As a first variation of the ICG, we modified Step 3 of Algorithm 3 to select not the k largest elements if $S^{\natural} = k$ holds, but the $k-1$ largest elements ($i \in S^{\natural} \cup S^{(t)}$ with respect to r_i). Considering the reduced problem's definition in (2.5), with this modification we get the k-th element's function value when we add it to the set. Thus the problem class does not change, ICG($k-1$) is a general solution method for non-decreasing submodular functions.

According to our empirical investigations, to be shown in Section 2.5, ICG($k-1$) is more efficient in terms of average running time than ICG, thus in the following we use this version of the algorithm as a base for further improvements.

2.4.3 ICG using graph structure (GCG)

Our second variant uses the structure of the input graph instances to calculate the p_i value in (2.8) that is needed to generate r_i by random, see Step 2 in Algorithm 3. For this we distinguish the problems defined on fully connected graphs (that is the LOC problem in our case) and on non-fully connected graphs (those are the COV and INF problems we consider). Therefore, the problem class is changed here, and the algorithm is applied to maximize non-decreasing submodular functions with graph representation.

In the case when the problem is a fully connected bipartite graph, for every vertex $j \in M$ we calculate the median m_j of the outgoing edges' weights. Recall that M refers to the set of nodes with outgoing edges only. To calculate the value of p_i, we add up the weights of the incoming edges at node $i \in N$ normalized with the degree of the node in M corresponding to the edge:

$$p_i = \begin{cases} \displaystyle\sum_{\substack{j:(j,i)\in E(G),\\ w_{ji}\geq m_j}} \frac{w_{ji}}{d_j} & \text{if } G \text{ is fully connected bipartite graph,} \\ \displaystyle\sum_{j:(j,i)\in E(G)} \frac{w_{ji}}{d_j} & \text{otherwise,} \end{cases} \tag{2.9}$$

where G is the input graph of the optimization problem, $E(G)$ is the set of edges of G, the edges have w_{ji} weights and d_j is the degree of the node $j \in M$. This defines the value of p_i and based on this we set the value of r_i by uniformly at random such that the relation $0 \leq r_i \leq p_i$ holds.

As an illustrative example, see the graph on Figure 2.1, where the labels of the nodes are indicated as black numbers and the sets N and M are marked above the vertices. The results of equation (2.9) are shown in Figure 2.1: the value of p_i for the node is to the left.

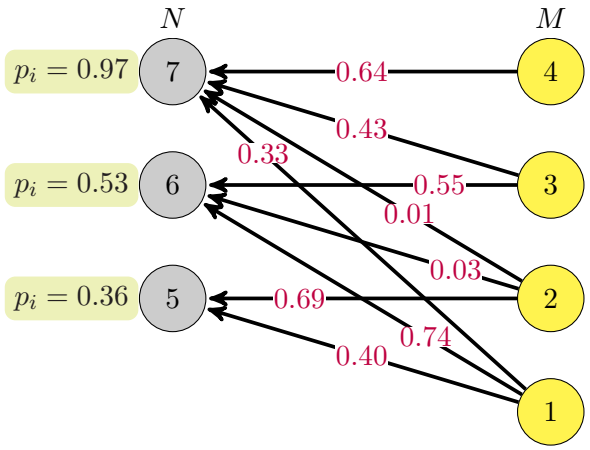


Figure 2.1: *Example graph to calculate the p_i values*

Although we choose k nodes from N we give a selection probability p_i by using nodes in M. The weight of outgoing edges of $j \in M$ are divided by the degree of j, which is used to normalize the effect of the edges. We sum this normalized edges weights for each node $i \in N$, which expresses the average impact of selecting node $i \in N$ relative to the other N nodes.

Let's revisit the example in Figure 2.1. There is only one edge from node 4 to node 7 so node 7 is important: node 4 can be served only if node 7 is selected. For this reason, we add to the selection probability value (p_i) an edge weight divided by 1 (i.e., it remains itself), so that the probability of selection of node 7 becomes high. In contrast, node 1 is connected to three nodes, indicating that it can be reached from nodes $5, 6$, and 7. These nodes are interchangeable for reaching node 1 and therefore less important. If we investigate the p_i value which are in Figure 2.1, next to the nodes, then we can see that those values are corresponding to the number and the weight of edges from nodes in set N. Accordingly, node 7 has the highest value because node 3 and 4 have one and two edges, respectively. Node 5 has the smallest p_i value since node 2 and 1, which are connected to it have edges to all vertices in N.

2.4.4 ICG using enumeration (ECG)

The formal description of our third approach is shown in Algorithm 4. The first four steps, namely Step 1 - 4 are similar to those used in the previous variants. Step 5 selects a feasible solution $S^\natural \in Q$ uniformly at random which satisfies the equation (2.7). Then, in Step 6 the set Σ is initialized as the union of the sets $S^\natural$ and $S^{(t)}$. The size of Σ is controlled by the parameter κ. In the subsequent steps the algorithm deals with the subsets of Σ (this is the enumeration part), where we need to balance between computational cost and the benefit of obtaining lower bounds of high quality. Thus, if $\mid \Sigma \mid > \kappa$, then we keep at most κ elements with the largest p_i values which is given by equation (2.9). Step 7 defines the set $\mathcal{P}_{k-1} \subseteq 2^{\Sigma}$. From all elements in $\mathcal{P}_{k-1}$ we keep only those with cardinality at most $k-1$ and calculate their function values. In Step 8, based on their function values we keep at most λ elements in $\mathcal{P}_{k-1}$, where λ is another control parameter of the algorithm.

The motivation behind Steps 5 - 8 is based on the following facts. Firstly, in contrast to ICG and GCG, we do not iterate any sub-algorithm inside the main algorithm, which could lead to less computational time. Secondly, in order to select the elements (i.e., nodes of the input graph) for Σ we use the p_i values, which are based on the degree properties of the input graph. Finally, the algorithm generates a good amount of promising feasible solutions, and, similar to ICG, these solutions can help decreasing the upper bound of problem (1.9), see (2.6). This depends on the strategy to be used for keeping the elements in $\mathcal{P}_{k-1}$ in Step 8. Note that in our experiments we used the strategy of keeping the elements with the smallest function values, as this turned out to be numerically efficient, see Section 2.5.

Since the algorithm also uses the basic idea of GCG, i.e., graph structure, it is also appli-

cable to non-decreasing submodular functions with graph representation.

Algorithm 4 ECG $(S^{(0)}, \lambda, \kappa)$

Input The initial feasible solution $S^{(0)}$, the number of feasible solution λ, and κ is the number of the elements of Σ.

Output The optimal solution S^*.

Step 1: Set $Q \leftarrow S^{(0)}$, $Q^+ \leftarrow \{S^{(0)}_{[0]}, \ldots, S^{(0)}_{[k]}\}$, $S^* \leftarrow S^{(0)}$, and $t \leftarrow 1$.

Step 2: Solve $MIP(Q^+)$. Let $S^{(t)}$ and $z^{(t)}$ be an optimal solution and the optimal value of $MIP(Q^+)$.

Step 3: If $f(S^{(t)}) > f(S^*)$, then set $S^* \leftarrow S^{(t)}$.

Step 4: If $z^{(t)} = f(S^*)$ holds, then output the solution S^* and exit.

Step 5: Select a feasible solution $S^\natural \in Q$ satisfying the equation (2.7) at random.

Step 6: Set $\Sigma = S^\natural \cup S^{(t)}$. If $\mid \Sigma \mid > \kappa$ then let Σ be the first κ elements with the largest p_i values from $S^\natural \cup S^{(t)}$, where p_i is defined in (2.9).

Step 7: Let $\mathcal{P}_{k-1}$ be the set of all the subsets of the power set of Σ which have at most $k-1$ elements and assign the corresponding function values to these subsets.

Step 8: Keep at most λ elements in $\mathcal{P}_{k-1}$. So at this point $\mid \mathcal{P}_{k-1} \mid \leq \lambda$.

Step 9: Set $Q \leftarrow Q \cup \{S^{(t)}\}$, $Q^+ \leftarrow Q^+ \cup \{S^{(t)}\} \cup \mathcal{P}_{k-1}$ and $t \leftarrow t+1$.

Step 10: For each feasible solution $S \in \mathcal{P}_{k-1}$, if $f(S) > f(S^*)$ holds, then set $S^* \leftarrow S$. Return to Step 2.

2.5 Numerical experiments

2.5.1 Computational environment

The implementation of all the investigated algorithms and models were done in AMPL [39]. For the numerical experiments the solver CPLEX 20.1.0.0 was used with the default options. The computer used had Intel Core CPU i5-6500 at 3.20GHz with 64G memory running Ubuntu Linux 22.04.

2.5.2 Test graphs

The authors of [106] made the graphs they used in their paper available online, thus for benchmarking the proposed methods we used those instances for the LOC and the COV problems. For the INF problems we generated new random graphs because the original graphs become rather easy to solve, all of the algorithms attain the optimal solutions. Our bipartite INF graphs were generated using the Erdős-Rényi (ER) model with probability $p = 0.3$. Note that [95] also used $p = 0.3$ in their experiments, whereas in [106] the parameter $p = 0.1$ was applied resulting sparser graphs.

Following the approach in [106] we had:

- $N = 20, 30, 40, 50,$ and 60 for LOC instances and $N = 20, 40, 60, 80,$ and 100 for COV and INF instances;
- $M = N + 1$ and $k = 5, 8$ for LOC, COV, and INF.
- For LOC instances, g_{ij} is a random value taken from the interval $[0, 1]$;
- for COV instances, a sensor $j \in N$ randomly covers an item $i \in M$ with probability 0.15, and w_i is a random value taken from the interval $[0, 1]$; and
- for INF instances, p_j is a random value taken from the interval $[0, 1]$.
- We had $\lambda = 10 \cdot k$.
- Finally, for controlling the cardinality of set Σ in ECG we had $\kappa = 12$ based on computational results of prior experiments.
- Note that all the random parameters were generated with uniform distribution.

Regarding the analysis of the parameters λ and κ we have created a Supplementary Information file which is available online [14].

2.5.3 Benchmarking results

The results are summarized in Tables 2.1 - 2.8. For each class 5 problem instances were tested, indicated in the last digit of the name of the instance. For every instance all the algorithms were run 5 times using different random seeds for the heuristics choices. The time limit was set to two hours (7 200 seconds). The solution of the greedy algorithm was the input initial feasible solution $S^{(0)}$ for every algorithm.

Tables 2.1 - 2.6 report the average running times (in seconds) and the mean values of the number of iterations (in brackets). The cases when an algorithm was running out of the time limit are indicated by the 🕒 symbol. Note that in these cases the number of iterations is not reported. Those cases when an algorithm was able to solve the problem for less than 5 different runs are indicated with a star ($\star$) next to the reported running time. Only those

instances are reported for which we had at least one algorithm solving the problem at least one of the cases.

Tables 2.7 and 2.8 report the mean relative gap[2] and how many times the algorithm was *not* able to solve the instance out of the 5 independent runs (in brackets). Note that only those instances are reported for which we had at least one algorithm which was running out of time at least once. For clarity, the results 0.00 (0) are replaced by the symbol $\diamondsuit$.

In the following we give detailed analysis of the reported results. The performance metrics of the proposed algorithms are discussed for the benchmark problems. For the different k values we compare our methods with ICG. The result of the fastest algorithm is indicated with boldface.

LOC, $k = 5$ The results are reported in Table 2.1. The ICG($k-1$) algorithm reduced the running time for almost all the cases (except once). To be precise, it was 3.46 times faster on average than ICG and it reduced the number of iterations to 82.62%. Furthermore, GCG was 4.49 times faster on average and reduced the number of iterations to 64.77%, while ECG was 6.10 times faster than ICG and the number of iterations were almost halved (53.71%). There are three cases where GCG was faster than ECG.

The relative gap values for those three graphs for which the ICG did not get the results are reported in Table 2.7. We can see that the gaps were below 1%.

LOC, $k = 8$ The results are reported in Table 2.2. Generally, the ICG($k-1$) obtained the result 1.79 times faster while the number of iteration was 94.76% of the ICG's iterations. GCG was 2.58 times faster and the number of iterations was reduced to 76.83%. Note that while ECG reduced the iterations to a similar extent (76.21%), it was faster only by 1.86 times compared to ICG. This confirms the fact that the technical time for generating the set of sets in the ECG increases when $k = 8$ (compared to $k = 5$), since GCG is faster with almost the same number of iterations.

For the unresolved instances, Table 2.7 shows the relative gap values. Note that none of the algorithms were able to solve the $N = 60$ problems. Most of the cases all of our algorithms achieved a smaller gap than ICG. Overall, the relative gaps remained below 1% for all reported problems.

COV, $k = 5$ The results are reported in Table 2.3. All of our proposed algorithms were able to solve the instances within the time limit, whereas ICG was running out of time for 7 cases, mostly for the largest dimensional problems (for these problems the relative gaps are reported in Table 2.7). For the successful instances, GCG was able to speed-up the running time the most, it was 5.10 times faster, while the iterations were decreased to 48.39% compared to ICG. Next was ICG($k-1$) with 3.79 times speed-up and 70.82% reduction in the number

[2] $(z_{UB} - z_{LB})/z_{LB} \times 100$, where z_{UB} and z_{LB} are the upper and lower bounds reported by the algorithms, respectively

Table 2.1: *Results for LOC $k = 5$. Mean running time in seconds (average number of iterations in brackets)*

graphs	ICG		ICG($k-1$)		GCG		ECG	
L.20.5.1	0.96	(9.4)	0.75	(8.6)	0.50	(5.6)	**0.49**	(5.0)
L.20.5.2	0.34	(5.8)	0.29	(5.0)	0.27	(3.8)	**0.21**	(3.0)
L.20.5.3	0.34	(5.8)	0.23	(4.4)	0.25	(4.0)	**0.15**	(2.0)
L.20.5.4	0.16	(3.4)	0.19	(4.0)	0.18	(3.0)	**0.12**	(2.0)
L.20.5.5	0.36	(5.6)	0.33	(5.0)	0.29	(4.2)	**0.24**	(3.0)
L.30.5.1	8.24	(17.8)	3.75	(12.8)	3.42	(10.4)	**2.31**	(8.0)
L.30.5.2	5.33	(16.6)	1.89	(10.8)	1.76	(8.6)	**1.15**	(6.0)
L.30.5.3	4.16	(14.4)	1.85	(10.4)	1.78	(8.2)	**1.35**	(6.0)
L.30.5.4	6.94	(18.4)	3.26	(13.2)	2.24	(9.6)	**2.59**	(10.0)
L.30.5.5	2.34	(12.2)	1.39	(10.0)	**1.21**	(7.0)	1.48	(7.0)
L.40.5.1	50.97	(27.6)	22.63	(22.0)	15.39	(16.2)	**13.77**	(15.0)
L.40.5.2	267.48	(49.0)	54.18	(30.0)	43.23	(25.6)	**26.36**	(21.0)
L.40.5.3	531.63	(54.4)	79.39	(32.4)	66.00	(27.4)	**30.49**	(19.0)
L.40.5.4	95.76	(38.2)	28.15	(24.8)	19.96	(19.0)	**12.44**	(15.0)
L.40.5.5	469.13	(58.4)	57.55	(31.2)	50.59	(26.6)	**39.25**	(24.0)
L.50.5.1	836.00	(57.2)	180.02	(35.2)	109.65	(28.2)	**107.15**	(25.0)
L.50.5.2	2 502.17	(81.8)	423.52	(47.0)	300.89	(37.0)	**185.98**	(29.0)
L.50.5.3	4 206.50	(89.8)	511.73	(46.4)	413.37	(38.0)	**396.25**	(37.0)
L.50.5.4	183.40	(35.8)	51.46	(25.2)	31.78	(18.6)	**26.12**	(16.0)
L.50.5.5	6 621.33	(22.4)	1 079.33	(64.4)	776.84	(52.0)	**490.68**	(42.4)
L.60.5.1	🕑	(−)	6 016.81	(84.6)	4 988.19	(96.6)	**3 217.13**	(75.0)
L.60.5.2	🕑	(−)	1 643.06	(69.4)	1 351.41	(60.8)	**1 030.39**	(52.0)
L.60.5.3	485.37	(40.8)	184.59	(30.4)	**128.39**	(24.0)	143.99	(27.0)
L.60.5.4	🕑	(−)	4 394.68	(92.4)	3 075.40	(75.4)	**1 968.20**	(62.0)
L.60.5.5	170.65	(30.4)	66.25	(22.0)	**49.76**	(16.6)	51.67	(19.0)

of iterations. ECG was 3.23 times faster and solved the problems in slightly less than half (49.46%) iterations.

COV, $k = 8$ The results are reported in Table 2.4. In this scenario GCG is the only one which is overall faster than ICG. Namely, GCG was 1.06 faster and reduced the number of iterations to 93.56%. ICG($k-1$) was 0.87 slower whereas ECG was 0.49 slower than ICG. Even the number of iterations increased. Note that this is the only group of problems where the baseline ICG algorithm showed better performance metrics in some of the cases.

The same phenomenon can be seen when examining the relative gap values, see Table 2.7. Compared to ICG, the ICG($k-1$) and ECG resulted in larger average gaps, only GCG ended up with smaller gaps on average.

Table 2.2: *Results for LOC $k = 8$. Mean running time in seconds (average number of iterations in brackets)*

graphs	ICG		ICG($k-1$)		GCG		ECG	
L.20.8.1	0.34	(3.6)	0.31	(4.0)	**0.26**	(3.2)	0.78	(2.0)
L.20.8.2	0.27	(3.2)	0.30	(4.0)	**0.27**	(3.0)	1.75	(4.0)
L.20.8.3	0.20	(2.2)	0.19	(3.0)	**0.15**	(2.0)	0.81	(2.0)
L.20.8.4	**0.24**	(3.0)	0.25	(4.0)	0.25	(3.0)	1.43	(3.0)
L.20.8.5	0.23	(3.0)	**0.20**	(3.0)	0.26	(3.0)	0.97	(3.0)
L.30.8.1	35.93	(25.6)	14.65	(17.0)	**10.04**	(13.8)	17.15	(12.0)
L.30.8.2	2.30	(7.0)	1.65	(6.8)	**1.44**	(5.4)	6.69	(6.0)
L.30.8.3	2.62	(7.8)	2.05	(7.8)	**1.81**	(6.4)	6.65	(7.0)
L.30.8.4	8.56	(12.6)	7.08	(12.4)	**5.18**	(10.2)	14.48	(10.0)
L.30.8.5	1.45	(6.2)	1.45	(6.6)	**1.15**	(5.2)	5.24	(5.0)
L.40.8.1	1 502.32	(48.0)	563.23	(34.2)	**323.94**	(29.8)	568.86	(35.4)
L.40.8.2	3 831.26	(71.8)	1 108.92	(46.4)	598.51	(35.8)	**429.17**	(35.0)
L.40.8.3	76.76	(18.4)	35.70	(15.8)	**19.96**	(13.0)	38.46	(13.0)
L.40.8.4	229.16	(30.0)	78.09	(23.0)	**47.64**	(19.2)	76.09	(20.0)
L.40.8.5	147.42	(28.2)	56.17	(20.8)	**47.54**	(18.4)	55.27	(14.0)
L.50.8.1	🕑	(−)	🕑	(−)	6 357.27	(47.2)	**6 186.57**	(62.0)
L.50.8.2	3 483.92	(50.0)	1 359.93	(37.6)	1 010.56	(33.8)	**946.79**	(31.0)
L.50.8.3	556.33	(24.2)	360.34	(21.6)	218.02	(18.2)	**149.28**	(17.0)
L.50.8.5	🕑	(−)	🕑	(−)	**6 936.66***	(64.5)	🕑	(−)

Table 2.3: *Results for COV $k = 5$. Mean running time in seconds (average number of iterations in brackets)*

graphs	ICG		ICG($k-1$)		GCG		ECG	
C.20.5.1	0.11	(3.0)	0.11	(3.0)	**0.08**	(2.0)	0.10	(2.0)
C.20.5.2	0.10	(3.0)	0.10	(3.0)	**0.07**	(2.0)	0.11	(2.0)
C.20.5.3	0.15	(4.4)	0.09	(3.0)	**0.09**	(2.0)	**0.09**	(2.0)
C.20.5.4	0.14	(4.0)	0.10	(3.0)	**0.08**	(2.0)	**0.08**	(2.0)
C.20.5.5	0.10	(2.4)	0.10	(3.0)	**0.08**	(2.0)	0.10	(2.0)
C.40.5.1	1.49	(8.4)	0.77	(6.0)	**0.68**	(4.4)	1.33	(5.0)
C.40.5.2	2.04	(9.0)	**0.95**	(7.2)	1.02	(5.2)	1.32	(5.0)
C.40.5.3	1.48	(9.0)	0.73	(6.0)	**0.52**	(3.8)	0.93	(4.0)
C.40.5.4	4.07	(13.0)	**1.16**	(7.4)	1.23	(6.0)	1.29	(5.0)
C.40.5.5	3.11	(11.8)	**0.79**	(6.4)	0.81	(4.6)	1.01	(4.0)
C.60.5.1	36.29	(20.6)	9.39	(13.0)	**8.11**	(9.2)	11.02	(9.0)
C.60.5.1	29.78	(18.6)	11.54	(14.2)	**8.23**	(9.4)	8.79	(8.0)
C.60.5.1	22.26	(16.2)	8.37	(11.4)	**4.19**	(6.8)	8.43	(8.0)
C.60.5.1	34.15	(20.6)	8.43	(12.2)	**6.21**	(8.6)	10.73	(9.0)
C.60.5.1	95.97	(29.0)	18.70	(16.8)	**11.08**	(10.6)	24.61	(14.0)
C.80.5.1	5 296.05	(85.6)	339.04	(34.8)	**273.19**	(29.0)	398.06	(29.0)
C.80.5.2	🕑	(−)	666.73	(48.8)	**489.90**	(39.4)	941.31	(44.0)
C.80.5.3	168.03	(26.4)	26.40	(15.4)	**19.08**	(10.4)	43.37	(12.0)
C.80.5.4	🕑	(−)	718.39	(46.2)	**523.22**	(35.4)	780.66	(38.0)
C.80.5.5	534.24	(38.6)	63.66	(19.8)	**37.34**	(12.6)	69.62	(15.0)
C.100.5.1	🕑	(−)	676.98	(41.2)	**377.66**	(28.2)	756.57	(35.0)
C.100.5.2	🕑	(−)	904.86	(44.0)	**452.01**	(28.2)	1 018.30	(35.4)
C.100.5.3	🕑	(−)	498.38	(36.8)	**405.62**	(28.6)	555.21	(29.0)
C.100.5.4	🕑	(−)	1 331.61	(49.0)	**973.07**	(37.0)	2 906.53	(49.0)
C.100.5.5	🕑	(−)	1 446.34	(52.8)	**997.95**	(39.0)	1 781.81	(43.0)

