Modeling and Optimizing for NP-hard Problems in Graph Theory

Ph.D. Thesis

Ahmad Anaqreh

Supervisors: Dr. Boglárka G.-Tóth and Dr. Tamás Vinkó

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

Szeged
2024

Contents

List of Figures

List of Tables

Abbreviations

A Adjacency Matrix 10

B Betweenness Centrality 9

B&B Branch and Bound Algorithm 16, 56

B&C Branch and Cut Algorithm 16, 50, 55, 56

BA Barabási-Albert Model 13, 26, 39, 44

BILP Binary Integer Linear Programming 14, 18, 20, 23, 26, 34, 37, 44, 46, 68, 69, 71

C Closeness Centrality 9

CC Clustering Coefficient 10, 13

CGP Cartesian Genetic Programming 38–42, 73

D Degree Matrix 10

DE Density 9

deg Degree Centrality 9, 10

DFS Depth-First Search Algorithm 56, 57

E Set of Edges 10, 20, 39, 65

ER Erdős-Rényi Model 13, 26, 39, 44

G Graph 10, 20, 21, 23, 39, 65

g(G) Geodetic Number 20, 21, 24

ILP Integer Linear Programming 14–16, 19, 50

L Laplacian Matrix 10, 65, 66, 68

L(S) Grounded Laplacian Matrix 66

LP Linear Programming 13, 15, 16

MILP Mixed Integer Linear Programming 14–16, 51, 56

SR Symbolic Regression 37, 38, 73

V Set of Vertices 10, 20, 21, 24, 39, 65

WS Watts-Strogatz Model 13, 26, 39, 44

Chapter 1

Introduction

Optimization involves the search for the optimal solution for a given problem. It stands as a fundamental concept among diverse disciplines such as mathematics, computer science, engineering, economics, and beyond. Selecting the appropriate optimization technique depends on the problem's category as well as the trade-off between the quality of the solution and the time sufficient for computation.

In combinatorial optimization, recommended approaches to achieve the best possible solution include standard methods like integer linear programming, binary integer linear programming, and linear programming with the relaxation for certain or all variables, yet still achieving an integer solution. Additionally, decomposition methods like Benders decomposition and Dantzig–Wolfe decomposition are efficient for some problem classes. Conversely, if reaching the optimal solution demands significant computational resources then the focus is on attaining a good enough solution within a reasonable time, thus metaheuristics and heuristics gain substantial importance.

Graph theory is the study of graphs, which serve as mathematical structures used to represent and analyze connections between objects. Its practical implementations include diverse domains, including computer science, network science, social science, biology, and more. The connection between optimization and graph theory has long been a significant focal point for researchers, involving the utilization of optimization techniques to deal with graph theory problems.

The primary objective of this thesis is to address three NP-hard problems from graph theory. These problems cannot be solved by deterministic algorithms in polynomial time, making them particularly challenging. The thesis explores different techniques to solve these problems and demonstrates the efficiency of the proposed methods compared to the methods presented in the literature.

1.1 Graphs

1.1.1 Graph Representations

In graph theory, a graph is fundamentally defined as a set of vertices and edges. The vertices can represent diverse types of data, and the edges serve as connections between pairs of vertices, forming a graph topological representation. Simply, a graph is either directed or undirected. Directed graphs have edges with specified directions, while undirected graphs have edges without any particular direction. A simple graph is represented by having, at most, one edge between any two vertices and the absence of any edge that starts and ends at the same vertex.

On the other hand, spectral representation provides an alternative way to represent graphs, which allows the graph to be defined using linear algebra elements, i.e., matrices and vectors, such as Adjacency matrix [82], Degree matrix, and Laplacian matrix [64]. This specific representation holds significant importance in clarifying both the structural and functional characteristics of the graph.

1.1.2 Graph Paths and Cycles

Graph paths and cycles are foundational concepts within the domain of graph theory [19]. A *walk* refers to a sequence of vertices and edges that traverse the graph, permitting revisits to both vertices and edges. If a walk doesn't have repeated edges, it is termed a *trail*. On the other hand, a *path* denotes a walk where no repetition of vertices or edges occurs. Lastly, a *cycle* in the graph is a specific type of path where the initial and final vertices are the same, resulting in a closed loop within the graph. The *length of a path* is defined by the number of edges it contains. Among all the paths between two vertices, the *shortest path* is the one with the minimum number of edges. The graph *diameter* is the length of the longest path among all the shortest paths within the graph.

The study of paths and cycles with specific structures serves as the basis for many graph theory problems that have been studied over the years. An instance of such a problem is the *Eulerian walk*, which involves finding a cyclic walk that traverses each edge precisely once [89]. Another well-known problem is the *Hamiltonian cycle*, which seeks a cycle that visits each vertex exactly once [3]. These problems have practical applications in various fields, making them subjects of great interest in graph theory research.

1.1.3 Graph Measures and Metrics

In graph theory, diverse measures and metrics are available. Some of them have a global scope, providing insights for the entire graph, while others are assigned to individual components like vertices and edges. An essential notion in graph theory is centrality, which emphasizes the importance of graph elements by utilizing a function that assigns values to each element.

Throughout, Let $G = (V,E)$ be a graph with vertex set V and edge set E. *Degree centrality* [92] is the first to mention. This metric describes a vertex's importance based on the number of edges that connect it to other vertices within the graph, which is denoted as deg_v for a given vertex v.

Betweenness centrality (B) [14] measures the proportion of all the shortest paths passing through a particular vertex. To elaborate, let g_{st} be the total number of shortest paths between vertices s and t, and for a vertex i let n^i_{st} be the number of shortest paths between s and t that go through vertex i. The computation of B_i is defined as:

$$B_i = \sum_{s,t \in V, s \neq t} \frac{n^i_{st}}{g_{st}}$$

Closeness centrality [72] computes the average distance from a specific vertex to all other vertices in the graph. Let d_{ij} be the shortest path length between vertex i and all other vertices in the graph, then the concept can be expressed as follows:

$$C_i = \frac{n}{\sum_{j \in V} d_{ij}}$$

Eigenvector centrality [18] is a metric that measures the importance of a vertex by considering the importance of its neighbors. A vertex with high eigenvector centrality indicates that it is connected to vertices with high eigenvector centrality. The computation of eigenvector centrality requires solving the following equation:

$$Ax = \lambda x \quad (x \neq 0)$$

where x is the eigenvector and λ is its corresponding eigenvalue. Thus the centrality of vertex v can be defined as:

$$eig_v = \frac{1}{\lambda} \sum_{t \in M(v)} eig_t$$

where $M(v)$ is the set of the neighbors of v.

Density of a graph can be described as a global measurement to evaluate the ratio between the number of vertices and the number of edges. The average degree of the graph can be outlined as follows:

$$a = \frac{2m}{n}$$

Consequently,

$$DE = \frac{a}{n-1}$$

A graph is classified as sparse when the density is close to zero, whereas it is considered dense when the density is close to one.

Clustering coefficient [15] is an additional metric that estimates the likelihood of two neighbors of a vertex being connected, representing the number of triangles in a graph. It can be calculated using the following formula:

$$CC = \frac{\text{number of closed triplets}}{\text{number of triplets (open and closed)}}$$

In this context, a closed triplet refers to a triangle, whereas an open triplet is a path of length two.

1.1.4 Graph Matrices

For a simple, undirected graph denoted by $G = (V, E)$, where V represents the set of vertices and E represents the set of edges, with $|V| = n$ vertices and $|E| = m$ edges, the *adjacency matrix* A is defined as

$$A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Adjacency matrix is a square matrix in which a non-zero element indicates that the corresponding nodes are adjacent. Implementations of well-known algorithms like Dijkstra's algorithm [48] and Floyd's algorithm [81] usually use the *adjacency matrix* to calculate the shortest paths for a given graph.

The *Degree matrix D* is $n \times n$ diagonal matrix defined as:

$$D_{ij} = \begin{cases} deg_i & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

The *Laplacian matrix L* is defined as follows:

$$L_{ij} = \begin{cases} deg_i & \text{if } i = j, \\ -1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise,} \end{cases}$$

Laplacian matrix is a square matrix that can be used to calculate, e.g., the number of spanning trees for a given graph. The eigenvalues of the *Laplacian matrix* are non-negative, and they do not exceed twice the maximum node degree[11].

1.1.5 Special Graphs

Connected Graph A graph is considered connected when there is at least one path connecting every pair of vertices. On the other hand, a graph is disconnected if certain vertices cannot be reached from other vertices. Figure 1.1 shows a connected graph.