Table 2.4: *Results for COV $k = 8$. Mean running time in seconds (average number of iterations in brackets)*

graphs	ICG		ICG($k-1$)		GCG		ECG	
C.20.8.1	0.14	(2.2)	**0.08**	(2.0)	0.11	(2.0)	0.62	(2.0)
C.20.8.2	**0.05**	(1.0)	**0.05**	(1.0)	**0.05**	(1.0)	**0.05**	(1.0)
C.20.8.3	**0.05**	(1.0)	**0.05**	(1.0)	**0.05**	(1.0)	**0.05**	(1.0)
C.20.8.4	0.10	(2.0)	**0.07**	(2.0)	0.10	(2.0)	0.21	(2.0)
C.20.8.5	0.09	(2.0)	**0.08**	(2.0)	0.13	(2.0)	0.20	(2.0)
C.40.8.1	1.23	(4.6)	1.24	(4.8)	**0.80**	(3.2)	9.20	(4.8)
C.40.8.2	0.42	(3.0)	0.39	(3.2)	**0.35**	(2.4)	5.15	(3.4)
C.40.8.3	**1.24**	(4.4)	8.76	(6.8)	4.23	(4.2)	14.97	(5.6)
C.40.8.4	0.88	(4.0)	1.12	(4.6)	**0.81**	(3.2)	7.41	(4.2)
C.40.8.5	**0.06**	(1.0)	**0.06**	(1.0)	**0.06**	(1.0)	0.07	(1.0)
C.60.8.1	**21.25**	(7.0)	44.72	(8.4)	32.16	(6.6)	136.43	(10.8)
C.60.8.2	0.09	(1.0)	**0.08**	(1.0)	**0.08**	(1.0)	0.09	(1.0)
C.60.8.3	**0.08**	(1.0)	**0.08**	(1.0)	**0.08**	(1.0)	0.09	(1.0)
C.60.8.4	**7.63**	(6.6)	12.43	(7.2)	10.94	(5.4)	52.95	(8.0)
C.60.8.5	11.78	(6.8)	11.16	(7.6)	**7.09**	(5.6)	51.11	(8.4)
C.80.8.1	284.70	(10.4)	621.12	(14.2)	**276.53**	(9.6)	705.20	(13.4)
C.80.8.2	470.94	(14.0)	579.62	(16.2)	**380.11**	(13.8)	721.66	(17.4)
C.80.8.3	**199.77**	(11.4)	362.43	(14.4)	221.57	(11.4)	546.51	(15.2)
C.80.8.4	1 529.46	(18.4)	2 067.69	(22.2)	**1 010.48**	(17.8)	2 382.21	(22.6)
C.80.8.5	**188.68**	(11.4)	361.91	(15.0)	194.43	(11.0)	473.23	(14.6)
C.100.8.1	2 797.19	(19.0)	4 166.53	(22.8)	**2 054.98**	(17.8)	5 078.67	(24.6)
C.100.8.2	5 483.76	(22.4)	🕑	(−)	**4 686.29**	(23.8)	6 596.13⋆	(27.0)
C.100.8.3	🕑	(−)	🕑	(−)	🕑	(−)	🕑	(−)
C.100.8.4	🕑	(−)	🕑	(−)	🕑	(−)	🕑	(−)
C.100.8.5	🕑	(−)	🕑	(−)	**6 113.83**⋆	(−)	🕑	(−)

INF, $k = 5$ The results are shown in Table 2.5. The ICG algorithm could not solve the cases where $N = 60, 80$, and 100 (except one of them). Notably, the most efficient algorithm on these problems were ECG in terms of overall running time and success rate, as it was able to solve all the instances. Compared to the baseline (where ICG was able to finish within the time limit) the average running time of ECG was 296.24 times faster, and the iteration number decreased to 24.09%. The other two variants, ICG($k-1$) and GCG also had nice results: ICG($k-1$) was 14.94 times faster and the number of iterations decreased to 39.23%, while the speed up of GCG was 20.26 with iteration's ratio of 27.30%.

Investigating the gap values, as reported in Table 2.8, we can see that the results of ICG show the biggest gaps, for the larger dimensional instances they even go above 10%. The ICG($k-1$) variant resulted in much smaller relative gaps, and GCG also had tighter final results than ICG.

INF, $k = 8$ These are the problem instances which were the most difficult ones for the tested algorithms. As it can be seen in Table 2.6, the algorithms could solve only the smallest graphs ($N = 20$) and some of the graphs with $N = 40$. ECG is the only one which could solve all the problems where $N \leq 40$. However, examining the relative gap values in Table 2.8 we can see that the results of the ECG were not the smallest ones and even ICG obtained tighter gaps in some instances. Overall, GCG achieved the best gap values.

2.6 Conclusions

We proposed three different algorithm variants for the non-decreasing submodular function maximization problem based on a MIP formulation using constraint generation approach. The work was inspired by a recent paper[106], the algorithm introduced here is named as ICG as it is an improved version of the standard constraint generation method. We presented an algorithm, ICG($k-1$), which uses sets of cardinality $k-1$, where we calculate the case when adding the k-th element to the set with (2.7). The results of ICG($k-1$) are notable because they work in the general case, for non-decreasing submodular functions with no graph representation, and are faster in run-time than ICG. Another idea was to exploit the structural properties of the input graph to select nodes, we called this algorithm as GCG. Finally, we proposed ECG procedure which generates the subsets of sets instead of using an iterative sub-algorithm.

According to our benchmarking results, we cannot declare a clear winner among the algorithms and it is not surprising as the investigated problem is NP-hard. However, for every instances there exists at least one of our algorithms which is computationally more efficient than the ICG algorithm.

Table 2.5: *Results for INF $k = 5$. Mean running time in seconds (average number of iterations in brackets)*

graphs	ICG		ICG($k-1$)		GCG		ECG	
I.20.5.1	0.35	(7.2)	0.14	(4.0)	**0.06**	(2.0)	0.16	(4.0)
I.20.5.2	2.71	(16.0)	0.65	(7.8)	0.45	(5.4)	**0.23**	(5.0)
I.20.5.3	0.83	(9.6)	0.27	(5.0)	**0.14**	(3.0)	0.17	(4.0)
I.20.5.4	1.12	(13.4)	0.21	(5.0)	0.17	(3.2)	**0.10**	(3.0)
I.20.5.5	0.63	(9.8)	0.12	(3.4)	0.10	(2.6)	**0.09**	(3.0)
I.40.5.1	🕑	(−)	211.64	(57.4)	155.88	(45.4)	**5.91**	(17.0)
I.40.5.2	1 230.13	(101.0)	68.32	(35.4)	51.35	(29.2)	**5.91**	(16.0)
I.40.5.3	2 517.61	(114.6)	77.88	(35.4)	62.79	(29.4)	**2.57**	(11.0)
I.40.5.4	2 036.73	(131.2)	62.04	(37.2)	42.11	(27.8)	**3.51**	(13.0)
I.40.5.5	2 420.70	(121.6)	84.12	(38.8)	71.94	(33.0)	**3.84**	(13.0)
I.60.5.1	🕑	(−)	1 872.07	(87.4)	1 447.89	(71.6)	**14.06**	(19.0)
I.60.5.2	🕑	(−)	🕑	(−)	🕑	(−)	**99.67**	(41.0)
I.60.5.3	🕑	(−)	🕑	(−)	7 122.45⋆	(138.0)	**46.91**	(31.0)
I.60.5.4	4 115.66	(112.0)	239.77	(42.2)	158.94	(30.2)	**7.81**	(15.0)
I.60.5.5	🕑	(−)	5 101.47	(132.2)	4 346.57	(117.2)	**38.17**	(30.0)
I.80.5.1	🕑	(−)	🕑	(−)	🕑	(−)	**63.90**	(29.0)
I.80.5.2	🕑	(−)	🕑	(−)	🕑	(−)	**631.07**	(73.0)
I.80.5.3	🕑	(−)	🕑	(−)	🕑	(−)	**137.34**	(42.0)
I.80.5.4	🕑	(−)	🕑	(−)	🕑	(−)	**543.79**	(70.0)
I.80.5.5	🕑	(−)	🕑	(−)	🕑	(−)	**137.87**	(43.0)
I.100.5.1	🕑	(−)	🕑	(−)	🕑	(−)	**693.71**	(66.0)
I.100.5.2	🕑	(−)	🕑	(−)	🕑	(−)	**1 674.73**	(99.0)
I.100.5.3	🕑	(−)	🕑	(−)	🕑	(−)	**655.78**	(65.0)
I.100.5.4	🕑	(−)	🕑	(−)	🕑	(−)	**1 715.59**	(98.0)
I.100.5.5	🕑	(−)	🕑	(−)	🕑	(−)	**636.56**	(64.0)

Table 2.6: *Results for INF $k = 8$. Mean running time in seconds (average number of iterations in brackets)*

graphs	ICG		ICG($k-1$)		GCG		ECG	
I.20.8.1	4.44	(12.6)	1.97	(8.6)	2.10	(8.0)	**1.87**	(8.0)
I.20.8.2	1.58	(8.0)	0.95	(6.4)	**0.64**	(4.6)	2.80	(11.0)
I.20.8.3	1.28	(7.4)	**0.71**	(5.6)	0.97	(6.0)	1.34	(7.0)
I.20.8.4	2.61	(10.6)	1.27	(7.6)	1.01	(5.8)	**0.93**	(6.0)
I.20.8.5	13.37	(21.6)	3.91	(11.8)	**2.67**	(9.0)	3.28	(11.0)
I.40.8.1	🕒	(−)	🕒	(−)	🕒	(−)	**4 111.21**	(60.0)
I.40.8.2	🕒	(−)	🕒	(−)	6 658.51	(78.0)	**2 538.11**	(55.0)
I.40.8.3	🕒	(−)	🕒	(−)	🕒	(−)	**6 242.53**	(78.0)
I.40.8.4	🕒	(−)	🕒	(−)	🕒	(−)	**2 256.41**	(56.0)
I.40.8.5	🕒	(−)	6 871.58	(81.0)	5 548.98	(73.0)	**1 369.95**	(50.0)

Table 2.7: *Relative gaps for LOC and COV problems (the number of unsuccessful runs in brackets)*

graph	ICG	ICG($k-1$)	GCG	ECG
L.60.5.1	0.58 (5)	◊	◊	◊
L.60.5.2	0.12 (5)	◊	◊	◊
L.60.5.4	0.46 (5)	◊	◊	◊
L.50.8.1	0.27 (5)	0.10 (5)	◊	◊
L.50.8.4	0.26 (5)	0.22 (5)	0.11 (5)	0.20 (5)
L.50.8.5	0.22 (5)	0.02 (5)	0.02 (2)	0.09 (5)
L.60.8.1	0.39 (5)	0.33 (5)	0.17 (5)	0.29 (5)
L.60.8.2	0.33 (5)	0.38 (5)	0.16 (5)	0.18 (5)
L.60.8.3	0.26 (5)	0.25 (5)	0.21 (5)	0.34 (5)
L.60.8.4	0.77 (5)	0.93 (5)	0.50 (5)	0.70 (5)
L.60.8.5	0.77 (5)	0.71 (5)	0.62 (5)	0.57 (5)
C.80.5.2	0.76 (5)	◊	◊	◊
C.80.5.4	0.55 (5)	◊	◊	◊
C.100.5.1	0.43 (5)	◊	◊	◊
C.100.5.2	0.42 (5)	◊	◊	◊
C.100.5.3	0.28 (5)	◊	◊	◊
C.100.5.4	1.33 (5)	◊	◊	◊
C.100.5.5	1.69 (5)	◊	◊	◊
C.100.8.2	◊	0.13 (5)	◊	0.15 (3)
C.100.8.3	1.49 (5)	1.79 (5)	1.41 (5)	1.29 (5)
C.100.8.4	1.84 (5)	2.94 (5)	1.96 (5)	2.55 (5)
C.100.8.5	0.54 (5)	0.93 (5)	0.45 (4)	0.95 (5)

Table 2.8: *Relative gaps for INF problems (the number of unsuccessful runs in brackets)*

graph	ICG	ICG($k-1$)	GCG	ECG
I.40.5.1	0.29 (5)	◊	◊	◊
I.60.5.1	2.94 (5)	◊	◊	◊
I.60.5.2	6.39 (5)	1.35 (5)	1.02 (5)	◊
I.60.5.3	5.92 (5)	1.17 (5)	0.82 (4)	◊
I.60.5.5	4.71 (5)	◊	◊	◊
I.80.5.1	5.38 (5)	1.27 (5)	0.71 (5)	◊
I.80.5.2	9.91 (5)	5.39 (5)	4.83 (5)	◊
I.80.5.3	8.26 (5)	3.74 (5)	3.16 (5)	◊
I.80.5.4	10.89 (5)	6.19 (5)	5.46 (5)	◊
I.80.5.5	7.83 (5)	3.26 (5)	2.67 (5)	◊
I.100.5.1	11.48 (5)	1.42 (5)	6.16 (5)	◊
I.100.5.2	11.44 (5)	1.58 (5)	6.72 (5)	◊
I.100.5.3	10.79 (5)	1.60 (5)	7.43 (5)	◊
I.100.5.4	13.31 (5)	1.79 (5)	8.43 (5)	◊
I.100.5.5	10.98 (5)	1.36 (5)	6.00 (5)	◊
I.40.8.1	2.33 (5)	1.07 (5)	0.89 (5)	◊
I.40.8.2	1.10 (5)	0.19 (5)	0.07 (3)	◊
I.40.8.3	1.83 (5)	1.04 (5)	0.80 (5)	◊
I.40.8.4	1.03 (5)	1.02 (5)	◊	◊
I.40.8.5	0.91 (5)	0.62 (3)	◊	◊
I.60.8.1	7.36 (5)	6.79 (5)	6.23 (5)	6.50 (5)
I.60.8.2	7.98 (5)	7.68 (5)	7.31 (5)	8.95 (5)
I.60.8.3	8.57 (5)	8.16 (5)	7.50 (5)	8.43 (5)
I.60.8.4	3.35 (5)	2.96 (5)	2.66 (5)	3.51 (5)
I.60.8.5	6.21 (5)	5.65 (5)	5.36 (5)	6.59 (5)
I.80.8.1	9.12 (5)	8.57 (5)	8.15 (5)	9.23 (5)
I.80.8.2	10.21 (5)	9.91 (5)	9.35 (5)	11.06 (5)
I.80.8.3	9.33 (5)	9.19 (5)	8.72 (5)	10.22 (5)
I.80.8.4	8.58 (5)	8.23 (5)	7.79 (5)	9.56 (5)
I.80.8.5	10.24 (5)	10.18 (5)	9.49 (5)	10.45 (5)
I.100.8.1	14.12 (5)	14.11 (5)	13.40 (5)	15.29 (5)
I.100.8.2	12.59 (5)	13.01 (5)	12.56 (5)	14.49 (5)
I.100.8.3	13.87 (5)	13.78 (5)	13.17 (5)	15.78 (5)
I.100.8.4	13.06 (5)	13.01 (5)	12.51 (5)	14.65 (5)
I.100.8.5	13.50 (5)	13.41 (5)	13.08 (5)	15.24 (5)

Chapter 3

On the initial set of constraints

3.1 Introduction

Nemhauser and Wolsey concluded in their seminal paper [82] that four problems are fundamental to solving an integer linear problem in terms of practicality: formulation of the model, selection of an initial set of constraints, decisions in a branch-and-bound algorithm, and finding good feasible solutions. Among these essential issues, we are concerned with that of the starting point selection, i.e., the selection of initial constraints that would make the algorithms more efficient in terms of runtime.

There are integer or mixed-integer linear programs that require exponentially large number of linear constraints. A well-known example is the traveling salesman problem, which has exponentially many constraints to eliminate subtours [21]. A general group of these problems are mixed integer linear programs to be solved by Benders decomposition. The basic approach is to start with a reduced relaxed problem with a small number of constraints. During the algorithm, additional constraints are generated if the existing ones are violated. When selecting the initial constraints, care should be taken to ensure that they are important or significant in some sense during the solution process. The recommended initial constraints are the solution of the greedy algorithm [80, 82].

3.1.1 From where to start the CG algorithm?

It can be observed that the choice of the starting point plays a role in the efficiency (i.e., its runtime) of the CG algorithm. More precisely, starting from a high function-valued initial point *might not* provide the fastest runtime. This effect is illustrated in Figure 3.1. Exact details of the test problem (called $C.60.5.3$) will be given later in Section 3.5, right now it is enough to understand that the task of the CG algorithm was to select the optimal $k = 5$ nodes from a graph of 60 vertices. The globally optimal solution for this benchmark example was known.

We created 250 test cases, part of which started from a random starting point, while in the other part we chose 3 of the best 5 vertices belonging to the global optimum, fixed them and randomly added 2 other nodes. The reason for this is that we did not get an initial set with a larger function value in the random choices, so we have also biased the sensitivity analysis a bit towards the more interesting scenarios.

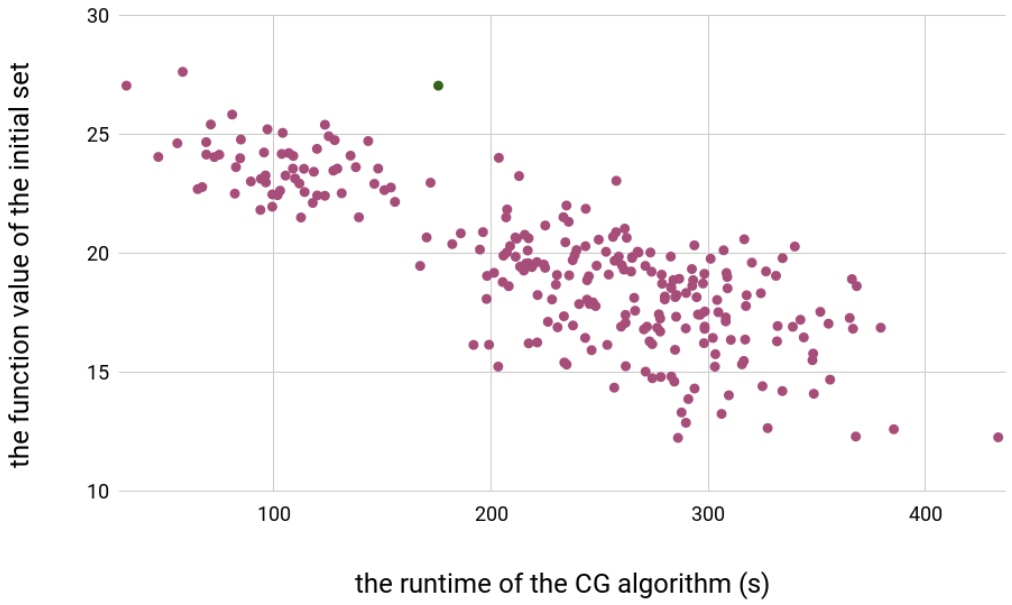


Figure 3.1: *Visualization of the sensibility of the starting point: starting points with similar function values can have rather different running times*

Figure 3.1 shows the scatter plot of the 250 test cases. It is important to note that in this figure, the x-axis shows the runtime of the CG algorithm and the y-axis gives the function value of the initial set. In the figure, two sets of points can be roughly separated, due to the semi-random chosen test cases. The green dot indicates the original CG algorithm starting from the initial point proposed by the greedy algorithm. Note that for the other results, we used all subsets of randomly generated points as starting point since we did not have any order. This setup is appropriate, because for the NS method, which will be presented later, the starting point is all the subset of the selected vertices. This figure perfectly illustrates that the running time of a CG algorithm can be very different even if the function value of the initial points are similar.

The phenomenon introduced informally above is the main motivation for this work.

Roadmap: The relevant literature summary follows in Section 3.2. The algorithms investigated in this chapter (and previously not presented) are reviewed in Section 3.3. A new starting point generation rule for accelerating the algorithms is discussed in Section 3.4. In Section 3.5, we present our numerical experiments, the benchmark instances that we used for

testing the algorithms, including a description of the computational environment, the properties of the test graphs and then the benchmarking results. Finally, Section 3.6 concludes the work.

3.2 Related work

CG-type algorithms are often used to solve mixed-integer linear problems (MILP), hence their efficiency is of high interest. A popular approach is to use machine learning to improve the speed of these algorithms. By achieving a good initial set, the number of iterations is reduced, which reduces the computational cost. In this spirit, a learning strategy was proposed in [86] that employs a modified nearest neighbor method to filter out redundant constraints in the Unit Commitment (UC) problem. In [56], a machine-learning-aided warm-start constraint generation algorithm was introduced which speeds up the search for the optimal solution of a MILP. The method is based on the offline detection of the invariant constraint sets of earlier occurrences of the target MILP. This significantly improves the prediction of the invariant constraint sets for instances that have not yet been seen. Thus, much fewer iterations are required to run the constraint generation algorithm and the online computational burden is significantly reduced. A similar idea for solving MIP problems can be found in [111], where a machine learning technique has been proposed. This is based on extracting efficient data from previous instances in order to improve the solution for similar instances. Good initial feasible solutions, affine subspaces, and redundant constraints were predicted based on statistical data, leading to a significant reduction in problem size.

Focusing on the submodular function maximization problem, most solution algorithms use the result of the greedy approach as a starting point, i.e., the constraints of the initial set, as proposed in [80, 82]. Correspondingly, in [106], the constraint generation procedure and its improved versions used the subsets generated from the greedy result as starting points. The study in [108] investigated fairness and balancing utility in submodular maximization, which was formulated as a bicriteria optimization problem. For this, two instance-dependent approximation algorithms were introduced. In these solving methods, the initial set is also the solution of the greedy algorithm. The well-known benchmarks (maximum coverage, influence maximization and facility location) were used to test the efficiency.

A cutting-plane algorithm for submodular maximization problems was described in [63], which is in fact an iterative binary-integral linear programming model. They used a so-called submodular cut plane, based on the submodularity of a set function via the Lovász extension, which ensures that the algorithm converges to the optimum in finite iterations. In the experiments, they used the solution by a greedy algorithm as initial subset.

3.3 A modern implementation

In this subsection, we would like to present a brief summary of a relevant and efficient method for submodular maximization, as found in [97]. A modern implementation of the MIP model of Nemhauser and Wolsey based on lazy constraint generation was presented. These procedures were also used to test the new initial point selection strategy. An analogy was discovered between the MIP model of submodular function maximization and the Benders decomposition [5]. Specifically, they were able to exploit some of the algorithmic improvements proposed for the Benders decomposition. That is, they took advantage of the support of modern MIP solvers for lazy constraints, so they could provide a stronger initial primal constraint and could also improve the dual constraint faster.

In terms of runtime, these algorithms are much faster than the algorithms presented previously, as can be seen from the numerical results. However, we would like to note that we are not demonstrating the effectiveness of the solution algorithms, but to show how the starting point selection strategy works.