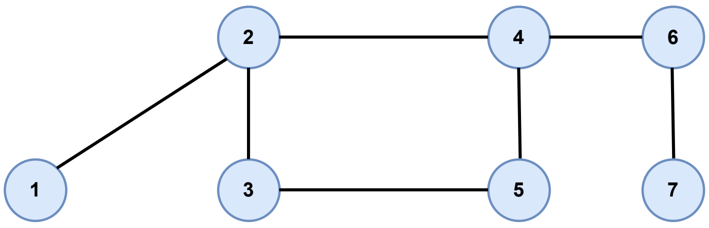


Figure 1.1: *Connected graph*

Regular Graph Within the domain of graph theory, a regular graph indicates a graph in which all vertices have an equal number of neighbors i.e. every vertex has the same degree. If the vertices in the graph have a degree of k, the graph is referred to as a k-regular graph. 2-regular graph shown in figure 1.2

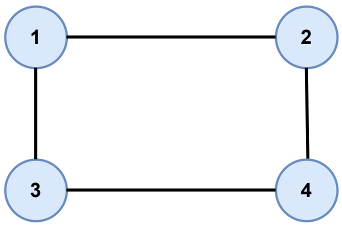


Figure 1.2: *2-regular graph*

Complete Graph A complete graph, as displayed in figure 1.3, is defined as a graph in which each vertex is connected to every other vertex. Consequently, each vertex has a degree of $n-1$, and the overall number of edges in the graph is $\frac{n(n-1)}{2}$

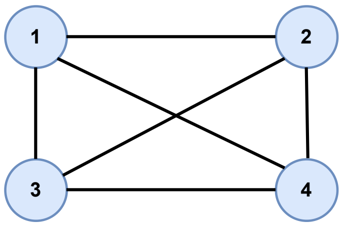


Figure 1.3: *Complete graph*

Acyclic Graph A graph that has no cycles is known as an acyclic graph. When the graph is acyclic and connected, it takes on the name of a tree. Conversely, an acyclic graph that is disconnected is referred to as a forest (a collection of trees). In the tree graph, the number of edges is $n-1$. Figure 1.4 illustrated acyclic graph.

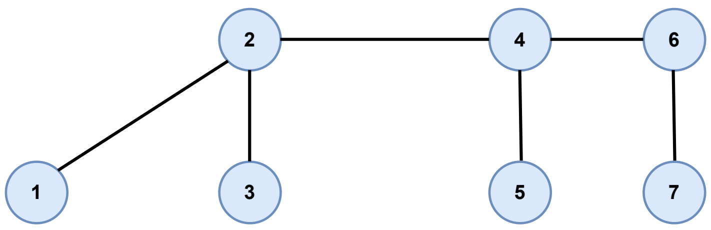


Figure 1.4: *Acyclic graph*

Bipartite Graph In a bipartite graph, the vertices are separated into two disjoint sets, with edges connecting vertices from the different sets. This implies that no edges exist between vertices within the same set. An example of a bipartite graph is displayed in figure 1.5

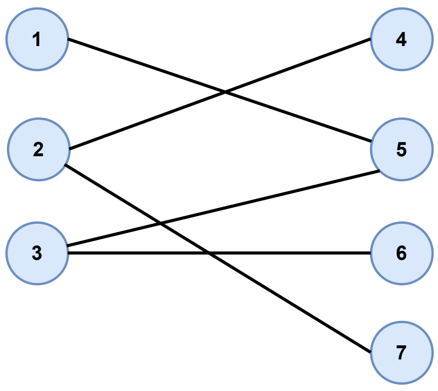


Figure 1.5: *Bipartite graph*

Planar Graph A graph is considered planar if it can be shown on a plane in such a way that none of its edges intersect. a planar graph shown in figure 1.6

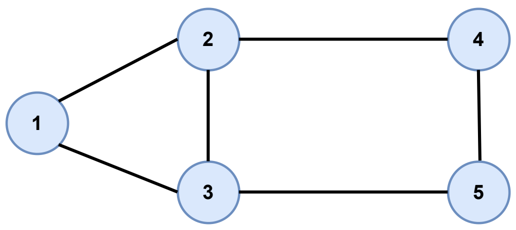


Figure 1.6: *Planar graph*

1.1.6 Random Graphs

Random graph models serve as a fundamental approach for generating graphs. Typically, the characteristics and various graph features can be predicted by creating a graph based on a specific model. There are three well-known models outlined as follows:

Erdős-Rényi (*ER*) model [35], a graph is chosen uniformly at random from the set of all graphs containing n vertices and m edges in case of $G(n,m)$. In the case of $G(n,p)$ the graph is constructed by connecting nodes randomly with each edge included in the graph with probability p independently from every other edge.

Watts-Strogatz (*WS*) model [88] generates graphs with small-world attributes, indicating that the average of all shortest paths in the graph is relatively low and the graph demonstrates a high CC.

Barabási-Albert (*BA*) model [4] constructs graphs using a preferential attachment growing mechanism, where vertices with more connections have a higher probability of receiving new links. As a result, this model exhibits the scale-free property, characterized by a power-law distribution of the form $p_k \sim k^{-\alpha}$, where p_k represents the fraction of vertices with degree k, and α is a parameter typically falling within the range $2 < \alpha < 3$.

1.2 Optimization

1.2.1 Linear Programming

Linear programming (*LP*) stands as the base for many optimization methods and constitutes a field of study that was referenced during the 1930s [49]. A substantial advancement occurred with the introduction of the simplex algorithm by George Dantzig in the 1940s [32], which proved the capability of solving systems of linear inequalities. The standard formulation of a linear program is as follows:

$$\min \ c^T x \tag{1.1}$$

subject to

$$Ax \geq b \tag{1.2}$$

$$x \geq 0 \tag{1.3}$$

Within this context, the decision variables are represented by the n-vector x, A stands for an $m \times n$ matrix, b is a vector of length m, and c forms a vector of length n. The formal definition of the objective function is presented in (1.1), and the set of inequalities marked as (1.2) and (1.3) represents the constraints. The feasible region of a linear program denotes the region containing all potential solutions that satisfy the constraints of the problem. Solving the linear program involves determining values for x within the feasible region, achieving a minimization of the objective function while simultaneously fulfilling the constraints. If no value of x fulfills the constraints $Ax \geq b$ and $x \geq 0$, the problem is classified as infeasible. Conversely, if the constraints do not place bounds on the feasible region, such that the objective can tend to $-\infty$ the problem is described as unbounded.

The linear program in which the decision variables are constrained to integer values is referred to as an Integer Linear Program (*ILP*):

$$\min \ c^T x \tag{1.4}$$

subject to

$$\begin{aligned} Ax &\geq b && (1.5)\\ x &\in \mathbb{Z}^n && (1.6) \end{aligned}$$

A more restricted version, termed Binary Integer Linear Programming (*BILP*), is when the decision variables are restricted to exclusively binary values, either zero or one.

$$\min \ c^T x \tag{1.7}$$

subject to

$$\begin{aligned} Ax &\geq b && (1.8)\\ x &\in \{0,1\}^n && (1.9) \end{aligned}$$

In the case where certain decision variables are required to be integers while others are continuous, the program is denoted as a Mixed Integer Linear Programming (*MILP*).

$$\min \ c^T x + d^T y \tag{1.10}$$

subject to

$$\begin{aligned} Ax + Dy &\geq b && (1.11)\\ x &\geq 0 && (1.12)\\ x &\in \mathbb{Z}^n && (1.13) \end{aligned}$$

1.2.2 Duality

Duality plays a crucial role in the domain of optimization and mathematical programming [13]. In an optimization problem, the purpose is to find the optimal solution—be it maximizing or minimizing the objective function—while fulfilling the constraints. This main problem is known as the primal problem. The dual problem derives from the primal problem, involving the determination of a set of dual variables that correspond to the constraints of the primal problem. The optimal solution of the dual problem specifies bound on the optimal solution of the primal problem. For the following *LP*

$$\min \; c^T x \tag{1.14}$$

subject to

$$Ax \geq b \tag{1.15}$$

$$x \geq 0 \tag{1.16}$$

the dual problem is defined as:

$$\max \; b^T y \tag{1.17}$$

subject to

$$A^T y \leq c \tag{1.18}$$

$$y \geq 0 \tag{1.19}$$

The essence of duality is the duality theorem, which proves a robust connection between the optimal solutions of the primal and dual problems. To study the concept, let's consider the primal and dual problems as defined previously:

Theorem 1 (Weak Duality Theorem) *If x is a primal feasible solution and y is a dual feasible solution for primal and dual problems respectively,*

- *If equality holds in $c^T x \geq b^T y$ then x is primal optimal solution and y is dual optimal solution.*
- *If either primal or dual is unbounded, then the other problem is infeasible.*

Theorem 2 (Strong Duality Theorem) *If x is a primal feasible solution and y is a dual feasible solution for primal and dual problems respectively,*

- *x is primal optimal solution and y is dual optimal solution such that $c^T x = b^T y$.*
- *If either primal or dual is infeasible, then the other problem is either unbounded or infeasible.*

1.2.3 Cutting Plane

The cutting plane is a technique employed to enhance the efficiency of solving *ILP* and *MILP* problems. This method was initially introduced by Gomory [42]. The approach revolves around solving the problem, and when the solution is not an integer, a valid constraint is added to the problem. This constraint is satisfied by all feasible integer solutions but is violated by the current non-integer solution. By combining the cut into the problem and resolving it, a new solution is obtained. This iterative process is repeated until an integer solution is attained. Gomory demonstrated that this method terminates within a finite number of steps. Thus, Gomory's cuts are applied to cut down the search space and cut off the non-integer solutions while preserving the integer solution.