3.3.1 GRASP heuristic

At the end of the greedy phase, a local search was inserted using a neighborhood structure. This was based on the fact that replacement neighbourhoods were examined: subsets of elements that had been dropped and a new element not yet in the set added. The GRASP heuristic can be obtained with the greedy algorithm extended by local search, based on [37]. Then, a randomized component is added to the greedy procedure, i.e., a candidate is chosen randomly, which is then corrected by local search. This is iterated until an iteration/resource limit is reached.

3.3.2 Separating fractional solutions

Separating fractional solutions is not trivial, but separating integer solutions is easy. This is because for fractional solutions, the point to be cut cannot be mapped to a subset of the ground set. This was the motivation for proposing two heuristic solutions for decomposition in [97]. One is based on the greedy algorithm, while the other is based on the Lovász-expansion.

Of the methods summarized above, 3 algorithms have been constructed and tested in [97]: $base$ is the basic constraint generation method, bc has constraints separated as lazy constraints on the fly without using custom branching, while $bc+$ is the improved version of bc and custom branching are included. bc and $bc+$ use GRASP as a warm start, so we used these algorithms to test our new starting point selection strategy.

3.4 A new starting point for constraint generating algorithms

To find a new starting point for a CG-type algorithm we use the input graph's structure. The idea is based on our work presented in the previous chapter, more specifically on the procedure for computing the value p_i, where we use the bipartite graph structure $G = (M, N; E)$. Thus, the algorithm presented below is appropriate for the maximization of non-decreasing submodular functions with graph representation (the benchmarks typically used in the literature and presented earlier were used for testing). But the procedure used there is not directly appropriate and in this method, the nodes are dynamically selected.

To choose k node as a new starting point, first of all, we calculate a new centrality value ns_i to every node $i \in N$. This centrality is adding up the weights of the incoming edges at node $i \in N$ normalized with the degree of the targets node, then multiplying the sum with the degree of the source node:

$$ns_i = d_i \cdot \sum_{j:(j,i)\in E(G)} \frac{w_{ji}}{d_j}, \tag{3.1}$$

where G is the input graph of the optimization problem, $E(G)$ is the set of edges of G, the edges have w_{ji} weights, d_j is the degree of the node $j \in M$ and d_i is the degree of node $i \in N$. Observe that this formula works well for graphs that are not fully connected, because weighting by the degree of the vertices only makes sense then.

Choose node i with highest ns_i value and delete node i with their edges and recalculate all the ns_i for every node $i \in N$. The next vertex is chosen for the starting point based on the recomputed centrality value. Repeat this method until k nodes are selected.

See the graph on Figure 3.2 as a small illustrative example, where the labels of the nodes are marked with black numbers and the sets N and M are marked above the vertices. The vertices signed by their labels and their corresponding ns_i values calculated by (3.1) are shown next to them highlighted by tanning color. Taking the ns_i values into account, we first choose node 7 and then delete this vertex with its edges. Then, the result graph is shown on Figure 3.3 with its recalculated ns_i values. Accordingly, the next selected vertex is 5 and not node 6, but note that in the first step it seemed that node 6 is the better choice.

Although we choose k nodes from N, we also use the vertices from M to calculate the new centrality metric. We divide the weight of the outgoing edges $j \in M$ by the degree of j, which is used to normalize the effect of the edges. We sum this normalized incoming edge weights for each $i \in N$ nodes, which expresses the average effect of selecting $i \in N$ nodes relative to the other nodes. This is weighted by the degree of node i, which helps to better scale the effect of the N vertices.

Let's revisit the example in Figure 3.2. Node 4 can be served only if node 7 is selected, because there is only one edge from node 4 to node 7, so node 7 is important. When computing the centrality metric, we add an edge weight to the value of this node, which is divided

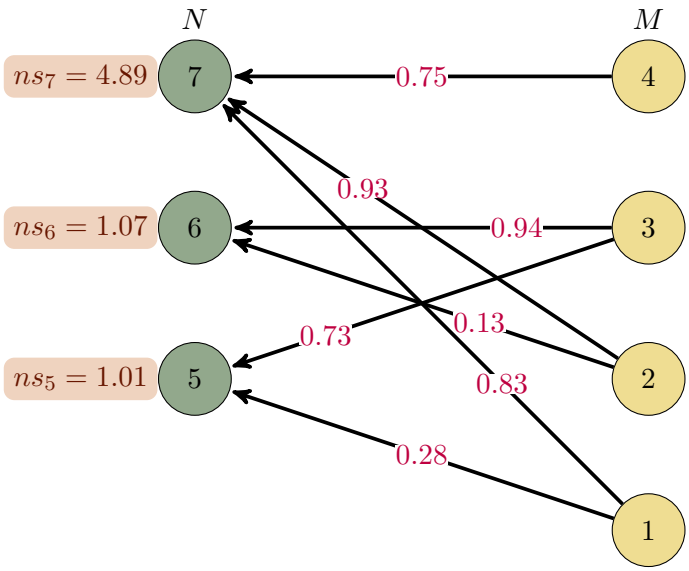


Figure 3.2: *Example graph to calculate the ns_i values; initial step*

by 1 (i.e., it remains itself), thus greatly increasing the value of ns_7. There are two other vertices connected to vertex 7 with two edges, but in both cases the edge with the higher weight is connected to the node 7. This effect is further increased by multiplying the value by the degree of the node.

Next, we delete node 7 with its edges, thus obtaining a new graph showing the case where vertex 7 no longer serves a function. This is the reason why, in the case of the graph in Figure 3.3, we now choose vertex 5. In the current state of the graph, the value of ns_5 is larger, i.e., the vertex is more important, because there is only one edge coming out of vertex 1, and its value is larger than the edge coming into vertex 6 from vertex 2. Notice that we always choose the most important vertex of the current ones. This is regardless of the edge weight going to the previously selected vertex from vertices $j \in M$ (e.g., the vertex 7 has a high weight edge going to it from 3). This is crucial for the strength of the method.

Finally, we generate all subsets from the k vertices proposed by the new centrality metric. We start a CG-type algorithm from the constraints defined by these subsets.

3.5 Numerical experiments

3.5.1 Computational environment

The implementation of the above proposed new centrality was made in R version 4.3.2 using its `igraph` 1.4.2 package. For numerical experiments, CG-type algorithms [17] are implemented in AMPL [39], using the original code with the parameters in [17]. The solver CPLEX 22.1.1.0 was called from AMPL using default options. The modern implementa-

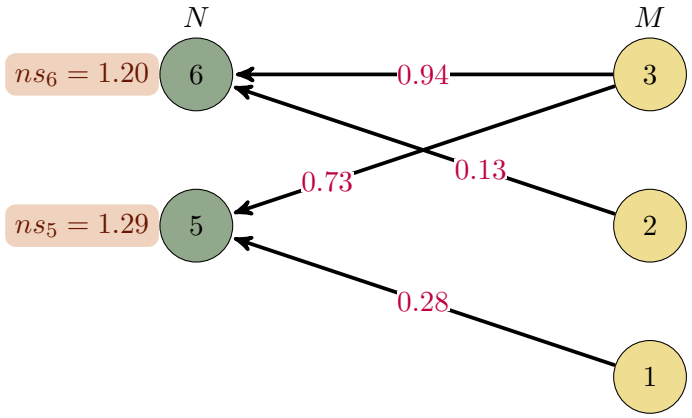


Figure 3.3: *Example graph to calculate the ns_i values; result of the first iteration*

tions, the bc and $bc+$ algorithms, were previously provided by the author of [97]. Thus the original C++ code was used for testing and the settings in [97] were not changed. The MIP solver is IBM ILOG CPLEX 22.1.1 [54]. The computer used had Intel Core CPU i5-6500 at 3.20 GHz with 64 G memory running Ubuntu Linux 22.04.3.

3.5.2 Problem instances

Among the problem instances detailed in Section 2.3, we used weighted coverage (COV) and bipartite influence (INF). Note that the facility location problem has a complete bipartite graph representation and is therefore not suitable for testing this heuristic.

3.5.3 Test graphs

All the algorithms presented in Section 2.4 and Algorithm 1 were re-run during testing to ensure a fair comparison with the procedures started from the new point (i.e., same configurations and software versions). We also tested the algorithms used in [97] and his versions started from a new initialization point, a short summary of this is given below. For both problems we used the benchmarks from [106], since they are available online.

Following the approach in [106] we had:

- $N = 20, 40, 60, 80,$ and 100;
- $M = N + 1$ and $k = 5, 8$.
- For COV instances, a sensor $j \in N$ randomly covers an item $i \in M$ with probability 0.15, and w_i is a random value taken from the interval $[0, 1]$; and
- for INF instances, an edge is randomly generated with probability $p = 0.1$ and p_j is a random value taken from the interval $[0, 1]$.

- We had $\lambda = 10 \cdot k$.
- The cardinality of set Σ in ECG: $\kappa = 12$, see [17] for details.
- Finally, all the random parameters were generated with uniform distribution.

3.5.4 Benchmarking results

Detailed results are available in an electronic supplement [15] containing 24 tables, for the transparency of the publication. The average results for each instance are given in Table 3.1 and Table 3.2.

Five instances were tested for each class, indicated by the last digit of the instances. All algorithms, for each task, were run 5 times using different random seeds for the heuristic choices. The time limit used for the runs was 7,200 seconds (2 hours). If, for any instance, not all of the 5 runs were completed within the time limit, the number of successful runs was indicated in brackets. The instances that did not run within the limit were described by the the mean relative gap[1] with the average number of cases counted in brackets behind it.

The following is a textual assessment of the tables. We denote new start algorithms with NS prefix. Where all of the instances were successfully solved, we have highlighted in the table the one that was faster. In the textual evaluation, we use the ratio $(time/NS\text{-}time)$ and $(NS\text{-}cons.nr./cons.nr.)$ to express the change in time and the number of constraints. For the comparison of iteration numbers, we used $(NS\text{-}iter.nr./iter.nr.) \times 100$ percent.

For the average values in Table 3.1, only graph instances were considered which could ran within the given time limit in all 5 cases using the given algorithm and its NS variant. The table shows that the NS algorithm was faster on average in most cases.

COV, $k = 5$ The results are reported in Tables 3-8 in the supplement [15]. Considering all of these methods together, the algorithms started from the new point were able to solve the problems within the time limit in more cases. For the successful instances, NS-CG was able to speed-up the running time the most, it was 2.46 times faster, while the number of iterations were decreased to 68.22% and the number of constraints was 1.70 times more compared to CG. Next was NS-ECG with 2.00 times faster, 71.06% reduction in the number of iterations and 1.54 times the number of constraining conditions. NS-ICG was 1.54 times faster and reduced the number of iterations to 85.85%, while using 1.05 times more constraints to solve the problem. NS-ICG($k-1$) and NS-GCG achieved similar results: NS-ICG($k-1$) achieved 1.37 times faster with less iterations (90.23%) and 1.05 times more constraints; NS-GCG was 1.32 times faster, reduced the number of iterations to 93.41% and used 1.07 times more constraints.

[1] $(z_{UB} - z_{LB})/z_{LB} \times 100$, where z_{UB} and z_{LB} are the upper and lower bounds reported by the algorithms, respectively

Table 3.1: *Summary of results: starting from $greedy$ versus NS (using CG-type algorithms)*

inst., k	algorithm	time	iter.nr.	cons.nr.	NS-time	NS-iter.nr.	NS-cons.nr.
COV, 5	CG	478.14	95.65	100.65	**115.82**	63.00	94.00
	ICG	302.05	17.80	597.28	**204.51**	15.92	542.02
	ICG($k-1$)	261.33	20.53	715.82	**186.86**	18.80	664.24
	GCG	184.53	14.79	649.67	**138.70**	14.09	628.55
	ECG	955.36	45.38	383.42	**583.16**	36.92	363.92
COV, 8	CG	**290.41**	22.78	30.78	565.18	24.17	279.17
	ICG	**541.56**	7.11	432.98	666.79	7.85	707.11
	ICG($k-1$)	441.73	7.61	478.57	**427.16**	8.85	778.79
	GCG	**394.74**	6.39	442.13	542.99	7.37	767.75
	ECG	591.54	8.65	398.30	**494.53**	8.31	745.98
INF, 5	CG	426.19	197.00	203.00	**243.38**	104.10	135.10
	ICG	904.53	51.76	1 219.89	**15.92**	13.75	300.24
	ICG($k-1$)	560.43	38.77	1 025.61	**68.78**	15.20	388.13
	GCG	591.83	33.28	1 086.53	**72.16**	14.18	460.08
	ECG	0.86	3.68	88.68	**0.85**	3.00	93.60
INF, 8	CG	20.52	81.00	90.00	**2.93**	14.33	269.33
	ICG	524.63	26.04	1 194.26	**6.66**	7.56	441.08
	ICG($k-1$)	202.74	18.12	917.20	**4.10**	5.30	425.83
	GCG	947.46	25.21	1 486.46	**40.66**	9.86	770.40
	ECG	442.63	17.22	1 322.61	**36.65**	6.52	703.26

Table 3.2: *Summary of results: starting from $GRASP$ versus NS (using the modern implementation)*

inst., k	algorithm	time	NS-time
COV, 5	bc	3.04	**2.40**
	$bc+$	2.84	**1.66**
COV, 8	bc	49.09	**48.18**
	$bc+$	34.31	**33.34**
INF, 5	bc	0.35	**0.11**
	$bc+$	0.32	**0.13**
INF, 8	bc	**3.62**	4.31
	$bc+$	**5.96**	9.80

Examining the bc and $bc+$ algorithms, we can see that by replacing the GRASP heuristic with the NS starting point, bc is 3.56 times faster, while $bc+$ is 3.53 times faster on average. In the case of bc, there are 2 graph instances where the GRASP heuristic algorithm runs faster.

COV, $k = 8$ The results are reported in Tables 9-14 in the supplement [15]. The results obtained for this instance are very interesting. Although we started all NS algorithms from the same new point, we did not achieve improvements in many cases. One reason for this is that the number of all the subsets of the $k = 8$ selected vertices is large, so there are already many constraints when starting the reduced MIP problem. Another reason is that algorithms generate new constraints in different ways per each iteration after the initialization. This results in an average 1.01 times faster for NS-ECG, while no speedup was achieved for the other algorithms. For NS-ECG, there were 176.15% more iterations and 15.08 times more constraint conditions compared to ECG. NS-CG was 0.38, NS-ICG was 0.57, NS-ICG($k-1$) was 0.59 whereas NS-GCG was 0.61 slower than the corresponding greedy initiated methods. Even the number of iterations and constraints has increased. Note that this is the only group of problems where the algorithms from the new starting point did not show absolute success. Much the same phenomenon can be observed when examining the relative gap values.

Using the NS starting point instead of the GRASP heuristic, the average speedup is 3.58 for bc and 4.25 for $bc+$. Note that, there are a few cases where $bc, bc+$ runs faster than NS-bc, NS-$bc+$; to be precise, 8 graphs for bc and 8 graphs for $bc+$.

INF, $k = 5$ The results are reported in Tables 15-20 in the supplement [15]. The best results were obtained for NS-ICG, an average 32.11 times faster, 32.66% reduction in the number of iterations and 80.28% reduction in the number of constraints. As with NS-ICG, NS-ICG($k-1$), and NS-GCG ran faster than the original algorithm for all examples and reduced both the number of iterations and the number of constraints. In numerical terms, ICG($k-1$) achieved 8.84 times faster runtime and reduced the number of iterations to 40.71% and the number of constraint conditions to 66.54%; GCG similarly achieved 6.82 times faster runs and reduced the number of iterations to 40.71% and the number of constraint conditions to 54.99%. NS-CG and NS-ECG did not win in terms of runtime for all graphs, but when looking at the average runtime results, they still ran faster. CG achieved a 3.82 times faster, while increasing the number of iterations and constraint conditions (by 1.08 times and 1.82 times, respectively). For ECG, on average 1.18 times faster, but note that the ECG solution time for these examples is under 5 seconds.

The gap values are similar: the gap values of the NS algorithms are smaller, and note that there were several times when the original algorithm could not run within the time limit, while the NS algorithms solved the problem in a short time.

For these cases, bc achieved 3.31 times faster running times from the new starting point, while $bc+$ achieved 2.64 times faster.

INF, $k = 8$ The results are reported in Tables 21-26 in the supplement [15]. For these instances, there were many cases where the algorithms could not solve within the time limit. That is why we can see several instances which only the NS procedure solved. The best average speedup here was also achieved with NS-ICG, exactly 8.33 times, while the number of iterations was less than halved (44.41%), but the number of constraints was almost 4 times more (3.89). We achieved similarly good results with NS-ICG($k-1$), with 15.50 times faster runs. NS-GCG also achieved 9.63 times faster results with half as many iterations (50.44%). ECG ran the most graphs, so we were able to make the most comparisons here and was 6.84 times faster with NS-ECG with no increase in the number of constraints (99.07%). In contrast, CG used 9.04 more constraints but also achieved a run time 8.33 times faster. Comparing the gap values, we can see that in most cases the NS algorithms achieved smaller gap values. In fact, there were 23 cases where the NS ran within the time limit in all 5 cases, while starting from greedy, the algorithm could not.

For the bc and $bc+$ algorithms, the average speedup is 2.09 and 2.04, respectively. For these instances, both for the bc and $bc+$ algorithms, the average runtime is lower when starting from the GRASP heuristic. Accordingly, this is the situation, where most of the graph instances were where the GRASP heuristic algorithm proved to be faster (7 cases for bc and 8 cases for $bc+$).

3.6 Conclusion

A new centrality metric based on the input graph structure was proposed. The graphs under consideration are not fully connected bipartite graphs, for which we have used both the edge weights and the degrees, taking into account the baseline problem. The centrality metric is dynamically recalculated after selecting and deleting a vertex, and the resulting node ordering is used to select the initial set of submodular function maximization problems. The importance of the choice of the starting point was already stated by Nemhauser and Wolsey in [82]. In most cases, the solution proposed by the greedy method or randomly selected feasible solution is used as the starting point for solving algorithms. But there are other proposals in the literature, more precisely, we have presented here the GRASP [97] heuristic.

We used five different algorithm variants for the non-decreasing submodular function maximization problem based on a MIP formulation using constraint generation approach which we started from the greedy's solution and also from the new starting point proposed by the centrality metric. Furthermore, we used two modern implementations of Nemhauser and Wolsey's MIP model for submodular function maximization problem based on lazy constraint generation, which we started from the GRASP heuristic and also from the new starting point proposed by the centrality metric. According to our benchmarking results, algorithms

starting from the new initial set reduced the runtime by a factor of 5.37 for all test cases. Overall, we can conclude the initial set suggested by the new centrality metric is worth using, as shown by our run-time tests and, in their absence, the relative gap tests.

Part II

Influence maximization under deterministic linear threshold model

Basic concepts and definitions

Influence maximization (IM)

Influence maximization is a combinatorial optimization problem and perhaps one of the most actively studied problems in network science. It studies a social network represented as a graph $G = (V, E, W)$, where V is the set of nodes in G, E is the set of directed edges in G and $W : E \to \mathbf{R}_+$ is a non-negative weight function. The goal of the problem is to find a $k \geq 1$ sized set of so-called seed nodes $v_1, \ldots, v_k \in V$ with the maximum influence in graph G in such a way that a weighted directed graph G, a diffusion (or spreading) model and the integer k are given [64].

The following notations will be used: $n = |V|$, for a node $j \in V$ the set of its out-neighbors is denoted by $N_{out}(j)$, and for $j \in V$ the set of in-neighbors is denoted by $N_{in}(j)$. Let $S \subset V$ of size $0 < k \leq n$ be the set of seeds and the function $\sigma(S)$ is the number of influenced nodes started from $S \subset V$ seeds by executing the diffusion model. The formal definition of the optimization problem is therefore

$$\max_{S \subset V, |S|=k} \sigma(S).$$

Diffusion models are usually used with stochastic parameters to solve influence maximization [64]. Thus, $\sigma(S)$ is the expected number of influenced nodes. The nodes with influenced and uninfluenced states will also be called as active and inactive nodes, respectively.

Diffusion models

Several relevant diffusion models can be found in the literature, among which the independent cascade model [44], triggering model [64], time-aware model [73], and the linear threshold model [48] are the most popular ones. Our model is based on the linear threshold model (LTM) which solves the problem by iterating over a $t \in \mathbb{N}$ value, starting with $t = 1$. Let $b_{i,j} \in (0, 1)$ be the edge weight between node i and j, $\theta_i \in (0, 1]$ be the threshold value for each $i \in V$ that determines how easy it is to make the vertex active, and set $\hat{N}_{in}(i)$ be the already influenced in-neighbors of node i. The steps of the LTM is shown in Algorithm 5.

The value of σ in the LTM is estimated by executing R runs using θ_i values uniform at

Algorithm 5 Linear threshold diffusion model

Step 1 Let $t = 1$, $0 < k \leq n$ be fixed and $\mathcal{V}_0$ be a seed set containing k nodes, $\mathcal{V}_1 = \emptyset$.

Step 2 If

$$\sum_{j \in \hat{N}_{in}(i)} b_{j,i} \geq \theta_i$$

holds for any $i \in V$ uninfluenced nodes, then put node i into the set $\mathcal{V}_t$. Mark those nodes as active at the end of this step.

Step 3 If there is no chance to influence more node that is $\mathcal{V}_t = \emptyset$ holds, then STOP. $\mathcal{V} = \mathcal{V}_0 \cup \ldots \cup \mathcal{V}_{t-1}$ and $\sigma(\mathcal{V}_0) = |\mathcal{V}|$ by this time.

Otherwise, let $t := t + 1$ and go back to Step 2.

random, then taking the average value of the influence to obtain the expected value of σ.

The threshold value θ_i determines the influenceability of node i. According to the original paper of [64], it is arguably difficult to measure (e.g., in social networks) the value of this thresholds. Hence, the evaluation of the LT model is done by executing it R times and then the average influence value is taken; this is how expected value of σ is obtained. In this case it can be shown that the function $\sigma(\cdot)$ has *submodularity* property [64], which has the important consequence that a greedy algorithm guarantees that

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

holds for any seed set S, where S^* is the optimal seed set [83].

Deterministic linear threshold model (DLTM)

In our work we used the deterministic linear threshold model (DLTM). It differs in that the θ_i values of the nodes are fixed, and thus no need for running the diffusion model multiple times. Note, that in [48] the original LT model is also deterministic. DLTM has been investigated also in the recent years, see e.g. in [1, 60, 77, 78, 112]. One of the key properties of DLTM is that the submodularity property does not hold [2], so there is no guarantee of the efficiency of the greedy algorithm.

Note that IM under DLTM is a bilevel optimization problem since we need to find the maximum number of active nodes together with the minimum time t. Even using state-of-the-art optimization methods it is not possible to solve this kind of problem. Hence, an iterative solution method needs to be applied using the binary linear program which details are given later.