1.2.4 Branch-and-Bound

The Branch and Bound algorithm (*B&B*) [30] stands as a notably efficient optimization method primarily utilized for handling *ILP* and *MILP* problems. It adopts a divide-and-conquer approach to partition the search space of the initial problem into sub-problems with smaller search space, consequently constructing a search tree.

The initial phase of the *B&B* begins with the solution of the root node located at the top of the search tree. At this stage, all variables are relaxed, assuming continuous values. If the relaxation of the root node generates an infeasible solution, it implies the original problem is similarly infeasible. Conversely, if the linear relaxation of the root node generates an optimal solution where all integer variables have integer values, then this solution is optimal for the original problem.

However, if the solution resulting from the relaxation includes at least one non-integer variable, branching is employed. Through branching, two sub-problems are generated, each with an extra constraint. Let x_i^* be the solution from the relaxation, the new constraints take the form of $x_i \leq \lfloor x_i^* \rfloor$ in the first sub-problem and $x_i \geq \lceil x_i^* \rceil$ in the second sub-problem. These sub-problems are subsequently tackled recursively.

The efficacy of the *B&B* relies on its utilization of the pruning technique. For instance, in a minimization problem, solutions obtained from the relaxations serve as lower bounds, while integral solutions serve as upper bounds. As a result, a sub-problem can be disregarded if its solution is equal to or greater than the upper bound. This mechanism simplifies the search procedure and effectively narrows down the search space of feasible solutions.

1.2.5 Branch-and-Cut

Branch-and-Cut algorithm (*B&C*) [68] is an enhanced variant of the *B&B*, representing a hybrid approach that combines the *B&B* with cutting planes. The primary objective is to attain better linear relaxation solutions, i.e. to achieve a superior bound, by getting more efficient pruning of the search space using the cuts.

In this algorithm, a search tree is constructed similarly to the *B&B*. On the other hand, a cut is added to the problem to limit the solution domain for the *LP* relaxation problems while preserving valid integer solutions. By including these cuts, significant performance improvements can be achieved, as they reduce the number of branches that need to be explored in the search tree.

1.2.6 Heurstics and Metaheuristics

The term heuristic derives from the Greek word ”heuriskein” meaning ”to discover” [91]. In this context, a heuristic approach suggests methodologies intended to facilitate problem-solving. Consequently, heuristic optimization is applied to find approximate solutions without guaranteeing the achievement of the optimal solution, but rather aiming to ensure a good solution within a reasonable time. Heuristics are recommended when no reliable exact method exists to solve the problem, or when an exact method exists but proves computationally intensive, demanding substantial time and/or storage resources. Furthermore, heuristics can be utilized to provide initial solutions for solvers. Well-known heuristic strategies encompass greedy algorithms [84] and hill climbing [29]. These techniques are especially beneficial for problems with specific structures.

Similar to heuristic, metaheuristics optimization [1, 16] constitutes an approach to problem-solving wherein algorithms are designed and implemented to achieve approximative solutions for complicated optimization problems. Metaheuristics emphasizes two strategies: exploiting the current search region for the most promising solution and exploring new search regions to efficiently find solutions. These methodologies are adaptable and applicable to a diverse spectrum of optimization problems, including combinatorial optimization, continuous optimization, and multi-objective optimization. Some common examples of metaheuristics optimization algorithms include Genetic Algorithms [5], Ant Colony Optimization [33], Grey Wolf Optimization [67], and an expanding array of algorithms classified as nature-inspired due to their simulation of mechanisms observed in the nature.

The primary distinction between heuristics and metaheuristics lies in their problem dependency. Heuristics are highly designed for specific problems, enabling efficient solutions for those particular cases but often proving to be not good enough for other problems. On the opposite, metaheuristics are conducted as general-purpose algorithms that can be applied effectively to almost all types of optimization problems, irrespective of their specific characteristics.

1.3 Contributions

The primary purpose of this thesis is to tackle three NP-hard graph problems by utilizing various optimization techniques, including standard methods, heuristics, and metaheuristics. The evaluation of the proposed methods involved comparing their results and execution times with those presented in the literature. The concepts, figures, tables, and results presented in this thesis have been previously published in scientific papers. In summary, the contributions can be outlined as follows:

Chapter 2.: I introduced novel algorithmic approaches for obtaining upper bounds of the graph geodetic number, which is known to be an NP-hard problem. To demonstrate the efficiency of these algorithms, I conducted experiments on a diverse set of graphs with varying structures. Through a comparison of the results with the *BILP* formalism from recent literature, I illustrate that my approaches outperform the *BILP* in terms of computational speed while maintaining a sufficiently precise upper bound relative to the optimal solution.

Chapter 3.: I proposed an approach that combines Symbolic Regression with an evolutionary algorithm known as Cartesian Genetic Programming, to formulate equations capable of approximating the graph geodetic number. To evaluate the efficacy of these equations, I performed assessments on both randomly generated and real-world graphs. The results demonstrated that the derived equations showed a reasonable approximation in comparison to the optimal solution.

Chapter 4.: I introduced three integer linear programs to attain optimal solutions for the longest induced cycle problem, a well-known graph problem categorized as NP-complete. The numerical efficiency of these proposed approaches was validated through experiments conducted on real-world graphs and compared to the methods proposed in the literature.

Chapter 5.: I presented two methods to tackle the NP-hard problem of maximizing the smallest eigenvalue of the grounded Laplacian matrix. Degree centrality is used as the base for the first method. In addition, the vertex cover problem was employed as an additional method of solving the problem. To evaluate the efficiency of the proposed methods, I executed experiments on real-world graphs, comparing the obtained solutions and the execution times with the methods proposed in the literature.

Chapter 2

Algorithmic Upper Bounds for Graph Geodetic Number

2.1 Introduction

Path-covering problems hold significant value both in theory and practical applications due primarily to their straightforward interpretability. These problems encompass various complicated notions of coverage, with one notable subset revolving around the determination of shortest paths. Within this problem domain lies the concept of the graph geodetic number, which is a global measure for simple connected graphs to find the minimal-cardinality set of vertices, such that all shortest paths between its elements cover every vertex of the graph. Geodetic number problem initially introduced in the work by Harary [46]. This problem finds applications in several fields, with one particular example outlined in Manuel's work [59], where it is framed as a social network problem.

Notably, it has been shown that computing the geodetic number is a computationally challenging task for general graphs, as it falls into the NP-hard problem category [12]. As is often the case with graph theoretical problems, an *ILP* formulation is viable. Hansen and van Omme presented such a model in a recent paper [45], which also featured the first computational experiments conducted on a collection of moderately sized random graphs.

Chakraborty *et al.* [25] proposed an algorithm for approximating the geodetic number on edge color multigraph. An efficient polynomial algorithm to compute the geodetic number of interval graphs was presented in [34].

Geodetic sets and geodetic numbers have diverse applications. They find utility in computational sociology, as suggested in [22, 86]. The concept of the convexity of a set of vertices in a graph, as defined in [47], represents a somewhat converse property to geodetic sets. Related concepts include the graph hull number [36] and the domination number [44], all of which have practical applications. For instance, they are employed in public transportation design [25], achievement and avoidance games [24], location problems [75], maximizing switchboard numbers in telephone tree graphs [69], mobile ad hoc networks [76], and the design of efficient typologies for parallel computing [73].

Drawing inspiration from these results, this work, which is published in [9], conducts an empirical investigation into upper bound algorithms. These algorithms, as demonstrated by the experiments, consistently produce results with minimal gaps on diverse sets of graphs. Notably, these methods require relatively low computational time, even when applied to random graphs comprising 150 vertices, as well as on large-scale real-world graphs.