Chapter 4

An exact method

4.1 Introduction

Social networks have become increasingly popular in recent years and their use is growing. With the help of modern technology, the use of online social networks for various purposes, such as the diffusion of information, ideas, influences, advertisements and many others through networks has become commonplace. There are many instances when an idea or trend starts to spread in the community, such as a mobile app, a political movement, an ideal and even the challenges that are popular nowadays. Their lifetime is not necessarily predictable: some end quickly, while others take a long time. An interesting topic is the study of these phenomena, for example, the extent to which people influence each other, how different friends, neighbours, colleagues influence an individual's opinion on a particular issue. One question is whether information reaches people and, if so, from where. There has been a lot of related research, for example on the spread of innovation around the world or the study of viral marketing. A related research model on the spread of influence has inspired the work in this chapter.

In a recent paper [65] an integer programming formalization of IM using the so-called deterministic linear threshold diffusion model was proposed. In fact, it is a special 0-1 linear program in which the objective is to maximize influence while minimizing the diffusion time. In this chapter, by rigorous analysis, we show that the proposed algorithm can get stuck in locally optimal solution or cannot even start on certain input graphs. The identified problems are resolved by introducing further constraints which then lead to a correct algorithmic solution.

Roadmap: In this chapter, Section 4.2 summarizes the related works. Section 4.3 introduces the original model and a proposed algorithm, followed by Section 4.4, which contains the analysis of this model and the verification of the correctness of the model. Numerical results are given in Section 4.6 and final thoughts are summarized in Section 4.7.

4.2 Related works

Among the vast amount of scientific contributions related to the IM problem, the most relevant publications to our work are the ones using ILP models and/or based on the DLT diffusion model.

For influence *minimization* [113] gives an ILP model, which is another problem. Using the LT model [90] gives such ILP formalism in which the aim is to determine the set of nodes which will never get influenced. This information might be used for solving the original problem. The already mentioned paper of [2] considers the problem as constraint satisfaction and investigates the efficiency of belief propagation algorithm.

In [50] a binary integer program is developed that approximates the IM using independent cascade diffusion model (which is different from the one used in this chapter) by Monte Carlo sampling together with a linear programming relaxation based method with a provable worst case bound.

The stochastic version of the IM problem was investigated by [110]. They developed a two-stage stochastic programming framework using a delayed constraint-generation algorithm. The paper [59] focuses on competitive IM based on probabilistic independent cascade model in which the seed individuals of one entity is already known, while another entity wants to choose its seed set of individuals that triggers an influence cascade of maximum impact. An algorithmic framework based on a Benders decomposition is developed which enables to handle graphs with thousands of nodes and edges. Note that the full ILP model in [65] also considers competition explicitly. Another reformulation based on Benders decomposition of the IM problem using probabilistic independent cascade model was developed in [51].

The paper [81] investigates a robust optimization problem using the IM. It is assumed that the nodes' thresholds and the edge weights can change within a certain domain. The problem what we are studying is a special version of this general one. They construct such an ILP model in which the time parameter t does not play a role (in contrast to our work), moreover, the number of variables grow exponentially. The IM was also used in [10] where an arc-based mixed-integer programming model has been developed for the so-called Least Cost Influence Maximization Problem and thus it is a different problem. A possible extension of the IM problem was defined in [52]. Namely, the Targeted and Budgeted Influence Maximization problem under IM was developed, which allows different nodes to carry different cost and return values. This problem was investigated by using a scalable greedy approach.

In [43] a new approach was presented called MLPR (matrix multiplication, linear programming, randomized rounding) with linear programming used as its core in order to solve the IM problem with LT model. Although the method was shown to be efficient both in running time and in the quality of the result, it does not have an approximation guarantee.

It was shown by [13] that the IM problem using LT diffusion model is equivalent to the so-called targeted immunization problem. A mixed-integer linear programming formalism

was developed together with Benders decomposition approach. A much more general IM problem was introduced by [114] to study the spread of infectious disease process. The considered model takes the cumulative effect of LT into account.

The paper [58] studied three new variants of the competitive influence maximization problem (CIMP) which consider passive (viewing-only) nodes, node resistance, and customer choice behavior. For solving these problems a mixed-integer nonlinear programming model was proposed.

Obtaining realistic parameters, such as nodes' threshold values and edge weights for real-world graphs could be challenging. A mixed-integer linear programming model and an approximate method using an artificial neural network have been proposed to learn the edge weights in the LTM for synthetic and real data by [88]. Regarding the estimation of threshold values [105] comprehensively surveys the different threshold values used in various IM models and develops four threshold estimation models based on edge weight and degree distribution.

Finally, for a detailed overview on the IM problem and its definitions, computational complexities, heuristic solution methods the reader is referred to the survey [72].

4.3 A model and a proposed algorithm

4.3.1 A 0-1 linear programming model

Two integer linear programming formulations of influence maximization based on the IM were recently proposed and studied in [65]. The first one, referred as *basic* model, includes a single party trying to find the initial seed nodes to maximize the spread of influence; while the second one, referred as *competition* model, extends the first one by introducing an enemy trying to spread its own influence. In the current chapter, we investigate the basic model of [65], without considering the cost of selecting a seed node.

The formulation of the basic model is a special 0-1 LP, in which $\mathbf{x} \in \{0,1\}^{n\times\mathcal{T}}$ is the decision variable, $n = |V|$, and the index $\mathcal{T} > 1$ is also part of the optimization problem. Hence, $\mathbf{x}$ is a binary matrix in which choosing the rows in the first column to be equal to 1 represents the selection of the seed nodes. This should be done in such a way that, given certain constraints dictated by IM, the sum of the last column is to be maximized.

Assuming that $\mathcal{T} > 1$ is a given integer constant, let $T = \{2, \ldots, \mathcal{T}\}$ be the set of time periods describing the diffusion process. Let integer $k > 0$ be the number of seed nodes to be selected. The set of in-neighbors of node i is denoted by $N_{in}(i)$.

In the following the binary LP formulation is given, inspired by the basic model of [65],

where the cost of selecting a seed node is equal to 1.

$$S \max \sum_{i=1}^{n} x_{i,\mathcal{T}} \tag{4.1}$$

$$\sum_{i=1}^{n} x_{i,1} \leq k \tag{4.2}$$

$$\sum_{j \in N_{in}(i)} b_{j,i} x_{j,t-1} \geq \theta_i x_{i,t} \quad \forall (i \in V, t \in T) \tag{4.3}$$

$$\sum_{j \in N_{in}(i)} b_{j,i} x_{j,t-1} \leq \theta_i + x_{i,t} \quad \forall (i \in V, t \in T) \tag{4.4}$$

$$x_{i,t-1} \leq x_{i,t} \quad \forall (i \in V, t \in T) \tag{4.5}$$

$$\mathbf{x} \in \{0,1\}^{n \times \mathcal{T}}. \tag{4.6}$$

In the objective function (4.1) the number of influenced nodes are maximized in the last time period. The constraint (4.2) limits the number of seed nodes to be selected initially. The constraint (4.3) guarantees that node i cannot be influenced at time period t if the total weighted in-degree from the already influenced neighbors is below the threshold value of node i. Furthermore, by constraint (4.4), if node i's threshold at time period t is exceeded by the weighted in-degree from the already influenced neighbors, aims to enforce that node i gets influenced. In terms of the correctness of the model, the constraint can be omitted, only constraining the search space by forcing propagation if the conditions are satisfied. This will be discussed in more detail later in the thesis. It is important to emphasize here that it is assumed that the sum of in-weights of nodes cannot exceed 1. The constraint (4.5) ensures that influenced nodes remain to be so in later time periods, whereas constraint (4.6) restricts the solution matrix to be binary.

The objective function in fact has the form

$$\min_{\mathcal{T}} \max \sum_{i=1}^{n} x_{i,\mathcal{T}}$$

and together with constraints (4.2) - (4.6) we have a bilevel optimization problem. It is shown that linear bilevel problems are strongly NP-hard [53].

The AMPL modeling language [39], which we used for implementation and numerical experiments (see Section 4.6), is not suitable for directly describing bilevel optimization models. That would require to have declarations as `var T; var x{n,T};` which is not supported. Hence, we need to consider and treat $\mathcal{T}$ as a constant.

Remark 1. *The globally optimal solution for the bilevel problem is when we have the maximal influence within the shortest diffusion time. This will be referred as $(\sigma^*, \mathcal{T}^*)$ in the following.*

4.3.2 An iterative algorithm

The solution method for the bilevel optimization problem proposed in [65] is shown in Algorithm 6.

Algorithm 6

Step 1 Start the iteration from $\mathcal{T} := 2$.

Step 2 Solve the optimization problem (4.1) - (4.6) with fixed $\mathcal{T}$.

Step 3 If $\mathbf{x}_{i,\mathcal{T}} = \mathbf{x}_{i,\mathcal{T}-1} \quad \forall (i \in V)$, i.e., the last two columns of $\mathbf{x}$ are the same then STOP, the optimum is found. Otherwise, let $\mathcal{T} := \mathcal{T} + 1$ and go back to Step 2.

The subsequent two sections investigate the correctness of Algorithm 6 in which it appears that several corrections are needed. This iterative procedure would allow to find the minimum $\mathcal{T}$ value. Thus the value of $\mathcal{T}^*$ is given by the cycle variable $\mathcal{T}$. The correct algorithm is given towards the end of this chapter.

4.4 Analysis

In this section a thorough analysis of Algorithm 6 proposed in [65] and shown in Section 4.3.2 is given.

For a start, it turns out that the optimization problem (4.1)-(4.6) needs to be modified.

Proposition 4.4.1. *For the correctness of Algorithm 6, the constraint* (4.3) *has to be replaced by*

$$\sum_{j \in N_{in}(i)} b_{j,i} x_{j,t-1} \geq \theta_i (x_{i,t} - x_{i,t-1}) \ \ \forall (i \in V, t \in T). \tag{4.7}$$

Proof. The model (4.1)-(4.6) dictates that the optimal seed nodes are those which have the maximal number of influenceable neighbors. However, by constraint (4.3) these seed nodes need also be influenced, which is only possible if these selected seed nodes form a set of size k in which the node's weighted in-degree is larger than its threshold. This cannot be held in general, thus it might happen that we obtain sub-optimal solution or it even becomes impossible to select seed nodes, and hence the influence spreading cannot be started.

On the other hand, by replacing constraint (4.3) with (4.7) all nodes could be selected as seed node, and thus the optimal solution could be found.

□

As an illustrative example, see the graph on Figure 4.1. The labels of the nodes are indicated as red numbers. By constraint (4.3) the influence spreading cannot be started. The

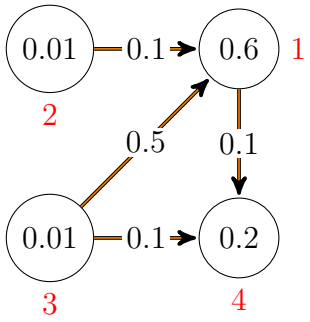


Figure 4.1: *Example graph to show the need of the new constraint* (4.7)

matrix $\mathbf{x}$ corresponding to the correct global optimum for this graph is

$$\mathbf{x} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

As it can be seen, the nodes represented by row 2 and row 3 are selected as seed nodes, which both have threshold value 0.01. These two nodes are not connected to each other, hence the model (4.1)-(4.6) is infeasible at time $\mathcal{T} = 2$.

Remark 2. *Note that constraint* (4.7) *is equivalent to constraint* (4.3) *together with adding loop edges to all the nodes. However, it turns out that from the computational efficiency point of view using* (4.7) *directly is more beneficial.*

In the following we show that Algorithm 6 can get stuck in locally optimal solution even if the newly added constraint (4.7) is taken into account.

Proposition 4.4.2. *For the optimization problem* (4.1), (4.2), (4.4) - (4.7), *there is a graph for which*

$$(\sigma, \mathcal{T}) = (\sigma, \mathcal{T}+1) \quad \textit{and} \quad (\sigma, \mathcal{T}+1) < (\sigma, \mathcal{T}+2).$$

Proof. Such a graph is shown on Figure 4.2, the solution matrix for $\mathcal{T} = 3$ is

$$\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

By choosing nodes $\{1, 4\}$ as seed nodes, for $\mathcal{T} = 2$ we have $\sigma = 4$ and the algorithm cannot increase the number of influenced nodes from $\mathcal{T} = 2$ to $\mathcal{T} = 3$ because the graph structure

does not allow to increase the number of active nodes by changing the seed nodes. Moreover, any other seed nodes result less or equal number of active nodes at $\mathcal{T} = 3$. On the other hand, it can increase the number of active nodes at $\mathcal{T} = 4$ by changing the seed nodes to $\{1, 3\}$ and on this case, all of the nodes will be active. □

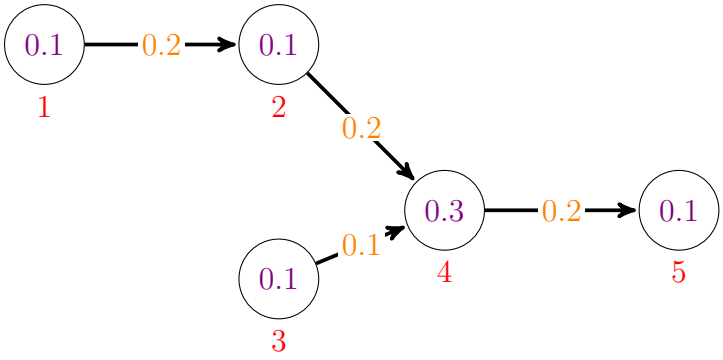


Figure 4.2: *Example graph for Proposition 4.4.2*

We conclude that an extension of the optimization model (4.1), (4.2), (4.4) - (4.7) is needed in order to have a strategy about when to stop the iterative algorithm to be sure that it indeed reached the globally optimal solution. At that end, the following constraint is added:

$$\sum_{i=1}^{n} x_{i,\mathcal{T}-1} + 1 \leq \sum_{i=1}^{n} x_{i,\mathcal{T}}. \tag{4.8}$$

The purpose of constraint (4.8) is to force that for a given $\mathcal{T}$, the last step of the diffusion must have at least one more influenced node than in the previous step. We can thus guarantee no repetition in the last two columns of matrix $\mathbf{x}$.

Remark 3. *Note that adding constraint* (4.8) *to the binary ILP model is in a direct contradiction to Algorithm 6, thus from now on we are developing an alternative version.*

Remark 4. *The constraint (4.8) can easily be extended to any two consecutive columns in matrix* $\mathbf{x}$*. The overall performance of that version is discussed in Section 4.6.*

Remark 5. *On Figure 4.3 the graph is the same as that on Figure 4.2. By choosing* $\{1, 3\}$ *as seed nodes, all of the nodes are active at* $\mathcal{T} = 4$ *with using constraint* (4.8)*. The gray shading of the nodes indicate the time when the node gets activated.*

The following proposition claims that although constraint (4.8) guarantees no repetition in the last two columns of $\mathbf{x}$, we can obtain such result in which column duplication appears inside the solution matrix.

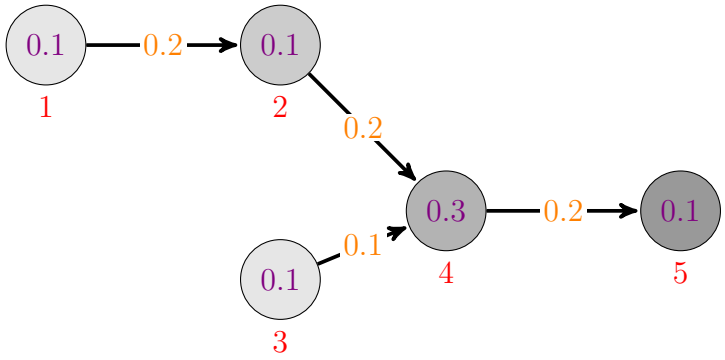


Figure 4.3: *Solution of example graph for Proposition 4.4.2 with* (4.8)

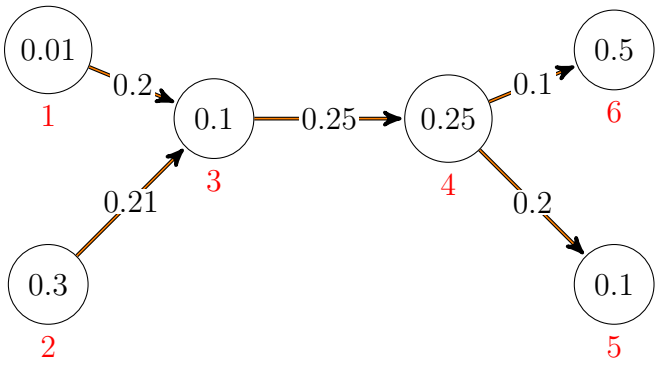


Figure 4.4: *Example graph for Proposition 4.4.3*

Proposition 4.4.3. *For increasing $\mathcal{T}$ values the solutions of* (4.1), (4.2), (4.4) - (4.8) *do not necessarily form a monotonically increasing sequence. Moreover, it can also happen that repetition occurs for consecutive columns in matrix* $\mathbf{x}$.

Proof. As an example we refer to the graph shown in Figure 4.4. The global optimum needs $\mathcal{T} = 4$ diffusion steps. By allowing further iteration steps to be taken by the algorithm, we expect to obtain an infeasible solution. However, the solution matrix for $\mathcal{T} = 5$ is

$$\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

which contains repetition in its second and third column, hence lengthening the spreading up to $\mathcal{T} = 5$. This column repetition could go on forever. Note that this solution matrix is not representing the global optimum simply because the minimum time is $\mathcal{T}^* = 4$. This

phenomenon is caused by constraints (4.4) and (4.7), which is explained in the graph shown on Figure 4.4. According to matrix $\mathbf{x}$ reaching the node with the threshold value 0.25 could be delayed, thus we examine that case. Before reaching that node, constraint (4.7) always gets trivially satisfied, given the fact that its right hand side equals to 0. When the neighbor of the node in question is already activated, then on the left hand side of the constraint (4.4) the weight of the incoming edge appears, while we have either 0 or the threshold of the node on its right hand side. Since both values satisfy constraint (4.4), the algorithm allows to have the activation of the node after getting the global optimum. □

The example above shows that adding (4.8) to the ILP model can cause infinite loop in the iterative approach. It is caused by the possibility of column repetition inside the matrix $\mathbf{x}$, as it is explained in the proof of Proposition 4.4.3. This can be avoided by changing constraint (4.4) into

$$\sum_{j \in N_{in}(i)} b_{j,i} x_{j,t-1} \leq \theta_i + x_{i,t} - \varepsilon \qquad \forall (i \in V, t \in T), \tag{4.9}$$

where $\varepsilon > 0$ is a small constant to make sure that the node is activated when the sum of the edge weight of the already influenced in-neighbors of node is equal to the threshold. The choice for ε is discussed in Section 4.6.

The following proposition claims that adding constraint (4.8) to the ILP model does not prune the globally optimal solution.

Proposition 4.4.4. *The globally optimal solution of* (4.1)*,* (4.2)*,* (4.5) - (4.7)*,* (4.9) *satisfies constraint* (4.8) *as well.*

Proof. The globally optimal solution $(\sigma^*, \mathcal{T}^*)$ of (4.1), (4.2), (4.5) - (4.7), (4.9) cannot contain repetitions in the last $m > 1$ columns in its matrix $\mathbf{x}$ because in that case $(\sigma^*, \mathcal{T}^* - m + 1)$ would be a better solution. Due to constraint (4.9) matrix $\mathbf{x}$ cannot contain more column repetitions. Thus constraint (4.8) is satisfied. □

In addition to the previous proposition, it can also be shown that adding constraint (4.8) to the ILP model does not change the globally optimal solution.

Proposition 4.4.5. *The diffusion value $\mathcal{T}^*$ and influence value σ^* corresponding to the globally optimal solution of* (4.1)*,* (4.2)*,* (4.5) - (4.7)*,* (4.9) *are respectively the same as the values $\mathcal{T}^{**}$ and σ^{**} corresponding to the global optimum of* (4.1)*,* (4.2)*,* (4.5) - (4.9)*.*

Proof. By introducing the constraint (4.8) such an optimization problem is obtained in which the value of $\mathcal{T}$ cannot be increased forever: at a certain point it gets an infeasible solution.

Firstly, let us see if $\sigma^{**} = \sigma^*$ holds. We have to check two cases.

- Assume that $\sigma^* < \sigma^{**}$. By dropping the constraint (4.8), we have a better solution for the problem (4.1), (4.2), (4.5) - (4.7), (4.9), which is not possible, since $(\sigma^*, \mathcal{T}^*)$ is the globally optimal solution.

- Assume that $\sigma^* > \sigma^{**}$. By Proposition 4.4.4 we know that a solution of (4.1), (4.2), (4.5) - (4.7), (4.9) also satisfies constraint (4.8) as well. Thus σ^* would be a better solution for the problem (4.1), (4.2), (4.5) - (4.9), which is not possible as σ^{**} is maximal.

We have contradictions for both cases, thus $\sigma^* = \sigma^{**}$.

Secondly, we check whether $\mathcal{T}^{**} = \mathcal{T}^*$ holds. We have to check again two cases.

- Assume that $\mathcal{T}^* < \mathcal{T}^{**}$. By Proposition 4.4.4 this is not possible as constraint (4.8) would not be satisfied.

- Assume $\mathcal{T}^* > \mathcal{T}^{**}$. We know that σ^* is global optimum for the problem (4.1), (4.2), (4.5) - (4.9) as well. This solution cannot be found with smaller amount of iteration steps under the constraints (4.8).

We have contradictions again, thus $\mathcal{T}^* = \mathcal{T}^{**}$. □

Now we need to find stopping conditions to the iterative procedure. Clearly, one of them is when all nodes are influenced. The other one is when the model becomes infeasible.

Proposition 4.4.6. *If the problem* (4.1)*,* (4.2)*,* (4.5) - (4.9) *becomes infeasible for a given* $\mathcal{T}$ *value, then it remains to be infeasible for the further iteration steps as well.*

Proof. We show that if a solution is feasible then it was so in earlier iteration steps.

- $\mathcal{T} = 2$: The algorithm was able to do the first iteration, thus it could select the seed nodes. Hence, it has a feasible solution at $\mathcal{T}$ = 1.

- $\mathcal{T} = 3$: By constraint (4.8) the last two columns cannot be the same. The first two columns of matrix $\mathbf{x}$ are certainly feasible, the corresponding nodes can be reached within this time frame.

- $\mathcal{T} = m$: In case we remove the m-th column (where $m > 3$), a feasible solution is obtained since the nodes in the $(m-1)$-th column could be reached and activated in $\mathcal{T} - 1$ steps. It is important to see that this solution is not necessarily globally optimal for all $t \in T$.

□

Finally, implicated by Proposition 4.4.5 and 4.4.6 we have the following consequence.