2.1.1 Problem description

A simple connected graph is denoted by $G = (V, E)$. Assume that $n = |V|$ and $m = |E|$. Given $i, j \in V$, the set $I[i, j]$ contains all $k \in V$ which lies on any shortest path (*geodetics*) between i and j. The union of all $I[i, j]$ for all $i, j \in S \subseteq V$ is denoted by $I[S]$, which is called *geodetic closure* of $S \subseteq V$. Formally

$$I[S] := \{k \in V : \exists i, j \in S, k \in I[i, j]\}.$$

The *geodetic set* is a set S for which $I[S] = V$. The *geodetic number* of G is

$$g(G) := \min\{|S| : S \subseteq V \text{ and } I[S] = V\}.$$

2.1.2 Binary integer linear programming model

In the paper by Hansen *et al.* [45], a *BILP* formulation is introduced as follows. They use d_{uv} to represent the length of the shortest path between any two vertices u and v in the graph, where $u, v \in V$.

For each vertex $k \in V$, they define the set P_k as:

$$P_k := \{(i, j) \in V \times V \mid d_{ik} + d_{kj} = d_{ij}\}.$$

Here, P_k contains all pairs of vertices for which the shortest path goes through vertex k.

The model is formulated as follows:

$$\min \sum_{k=1}^{n} x_k \tag{2.1}$$

subject to

$$1 - x_k \le \sum_{(i,j) \in P_k} y_{ij} \qquad \forall\, k, i, j \in V, i < j \tag{2.2}$$

$$y_{ij} \le x_i \qquad \forall\, i, j \in V, i < j \tag{2.3}$$

$$y_{ij} \le x_j \qquad \forall\, i, j \in V, i < j \tag{2.4}$$

$$x_i + x_j - 1 \le y_{ij} \qquad \forall\, i, j \in V, i < j \tag{2.5}$$

$$x_i \in \{0,1\} \quad \forall i \in V \tag{2.6}$$

$$y_{ij} \in \{0,1\} \quad \forall i,j \in V, i<j \tag{2.7}$$

The variable x_k serves as an indicator of whether vertex k belongs to the set S or not. Consequently, the objective is to minimize the sum of these binary variables. The auxiliary variables y_{ij} represent the bilinear term $x_i x_j$. Thus, in equation (2.2), it is implied that x_k can only be 0 if there exist vertices i and j in set S such that vertex k is part of their shortest path. The McCormick conditions, as detailed in equations (2.3)-(2.5), establish a correspondence $y_{ij} = x_i x_j$.

2.2 Upper bounds

The most straightforward upper bound for the geodetic number is $g(G) \leq n$, a bound that holds precisely for complete graphs. Chartrand *et al.* [26], establish a more refined upper bound, $g(G) \leq n-d+1$, where d represents the diameter of graph G. Additional upper bounds are detailed in [21, 83, 86], although these are specific to certain graph structures. The aim is to derive upper bounds applicable to general graphs through algorithmic methods. This pursuit leads to the development of two upper bound algorithms: the first one uses Floyd's algorithm [81], while the second algorithm is based on Dijkstra's algorithm [48].

2.2.1 Greedy algorithm

The core concept behind this algorithm is to construct a geodetic set in an iterative greedy manner. At each iteration, the algorithm selects a vertex, denoted as i, to be added to set S that leads to the maximum increase in $I[S]$. In other words, it chooses the vertex for which the cardinality $\mid I[S \cup \{i\}] \setminus I[S] \mid$ is maximal.

Initialization. The initialization phase is shown in Algorithm 1. In lines 4−5, the algorithm computes all-pair shortest paths, which are subsequently utilized in lines 6−8 to define sets denoted as I_{ij}. These sets, at this stage, are essentially the same as $I[i,j]$, which was previously introduced in Section 2.1.1. In other words, they contain all vertices on any shortest path between vertices i and j. A distinct notation is employed here intentionally, as the sets I_{ij} may change later in the algorithm. In line 9, vertices with a degree of at most one are placed into the initially empty geodetic set S. A vertex with a degree of one is the vertex that is connected by a single edge in the graph, and these vertices are required to be part of the geodetic set S, as it is established in [46]. The geodetic closure $I[S]$ is initialized for the set S in lines 10−11, though it is important to note that this set could potentially be empty at this stage. Finally, in lines 12−13, all vertices that have already been covered are removed from the sets I_{ij}.

Algorithm 1: Greedy algorithm - Initialization

```
1  Function GreedyInit
2      $S = \emptyset, I[S] = \emptyset, I_{ij} = \emptyset \quad \forall i,j \in V$
3      $d_{ij} = 1 \quad \forall (i,j) \in E,\ d_{ij} = \infty \quad \forall (i,j) \notin E$
4      for $\forall k \in V, i \in V, j \in V$ do                        // distance calculation
5          $d_{ij} = \min\{d_{ij}, d_{ik} + d_{kj}\}$                      // by Floyd algorithm
6      for $\forall i \in V, j \in V, k \in V$ do                         // calculation of $I_{ij}$
7          if $d_{ij} = d_{ik} + d_{kj}$ then
8              $I_{ij} = I_{ij} \cup \{k\}$
9      $S = \{k \in V \mid \deg_k \leq 1\}$           // unreachable vertices must be in S
10     for $\forall i \in S, j \in S$ do                                    // build I[S] for S
11         $I[S] = I[S] \cup I_{ij}$
12     for $\forall i \in V, j \in V$ do                          // Update $I_{ij}$-s by removing
13         $I_{ij} = I_{ij} \setminus I[S]$                                  // all covered vertices
```

Auxiliary functions. The description of the greedy approach is extended in Algorithm 2, where the functions `LargestIncrease` and `LargestIncreasePair` are introduced. These functions are responsible for determining the vertex (or vertex pair) that if they are included in set S, resulting in the most expansion of the covered set $I[S]$. Two sets of notations are utilized: the sets $I_i[S]$ encompass vertices that would be covered if vertex i were included in set S, and sets $I_{ij}[S]$ consist of vertices that would be covered if both vertices i and j were included in set S. The initialization of sets $I_i[S]$ as empty occurs in line 16. Subsequently, in lines $15-18$, these sets $I_i[S]$ are constructed based on the vertices currently present in I_{ij}, where vertex i is not a member of set S while vertex j is part of set S. Further, in line 19, the variable ℓ is defined as the vertex from the set $V \setminus S$ for which $I_i[S]$ is maximized. Similarly, within the function `LargestIncreasePair(V,S,I)`, lines $21-22$ are responsible for computing the sets $I_{ij}[S]$ using vertices that would yield the greatest expansion of $I[S]$ if the pair (i,j) included. Lastly, in line 23, a pair of vertices is selected such that the set $I_{ij}[S]$ is the largest among the options.

Main algorithm. The primary loop of the greedy approach is presented in Algorithm 3. The condition outlined in line 27 verifies if there are any uncovered vertices remaining. In lines $28-33$, a heuristic rule is applied, which involves a straightforward check to see if the size of set $I_\ell[S]$ is at least half the size of set $I_{kh}[S]$. This constitutes a greedy choice, although it is worth noting that alternative conditions could also be employed to determine whether to add a single vertex or a pair of vertices to set S. Subsequently, in lines $34-35$, the sets I_{ij} are updated by removing all the covered vertices from them. In line 36, the selection of the vertex that results in the greatest expansion of $I[S]$ needs to be chosen again. Finally, the execution of lines $37-40$ depends on the value of the parameter *AddOne*. If it is set to

Algorithm 2: Greedy algorithm - Auxiliary functions

```
14 Function LargestIncrease($V,S,I$)
15     for $\forall i \in V \setminus S$ do                // compute $I_i[S]$, the set increasing
16         $I_i[S] = \emptyset$                              // $I[S]$ if $i$ is included
17         for $\forall j \in S$ do
18             $I_i[S] = I_i[S] \cup I_{ij}$
19     $\ell = \operatorname{argmax}_{i \in V \setminus S} |I_i[S]|$      // find vertex for which $I[S]$ would grow most
       return $\ell, I_\ell[S]$
20 Function LargestIncreasePair($V,S,I$)
21     for $\forall i \in V \setminus S, j \in V \setminus S$ do     // compute $I_{ij}[S]$, the set increasing
22         $I_{ij}[S] = I_{ij} \cup I_i[S] \cup I_j[S]$            // $I[S]$ if $i,j$ are included
23     $(k,h) = \operatorname{argmax}_{i,j \in V \setminus S} |I_{ij}[S]|$ // pair of vertices which $I[S]$ would grow most
       return $k,h,I_{kh}[S]$
```

$AddOne = 0$, then the algorithm chooses the best pair of vertices to add to set S by invoking the `LargestIncreasePair` function. In contrast, when $AddOne = 1$, this function call is skipped, and the adjustments made in line 38 are applied to ensure consistency.

It is important to note that, since the input graph G is undirected, similar to the description of the *BILP* in Section 2.1.2, one can safely assume the condition $i < j$ for all relevant cases. In practice, this assumption is employed in the actual implementation of the greedy algorithm to enhance its efficiency. However, these technical details have been omitted for the sake of simplicity in understanding the pseudocodes.

Computational complexity

The greedy heuristic employs Floyd's algorithm to compute distances in the input graph, which requires $\mathcal{O}(n^3)$ time. The construction of I_{ij} involves nested loops, with a complexity of $\mathcal{O}(n^3)$. Calculating $I[S]$ and updating I_{ij} both have a time complexity of $\mathcal{O}(n^2)$.

Within the main loop of the algorithm, there are nested loops. Initially, there is an outer loop that checks if there are any non-empty $I_\ell[S]$ remaining, which can iterate up to n times in the worst case. Subsequently, the inner loop for updating all I_{ij} has a time complexity of $\mathcal{O}(n^2)$. Both auxiliary functions used to identify the vertex or vertices that maximize the growth of $I[S]$ essentially involve two loops, resulting in a time complexity of $\mathcal{O}(n^2)$ for each function. Considering the combination of the outer and inner loops, the overall time complexity of this part is $\mathcal{O}(n^3)$. Consequently, the computational complexity of the greedy algorithm is $\mathcal{O}(n^3)$.

Algorithm 3: Greedy algorithm - Main

```
24 GreedyInit                                        // initialize $I_{ij}$ sets and $S$
25 $[\ell, I_\ell[S]] =$ LargestIncrease($V,S,I$)       // $\ell$ would make $I[S]$ grow most
26 $[k,h,I_{kh}[S]] =$ LargestIncreasePair($V,S,I$)     // $k,h$ make $I[S]$ grow most
27 while $|I_\ell[S]| + |I_{kh}[S]| > 0$ do              // the set is not geodetic yet
28     if $|I_\ell[S]| > |I_{kh}[S]|/2$ then  // balance adding one or two vertices to $S$
29         $S = S \cup l$
30         $I[S] = I[S] \cup I_\ell[S]$                                  // update $I[S]$
31     else
32         $S = S \cup \{k,h\}$
33         $I[S] = I[S] \cup I_{kh}[S]$                                 // update $I[S]$
34     for $\forall i \in V, j \in V$ do                   // Update $I_{ij}$-s by removing
35         $I_{ij} = I_{ij} \setminus I[S]$                         // all covered vertices
36     $[\ell, I_\ell[S]] =$ LargestIncrease($V,S,I$)                  // recompute $I_\ell[S]$
37     if $AddOne$ then                          // $AddOne$ is a control parameter
38         $k = h = 0; I_{kh}[S] = \emptyset$
39     else
40         $[k,h,I_{kh}[S]] =$ LargestIncreasePair($V,S,I$)        // recompute $I_{kh}[S]$
```

2.2.2 Locally greedy algorithm

The locally greedy algorithm serves the same purpose as previously described, which is to efficiently find an upper bound on $g(G)$. Moreover, it aims to achieve speed improvements compared to the method introduced in Section 2.2.1 by utilizing only local information. Instead of calculating all shortest paths using Floyd's algorithm, this approach calculates distances from a specific vertex to all vertices not yet included in the geodetic set S using Dijkstra's algorithm.

The specifics of the locally greedy algorithm are clarified in Algorithm 4. This algorithm takes vertex v as input, which can be either a degree-one vertex, as discussed in Section 2.2.1, or a simplicial vertex. A simplicial vertex is defined as a vertex whose neighbors collectively form a clique, meaning that every pair of neighbors are adjacent. It has been proved in [2] that simplicial vertices are always part of the geodetic set.

Vertex v serves as the initial element of set S. The `LargestLocalIncrease` function, as indicated in line 42, returns the vertex u that would maximize the growth of set $I[S]$. This function is further detailed in lines 53-59. Initially, it calculates the distances from vertex v to all other vertices in the graph, along with the corresponding shortest paths, and fills the sets $I_{v,:}$. Dijkstra's algorithm, a well-known algorithm, is used for this purpose, although its detailed implementation is not provided in the pseudocode. Subsequently, in lines 55-57, the function computes the sets $I_j[S]$ for all $j \in V \setminus S$. In line 58, the function identifies the vertex u whose inclusion in S would result in the greatest expansion of $I[S]$.

In lines 43-45, the algorithm adds vertex u to the geodetic set S, updates $I[S]$, and then removes $I[S]$ from the set R, which represents the remaining vertices yet to be covered. The main loop of the algorithm, as described in lines 47-52, checks in each iteration whether set R is empty, indicating if there are still any uncovered vertices. As long as there are uncovered vertices remaining, the algorithm will continue to execute, calling the `LargestLocalIncrease` function on vertex w in line 48.

Algorithm 4: Locally Greedy algorithm

```
Input: $v$ a degree-one or simplicial vertex
41 $R = V, S = v, I[S] = \emptyset, I_i[S] = \emptyset \quad \forall i \in V, I_{ij} = \emptyset \quad \forall i,j \in V$
42 $[u, I_u[S]] = $ LargestLocalIncrease$(v, V, S, I)$   // $I[S]$ would grow most for $u$
43 $S = S \cup \{u\}$                                          // update $S$
44 $I[S] = I[S] \cup I_u[S] \cup \{v, u\}$                          // update $I[S]$
45 $R = R \setminus I[S]$                 // update $R$ by removing covered vertices
46 $w = u$
47 while $|R| > 0$ do                            // the set is not geodetic yet
48     $[u, I_u[S]] = $ LargestLocalIncrease$(w, V, S, I)$          // compute $I_u[S]$
49     $S = S \cup \{u\}$                                      // update $S$
50     $I[S] = I[S] \cup I_u[S] \cup \{u\}$                          // update $I[S]$
51     $R = R \setminus I[S]$              // update R by removing covered vertices
52     $w = u$
53 Function LargestLocalIncrease$(v, V, S, I)$
54     $I_{v,:} = $ Dijkstra$(v)$          // shortest paths by Dijkstra algorithm
55     for $\forall j \in V \setminus S$ do
56         for $\forall i \in S$ do
57             $I_j[S] = I_j[S] \cup I_{ij}$
58     $u = \operatorname{argmax}_{j \in V \setminus S} |I_j[S] \setminus I[S]|$ // find the vertex which $I[S]$ would grow most
59     return $u, I_u[S]$
```

Computational complexity

The locally greedy algorithm employs Dijkstra's algorithm to calculate distances within the input graph, which operates in $\mathcal{O}(n^2)$ time. The construction of $I_j[S]$ using nested loops has a complexity of $\mathcal{O}(n^2)$. Consequently, the `LargestLocalIncrease` function, taken as a whole, requires $\mathcal{O}(n^2)$ time.

The primary while loop of the algorithm serves to fill the geodetic set S by invoking the `LargestLocalIncrease` function repeatedly until set R becomes empty. This loop iterates at most n times. Hence, the total computational complexity of the locally greedy algorithm is $\mathcal{O}(n^3)$, which is consistent with the computational complexity of the other greedy methods proposed in Section 2.2.1.

2.3 Numerical experiments

To assess the performance and efficiency of the upper bound algorithms discussed in Section 2.2, both the *BILP* and the algorithms were implemented in AMPL [38]. Gurobi [43] served as the solver, with parameters configured as `mipfocus=1` and `timelim=3600`. The experiments were conducted on a computer with a 3.10 GHz i5-2400 CPU and 8GB of memory.

The obtained results are presented in Tables 2.1, 2.2, 2.3, and 2.4. The columns in these tables give the following information:

graph: Indicates the size of the graphs, specified by the number of vertices (n) and the number of edges (m);

exact: Shows the exact geodetic number (or best solution found by Gurobi in case of running out of time) and the time taken in seconds to find this solution;

greedy: Displays the upper bound determined by the greedy algorithm (i.e., Algorithm 3) using $AddOne = 0$, along with its execution time in seconds;

greedy (`AddOne`): Presents the upper bound identified by the greedy algorithm (Algorithm 3) when $AddOne = 1$, i.e., the addition of only one vertex at a time. The execution time in seconds is also included;

locally greedy: Represents the upper bound founded by the locally greedy algorithm (Algorithm 4) and the time taken in seconds to compute it.

2.3.1 Graph instances

The algorithms were provided with a variety of input graphs, including randomly generated graphs with diverse structural characteristics. This approach aimed to gain a deeper understanding of how the solution to the geodetic number problem is influenced by the graph structure. Additionally, a limited selection of real-world graphs was included in the benchmark for evaluation.