Corollary 4.4.6.1. *The problem* (4.1)*,* (4.2)*,* (4.5) - (4.9) *is feasible in the iteration steps* $2, \ldots, \mathcal{T}^*$*, i.e., before finding the global optimum.*

4.5 A correct model and the solution algorithms

Model Summarizing the above analysis, the correct model is:

$$\max \sum_{i=1}^{n} x_{i,\mathcal{T}} \tag{4.10}$$

$$\sum_{i=1}^{n} x_{i,1} \leq k \tag{4.11}$$

$$\sum_{j \in N(i)} b_{j,i} x_{j,t-1} \geq \theta_i (x_{i,t} - x_{i,t-1}) \ \ \forall (i \in V, t \in T) \tag{4.12}$$

$$\sum_{j \in N(i)} b_{j,i} x_{j,t-1} \leq \theta_i + x_{i,t} - \varepsilon \ \ \forall (i \in V, t \in T) \tag{4.13}$$

$$x_{i,t-1} \leq x_{i,t} \ \ \forall (i \in V, t \in T) \tag{4.14}$$

$$\sum_{i=1}^{n} x_{i,\mathcal{T}-1} + 1 \leq \sum_{i=1}^{n} x_{i,\mathcal{T}} \tag{4.15}$$

$$\mathbf{x} \in \{0,1\}^{n \times \mathcal{T}}. \tag{4.16}$$

The iterative algorithm The correct iterative algorithm to find the globally optimal solution of the influence maximization problem under deterministic linear threshold diffusion model is given in Algorithm 7.

Algorithm 7

Step 1 Start the iteration with $\mathcal{T} := 2$.

Step 2 Solve the problem defined by the set of equations {(4.10) - (4.16)} for the diffusion time value $\mathcal{T}$.

Step 3 If the solution becomes infeasible or all the nodes are influenced then STOP, the global optimum is found. Otherwise, let $\mathcal{T} = \mathcal{T} + 1$ and go back to Step 2.

Greedy Although the so-called submodularity property does not hold for the DLTM, the greedy approach [64] is still a favorable method for solving the IM problem. We have adapted the greedy strategy into our ILP framework. The formal description of the greedy approach is given in Algorithm 8.

Algorithm 8 Greedy algorithm using the ILP model

Step 1 Let $k := 1$ be the number of seed nodes.

Step 2 Let $\mathcal{T} := 2$ and start the iteration with diffusion time.

Step 3 Solve the ILP defined by $\{(4.10)$ - $(4.16)\}$ for the diffusion time value $\mathcal{T}$.

Step 4 If all the nodes are influenced or the model becomes infeasible then the optimum is found and stop the iteration with $\mathcal{T}$ and go to Step 5. Otherwise, let $\mathcal{T} = \mathcal{T} + 1$ and go back to Step 3.

Step 5 If $k = |S|$ where S is set of seeds, then stop the algorithm. Otherwise, fix the selected seed nodes for the remaining iterations. Let $k := k + 1$ and go to Step 2.

Random In this simple method, seed nodes were randomly selected 20 times. Starting the solution method from those random nodes get the number of influenced vertices. As the final result the best solution was selected from those.

4.6 Numerical experiments

4.6.1 Computational environment

The implementation of all the investigated ILP models were done in AMPL [39]. For the numerical experiments the solver Gurobi 9.5 was used with the non-default options: `threads=1 lpmethod=0 cuts=0 mipgapabs=1e-2`, which, compared to the default options, turned out to be much more efficient for these particular models. The computer used had Intel Xeon CPU E5-2660 at 2.00 GHz with 64 G memory running Ubuntu Linux 18.04.5.

4.6.2 Test graphs

For benchmarking the proposed algorithm some random graphs were generated. Two types of random graphs were used: Watts-Strogatz (WS) small-world graphs [109] and so-called LFR graphs with prescribed community structures [71]. For both types $5-5$ graph instances were generated.

WS graphs These graphs were generated by using the package `R/igraph`. The parameters were:

- number of nodes is 60,

- number of neighbors in the starting graphs are $s = 4, 8$, and 12,
- and the rewiring probabilities (i.e., the probability of changing a directed edge $(v_1, v_2) \in E$ into a new edge (v_1, v_3), where $v_1 \neq v_2 \neq v_3$) are $\beta = 0.1$ and 0.3.

The `mutual` parameter was used which makes the graphs directed by doubling the undirected edges. Then 45% of randomly selected edges got removed. The edge weights were assigned as follows.

- First, for each edge a uniform at random number were generated in the interval $[0, 1]$.
- Nodes with larger than 1 in-weights were normalized to 1.
- Moreover, we applied a multiplication with a factor r_w which was a uniform random number in the interval $[0.6, 1]$.

The threshold values of the nodes were generated uniform at random in the interval $[0.15, 0.4]$.

Using this particular procedure we were able to find such WS graphs on which the greedy algorithm found suboptimal solutions.

LFR graphs These graphs were generated by the code from [71], obtaining weighted directed graphs with community structure (thus, resembling social networks). The weights were assigned to the edge using the followings.

- Nodes with in-weights larger than 1 (generated by the LFR method) were normalized to 1.
- Moreover, we applied a multiplication with a factor r_w which was a uniform random number in the interval $[0.6, 1]$.

The threshold values of the nodes were generated uniform at random in the interval $[0.05, 0.4]$. Two configurations were made:

- number of nodes is fixed to $n = 120$,
- average degree $avgk = 6, 7$,
- maximum degree $maxk = 13, 10$,
- mixing parameter $\mu_w = 0.1$,
- minimal community size $minc = 7, 5$,
- maximal community size $maxc = 21, 42$.

4.6.3 Benchmarking results

The testing of Algorithm 6 using (4.7) and Algorithm 7 is shown by not only comparing the execution times of these two versions but also the results obtained by the greedy and random algorithms presented in the 4.5 Section. In all experiments the number of seed nodes were fixed to $k = 2$.

Constraint (4.9) needs to set up the constant $\varepsilon > 0$. Practically, this should be fixed to slightly bigger than the constraint tolerance value for the solver in use. Since in Gurobi 9.5 the default value for both the feasibility of primal constraints and feasibility of dual constraints is `1e-6` we chose $\varepsilon =$`1e-5` in our experiments.

General observations We have done some experiments on different formalism and found the following results.

- As it was remarked after the proof of Proposition 4.4.1 constraint (4.8) can be replaced by (4.3) together with adding for each node i a loop edge with weight equal to θ_i. This version was about 18% longer than using Algorithm 7 as proposed.
- We also investigated the idea of replacing (4.8) and (4.9) by

$$\sum_{i=1}^{n} x_{i,t-1} + 1 \leq \sum_{i=1}^{n} x_{i,t} \quad (\forall t \in T).$$

  This formalism, on average, resulted in about two times less running time.

WS graphs The results obtained for the Watts-Strogatz test graphs are reported in Table 4.1 and 4.2.

The random algorithm were able to find the optimal σ^* value in 6.6% of the cases. However, it always missed the minimal diffusion time $\mathcal{T}^*$. The greedy algorithm found the globally optimal (σ^*, $\mathcal{T}^*$) pairs in 33% of the cases. Note the effect of the fact that the submodularity (see the introduction in Part II) does not hold for the greedy algorithm due to the DLT diffusion model. Algorithm 6 from [65] using constraint (4.7) missed the globally optimal solution in 5 cases (meaning 83.3% success rate).

Regarding the running time, see Table 4.2, obviously the random and the greedy algorithms were really fast. Comparing Algorithm 6 and 7 it can be seen that the corrected version resulted in usually much longer running time. Our Algorithm 7 can be up to 31 times slower. Closer inspection into the results reveal that, for example, for the case $s = 4, \beta = 0.1, i = 3$ our proposed algorithm needed $1,271$ seconds to prove that there is no better solution than $(59, 15)$. For $t > 15$ values the σ value got decreasing. Note that there are cases where the optimal seed set can make the entire graph influenced, i.e., where $\sigma^* = 60$, yet, our proposed algorithm is much slower. For example, in the case $s = 12, \beta = 0.3, i = 5$ it turns out

Table 4.1: *Benchmarking results for the small-world Watts-Strogatz graphs; optimum values*

			random		greedy		Alg. 6+ (4.7)		Algorithm 7	
s	β	$i.$	σ	$\mathcal{T}$	σ	$\mathcal{T}$	σ	$\mathcal{T}$	σ	$\mathcal{T}$
4	0.1	1	58	15	60	14	58	8	60	14
		2	60	14	60	9	60	9	60	9
		3	2	1	59	15	59	15	59	15
		4	59	15	60	11	59	9	60	11
		5	4	2	60	10	58	8	60	10
4	0.3	1	60	13	60	7	60	7	60	7
		2	3	2	59	12	58	9	59	12
		3	3	2	58	8	58	8	58	8
		4	6	3	58	13	58	9	58	9
		5	58	11	58	9	58	9	59	11
8	0.1	1	2	1	58	10	60	10	60	10
		2	2	1	11	4	60	11	60	11
		3	3	2	60	10	60	10	60	10
		4	3	2	60	9	60	9	60	9
		5	3	2	33	6	60	10	60	10
8	0.3	1	5	4	48	7	60	8	60	8
		2	2	1	60	7	60	7	60	7
		3	2	1	12	5	60	11	60	11
		4	2	1	7	3	60	12	60	12
		5	2	1	6	2	60	9	60	9
12	0.1	1	2	1	4	2	60	9	60	9
		2	2	1	3	2	60	10	60	10
		3	3	2	4	2	60	11	60	11
		4	2	1	2	1	60	12	60	12
		5	2	1	5	2	7	3	7	3
12	0.3	1	3	2	4	2	7	4	7	4
		2	2	1	4	2	8	5	8	5
		3	2	1	4	2	60	10	60	10
		4	2	1	8	3	60	9	60	9
		5	2	1	8	4	60	10	60	10

that our algorithm was struggling in the very last iteration – this is certainly caused by the constraint (4.8). On the other hand, there are five problem instances where our Algorithm 7 was faster.

LFR graphs The results obtained for the LFR graphs are shown in Tables 4.3 and 4.4.

The random and greedy algorithms were able to find the optimal σ^* value in 20% and 80% of the cases, respectively. However, the random selection of seed nodes always missed the corresponding minimal diffusion time $\mathcal{T}^*$. The greedy algorithm found the globally

Table 4.2: *Benchmarking results for the small-world Watts-Strogatz graphs; running times in seconds*

s	β	$i.$	random	greedy	Alg. 6+ (4.7)	Algorithm 7
4	0.1	1	0.28	4.55	8.8	16.5
		2	0.23	2.66	25.9	32.9
		3	0.01	7.55	105.4	1 271.6
		4	0.22	5.18	15.2	15.8
		5	0.01	3.17	22.9	21.1
4	0.3	1	0.19	2.22	8.3	9.3
		2	0.01	5.24	17.3	407.5
		3	0.02	4.75	6.9	220.6
		4	0.02	6.25	34.6	635.0
		5	0.18	5.70	12.2	70.3
8	0.1	1	0.01	4.06	212.3	704.4
		2	0.01	0.52	1 311.8	1 849.9
		3	0.02	3.83	2 354.0	1 391.3
		4	0.02	3.11	130.2	149.8
		5	0.02	1.25	480.3	1 651.4
8	0.3	1	0.04	1.91	112.8	87.6
		2	0.01	1.96	38.7	30.0
		3	0.01	0.78	1 233.4	5 569.3
		4	0.01	0.30	6 878.8	13 981.4
		5	0.01	0.12	196.6	212.9
12	0.1	1	0.01	0.12	650.1	1 026.1
		2	0.01	0.12	2 066.8	9 343.7
		3	0.02	0.13	5 346.7	22 485.2
		4	0.01	0.03	23 832.7	103 358.9
		5	0.01	0.13	13.2	32.2
12	0.3	1	0.02	0.13	39.3	43.4
		2	0.01	0.13	85.8	45.3
		3	0.01	0.13	3 549.9	4 640.9
		4	0.01	0.30	712.4	871.3
		5	0.01	0.52	974.5	8 511.0

optimal (σ^*, $\mathcal{T}^*$) pairs in 70% of the cases. Note that greedy reported larger diffusion time than the optimal one in two cases. We can see that Algorithm 6 using (4.7) missed the globally optimal solution for 3 graphs.

Regarding the running times, see Table 4.4, the random and greedy algorithms were again really fast. Comparing Algorithm 6 and 7 we can see that our proposed version can be up to 10 times slower. This is due to the same fact as mentioned for the WS graphs: it takes considerable time to prove the optimality of the found solution.

Table 4.3: *Benchmarking results for the LFR graphs; optimal values*

						random		greedy		Alg. 6+ (4.7)		Algorithm 7	
$avgk$	$maxk$	μ_w	$minc$	$maxc$	$i.$	σ	$\mathcal{T}$	σ	$\mathcal{T}$	σ	$\mathcal{T}$	σ	$\mathcal{T}$
6	10	0.1	5	42	1	110	13	120	13	110	9	120	13
					2	103	12	103	10	103	10	103	10
					3	120	12	120	9	120	8	120	8
					4	19	6	91	12	90	10	91	12
					5	49	15	101	11	101	11	101	11
7	13	0.1	7	21	1	75	10	94	11	96	10	96	10
					2	62	13	120	13	120	13	120	13
					3	23	6	120	15	120	15	120	15
					4	2	1	83	8	83	8	83	8
					5	51	9	111	10	111	10	120	18

Table 4.4: *Benchmarking results for the LFR graphs; runnig times in seconds*

$avgk$	$maxk$	μ_w	$minc$	$maxc$	$i.$	random	greedy	Alg. 6+ (4.7)	Algorithm 7
6	10	0.1	5	42	1	0.62	19.77	330.4	413.6
					2	0.49	20.05	785.6	5 099.7
					3	0.42	11.39	190.7	193.9
					4	0.19	11.37	135.8	398.8
					5	0.68	30.39	1 886.5	18 279.5
7	13	0.1	7	21	1	0.36	10.19	200.6	1 214.2
					2	0.59	23.74	320.0	388.9
					3	0.21	23.79	451.4	579.6
					4	0.02	15.16	108.2	979.8
					5	0.33	29.67	219.5	420.4

4.7 Conclusions

We proposed an exact 0-1 linear programming model for the influence maximization problem based on deterministic linear threshold model. By rigorous analysis the correctness was shown. The work was inspired by a recent paper [65]. In fact, our proposed model is an improved version in a sense that the model in [65] does not always find the global optimum. We demonstrated this fact in our analysis and by numerical testings.

According to our benchmarking results, even for relatively small graphs, finding the exact solution can only be done in very pessimistic running times. In one hand this is not surprising as the problem is strongly NP-hard. On the other hand, our exact model is the first one to computationally demonstrate how difficult is to find the global solution.

Chapter 5

A heuristic for seeds selection

5.1 Introduction

This chapter continues with the task of maximizing influence in a slightly different aspect. Chapter 4 shows a correct algorithm for solving this problem using a deterministic linear threshold model. This served as motivation, since although it gives the global optimum of the problem, we only obtain results for small graphs within a reasonable run-time. Since the solution describes how the diffusion happens for different time constraints, it is of interest to investigate how the various characteristics of the underlying graph relate to the result.

The aim is to find a correlation between the input graph and the solution method. More precisely, whether the seed vertices selected at initial time have some distinguishable property; or whether we can exclude vertices from the initial set based on a feature. A good approach to this can be to use graph centrality, which establishes an ordering between nodes based on the values assigned to the vertices.

Combining these, this chapter introduces two new centrality metrics that can be computed from the structure of the input graph. These can be used to minimize the number of possible seed nodes. The solver then chooses from the reduced set of possible seed nodes provided by the centrality metrics. By solving the problem in this way, the solution time is reduced.

Roadmap: The structure of the chapter is the following. A short review of the literature is presented in Section 5.2. The demonstration of the new centrality metrics and the seed selection method are presented in Section 5.3, while the algorithm used with the heuristic is in the Section 5.4. Then, we apply our methods on different benchmark sets: LFR-type and real-world graphs. The graph instances used for testing, the detailed and the discussion of the results are given in Section 5.5. The final thoughts are summarized in Section 5.6.

5.2 Related Works

In this section, we would like to present the related works that are relevant, i.e., those based on the deterministic linear threshold model using heuristics.

A fast algorithm for finding the most influential people was proposed in [89]. First, they were looking for communities of the graph and examining their limited number to reduce implementation time. The nodes to be excluded were selected using centrality measurements, community detection and new set computation. They then bounded the search space of the input network using the new set. The main advantage of it is that it reduces the number of nodes to be tested without compromising quality in order to reduce execution time.

The threshold-based greedy approach was presented in [67] for solving IM under DLTM. The greedy approach allows to obtain very good results compared to other combinatorial algorithms. It can also be used with the genetic algorithm and significantly improve its efficiency.

A new hybrid centrality metric based on closeness (harmonic) and decay measures was proposed in [102]. Two main application areas were presented. One hybridization was used to solve the coverage problem, while the other tries to find the most ideal node to reach the most people in the population, using the deterministic threshold model. In both cases, the centrality metric is based on a formulation of the weighted centrality measure of nodes.

The complexity of the IM under DLTM is studied in [78]. It is shown that for DLTM, active nodes are not approximable in $n^{1-\varepsilon}$ -factor polynomial time unless P=NP. In contrast, they are well approximable in the linear threshold model and the independent cascade model. It has also been demonstrated that for a given set of seeds, the number of influenced nodes can be determined in polynomial time.

The fact that the DLTM has no polynomial time $n^{1-\varepsilon}$ approximation unless P=NP, even when a person needs at most two active neighbors to become active was shown in [77]. However, there exists an $\varepsilon/(\varepsilon-1)$ polynomial-time approximation in the case where one of the neighbours has already become active and the person can be activated. It is shown to be the best approximation under plausible Complexity-Theoretic assumptions.

In [52], the authors have created Targeted and Budgeted Influence Maximization for DLTM. They advanced a scalable algorithm that allows some optional methods to solve the problem, it is the TArgeted and BUdgeted Potential Greedy (TABU-PG) algorithm. It is an iterative and heuristic algorithm that relies on investing in potential future gains when choosing seed nodes.

In [104], the selection of top-k nodes is investigated based on the measure corresponding to the social network under consideration. It relies on the Shapley measure to efficiently compute an approximate solution to the problem. Although explicitly not using DLTM, the algorithm is general in the sense that it does not exploit the submodular property of the function.

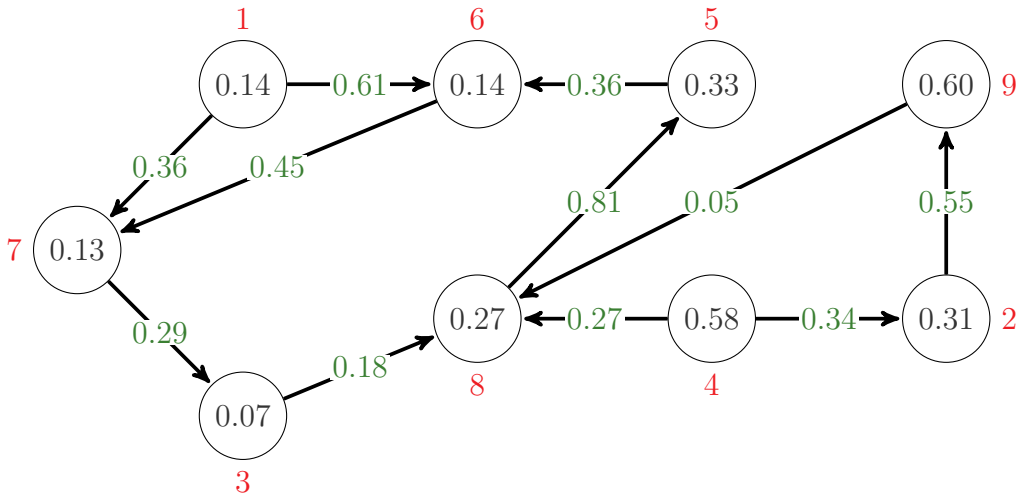


Figure 5.1: *Example graph*

5.3 New centralities

Two new centrality measures are proposed. Both of them are specifically developed for solving IM problem under DLTM. What makes them distinguished from other centralities is that they take into account not only the direction and the weight of the graph edges, but also the weight, i.e. threshold, of the nodes. To make it easier to understand our metrics and the calculation, we have made a small graph, which is shown in Figure 5.1.

5.3.1 Influenceability

Our first centrality is the influenceability, denoted by $\mathcal{I}_{in}$, which measures how easy is to activate a node. To calculate this, we examine the incoming edges from the neighbours, namely which edges and combinations of edges are able to reach or exceed the threshold value of the node. We define the weighted incidence, denoted by w_{to} as follows. Let $\mathcal{P}(i)$ be the set of all possible combinations of edges coming into the vertex $i \in V(G)$, except the empty set. Take the number of edge combinations that are able to activate the node by dividing by the number of edges in the edge combination and all occurrences of a given number of edge combinations:

$$w_{to}(i) = \sum_{K \in \mathcal{P}(i)} \left(\frac{\left[\sum_{b_{ji} \in K} b_{ji} > \theta_i \right]}{|K| \cdot \binom{|N_{in}(i)|}{|K|}} \right) \quad \forall i \in V(G), \tag{5.1}$$

where if $\left[\sum_{b_{ji} \in K} b_{ji} > \theta_i \right]$ is true, then its value is 1, otherwise it is 0. Finally, sum the w_{to} values and obtain $\mathcal{I}_{in}^{(p)} = \sum w_{to}(i)$, where $i \in V(G)$. The label $^{(p)}$ indicates that this is a "preliminary" value, which is then adjusted by the values of its neighbors.

Table 5.1 shows the calculation of $\mathcal{I}_{in}^{(p)}$ (including 5.1) for vertex 8 of the graph in Figure 5.1. The in-neighbors of vertex 8 are nodes $9, 4$, and 3. Note that vertex 4 can influence vertex 8 by itself, so combinations where vertex 4 is included will certainly be able to influence vertex 8.

Table 5.1: *The $\mathcal{I}_{in}^{(p)}$ value of node 8 of the graph in Figure 5.1*

combinations of edges $\mathcal{P}(8)$	sum	θ	$w_{to}(8)$
$9 \to 8$	0.05	0.27	0
$4 \to 8$	0.27	0.27	$1/(1 \cdot 3)$
$3 \to 8$	0.18	0.27	0
$9 \to 8, 4 \to 8$	0.31	0.27	$1/(2 \cdot 3)$
$9 \to 8, 3 \to 8$	0.23	0.27	0
$4 \to 8, 3 \to 8$	0.44	0.27	$1/(2 \cdot 3)$
$9 \to 8, 4 \to 8, 3 \to 8$	0.49	0.27	$1/(3 \cdot 1)$
			$\mathcal{I}_{in}^{(p)}(8) = 1$

Table 5.2 shows the calculated $\mathcal{I}_{in}^{(p)}$ values for the vertices of the small graph in Figure 5.1.