Random graphs

The random graphs were generated by using the standard models: *ER* model [35], *WS* model [88] and *BA* model [4].

Regarding the number of vertices and edges the following approaches were used:

- the number of vertices were $n = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100$, and
- the number of edges followed the scheme as in [45]:
  - for each case one can have maximum $\frac{n(n-1)}{2}$ edges,

  - and 20%, 40%, 60%, and 80% of the maximum number of edges has been taken.

- Apart from these graphs, some bigger graphs were created with $n = 115, 135, 150$ vertices, using the same procedure as above with the only difference that 25%, 50%, and 75% of the maximum number of edges were taken.

Real-world graphs

The graphs utilized in this study are considered benchmark graphs. The first set of graphs was sourced from the datasets available in the UCINET software[1], while the other graphs are well-known graphs from the Network Repository[2].

2.3.2 Discussion on results with random graphs

General observations

Before the discussion of the specific results for different types of random graphs, a general overview can be summarized as follows.

- In less than half of the cases, the exact solutions were obtained, mainly due to time constraints. In these instances, the reported solution serves as a lower bound for the geodetic number.

- Both versions of the greedy algorithm (Algorithm 3) consistently completed their execution in less than 2 seconds for every tested random graph. Their execution times were generally similar. The version considering pairs of vertices (`AddOne=0`) typically provided a superior or equal upper bound compared to the version considering a single vertex (`AddOne=1`).

- The locally greedy algorithm (Algorithm 4) showed remarkably fast execution times, typically taking less than 0.2 seconds for each tested graph. On average, this algorithm proved to be the fastest among the options.

- Unsurprisingly, as the number of vertices increased, both the geodetic number and its computation time also increased (for graphs with the same density). When considering averages for graphs with the same density, the geodetic number tended to decrease as density increased. The computational effort was most extensive when the density was 0.4 and minimal when it was 0.8.

[1]`https://sites.google.com/site/ucinetsoftware/datasets/`
[2]`http://networkrepository.com/`

Erdős-Rényi random graphs

The results for the Erdős-Rényi graphs are presented in Table 2.1. The solver successfully found the optimal solution within the time limit for 22 instances, which is 45% of the cases.

The greedy algorithm failed to obtain the optimal solution in 28 cases. Consequently, it reported the optimal value as an upper bound in 43% of the cases. For 20 graphs, it missed the optimum by just one additional vertex, and for 7 graphs, it fell short by two more vertices in the minimal geodetic set. Conversely, it managed to find a superior solution in one instance compared to the upper bound reported by Gurobi. In comparing the two versions of the greedy algorithm, they yielded the same values in 38 cases. `AddOne=1` provided better upper bounds for only 2 input graphs, while the default version outperformed it in 9 cases.

The locally greedy algorithm reported the optimal solution in 22 cases, amounting to 45% of the cases. In 16 cases, the upper bound missed the optimal solution by one additional vertex, in 6 cases by two more vertices, and in 5 cases by three or more vertices.

When comparing the greedy algorithm with the locally greedy algorithm, the two methods reported identical upper bounds in 32 cases. In 7 cases, the locally greedy algorithm provided superior upper bounds, whereas in 10 cases, the greedy algorithm was closer to the optimal solution.

The final row in Table 2.1 displays the mean values for the obtained geodetic number bounds as well as the average execution times. Although the differences are minimal, the greedy algorithm excels in achieving robust upper bounds, while the locally greedy approach stands out as the fastest method for Erdős-Rényi random graphs.

Table 2.1: *Numerical results for the Erdős-Rényi random graphs, time is given in seconds. The best results are highlighted.*

graph		exact		greedy		greedy (`AddOne`)		locally greedy	
n	m	value	time	value	time	value	time	value	time
10	9	4	0.004	4	0.004	4	0.003	4	0.001
10	18	4	0.004	4	0.004	4	0.004	4	0.004
10	30	4	0.007	4	0.002	4	0.002	4	0.002
10	36	3	0.011	4	0.004	4	0.004	3	0.004
20	37	5	0.063	5	0.009	5	0.008	6	0.004
20	76	4	0.073	4	0.011	4	0.011	4	0.006
20	114	4	0.767	5	0.01	5	0.011	5	0.007
20	152	3	0.197	3	0.011	3	0.012	3	0.006
30	87	6	0.906	7	0.018	8	0.017	7	0.011
30	174	6	3.258	8	0.019	8	0.019	6	0.013
30	261	4	2.198	5	0.016	5	0.015	5	0.011
30	348	4	3.259	4	0.02	4	0.018	4	0.009

Table 2.1: *(continued) Numerical results for the Erdős-Rényi random graphs, time is given in seconds. The best results are highlighted.*

graph		exact		greedy		greedy (`AddOne`)		locally greedy	
n	m	value	time	value	time	value	time	value	time
40	156	7	76.422	7	0.03	7	0.027	9	0.011
40	312	6	89.274	8	0.031	7	0.031	7	0.016
40	468	4	17.424	6	0.039	6	0.037	6	0.011
40	624	3	4.086	4	0.036	4	0.037	4	0.013
50	245	7	208.332	9	0.056	9	0.051	10	0.023
50	490	6	1169.8	7	0.059	8	0.052	8	0.018
50	735	5	832.257	6	0.068	6	0.059	5	0.014
50	1000	3	18.231	4	0.061	4	0.057	4	0.016
60	354	≤ 9	> 3600	10	0.092	11	0.083	11	0.028
60	708	≤ 7	> 3600	8	0.106	8	0.096	8	0.025
60	1062	≤ 5	> 3600	6	0.111	6	0.097	5	0.024
60	1416	4	429.412	4	0.104	4	0.101	4	0.016
70	483	≤ 8	> 3600	10	0.131	10	0.121	12	0.033
70	966	≤ 7	> 3600	9	0.164	9	0.139	9	0.038
70	1449	≤ 5	> 3600	6	0.156	7	0.152	6	0.031
70	1932	4	663.129	4	0.147	4	0.142	4	0.025
80	632	≤ 9	> 3600	9	0.193	10	0.181	13	0.048
80	1264	≤ 8	> 3600	9	0.235	10	0.208	9	0.047
80	1896	≤ 6	> 3600	6	0.242	6	0.215	6	0.039
80	2528	≤ 4	> 3600	4	0.225	4	0.214	4	0.029
90	801	≤ 10	> 3600	10	0.277	13	0.261	15	0.638
90	1602	≤ 8	> 3600	9	0.331	10	0.295	9	0.052
90	2403	≤ 5	> 3600	6	0.347	6	0.316	5	0.041
90	3204	≤ 4	> 3600	5	0.319	5	0.302	5	0.041
100	990	≤ 12	> 3600	11	0.375	14	0.348	15	0.076
100	1980	≤ 9	> 3600	9	0.449	9	0.387	9	0.061
100	2970	≤ 6	> 3600	6	0.479	6	0.423	6	0.047
100	3960	≤ 4	> 3600	4	0.433	4	0.425	4	0.038
115	1638	≤ 13	> 3600	15	0.627	15	0.572	14	0.101
115	3277	≤ 7	> 3600	8	0.746	8	0.648	8	0.074
115	4916	≤ 5	> 3600	5	0.704	5	0.672	5	0.068
135	2261	≤ 14	> 3600	15	0.943	14	0.823	16	0.152
135	4522	≤ 8	> 3600	8	1.107	8	0.953	8	0.107
135	6783	≤ 4	> 3600	5	1.101	5	1.042	5	0.088

Table 2.1: *(continued) Numerical results for the Erdős-Rényi random graphs, time is given in seconds. The best results are highlighted.*

graph		exact		greedy		greedy (`AddOne`)		locally greedy	
n	m	value	time	value	time	value	time	value	time
150	2793	≤ 15	> 3600	16	1.393	16	1.154	16	0.182
150	5587	≤ 8	> 3600	8	1.528	8	1.356	8	0.134
150	8381	≤ 5	> 3600	5	1.563	5	1.465	5	0.107
average		6.224	2055.492	6.898	0.309	7.122	0.279	7.184	0.053

Watts-Strogatz random graphs

The results for the Watts-Strogatz graphs are detailed in Table 2.2. In this case, the exact method reached its time limit in 23 instances, solving the problem for only 53% of the cases.

The greedy algorithm produced the same result as the exact method in 29% of the cases, specifically for 14 graphs. When it did not match the optimal (or best-reported) value, it provided a +1 difference for 25 graphs and a +2 difference for 10 graphs.

Among the two versions of the greedy algorithm, 38 cases resulted in identical values. `AddOne=1` yielded superior upper bounds in only one instance, while the default version outperformed it 10 times.

The locally greedy algorithm found the optimal solution in 8 cases. In 22 cases, the upper bound missed the optimal solution by just one additional vertex, in 12 cases by two more vertices, and in 7 cases by three or more vertices.

When comparing the performance of the greedy algorithm and the locally greedy algorithm, they reported the same upper bound in 29 cases. The locally greedy algorithm provided better upper bounds in 3 cases, while the default version yielded better results in 17 cases.

Regarding average performance, as reported in the last row of Table 2.2, the default greedy algorithm was the best in providing superior upper bounds, whereas the locally greedy approach proved to be the fastest for Watts-Strogatz random graphs.

Table 2.2: *Numerical results for the Watts-Strogatz random graphs, time is given in seconds. The best results are highlighted.*

graph		exact		greedy		greedy (`AddOne`)		locally greedy	
n	m	value	time	value	time	value	time	value	time
10	9	5	0.008	5	0.003	5	0.003	5	0.002
10	18	3	0.013	3	0.004	3	0.004	3	0.003
10	30	3	0.012	3	0.004	3	0.004	3	0.003
10	36	2	0.007	2	0.001	2	0.001	2	0.001
20	37	5	0.102	6	0.011	6	0.008	6	0.007
20	76	5	0.512	6	0.011	6	0.011	6	0.008
20	114	4	0.477	4	0.009	4	0.008	4	0.006
20	152	4	0.728	4	0.011	5	0.011	5	0.007
30	87	7	2.294	7	0.019	7	0.018	9	0.013
30	174	6	6.423	6	0.02	6	0.02	7	0.011
30	261	5	3.852	5	0.022	5	0.019	5	0.012
30	348	4	1.052	4	0.019	4	0.018	4	0.006
40	156	7	20.866	9	0.036	9	0.032	10	0.018
40	312	6	106.294	6	0.035	9	0.032	8	0.019
40	468	4	35.792	4	0.038	4	0.035	5	0.015
40	624	3	2.963	4	0.035	5	0.034	3	0.011
50	245	7	82.019	8	0.054	8	0.047	11	0.023
50	490	≤ 7	> 3600	9	0.067	9	0.058	9	0.024
50	735	5	255.426	6	0.072	6	0.063	6	0.019
50	1000	4	4.742	5	0.059	5	0.058	5	0.017
60	354	9	771.2	10	0.094	11	0.085	12	0.031
60	708	≤ 7	> 3600	9	0.102	9	0.094	9	0.031
60	1062	5	561.631	7	0.107	8	0.1	6	0.026
60	1416	4	10.923	5	0.097	5	0.095	5	0.021
70	483	≤ 8	> 3600	10	0.138	9	0.126	11	0.035
70	966	≤ 7	> 3600	9	0.161	9	0.137	9	0.037
70	1449	≤ 5	> 3600	6	0.157	6	0.141	6	0.026
70	1932	4	28.026	5	0.148	5	0.137	5	0.024
80	632	≤ 9	> 3600	10	0.208	10	0.183	13	0.047
80	1264	≤ 8	> 3600	9	0.238	10	0.207	10	0.048
80	1896	≤ 5	> 3600	7	0.239	7	0.216	7	0.042
80	2528	4	390.748	5	0.213	5	0.208	5	0.028
90	801	≤ 10	> 3600	11	0.276	13	0.265	13	0.058

Table 2.2: *(continued) Numerical results for the Watts-Strogatz random graphs, time is given in seconds. The best results are highlighted.*

graph		exact		greedy		greedy (`AddOne`)		locally greedy	
n	m	value	time	value	time	value	time	value	time
90	1602	≤ 8	> 3600	9	0.336	9	0.286	10	0.061
90	2403	≤ 6	> 3600	7	0.344	7	0.311	7	0.049
90	3204	4	106.156	6	0.327	6	0.302	5	0.041
100	990	≤ 11	> 3600	11	0.371	13	0.343	15	0.082
100	1980	≤ 8	> 3600	9	0.472	9	0.387	9	0.068
100	2970	≤ 6	> 3600	7	0.452	7	0.417	7	0.059
100	3960	4	350.92	6	0.415	6	0.398	6	0.043
115	1638	≤ 13	> 3600	14	0.614	14	0.574	15	0.106
115	3277	≤ 8	> 3600	9	0.715	9	0.648	9	0.091
115	4916	≤ 5	> 3600	6	0.693	6	0.645	6	0.071
135	2261	≤ 14	> 3600	15	0.927	16	0.832	15	0.135
135	4522	≤ 7	> 3600	9	1.112	9	0.952	9	0.116
135	6783	≤ 5	> 3600	6	1.065	6	0.997	6	0.086
150	2793	≤ 13	> 3600	14	1.287	16	1.164	15	0.183
150	5587	≤ 8	> 3600	8	1.497	8	1.338	9	0.141
150	8381	≤ 5	> 3600	6	1.502	6	1.411	6	0.106
average		6.265	1745.78	7.163	0.303	7.45	0.28	7.673	0.043

Barabási-Albert random graphs

Lastly, for the Barabási-Albert graphs, the computational results are presented in Table 2.3. Gurobi successfully found the optimal solution within the time limit for 25 instances, accounting for 51% of the cases.

The greedy algorithm failed to match the exact method's value in 31 cases, resulting in a success rate of 37%. It reported a +1 difference for 21 graphs, a +2 difference for 8 graphs, and a +3 difference for one graph compared to the value obtained by the exact method. Remarkably, for one graph instance, it achieved a better upper bound than Gurobi.

When comparing the two versions of the greedy algorithm, they produced identical values in 29 cases. `AddOne=1` provided a better upper bound in 2 cases, while the default version outperformed it 18 times.

The locally greedy algorithm found the optimal solution in 10 cases. In 19 cases, the upper bound missed the optimal solution by just one additional vertex, in 10 cases by two more vertices, and in 10 cases by three or more vertices.

The default version of the greedy algorithm and the locally greedy algorithm reported the same upper bound in 25 cases. In 4 cases, the locally greedy algorithm provided a better upper bound, whereas in 20 cases, the default version was superior.

The last row of Table 2.3 summarizes the average performance. Similar to the other two types of random graphs, the default greedy algorithm stands out as the most effective for obtaining upper bounds for the geodetic number, while the locally greedy algorithm excels in achieving this quickly for Barabási-Albert random graphs.

Table 2.3: *Numerical results for the Barabási-Albert random graphs, time is given in seconds. The best results are highlighted.*

graph		exact		greedy		greedy (`AddOne`)		locally greedy	
n	m	value	time	value	time	value	time	value	time
10	9	6	0.004	8	0.004	6	0.004	6	0.002
10	18	4	0.007	4	0.004	4	0.004	4	0.001
10	30	4	0.007	4	0.004	4	0.004	4	0.002
10	36	3	0.009	3	0.002	3	0.001	3	0.001
20	37	7	0.042	7	0.012	7	0.011	8	0.008
20	76	5	0.058	6	0.011	7	0.009	6	0.007
20	114	4	0.176	5	0.012	5	0.012	5	0.007
20	152	3	0.046	3	0.008	3	0.007	3	0.005
30	87	9	0.269	10	0.021	12	0.02	11	0.011
30	174	6	0.382	7	0.02	8	0.017	7	0.012
30	261	5	0.995	5	0.016	5	0.015	5	0.007
30	348	3	0.724	4	0.019	4	0.019	4	0.011
40	156	10	20.117	11	0.032	12	0.033	12	0.016
40	312	7	4.014	10	0.038	10	0.035	10	0.02
40	468	5	9.844	7	0.037	7	0.034	7	0.017
40	624	4	1.955	4	0.036	4	0.035	4	0.013
50	245	11	6.754	12	0.064	12	0.051	13	0.026
50	490	8	197.846	10	0.063	9	0.056	10	0.022
50	735	5	58.891	7	0.067	8	0.061	6	0.019
50	1000	4	12.052	6	0.059	6	0.053	5	0.014
60	354	≤ 11	> 3600	12	0.097	12	0.092	16	0.037
60	708	≤ 8	> 3600	10	0.103	10	0.096	9	0.031
60	1062	6	645.853	6	0.101	7	0.094	7	0.028
60	1416	4	26.435	4	0.092	4	0.089	4	0.018
70	483	≤ 11	> 3600	11	0.142	14	0.129	18	0.048
70	966	≤ 9	> 3600	11	0.164	11	0.139	11	0.043