Table 5.2: *The $\mathcal{I}_{in}^{(p)}$ values for the graph in Figure 5.1*

node:	1	2	3	4	5	6	7	8	9
$\mathcal{I}_{in}^{(p)}$:	0	1	1	0	1	1.5	1.5	1	0

The final centrality metrics are obtained by combining with the measure of node and its neighbors. The influenceability value of a node is obtained by adding to the value of $\mathcal{I}_{in}^{(p)}$ the approximation of the influenceability of its in-neighbours:

$$\mathcal{I}_{in}(i) = \mathcal{I}_{in}^{(p)}(i) + \sum_{j \in N_{in}(i)} \frac{\mathcal{I}_{in}^{(p)}(j)}{|N_{out}(j)| - 1} \quad \forall (i \in V). \tag{5.2}$$

Note that if $|N_{out}(j)| \leq 1$, then let $|N_{out}(j)| = 2$ for the divisor to be 1.

5.3.2 Ability-to-influence

The second centrality is the ability-to-influence, denoted by $\mathcal{I}_{out}$. This indicates the influencing role of the node on its neighbors. Specifically, we look at all the combinations of incoming edges to the neighbourhood which include the edge from the investigated node. Of these, we count the ones whose sum of weights reaches the threshold value of the node and calculate the weighted incidence value for this case, denote w_{from}. As calculated for

the influenceability, we divide the number of infecting edges by the number of edges in the edge combination and all occurrences of a given number of edge combinations. Finally, summarized w_{from} for each investigated combinations of edges.

The calculation of $\mathcal{I}_{out}^{(p)}$ value for node 4 is shown in Table 5.3. To do this, we need to look at the neighbours of node 4, i.e., the edges coming into the vertices 8 and 2. We have seen the combinations of edges coming into vertex 8 in Table 5.1, but only those that involve the edge coming from vertex 4 are needed.

To calculate this efficiently, we create a table with rows and columns representing the vertices of the graph. Row i and column j show the role of vertex j in the contamination of vertex i. The column sum of the table gives the $\mathcal{I}_{out}^{(p)}$ value of the vertices. We see this counting table in Table 5.4 which also shows the calculated $\mathcal{I}_{out}^{(p)}$ values for the vertices of the small graph in Figure 5.1.

Table 5.3: *The $\mathcal{I}_{out}^{(p)}$ value for node 4 of the graph in Figure 5.1.*

combinations of edges	sum	θ	$w_{from}(4)$
$4 \to 8$	0.27	0.27	$1/(1 \cdot 1)$
$9 \to 8, 4 \to 8$	0.31	0.27	$1/(2 \cdot 2)$
$4 \to 8, 3 \to 8$	0.44	0.27	$1/(2 \cdot 2)$
$9 \to 8, 4 \to 8, 3 \to 8$	0.49	0.27	$1/(3 \cdot 1)$
$4 \to 2$	0.34	0.31	$1/(1 \cdot 1)$
			$\mathcal{I}_{out}^{(p)}(4) = 2.83$

Table 5.4: *The $\mathcal{I}_{out}^{(p)}$ values for the graph in Figure 5.1.*

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	0	0	0	1.00	0	0	0	0	0
3	0	0	0	0	0	0	1.00	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1.00	0
6	1.50	0	0	0	1.50	0	0	0	0
7	1.50	0	0	0	0	1.50	0	0	0
8	0	0	0.58	1.83	0	0	0	0	0.58
9	0	0	0	0	0	0	0	0	0
$\mathcal{I}_{out}^{(p)}$	3.00	0.00	0.58	2.83	1.50	1.50	1.00	1.00	0.58

The ability-to-influence value of a vertex is obtained by adding to the value of $\mathcal{I}_{out}^{(p)}$ the approximation of the ability-to-influence of its out-neighbors:

$$\mathcal{I}_{out}(i) = \mathcal{I}_{out}^{(p)}(i) + \sum_{j \in N_{out}(i)} \frac{\mathcal{I}_{out}^{(p)}(j)}{|N_{in}(j)| - 1} \ \forall (i \in V). \tag{5.3}$$

Note that if $|N_{in}(j)| \leq 1$, then let $|N_{out}(j)| = 2$ for the divisor to be 1.

5.3.3 Potential seed selection

Using the two centrality values, we want to determine which vertices can be seeds. Therefore, first, the centrality values are normalized between 0 and 1 in a way that all the elements are divided with the maximum. Such normalization is denoted in each case by $||.||$. Then, we sort the nodes according to their centrality value. We put them in descending order according to their ability-to-influence value, since seed vertices should have good ability-to-influence's value. Conversely, we rank the vertices in ascending order according to their influenceability value, since seed vertices are unlikely to be easily infected. We take the weighted sum of the two order values for each node to get $\mathcal{I}$. This is shown in equation (5.4):

$$\mathcal{I}(i) = \alpha \cdot \text{ord}(||\mathcal{I}_{out}(i)||) + (1 - \alpha) \cdot \text{ord}(||\mathcal{I}_{in}(i)||) \ \forall (i \in V). \tag{5.4}$$

The normalized values of influenceability and ability-to-influence for the graph in Figure 5.1 are shown in Table 5.5. Also, the $\mathcal{I}$ values are calculated from them.

Finally, to form the set of potential seed nodes, choose the subset of $V(G)$ according to $\mathcal{I}$. This is controlled by a parameter $0 < r < 1$, thus the cardinality of the candidate seeds set is $r \cdot |V(G)|$.

Table 5.5: *The value of* $||\mathcal{I}_{in}||$, $||\mathcal{I}_{out}||$, *and* $\mathcal{I}(i)$ *with* $\alpha = 0.0$ *and* $\alpha = 0.8$ *for the graph in Figure 5.1*

node	$\|\|\mathcal{I}_{in}\|\|$	$\|\|\mathcal{I}_{out}\|\|$	$\mathcal{I}(i), \alpha = 0.0$	$\mathcal{I}(i), \alpha = 0.8$
1	0.00	1.00	1	1.0
2	0.33	0.11	3	7.8
3	0.83	0.20	7	7.0
4	0.00	0.61	2	2.0
5	0.67	0.55	5	3.4
6	0.83	0.45	8	4.8
7	1.00	0.29	9	6.6
8	0.67	0.45	6	5.2
9	0.33	0.20	4	7.2

5.4 Algorithms

Before details are given about our new heuristic method it needs to be emphasized that due to the nature of our problem, namely finding the maximum number of influenced nodes with minimum diffusion time steps, all algorithms work iteratively and use the ILP model $(4.10) - (4.16)$ and their stopping criteria is to run until infeasibility. This usually leads to a long running time.

5.4.1 Proposed heuristic: IAtI

Here we describe our proposal for a heuristic which selects a candidate seeder set of graph nodes based on the new centrality metrics introduced in Section 5.3. Since it is using the Influenceability and the Ability-to-Influence measures we refer to it as IAtI-heuristic.

The method is described in Algorithm 9. In its precondition phase, in Step 1 and 2 it calculates the two centrality metrics and the combination of them using the formulae given in Section 5.3. In Step 3 the algorithm collects the set of possible seed nodes. It is a parameter $0 < r < 1$ which controls the ratio of the nodes to be selected. Higher r value leads to higher probability for the seed nodes corresponding to the global optimum to be selected into the candidate set. However, high r value also leads to higher execution time, thus it needs to be set up with care. Steps $4 - 6$ describe the iterative part of the algorithm. This is essentially the same as the algorithm we proposed in Chapter 4. Note that the algorithm usually iterates until the ILP model becomes infeasible, as there is no other stopping criteria, unless all the nodes become active.

Algorithm 9 IAtI-heuristic(r, α)

Input A directed graph G with edge weights and node threshold values.

Step 1 Calculate $\mathcal{I}_{in}^{(p)}$ and $\mathcal{I}_{out}^{(p)}$ for all $i \in V$ and then $\mathcal{I}_{in}$ and $\mathcal{I}_{out}$ using Eq. (5.2) and (5.3), respectively.

Step 2 Form $\mathcal{I}$ for each vertex according to the Equation (5.4) using the input parameter α.

Step 3 Define $S \subseteq V(G)$ to be the set of possible seeds: choose the top $r \cdot |V(G)|$ number of nodes from $\mathcal{I}$.

Step 4 Let $\mathcal{T} := 2$ and start the iteration.

Step 5 Solve the ILP defined by $\{(4.10)$ - $(4.16)\}$ for the diffusion time value $\mathcal{T}$, so that the seed vertices can be chosen exclusively from the set S.

Step 6 If all the nodes are influenced or the solution becomes infeasible then stop the iteration. Otherwise, let $\mathcal{T} = \mathcal{T} + 1$ and go back to Step 5.

Discussion on parameters' choice for IAtI Now that we have Algorithm 9, let us see how to set up its parameters in order to have high chance to include the seed nodes of the global optimum and exclude those which might not be good candidates. As we have calculated the centrality metrics for the small test graph from Figure 5.1 as well as the globally optimal solution using $k = 2$ seed nodes, the scatterplot shown in Figure 5.2 clearly suggests that we shall aim for larger ability-to-influence value together with low influenceability. The explanation is that the seed nodes of the global optimum are colored red (node 4 and 1). Larger circle represents later activation in the diffusion in the globally optimal solution[1]. Obviously, this landscape of the centrality values is quite simple as our small test graph is of special structure. Discussion on some larger graphs will be given in Section 5.5.3.

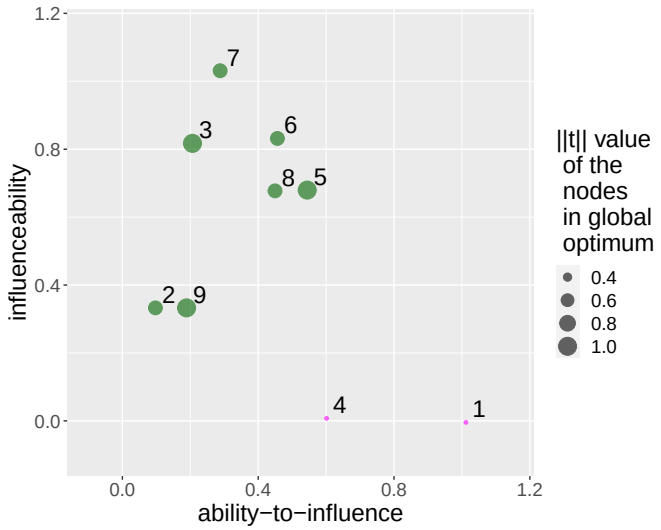


Figure 5.2: *Visualization of the new centralitites for the small test graph*

Two different α values were used from which potential seeds were selected in equal proportions. In the first case, the weight of the ability-to-influence property is set to a higher value, so $\alpha = 0.8$ and select elements with a ratio $r = 0.2$ which are the best elements according to $\mathcal{I}$. Moreover, we add the vertices with ability-to-influence value equal to 1. This is necessary because it has the highest chance of infection, yet in many cases it will not be the seed. There could be several reasons for this, for example another node could reach the same global optimum or such a node could be a neighbour of the seed that could infect the seed. In any case, choosing these nodes gives a better chance for finding the global optimum. Denote this set by S_1. In the second case, we select based on the influenceability value

[1]It must be emphasized here that the size of the circles on Figure 5.2 represents the activation time in the globally optimal solution. For non-optimal $\mathcal{T}$ value we might obtain different times.

alone, so $\alpha = 0$ and added nodes with 0 influenceability value. This is because nodes with an influence value of 0 can never be influenced unless they become seed nodes themselves. Therefore, any vertex with an influence value of 0 is chosen to be a seed vertex. The resulting set is denoted by S_2. Finally, take the intersection of S_1 and S_2 and obtain the set S. The seed nodes are chosen from this set S.

5.5 Numerical experiments

5.5.1 Computational environment

The implementation of the above proposed new centralities was done in R version 4.1.2 using its `igraph` 1.3.5 package. The numerical experiments were executed with Gurobi 10.0 called from AMPL [39] using non-default options: `threads=8 lpmethod=0 cuts=0 mipgapabs=1e-2`. The computer used had Intel Core i7-10700 CPU at 2.90 GHz with 16 GB memory running Ubuntu 22.04.2. Note that we used the multi-core setup of Gurobi (i.e., for solving the ILP models), whereas R was used with single threading. In the R implementation of the new centrality metrics we used the different `apply` functionalities to process matrices and lists efficiently.

5.5.2 Test graphs

Synthetic random graphs To generate random test graphs we used the LFR scheme, which creates networks with prescribed community structures [71]. Note that this procedure only provides the graphs of social-network type, the edge weights and node thresholds needed to be assigned in the second phase. Similarly to our previous work (in Chapter 4), the following procedure was used.

- Regarding the edge weights: nodes with in-weights larger than 1 (given by the LFR method) were normalized to 1; and we applied a multiplication with a factor r_w which was a uniform random number in the interval $[0.6, 1]$.

- The nodes' threshold values were generated with uniform random distribution in the interval $[0.05, 0.6]$.

- From the LFR method three parameters were fixed: mixing parameter $\mu_w = 0.1$, minimal community size $minc = 5$, and maximal community size $maxc = 42$.

- Three parameters were varied in the experiments: number of nodes n, the average degree $avgk$, and the maximum degree $maxk$.

Table 5.6: *A selection of real-world test graphs*

name	N	M	$d_{\max}$
soc-dolphins [79]	62	159	5
ca-sandi-auths	86	124	12
retweet [92, 93]	96	117	17
ca-netscience [84]	379	914	34

Real-world graphs We have tested our proposed method on a small set of real-world graphs [91], see Table 5.6 for the details. Note that the last column indicates the maximum degree of the given graph. For our problem setup $d_{\max}$ needs to be relatively small, since the exact ILP solver as well as our heuristic does not scale efficiently. Nevertheless, for these four social networks we did the following experiments.

The real graphs were undirected and unweighted graphs. The `mutual` parameter was used which makes the graphs directed by doubling the undirected edges. We then created 3 groups according to the percentage of edges that were deleted randomly and how we generated weight of edges and threshold values. These groups and their corresponding generating parameters are shown in Table 5.7. The edges and weights were generated uniform at random in the given interval. As for the weights of the edges, similar to the LFR graphs, nodes with weights greater than 1 were normalized to 1 and multiplied by a factor r_w which were randomly chosen from the interval $[0.6, 1]$.

Table 5.7: *The parameters used to generate real-world test graphs*

group	% of edges deleted	threshold	weights
#1	45	$[0.05, 0.5]$	$[0.01, 0.6]$
#2	50	$[0.1, 0.5]$	$[0.05, 0.5]$
#3	50	$[0.07, 0.55]$	$[0.075, 0.55]$

5.5.3 Results

For all the test graphs we ran the new IAtI-heuristic and compare its results with those obtained by Greedy, see Section 4.5. For reference we also solved the smaller sized problems with Gurobi to get the global optimal solution. Note that we fixed the seed set size to 2.

LFR graphs The results are reported in Table 5.8 and 5.9. If the obtained results (with respect to σ and $\mathcal{T}$) were different, then the better solution is emphasized in boldface. For the smaller sized problems we can see that there were 3 cases when neither our heuristic nor the greedy approach was able to find the global optimum, see Table 5.8. Greedy found better

solutions than our heuristic in 7 cases. Also in 7 cases (that is 23%) our heuristic missed the global optimum. On the other hand, Greedy lost the global optimum in 11 cases (37% of the cases). Regarding the running time, the greedy approach is by far the quickest method. On the other hand, the IAtI-heuristic is faster than the exact method, usually around $4-12$ times.

Table 5.8: *Benchmarking results for the LFR graphs; time in seconds*

graph parameters			IAtI-heuristic			greedy			global opt.		
n	$avgk$	$maxk$	σ	$\mathcal{T}$	time	σ	$\mathcal{T}$	time	σ^*	$\mathcal{T}^*$	time
100	3	8	87	13	22.2	87	13	11.9	87	13	164.2
100	4	8	74	17	77.1	74	17	7.6	74	17	1 442.3
100	4	10	100	15	11.7	100	15	5.5	100	15	20.6
100	5	8	88	24	339.1	88	24	12.4	88	24	4 287.9
100	5	10	89	10	168.7	89	10	5.9	89	10	1 742.7
100	6	8	88	17	348.1	88	17	9.6	88	17	4 353.3
105	3	8	58	11	8.9	58	11	5.4	58	11	92.0
105	4	8	57	10	15.5	57	10	2.4	57	10	169.0
105	4	10	**80**	**15**	16.0	79	14	6.2	80	15	42.2
105	5	8	85	16	20.1	85	16	7.4	85	16	86.9
105	5	10	66	20	41.7	66	20	8.7	66	20	108.0
105	6	8	61	14	27.7	61	14	4.8	**61**	**12**	119.5
110	3	8	75	20	110.0	75	20	10.7	**75**	**15**	655.8
110	4	8	**96**	**23**	411.9	87	15	9.4	96	23	5 510.2
110	4	10	91	16	20.9	91	16	10.1	91	16	188.0
110	5	8	90	18	38.9	90	18	9.4	90	18	313.7
110	5	10	70	15	236.5	70	15	7.1	70	15	430.7
110	6	8	**95**	**15**	221.0	95	19	16.6	95	15	1 085.0
115	3	8	56	13	6.6	56	13	4.6	56	13	59.3
115	4	8	**49**	**18**	13.5	45	14	6.4	49	18	170.5
115	4	10	43	11	12.3	**49**	**13**	4.2	49	13	64.3
115	5	8	**25**	**18**	24.9	21	7	1.1	25	18	219.5
115	5	10	**78**	**20**	81.7	68	14	6.3	78	20	193.1
115	6	8	**49**	**13**	18.2	46	11	4.3	49	13	292.7
120	3	8	86	26	59.7	**86**	**25**	23.4	86	25	1 402.6
120	4	8	**77**	**17**	30.9	75	15	6.9	77	17	307.2
120	4	10	113	20	36.4	**113**	**14**	18.2	113	14	274.9
120	5	8	70	12	97.0	70	14	9.6	**70**	**11**	469.5
120	5	10	82	11	67.6	**83**	**17**	9.9	83	17	326.7
120	6	8	**98**	**15**	113.7	98	17	12.5	98	15	1 160.0

For the larger graphs, reported in Table 5.9 we can see that the in terms of running time Greedy is much faster than our heuristic. Greedy found in only 7 cases better solution than our heuristic. On the other hand the IAtI-heuristic was in 13 cases more successful than the greedy approach and among these there are several where it was much better.

Table 5.9: *Benchmarking results for the LFR graphs for larger instances; time in seconds*

graph parameters			IAtI-heuristic			greedy			global opt.		
n	$avgk$	$maxk$	σ	$\mathcal{T}$	time	σ	$\mathcal{T}$	time	σ^*	$\mathcal{T}^*$	time
125	3	8	116	17	35.5	116	17	13.4	116	17	1 356.2
125	4	8	102	24	407.1	102	24	25.6	**118**	**34**	73 866.8
125	4	10	93	22	115.0	98	27	25.1	**99**	**28**	1 586.1
125	5	8	81	15	303.7	81	15	16.9	81	15	69 321.4
125	5	10	99	19	220.9	**99**	**18**	29.4	99	18	9 840.2
125	6	8	**104**	**13**	406.1	104	14	15.2	104	13	464 737.9
130	3	8	121	28	604.7	121	28	90.4	121	28	13 254.5
130	4	8	**96**	**14**	266.5	84	10	11.7	96	14	7 326.2
130	4	10	**51**	**9**	80.6	51	10	5.9	51	9	903.1
130	5	8	130	20	116.5	**130**	**19**	16.3	130	19	1 172.9
130	5	10	114	16	355.2	**116**	**16**	27.1	116	16	8 064.3
130	6	8	130	15	54.2	130	15	11.9	130	15	285.5
135	3	8	129	19	22.7	**135**	**18**	10.6	135	18	71.6
135	4	8	117	13	215.9	117	13	22.7	117	13	1 834.9
135	4	10	93	14	65.4	93	14	15.8	93	14	1 116.3
135	5	8	**74**	**21**	864.4	55	16	12.1	74	21	68 639.3
135	5	10	81	13	234.5	81	13	15.5	81	13	2 597.7
135	6	8	**113**	**23**	1 059.6	97	18	36.3	113	23	144 091.6
140	3	8	57	11	17.6	57	11	18.4	57	11	2 166.3
140	4	8	**116**	**20**	247.9	114	16	13.3	116	20	1 040.8
140	4	10	82	17	32.5	82	17	14.0	82	17	641.3
140	5	8	**93**	**18**	369.2	72	16	22.7	93	18	34 445.7
140	5	10	111	17	172.8	111	17	21.9	111	17	3 063.2
140	6	8	**130**	**32**	1 283.5	103	17	17.5	130	32	472 788.0
145	3	8	70	16	34.6	70	16	14.0	70	16	963.9
145	4	8	100	21	273.4	100	21	45.2	100	21	11 683.2
145	4	10	60	18	123.8	76	19	17.2	**84**	**17**	1 767.4
145	5	8	58	20	134.7	44	14	7.8	**87**	**26**	10 491.1
145	5	10	**110**	**21**	621.5	108	20	30.0	110	21	11 133.9
145	6	8	101	25	885.1	101	25	31.1	101	25	63 067.0
150	3	8	72	12	107.1	72	12	17.2	72	12	1 496.5
150	4	8	66	13	167.3	**66**	**12**	9.8	66	12	5 819.6
150	4	10	**84**	**16**	78.6	83	17	13.9	84	16	2 212.3
150	5	8	123	21	290.5	118	19	28.2	**123**	**20**	47 651.9
150	5	10	89	27	1 067.1	89	27	61.5	89	27	444 340.0
150	6	8	**130**	**22**	1 207.6	124	18	20.6	130	22	1 063 868.0

To discuss a particular example at which the IAtI-heuritic lost the globally optimal solution, whereas Greedy was able to find it, see Figure 5.3. The scatterplot shows the two centralities of the nodes together with their activation time at the optimal $\mathcal{T}^*$. The empty

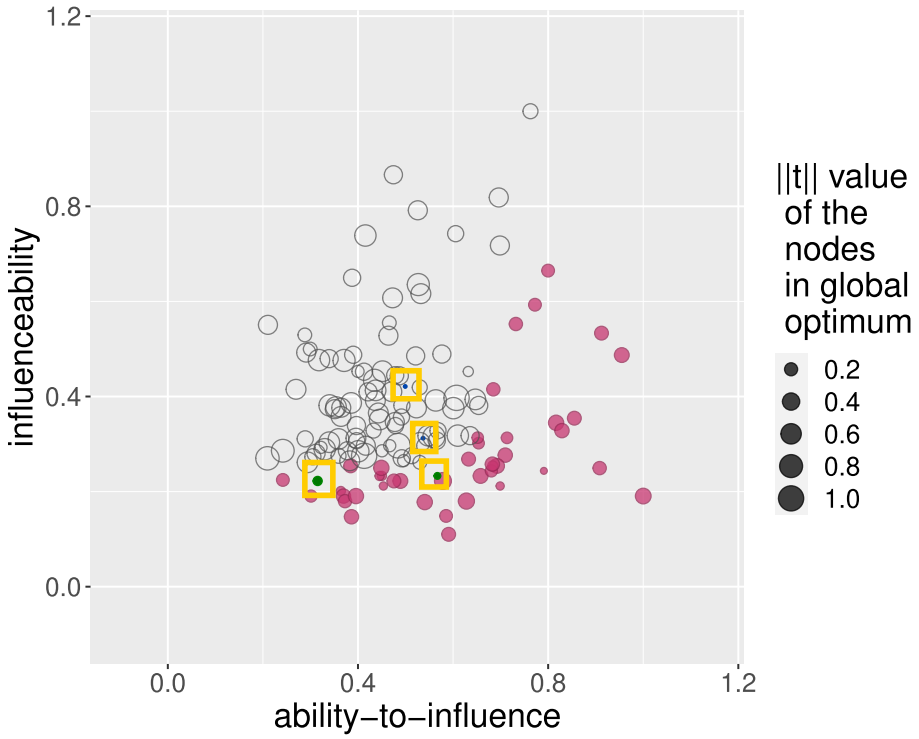


Figure 5.3: *Visualization of the new centralitites for the LFR.135.3.8 graph*

circles correspond to those nodes which were not selected by our heuristic to be candidates for seeds. As shown by the blue circles, we can see that IAtI dropped the optimal seed set, as only those were selected to be candidates which are shown with magenta color. Note that Greedy, instead, found those blue-colored nodes. The seed nodes chosen by the IAtI-heuristic are shown as green colored circles. It would be possible to parameterize the IAtI-heuristic in such a way that it would include the optimal seeds in the candidate set. Figure 5.3 suggests that in that case the cardinality of the candidate seed set would be necessarily larger which would result in much longer running time.

Real-world graphs Our experiments on some selected real world graphs are reported in Table 5.10. As in the earlier tables it is shown in boldface if a method obtained a better solution than the other one. We can observe that the trend of the IAtI-heuristic being much slower than Greedy remains to be the case also here. There was only one case when the greedy approach found a better solution than our heuristic. On the other hand, IAtI-heuristic was able to find better solution in 5 cases (out of 12), which corresponds to 42%.

Table 5.10: *Results for the small real-world graphs; time in second*

	IAtI-heuristic			greedy		
graph	σ	$\mathcal{T}$	time	σ	$\mathcal{T}$	time
soc-dolphins #1	**45**	**18**	11.9	41	10	5.1
ca-sandi-auths #1	**34**	**9**	6.9	30	9	1.7
retweet #1	22	7	5.5	22	7	0.9
ca-netscience #1	63	9	319.5	63	9	183.6
soc-dolphins #2	**34**	**9**	6.9	24	5	0.5
ca-sandi-auths #2	**13**	**5**	5.5	12	4	0.3
retweet #2	19	6	4.8	19	6	0.7
ca-netscience #2	25	7	3 047.7	**25**	**6**	13.8
soc-dolphins #3	27	8	5.5	27	8	1.5
ca-sandi-auths #3	14	5	7.6	14	5	0.5
retweet #3	17	6	4.9	17	6	0.6
ca-netscience #3	**41**	**11**	300.1	30	5	10.2

Finally, let us demonstrate again the seed node selection strategy of the IAtI-heuristic on the ca-sandi-auths graph as shown in Figure 5.4. The blue colored circles represent the nodes found by the Greedy approach, leading to a suboptimal result. The blue circle on the left was exluded by the IAtI-heuristic from the possible set of seed nodes. As before, the candidate set of seed nodes are colored with magenta. Among them, there are the two optimal seed nodes shown as green circles. Note that there are quite many nodes with 0 influenceability value. Those are definitely selected by IAtI as possible seed nodes as they are impossible to be activated by other nodes. As we can observe, one of them is indeed part of the optimal seed set.

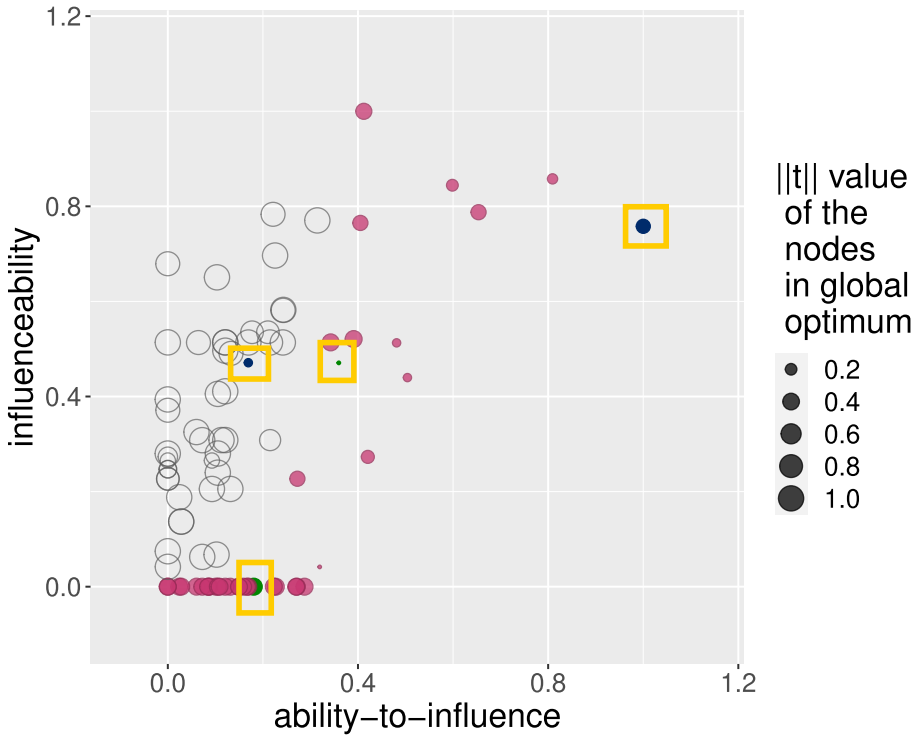


Figure 5.4: *Visualization of the new centralitites for the ca-sandi-auths #1 real graph*

5.6 Conclusion

We proposed two new centrality metrics for the influence maximization under deterministic linear threshold model. These metrics take into account the structure of the input network, more precisely the weight of edges, the combinations of edges and the threshold value of vertices. This is a great advantage of our method, because it is not usual to include the node's threshold in the centrality measure. Using the two centrality metrics, we selected vertices that have a high probability of being seed nodes. The solver now selects seed vertices only among these. This reduces the computational complexity of the task and therefore, compared to running the ILP solver on the unrestricted model, it speeds up the procedure. Using those metrics, we created the so-called IAtI algorithm. This was compared with Greedy and with the global optimum, also. The IAtI algorithm is slower than Greedy, but in many cases it gives a better solution and in most cases it finds the global optimum. Although most real-world networks are sparse graphs, for which our method works well, the disadvantage of this method is that it takes a lot of computing time to generate and use edge combinations for

large degree of nodes. Thus, it may take a long time for pre-processing and is therefore, in its current form, not well scalable. Our future work is to develop a version of the method that can handle graphs including nodes with relatively large degree.

Bibliography

[1] Daron Acemoglu, Asuman Ozdaglar, and Ercan Yildiz. Diffusion of innovations in social networks. In *50th IEEE Conference on Decision and Control and European Control Conference*, pages 2329–2334, 2011.

[2] Fabrizio Altarelli, Alfredo Braunstein, Luca Dall'Asta, and Riccardo Zecchina. Optimizing spread dynamics on graphs by message passing. *Journal of Statistical Mechanics: Theory and Experiment*, 2013(09):P09011, 2013.

[3] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[4] Stefano Benati. An Improved Branch & Bound Method for the Uncapacitated Competitive Location Problem. *Annals of Operations Research*, 122(1):43–58, 2003.

[5] Jacobus Franciscus Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962/63.

[6] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[7] Phillip Bonacich. Factoring and weighting approaches to status scores and clique identification. *The Journal of Mathematical Sociology*, 2:113–120, 1972.

[8] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30:107–117, 1998.

[9] Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

[10] Cheng-Lung Chen, Eduardo Pasiliao, and Vladimir Boginski. A cutting plane method for least cost influence maximization. In Sriram Chellappan, Kim-Kwang Raymond Choo, and NhatHai Phan, editors, *Computational Data and Social Networks*, pages 499–511. Springer International Publishing, 2020.

[11] Wenlin Chen, Yixin Chen, and Kilian Weinberger. Filtered search for submodular maximization with controllable approximation bounds. In *Artificial Intelligence and Statistics*, pages 156–164, 2015.

[12] Yongbo Chen, Liang Zhao, Yanhao Zhang, Shoudong Huang, and Gamini Dissanayake. Anchor selection for slam based on graph topology and submodular optimization. *IEEE Transactions on Robotics*, 38(1):329–350, 2022.

[13] Chun-Hung Cheng, Yong-Hong Kuo, and Ziye Zhou. Outbreak minimization vs influence maximization: an optimization framework. *BMC Medical Informatics and Decision Making*, 20(1):1–13, 2020.

[14] Eszter Csókás and Tamás Vinkó. Constraint generation approaches for submodular function maximization leveraging graph properties – supplementary information. `https://www.inf.u-szeged.hu/~tvinko/CsV-submodular_max-supplement.pdf`.

[15] Eszter Csókás and Tamás Vinkó. On the initial set of constraints for submodular function maximization – supplementary information. `https://www.inf.u-szeged.hu/~tvinko/CsV-submodular_max-supplement2.pdf`.

[16] Eszter Csókás and Tamás Vinkó. An exact method for influence maximization based on deterministic linear threshold model. *Central European Journal of Operations Research*, 31:269–286, 2023.

[17] Eszter Csókás and Tamás Vinkó. Constraint generation approaches for submodular function maximization leveraging graph properties. *Journal of Global Optimization*, 88:377–394, 2024.

[18] Eszter Csókás and Tamás Vinkó. On the initial set of constraints for graph-based submodular function maximization. *Acta Cybernetica*, 2024.

[19] Eszter Csókás and Tamás Vinkó. A heuristic for influence maximization under deterministic linear threshold model. *Informatica*, 48(4), 2025.

[20] George Dantzig. Origins of the simplex method. In *A history of scientific computing*, pages 141–151. Association for Computing Machinery, 1990.

[21] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[22] Abhimanyu Das and David Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of*

the 28th International Conference on Machine Learning, ICML'11, pages 1057–1064, 2011.

[23] Kousik Das, Sovan Samanta, and Madhumangal Pal. Study on centrality measures in social networks: a survey. *Social Network Analysis and Mining*, 8:1–11, 2018.

[24] Dingzhu Du, Panos Pardalos, Xiaodong Hu, and Weili Wu. *Introduction to combinatorial optimization*. Springer, 2022.

[25] Jack Edmonds. Matroids, submodular functions, and certain polyhedra. *Combinatorial Structures and Their Applications*, pages 69–87, 1970.

[26] Paul Erdős and Alfréd Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[27] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5(1):17–60, 1960.

[28] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Bulletin of the Institute of International Statistics*, 38:343–347, 1961.

[29] Paul Erdős and Alfréd Rényi. On the strength of connectedness of a random graph. *Acta Mathematica Academiae Scientiarum Hungarica*, 12(1):261–267, 1961.

[30] Paul Erdős and Alfréd Rényi. Asymmetric graphs. *Acta Mathematica Academiae Scientiarum Hungaricae*, 14(295-315):3, 1963.

[31] Paul Erdős and Alfréd Rényi. On random matrices. *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, 8(455-461), 1966.

[32] Paul Erdős and Alfréd Rényi. On the existence of a factor of degree one of a connected random graph. *Acta Mathematica Academiae Scientiarum Hungaricae*, 17(359-368):192, 1966.

[33] Paul Erdős and Alfréd Rényi. On random matrices II. *Studia Scientiarum Mathematicarum Hungarica*, 3:459–464, 1968.

[34] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

[35] Julius Farkas. Theorie der einfachen Ungleichungen. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1902(124):1–27, 1902.

[36] Uriel Feige, Vahab Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

[37] Thomas Feo and Mauricio Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

[38] Stephanie Forrest. Genetic algorithms. *ACM Computing Surveys*, 28(1):77–80, 1996.

[39] Robert Fourer, David Gay, and Brian Kernighan. *AMPL. A modeling language for mathematical programming*. Thomson, 1993.

[40] András Frank. Matroids and submodular functions. *Annotated Biblographies in Combinatorial Optimization*, pages 65–80, 1997.

[41] Linton Clarke Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.

[42] Tibor Gallai. Maximum-minimum Sätze über Graphen. *Acta Mathematica Academiae Scientiarum Hungaricae*, 9:395–434, 1958.

[43] Farzaneh Ghayour-Baghbani, Masoud Asadpour, and Heshaam Faili. MLPR: Efficient influence maximization in linear threshold propagation model using linear programming. *Social Network Analysis and Mining*, 11:1–10, 2021.

[44] Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12(3):211–223, 2001.

[45] Boris Goldengorin. Maximization of submodular functions: Theory and enumeration algorithms. *European Journal of Operational Research*, 198(1):102–112, 2009.

[46] Boris Goldengorin and Diptesh Ghosh. A multilevel search algorithm for the maximization of submodular functions applied to the quadratic cost partition problem. *Journal of Global Optimization*, 32:65–82, 2005.

[47] Daniel Golovin and Andreas Krause. Adaptive submodularity: A new approach to active learning and stochastic optimization. In *Proceedings of International Conference on Learning Theory*, pages 333–345, 2010.

[48] Mark Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420–1443, 1978.

[49] Martin Grötschel, Lovász László, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 06 1981.

[50] Evren Güney. An efficient linear programming based method for the influence maximization problem in social networks. *Information Sciences*, 503:589–605, 2019.

[51] Evren Güney, Markus Leitner, Mario Ruthmair, and Markus Sinnl. Large-scale influence maximization via maximal covering location. *European Journal of Operational Research*, 289(1):144–164, 2021.

[52] Furkan Gursoy and Dilek Gunnec. Influence maximization in social networks under deterministic linear threshold model. *Knowledge-Based Systems*, 161:111–123, 2018.

[53] Pierre Hansen, Brigitte Jaumard, and Gilles Savard. New branch-and-bound rules for linear bilevel programming. *SIAM Journal on Scientific and Statistical Computing*, 13(5):1194–1217, 1992.

[54] IBM. ILOG CPLEX 22.1.1 User's Manual. 2022.

[55] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM*, 48(4):761–777, 2001.

[56] Asunción Jiménez-Cordero, Juan Miguel Morales, and Salvador Pineda. Warm-starting constraint generation for mixed-integer optimization: A machine learning approach. *Knowledge-Based Systems*, 253:109570, 2022.

[57] Ari Juels and Martin Wattenberg. Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. *Advances in Neural Information Processing Systems*, 8, 1995.

[58] Michael Kahr, Markus Leitner, and Ivana Ljubić. The impact of passive social media users in (competitive) influence maximization. *Technical report. University of Vienna, Austria*, 2022.

[59] Michael Kahr, Markus Leitner, Mario Ruthmair, and Markus Sinnl. Benders decomposition for competitive influence maximization in (social) networks. *Omega*, 100:102264, 2021.

[60] Panagiotis Karampourniotis, Boleslaw Szymanski, and Gyorgy Korniss. Influence maximization for fixed heterogeneous thresholds. *Scientific Reports*, 9:5573, 2019.

[61] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[62] Richard Karp. *Reducibility among combinatorial problems*. Springer, 2010.

[63] Yoshinobu Kawahara, Kiyohito Nagano, Koji Tsuda, and Jeff A Bilmes. Submodularity cuts and applications. In *Advances in Neural Information Processing Systems*, volume 22, pages 1–9, 2009.

[64] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.

[65] Muhammed Emre Keskin and Mehmet Güray Güler. Influence maximization in social networks: an integer programming approach. *Turkish Journal of Electrical Engineering & Computer Sciences*, 26(6):3383–3396, 2018.

[66] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[67] Stepan Kochemazov. Comparative study of combinatorial algorithms for solving the influence maximization problem in networks under a deterministic linear threshold model. *Procedia Computer Science*, 136:190–199, 2018. 7th International Young Scientists Conference on Computational Science, YSC2018.

[68] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability*, 3:71–104, 2014.

[69] Andreas Krause and Carlos Guestrin. Optimal nonmyopic value of information in graphical models: Efficient algorithms and theoretical limits. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1339–1345, 2005.

[70] Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):1–20, 2011.

[71] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E-Statistical, Nonlinear, and Soft Matter Physics*, 78:046110, 2008.

[72] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1852–1872, 2018.

[73] Bo Liu, Gao Cong, Dong Xu, and Yifeng Zeng. Time constrained influence maximization in social networks. In *2012 IEEE 12th International Conference on Data Mining*, pages 439–448. IEEE, 2012.

[74] László Lovász. A characterization of perfect graphs. *Journal of Combinatorial Theory, Series B*, 13(2):95–98, 1972.

[75] László Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Mathematics*, 2(3):253–267, 1972.

[76] Cheng Lu, Wenguo Yang, and Suixiang Gao. A new greedy strategy for maximizing monotone submodular function under a cardinality constraint. *Journal of Global Optimization*, 83(2):235–247, 2022.

[77] Zaixin Lu, Wei Zhang, Weili Wu, Bin Fu, and Dingzhu Du. Approximation and inapproximation for the influence maximization problem in social networks under deterministic linear threshold model. In *31st International Conference on Distributed Computing Systems Workshops*, pages 160–165, June 2011.

[78] Zaixin Lu, Wei Zhang, Weili Wu, Joonmo Kim, and Bin Fu. The complexity of influence maximization problem in the deterministic linear threshold model. *Journal of Combinatorial Optimization*, 24(3):374–378, 2012.

[79] David Lusseau, Karsten Schneider, Oliver Boisseau, Patti Haase, Elisabeth Slooten, and Steve Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

[80] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques: Proceedings of the 8th IFIP Conference on Optimization Techniques Würzburg*, pages 234–243. Springer, 1977.

[81] Giacomo Nannicini, Giorgio Sartor, Emiliano Traversi, and Roberto Wolfler Calvo. An exact algorithm for robust influence maximization. *Mathematical Programming*, 183:419–453, 2020.

[82] George Nemhauser and Laurence Wolsey. Maximizing submodular set functions: formulations and analysis of algorithms. *Studies on Graphs and Discrete Programming*, pages 279–301, 1981.

[83] George Nemhauser, Laurence Wolsey, and Marshall Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

[84] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3):036104, 2006.

[85] Naoto Ohsaka and Tatsuya Matsuoka. Approximation algorithm for submodular maximization under submodular cover. In *Uncertainty in Artificial Intelligence*, pages 792–801, 2021.

[86] Salvador Pineda, Juan Miguel Morales, and Asunción Jiménez-Cordero. Data-driven screening of network constraints for unit commitment. *IEEE Transactions on Power Systems*, 35:3695–3705, 2019.

[87] Márton Pósfai and Albert-László Barabási. *Network Science*. Cambridge University Press, 2016.

[88] Zhecheng Qiang, Eduardo Pasiliao, and Qipeng Zheng. Model-based learning of information diffusion in social media networks. *Applied Network Science*, 4(1):1–16, 2019.

[89] Khadije Rahimkhani, Abolfazl Aleahmad, Maseud Rahgozar, and Ali Moeini. A fast algorithm for finding most influential people based on the linear threshold model. *Expert Systems with Applications*, 42:1353–1361, 2015.

[90] Daniele Rosa and Alessandro Giua. On the spread of innovation in social networks. *IFAC Proceedings Volumes*, 46(27):322–327, 2013.

[91] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[92] Ryan Rossi, David Gleich, Assefaw Gebremedhin, and Md Mostofa Ali Patwary. What if clique were fast? Maximum cliques in information networks and strong components in temporal networks. *arXiv preprint arXiv:1210.5802*, pages 1–11, 2012.

[93] Ryan Rossi, David Gleich, Assefaw Gebremedhin, and Md Mostofa Ali Patwary. Fast maximum clique algorithms for large graphs. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 365–366, 2014.

[94] Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

[95] Shinsaku Sakaue and Masakazu Ishihata. Accelerated best-first search with upper-bound computation for submodular function maximization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[96] Shinsaku Sakaue, Masaaki Nishino, and Norihito Yasuda. Submodular function maximization over graphs via zero-suppressed binary decision diagrams. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[97] Domenico Salvagnin. Some experiments with submodular function maximization via integer programming. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 488–501. Springer, 2019.

[98] Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80:346–355, 2001.

[99] Santiago Segarra and Alejandro Ribeiro. Stability and continuity of centrality measures in weighted graphs. *IEEE Transactions on Signal Processing*, 64:543–555, 2016.

[100] Marvin Shaw. Group structure and the behavior of individuals in small groups. *The Journal of Psychology*, 38:139–149, 1954.

[101] Jianbing Shen, Zhiyuan Liang, Jianhong Liu, Hanqiu Sun, Ling Shao, and Dacheng Tao. Multiobject tracking by submodular optimization. *IEEE Transactions on Cybernetics*, 49(6):1990–2001, 2019.

[102] Anuj Singh, Rishi Ranjan Singh, and S.R.S. Iyengar. Node-weighted centrality: a new way of centrality hybridization. *Computational Social Networks*, 7:Article nr. 6, 2020.

[103] Karl Robert Stromberg. *Introduction to classical real analysis*. Wadsworth International Group, Belmont, California, 1981.

[104] Rama Suri and Yadati Narahari. Determining the top-k nodes in social networks using the Shapley value. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1509–1512, 2008.

[105] Ashis Talukder, Md. Golam Rabiul Alam, Nguyen Tran, Dusit Niyato, Gwan Hoon Park, and Choong Seon Hong. Threshold estimation models for linear threshold-based influential user mining in social networks. *IEEE Access*, 7:105441–105461, 2019.

[106] Naoya Uematsu, Shunji Umetani, and Yoshinobu Kawahara. An efficient branch-and-cut algorithm for submodular function maximization. *Journal of the Operations Research Society of Japan*, 63(2):41–59, 2020.

[107] Andrew Vince. A framework for the greedy algorithm. *Discrete Applied Mathematics*, 121(1):247–260, 2002.

[108] Yanhao Wang, Yuchen Li, Francesco Bonchi, and Ying Wang. Balancing utility and fairness in submodular maximization. *CoRR*, abs/2211.00980, 2023.

[109] Duncan Watts and Steven Strogatz. Collective dynamics of 'small-world'-networks. *Nature*, 393(6684):440–442, 1998.

[110] Hao-Hsiang Wu and Simge Küçükyavuz. A two-stage stochastic programming approach for influence maximization in social networks. *Computational Optimization and Applications*, 69(3):563–595, 2018.

[111] Álinson Xavier, Feng Qiu, and Shabbir Ahmed. Learning to solve large-scale security-constrained unit commitment problems. *INFORMS Journal on Computing*, 33(2):739–756, 2021.

[112] Rupei Xu. An L_p norm relaxation approach to positive influence maximization in social network under the deterministic linear threshold model. In Anthony Bonato, Michael Mitzenmacher, and Paweł Prałat, editors, *Algorithms and Models for the Web Graph*, pages 144–155. Springer International Publishing, 2013.

[113] Lan Yang, Alessandro Giua, and Zhiwu Li. Minimizing the influence propagation in social networks for linear threshold models. *IFAC-PapersOnLine*, 50:14465–14470, 2017.

[114] Shunyu Yao, Neng Fan, and Jie Hu. Modeling the spread of infectious diseases through influence maximization. *Optimization Letters*, 16:1563–1586, 2022.

[115] Haifeng Zhang and Yevgeniy Vorobeychik. Submodular optimization with routing constraints. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016.

Summary

This dissertation summarizes the results of my work in the field of optimization and network science. Among my main goals was to improve existing solver algorithms for specific problems using input graphs after reviewing the literature. It is becoming a popular method nowadays to combine these two research areas in this way, which was a motivating force for me.

My thesis consists of two main parts. In the first one, I dealt with the maximization of submodular functions on problems that have graph representations. First, I presented the variants of our constraint generating algorithm, and then proposed a centrality metric to serve as a starting point for solving algorithms.

In the second part, I considered the influence propagation maximization, which is also interpreted in terms of a graph. First, we analysed in detail the exact solving procedure published in the literature, which turned out in the end to be not always correct, thus we presented a possible improvement. Finally, for this work, we have also constructed a centrality metric based on the input graph that proposes the vertices that can be chosen at the initial moment of propagation.

Submodular function maximization based on graph properties

For the submodular function maximization studied in the first part of this dissertation, we considered functions with even graph representation. We considered different solving algorithms, which are constraint condition generating type algorithms. The basis of this type of solvers is a reduced mixed integer problem, which is augmented with a new constraint condition at each iteration.

Constraint generation approaches

This work is inspired by a recent publication [106], which includes the Improved Constraint Generation (ICG solution method, an improved version of the standard constraint generation method. The idea is to add several constraints to the problem per iteration instead of just

one, thus speeding up the solution process. Based on this, we have developed three different algorithm variants for the maximization problem of nondecreasing submodular functions.

First, we present the ICG($k-1$) algorithm, which works with subsets of $k-1$ elements, thus investigating the value of adding the kth element to the set. Another approach was to exploit the structural properties of the input graph. We use a centrality metric for node selection to generate the constraints; we call this algorithm GCG. Finally, we proposed the ECG algorithm, which directly generates the subsets of the sets, that is, the constraint conditions, instead of the iterative internal algorithm embedded in the algorithm.

Based on the experiments, we cannot highlight only one algorithm as the best, which can be explained by the fact that the problem is NP-hard. However, in all tested cases we had at least one algorithm that proved to be computationally more efficient than the ICG algorithm.

On the initial set of constraints

The importance of choosing the starting point, or initial constraints, has already been emphasized by Nemhauser and Wolsey in their article [82], which served as a motivation for our work. In the literature, in most cases, the solution generated by the greedy algorithm or a randomly chosen set of feasible solutions is used as a starting point. However, alternative proposals can also be found in the literature, such as the GRASP heuristic [97], which was also tested in our work.

In order to introduce a new centrality metric, we further investigated the structure of the input graph: in this case, the graphs we consider are not fully-connected bipartite graphs, where we consider both the weights of the edges and the degree values of the vertices, according to the specificity of the input problem. The values are dynamically recalculated, i.e., after a vertex is selected, we then remove it and its associated edges from the graph. This recalculated centrality was used to determine the order of the initial vertices, which was used as the starting point when solving to maximize submodular functions.

In our research, we tested our initial selection strategy in different ways. First, we tested five algorithm variants, which are constraint-generating solver methods for maximizing nondecreasing submodular functions, as presented in Chapter 4. The algorithms are started both from the solution of the greedy procedure and from the starting point suggested by the new centrality metric. In addition, we used two state-of-the-art implementations of Nemhauser and Wolsey's MILP model proposed in [97], which use a lazy constraint generation approach. These were tested starting from the starting point obtained with the GRASP heuristic and the new centrality metric.

Based on our results, the runtime of algorithms started from the initial set proposed by the new centrality metric showed an average improvement of 5.37 times in all test cases. Based on the results, it can be concluded that the use of the starting point proposed by the new centrality metric provides a significant advantage, which is confirmed not only by the reduction in runtime, but also by the analysis of the relative gap values.

Influence maximization under deterministic linear threshold model

In the second part of the thesis, we dealt with the maximization of influence propagation, including models that use the deterministic linear threshold model as a propagation model. In this section we also considered problems with graphical representation, and more specifically their solution algorithms, which thus appear in all four main chapters.

An exact method

For this problem, a linear programming model of type 0-1 has been proposed in the paper [65]. This attracted our interest in the topic and motivated us to work on it. We performed a thorough analysis of the model, which allowed us to prove that the model presented by [65] does not always find the global optimum.

We have examined the proposed constraints and found them to be both appropriate and necessary. The research revealed that the model is not complete, including the possibility of graphs that cannot start infection or are stuck at local optimum. This is what led us to argue that the constraints need to be extended to achieve the correct model. To change the constraints and find the global optimum, the correct stopping condition is very important. To do this, we introduced a new constraint that forces the value of the influence to be larger at the last moment than the one before; we then modified the stopping criterion accordingly. Thus, in fact, the model we propose is an improved version of the model presented by [65].

From our extensive numerical results, it can be concluded that even for relatively small graphs, we can find an exact solution with extremely unfavorable running times. This follows from the fact that the problem is highly NP-complete and is a good illustration of how difficult it is to find the global optimum even with computational methods. These results have provided the motivation not to stop at this point in the research, but to look for ways to reduce the runtime. It can be seen that this is not an easy challenge, success is not guaranteed, but a detailed examination of the algorithm eventually provided the idea for developing a heuristic step.

A heuristic for seeds selection

The research question whether it is possible to find a vertex ranking (i.e. centrality) value that can be used to predict which vertices to select (as seed vertices) at the initial moment is very interesting and has attracted many researchers [52, 67, 89, 102].

Based on the deterministic linear threshold model, we proposed two new centrality metrics to maximize the influence spread. These take into account the structure of the input network, in particular the weight of edges, the combinations of edges and the threshold of vertices. This approach is unique in a sense that the threshold of the nodes is rarely taken

into account when determining centrality measures.

Based on the two centrality metrics, we selected vertices that have a high probability of being good choices for spreading influence at the initial moment as seed vertices. The optimization procedure then selects the seed nodes from these only. Using centrality metrics, we generated the IAtI algorithm, which was compared to both the Greedy algorithm and the global optimum. The IAtI algorithm is slower than the Greedy algorithm, but in many cases it provides a better solution and in most cases it finds the global optimum.

Choosing from a reduced set significantly decreases the computational complexity and thus the runtime of the solver algorithm. However, the current version of the method does not scale well, since generating and handling combinations of edges for vertices with a large number of degrees requires significant computation time, and therefore the preprocessing time can be long. Note, however, that the sparsity of graphs in the majority of real networks contributes to the robustness of our method. Our future goal is to develop a version that can efficiently handle graphs with a large number of degree vertices.

Contributions of the thesis

In the **first thesis group**, the contributions are related to Part I, the maximization of submodular functions. A detailed presentation is given in Chapters 2 and 3.

I/1. I developed a version of the constraint generating algorithm that works with subsets of $k-1$ elements.

I/2. I created another version of the constraint generation algorithm, GCG, which exploits the structure of the graph in a heuristic step.

I/3. I described another version of the constraint generating algorithm, ECG, which generates subsets directly.

I/4. I introduced a new centrality metric, which is an initial point selection strategy for solving submodular function maximization.

I/5. I show that the new starting point selection strategy is better than the commonly used greedy method or the recently published GRASP heuristic.

In the **second thesis group**, the contributions are related to Part II, the maximization of influence spread. A detailed presentation is given in Chapters 4 and 5.

II/1. I discovered that the integer model proposed in [65] is not correct in all cases.

II/2. I showed that the model needs to be completed to get the correct fit.

II/3. I demonstrated step by steps that the new model we have constructed is correct.

II/4. By examining the step-by-step solution of the correct model, I presented two new centrality metrics for Influenceability and Ability-to-influence.

II/5. I demonstrated that by using a combination of the two centrality metrics, we can reduce the number of possible seed nodes at the initial moment.

Összefoglalás

Ez a disszertáció az optimalizálás és a hálozattudomány területén végzett munkám eredményeit foglalja össze. Fő céljaim között volt, hogy a szakirodalom áttekintése után, az adott feladatokra meglévő megoldó algoritmusokat fejlesszem az input gráfok használatával. Egyre népszerűbb módszer napjainkban összekötni ilyen módon ezt a két kutatási területet, ami rám is motiváló erővel hatott.

A dolgozatom két fő részből áll. Az első részben a szubmoduláris függvények maximalizálásával foglalkoztam. Először az általunk létrehozott feltétel generáló algoritmus változatokat mutattam be, majd egy centralitási metrikát javasoltam, ami kiindulási pontként szolgálhat a megoldó algoritmusoknál. A tesztelést olyan szubmoduláris függvényeket használtam, melyek rendelkeznek gráfos reprezentációval (ezt néhány megoldó algoritmus igényli).

A második részben a befolyásterjedés maximalizálásával foglalkoztam, amit szintén gráfokon értelmezünk. Először részletesen elemeztünk egy a szakirodalomban megjelent egzakt megoldó eljárást, amiről kiderült a munkánk során, hogy nem minden esetben helyes, így ennek javítási lehetőségét mutattuk be. Végül ehhez a munkához is készítettünk egy olyan centralitási metrikát, amely az input gráfon alapszik és javaslatot tesz a terjedés kezdeti pillanatában választható csúcsokra.

Szubmoduláris függvények maximalizálása

A disszertáció első felében vizsgált szubmoduláris függvény maximalizálásnál olyan függvényeket vizsgáltunk, melyek páros gráfos reprezentációval rendelkeznek. Különböző megoldó algoritmusokkal foglalkoztunk, melyek korlátozó feltétel generáló típusú eljárások. Az ilyen típusú megoldók alapja, hogy egy redukált vegyes egész értékű feladatból indulnak ki, mely minden iterációban bővül egy új korlátozó feltétellel.

Korlátozó feltétel generáló megközelítések

Ezt a munkánkat egy közelmúltbeli publikáció [106] inspirálta, amiben szerepel az ICG (Improved Constraint Generation) megoldó eljárás, ami a standard feltétel generáló módszer továbbfejlesztett változata. Ennek lényege, hogy nem egy, hanem több korlátozást ad hozzá

a feladathoz iterációnként, ezzel gyorsítva a megoldás menetét. Erre alapozva, három különböző algoritmusváltozatot dolgoztunk ki a nem csökkenő szubmoduláris függvények maximalizálási problémájára.

Elsőként az ICG($k-1$) algoritmust mutattuk be, amely $k-1$ elemszámú részhalmazokkal dolgozik. Ezzel azt tudjuk vizsgálni, hogy mi az értéke annak, ha a k-adik elemet adjuk hozzá a halmazhoz. Egy másik megközelítésünk a bemeneti gráf szerkezeti tulajdonságainak kihasználása volt. A feltételek generáláshoz szükséges csúcs választáshoz egy centralitási metrikát használunk; ezt az algoritmust GCG-nek neveztük el. Végül javasoltuk az ECG algoritmust, amely az eljárásba beágyazott iteratív belső algoritmus helyett közvetlenül generálja a halmazok részhalmazait, azaz a korlátozó feltételeket.

A tesztek alapján nem tudjuk csak az egyik algoritmust kiemelni, mint a legjobbat, ami azzal magyarázható, hogy a feladat NP-nehéz. Azonban minden tesztelt esetben volt legalább egy algoritmusunk, amely számítási szempontból hatékonyabbnak bizonyult az ICG eljárásnál.

A kezdeti korlátozó feltételek választása

A kezdőpont, avagy kezdeti korlátozó feltételek megválasztásának fontosságát már Nemhauser és Wolsey is kiemelte [82] a cikkében, ami motivációként szolgált munkánk során. A szakirodalomban, a legtöbb esetben a mohó algoritmus által generált megoldás vagy egy véletlenszerűen kiválasztott fizíbilis megoldás halmaz szolgál kiindulópontként. Azonban az irodalomban alternatív javaslatok is találhatóak, például a GRASP heurisztika [97], amelyet szintén teszteltünk munkánk során.

Egy új centralitási metrika bevezetése érdekében az input gráf szerkezetét tovább vizsgáltuk: ebben az esetben az általunk vizsgált gráfok nem teljes páros gráfok. A metrika során mind az élek súlyait, mind a csúcsok fokszámait figyelembe vesszük, a kiindulási probléma sajátosságainak megfelelően. Az értékeket dinamikusan újraszámítjuk, azaz miután egy csúcsot kiválasztottunk, azután a gráfból eltávolítjuk azt és a hozzátartozó éleket. Ezt az újraszámított centralitást használtuk fel a kezdeti csúcsok sorrendjének meghatározásához, amelyet szubmoduláris függvények maximalizálására irányuló problémák megoldása során kezdeti halmazaként alkalmaztunk.

Kutatásunk során a kiindulópont választási stratégiánkat különböző módon teszteltük. Egyrészt vizsgáltunk öt különböző algoritmus változatot, amelyek a 4. Fejezetben bemutatott, nem-csökkenő szubmoduláris függvények maximalizálására alkalmas, korlátozó feltételeket generáló megoldó metódusok. Az algoritmusokat mind a mohó eljárás megoldásából, mind pedig az új centralitási metrika által javasolt kezdőpontból indítottuk. Emellett két modern implementációt alkalmaztunk Nemhauser és Wolsey MILP modelljéből, amiket a [97] cikkben javasoltak, amelyek lusta korlátozó generálási megközelítést használnak. Ezeket a GRASP heurisztikával, illetve az új centralitási metrikával kapott kezdőpontból indítva teszteltük.

Eredményeink alapján az új centralitási mutató által javasolt kezdőhalmazból indított algoritmusok futási ideje átlagosan 5.37-szeres javulást mutatott az összes tesztesetben. Az eredmények alapján megállapítható, hogy az új centralitási metrika által javasolt kiindulási pont alkalmazása jelentős előnyt nyújt, amit nemcsak a futási idő csökkenése, hanem a relatív eltérés értékeinek elemzése is alátámaszt.

Befolyásterjedés maximalizálása determinisztikus lineáris küszöbmodell használatával

A dolgozat második felében a befolyásterjedés maximalizálásával foglalkoztunk, azon belül is azokkal a modellekkel, amelyek a determinisztikus lineáris küszöbmodellt használják terjedési modellként. Ebben a részben is gráfos reprezentációval bíró feladatokat vizsgáltunk, egész pontosan azok megoldó algoritmusait.

Egy egzakt modell megoldásairól

A feladatra egy 0-1 típusú lineáris programozási modellt javasoltak a [65] publikációban. Ez keltette fel az érdeklődésünket a témával kapcsolatban és adott motivációt munkánkhoz. Alapos elemzést végeztünk a modellen, ami által sikerült igazolni, hogy a [65] által bemutatott modell nem minden esetben találja meg a globális optimumot.

Megvizsgáltuk a javasolt korlátozó feltételeket, majd beláttuk, hogy azok helyesek és szükségesek is. A kutatás során kiderült, hogy a modell nem teljes, többek között előfordulhat olyan gráf, amin el sem tudja kezdeni a fertőzést, vagy beragad lokális optimumban. Ez vezetett minket arra az állításra, hogy a korlátozó feltételeket ki kell egészíteni a helyes modell eléréséhez. A korlátozó feltételek változtatása és a globális optimum megtalálása érdekében nagyon fontos a helyes megállási feltétel. Ehhez bevezettünk egy új korlátozó feltételt, ami kikényszeríti azt, hogy a befolyás értéke az utolsó pillanatban nagyobb legyen, mint az azt megelőzőben; majd ennek megfelelően módosítottuk a megállási kritériumot is. Így valójában az általunk javasolt modell egy továbbfejlesztett változata a [65] által bemutatott modellnek.

A részletes numerikus eredményeinkből azt a következtetést lehet levonni, hogy még viszonylag kis gráfok esetén is csak rendkívül kedvezőtlen futási idővel tudunk pontos megoldást találni. Ez következik abból, hogy a feladat erősen NP-teljes és jó illusztráció arra, hogy számítógépes eszközökkel is milyen nehéz megtalálni a globális optimumot. Ezek az eredmények szolgáltattak motivációt arra, hogy ne álljunk meg a kutatás ezen pontján, hanem vizsgáljunk olyan lehetőségeket, amelyek csökkentik a futási időt. Látható, hogy ez nem könnyű feladat, nem biztosított a siker, de az algoritmus részletes vizsgálata nyújtott végül ötletet heurisztikus lépés kidolgozásához.

Heurisztika a kezdő csúcsok választásához

Nagyon érdekes és sokakat megmozgató az a kutatási kérdés, hogy lehet-e találni olyan csúcsrangsorolási (azaz centralitási) értéket, amellyel megjósolható, hogy a kezdeti pillanatban mely csúcsokat válasszuk ki (ún. seed csúcsnak). Az ötlet számos kutatót vonzott [52, 67, 89, 102].

A determinisztikus lineáris küszöbmodell alapján két új centralitási mutatót javasoltunk a befolyásterjedés maximalizálásához. Ezek a számítás során figyelembe veszik az input hálózat szerkezetét, különösen az élek súlyát, az élek kombinációit és a csúcsok küszöbértékét. Ez a megközelítésünk egyedi abban az értelemben, hogy a centralitási mértékek meghatározásakor ritkán veszik figyelembe a csúcsok küszöbértékét.

A két centralitási mutató alapján olyan csúcsokat választottunk ki, amelyek nagy valószínűséggel jó választások lehetnek a kezdeti időpillanatban (seed csúcsként) a befolyásterjedéshez. Az optimalizáló eljárás aztán kizárólag ezekből választja ki a kezdeti csúcsokat. A centralitási metrikák alkalmazásával hoztuk létre az IAtI algoritmust, amit összehasonlítottunk a mohó algoritmussal és a globális optimummal is. Az IAtI algoritmus lassabb, mint a Mohó, azonban sok esetben jobb megoldást ad, és a legtöbb esetben meg is találja a globális optimumot.

A szűkített halmazból történő választás jelentősen csökkenti a számítási bonyolultságot, így a megoldó algoritmus futási idejét. Ám a módszer jelenlegi formája nem jól skálázható, hiszen a nagy fokszámú csúcsok esetén az élek kombinációinak generálása és kezelése jelentős számítási időt igényel, és emiatt az előfeldolgozás ideje hosszú lehet. Azért vegyük észre, hogy a valós hálózatok többsége ritka gráf, ami hozzájárul a módszer sikerességéhez. Jövőbeni célul azt tűztük ki, hogy egy olyan verziót fejlesszünk, amely képes hatékonyan kezelni a nagy fokszámú csúcsokat tartalmazó gráfokat is.

A disszertáció tézisei

Az **első téziscsoportban** a hozzájárulásaim az I. részhez, a szubmoduláris függvények maximalizálásához kapcsolódik. A részletes bemutatás a 2. és 3. fejezetben található.

I/1. Kifejlesztettem a korlátozó generáló algoritmus azon változatát, amely $k-1$ elemszámú részhalmazokkal dolgozik.

I/2. A korlátozó generáló algoritmus egy másik változatát hoztam létre, a GCG-t, mely egy heurisztikus lépésben kihasználja a gráf szerkezetét.

I/3. Megalkottam a korlátozó generáló algoritmus egy további változatát, az ECG-t, ami közvetlen módon generálja a részhalmazokat.

I/4. Új centralitási metrikát vezettem be, mely kezdőpont választási stratégia a szubmoduláris függvények maximalizálásának megoldásához.

I/5. Bemutattam, hogy az új kezdőpont választási stratégia jobb, mint az általában használt mohó eljárás vagy a nemrég publikált GRASP heurisztika.

A **második téziscsoport** a II. részhez, a befolyásterjedés maximalizálásához kapcsolódik. A részletes bemutatás a 4. és 5. fejezetben található.

II/1. Beláttam, hogy a [65]-ben javasolt egész értékű modell nem helyes minden esetre.

II/2. Bemutattam, hogy a modellt ki kell egészíteni a helyes működéshez.

II/3. Lépésenként igazoltam, hogy az ily módon általunk felállított új modell helyes.

II/4. A helyes modell lépésenkénti megoldását vizsgálva, a befolyásolhatóságra és a befolyásoló képességre két új centralitási metrikát alkottam meg.

II/5. Kimutattam, hogy a két centralitási metrika kombinációját használva, redukálni tudjuk a kezdeti pillanatban lehetséges (seed) csúcsok számát.

Publications

Journal publications

[16] Eszter Csókás and Tamás Vinkó. An exact method for influence maximization based on deterministic linear threshold model. *Central European Journal of Operations Research*, 31:269–286, 2023

[17] Eszter Csókás and Tamás Vinkó. Constraint generation approaches for submodular function maximization leveraging graph properties. *Journal of Global Optimization*, 88:377–394, 2024

[19] Eszter Csókás and Tamás Vinkó. A heuristic for influence maximization under deterministic linear threshold model. *Informatica*, 48(4), 2025

[18] Eszter Csókás and Tamás Vinkó. On the initial set of constraints for graph-based submodular function maximization. *Acta Cybernetica*, 2024

Acknowledgments

> ”Unless the LORD builds the
> house, its builders labor in vain.”
>
> *Bible, Psalm 127:1*

I believe that this dissertation would not have been possible without the Lord's will and help: praise the Lord!

I would like to thank my supervisor, Tamás Vinkó, for sitting down with me on the corridor 5 years ago and for outlining a possible framework for a collaboration. That was the beginning of something that resulted in this dissertation. Thank you for not only showing me the way professionally and helping me through the difficulties of research life, but also for turning to me as a friend. We have had a great journey, and I hope that this dissertation is not the end of it, but just a milestone, and that we will always continue from where we left off.

I would like to thank my colleagues, especially the LANSO research team, both current and past members. I think that the joint seminars and brainstorming sessions have added a lot to the research of all of us professionally. Thank you for being part of my work in this way. Thanks also to my friends, including our church community, whose names I will not even begin to list. They have always been there for me, listening, inspiring, motivating or even encouraging me to take a break when I needed it. Not forgetting the occasions when we supported each other in a joint learning experience.

Last but not least, I would also like to highlight my family. Especially my parents, who never stopped praying for me; my brothers Laci and Tamás; and my twin sister Márti, who never understood what I was doing but memorized it and she is always proud of me. Without you I could not be here, thank you!

Special thanks to my husband Gábor, who was my travelling partner on this trip. You never cheered me on from the shore, you joined me in the boat, through sunny and stormy weather too, and for that I cannot be grateful enough.

Eszter Csókás, 2025.