Table 2.3: *(continued) Numerical results for the Barabási-Albert random graphs, time is given in seconds. The best results are highlighted.*

graph		exact		greedy		greedy (AddOne)		locally greedy	
n	m	value	time	value	time	value	time	value	time
70	1449	≤ 6	> 3600	6	0.151	6	0.135	6	0.025
70	1932	4	67.771	5	0.146	5	0.137	5	0.027
80	632	≤ 11	> 3600	11	0.207	12	0.187	18	0.068
80	1264	≤ 9	> 3600	9	0.224	9	0.203	11	0.051
80	1896	≤ 6	> 3600	7	0.231	7	0.208	7	0.041
80	2528	4	132.305	5	0.217	5	0.207	5	0.032
90	801	≤ 13	> 3600	14	0.305	17	0.283	20	0.081
90	1602	≤ 9	> 3600	10	0.315	11	0.281	11	0.064
90	2403	≤ 6	> 3600	7	0.334	8	0.311	7	0.047
90	3204	4	713.099	5	0.305	5	0.283	5	0.042
100	990	≤ 15	> 3600	14	0.392	18	0.363	18	0.101
100	1980	≤ 9	> 3600	10	0.421	13	0.378	13	0.081
100	2970	≤ 7	> 3600	8	0.435	8	0.402	8	0.063
100	3960	≤ 4	> 3600	5	0.416	5	0.387	5	0.052
115	1638	≤ 15	> 3600	16	0.626	18	0.566	18	0.121
115	3277	≤ 8	> 3600	8	0.673	8	0.608	10	0.081
115	4916	≤ 5	> 3600	5	0.648	5	0.632	6	0.062
135	2261	≤ 15	> 3600	15	0.938	18	0.828	21	0.176
135	4522	≤ 9	> 3600	10	1.064	11	0.927	10	0.115
135	6783	≤ 5	> 3600	6	1.053	6	0.963	7	0.106
150	2793	≤ 15	> 3600	17	1.282	17	1.192	20	0.221
150	5587	≤ 9	> 3600	9	1.443	11	1.291	10	0.137
150	8381	≤ 6	> 3600	6	1.415	7	1.342	6	0.118
average		7.27	1802.034	8.061	0.3	8.653	0.27	9.081	0.048

2.3.3 Discussion of the results on larger graph instances

The values reported in Table 2.4 demonstrate that *BILP* can require hours to determine the exact geodetic number for graphs containing thousands of vertices and edges. In contrast, the proposed algorithms were capable of providing acceptable upper bounds within a reasonable time. It is important to note that, for this particular set of graphs, there was no time limit imposed on Gurobi. Even for the largest graph instance (ia-email-univ), both versions of the greedy algorithm were able to generate slightly less accurate upper bounds than the exact value in under 700 seconds. Although the locally greedy algorithm proved to be the fastest, it missed the upper bound by a considerably larger margin for the larger graphs compared to the greedy method. This pattern is also reflected in the average performance data provided in the last row of Table 2.4.

Table 2.4: *Numerical results for real-world graphs, time is given in seconds unless indicated otherwise*

graph			exact		greedy		greedy (`AddOne`)		locally greedy	
name	n	m	value	time	value	time	value	time	value	time
karate	34	78	16	0.06	16	0.024	16	0.019	16	0.015
mexican	35	117	7	0.21	7	0.025	8	0.023	9	0.011
sawmill	36	62	14	0.09	14	0.022	14	0.019	15	0.015
chesapeake	39	170	5	0.12	5	0.026	5	0.023	5	0.008
ca-netscience	379	914	253	37 m	256	21.6	260	20.8	264	14.1
bio-celegans	453	2025	172	1 h	183	40.8	188	34.3	225	14.8
rt-twitter-copen	761	1029	459	6 h	459	101.7	459	103.4	490	112.6
soc-wiki-vote	889	2914	275	14 h	276	236.4	277	232.0	409	120.5
ia-email-univ	1133	5451	244	16 h	248	698.6	250	677.9	464	269.0
average			160.6	4 h	162.7	122.1	164.1	118.7	210.8	59.0

2.4 Concluding remarks

Given the fact that the graph geodetic number problem is computationally challenging, it is valuable to develop an algorithmic way capable of providing upper bounds of acceptable quality within a reasonable time.

The author of this Ph.D. thesis is responsible for the following contributions presented in this chapter:

II/1. I have introduced two greedy-type approaches. The first, known as the greedy algorithm, utilizes Floyd's algorithm, whereas the locally greedy algorithm is based on Dijkstra's algorithm.

II/2. I have empirically demonstrated that the proposed algorithms can efficiently obtain upper bounds that closely approximate the optimal solution obtained from the binary integer linear programming. Meanwhile, their computational time is a small fraction of that needed to obtain the exact geodetic number.

Chapter 3

Symbolic Regression for Approximating Graph Geodetic Number

3.1 Introduction

Geodetic number is a global measure for simple connected graphs to find the minimal-cardinality set of vertices, such that all shortest paths between its elements cover every vertex of the graph [45]. A *BILP* formulation for the geodetic number problem was introduced in [45], which also included the initial computational experiments conducted on a set of random graphs. The formal definition of the geodetic number and its applications has been introduced in Section 2.1.

Symbolic Regression (SR) is a mathematical modeling technique aimed at discovering a simple formula that accurately fits a given output based on a set of inputs. Unlike traditional regression methods, which start with a predefined model, Symbolic Regression does not begin with a specific model structure. Instead, in SR, initial formulas are generated randomly by combining input parameters, operators, and constants. Subsequently, new formulas are constructed by recombining existing formulas using evolutionary algorithms, with genetic programming being the focus of this chapter.

Symbolic Regression operates within an effectively infinite search space, as it can potentially generate an infinite number of formulas. However, this characteristic can be beneficial when utilizing an evolutionary algorithm like genetic programming, which relies on diversity to efficiently explore the search space and ultimately discover highly accurate formulas.

The input parameters and constants in SR are predetermined. SR combines these input elements using a set of predefined arithmetic operators, such as $(+, -, \times, \div$, etc.) to formulate a mathematical expression. Symbolic Regression has been applied in various domains, including physics, where it has been employed to identify physical laws based on experimental data [79], as well as to find analytical solutions for iterated functions of arbitrary forms [80]. While there are alternative algorithms in the literature that can be used for Symbolic Regression beyond genetic programming [63], genetic programming remains one of the most widely used and popular algorithms applied by Symbolic Regression [52].

Cartesian Genetic Programming (CGP) is one of the most famous genetic programming tools, developed by Miller [66]. CGP is an iteration-based evolutionary algorithm that works as it follows. CGP begins by creating a set of initial solutions, from which the best solution is chosen by evaluating the solutions based on a fitness function. Then these solutions will be used to create the next generation in the algorithm. The next generation's solutions will be a mixture of chosen solutions from the previous generations, where the new solutions should not be identical to the previous ones, which can be done by mutation. Mutation is used to change small parts of the new solutions and it usually occurs probabilistically for CGP. The mutation rate is the probability of applying the mutation to a specific solution. Eventually, the algorithm must terminate. There are two cases in which this occurs: the algorithm has reached the maximum number of generations, or the algorithm has reached the target fitness. At this point, a final solution is selected and returned. CGP has several parameters to set up, which certainly have effects on its performance. The specific parameters used in this work are detailed in Section 3.3.3.

In this work, which is published in [8], SR with CGP has been used to derive formulas capable of approximating the graph geodetic number. The obtained formulas are tested on random and real-world graphs. They demonstrated how various graph properties as training data can lead to diverse formulas with different accuracy.

3.2 Methodologies

While there are limited research papers exploring the use of SR to approximate graph properties, Märtens *et al.* [60] serves as a notable starting point. They utilized SR and CGP with inputs as eigenvalues of the Laplacian and adjacency matrices to optimize graph diameter and isoperimetric number on real-world graphs. In this study, the focus is on obtaining results for the geodetic number for both random and real-world graphs. Consequently, an investigation into graph properties closely associated with the geodetic number was carried out.

The implementation employed in this work was the CGP-Library, a cross-platform Cartesian Genetic Programming tool developed by Andrew Turner[1]. This C-based library is compatible with Linux, Windows, and MacOS.

To utilize CGP, a training dataset is required. Each dataset contains instances, with each instance comprising two main components: (i) parameters representing graph properties and selected constants as inputs and (ii) the exact value of the corresponding graph property as the output. CGP then aims to combine these parameters and constants using arithmetic operators to achieve the desired output. The set of arithmetic operators used consistently across all cases includes $(+,-,\times,\div,\sqrt{x},x^2,x^3)$. The chosen graph properties encompass eigenvalues of the Adjacency and Laplacian matrices, the count of degree-one vertices, the

[1]`http://www.cgplibrary.co.uk/`

count of simplicial vertices, the number of vertices, and the number of edges. It will be demonstrated that these parameters are closely related to the graph geodetic number, making them valuable inputs for CGP. The subsequent section will categorize these inputs for clarity.

3.3 Parameters of the Numerical Experiments

The main goal of the experiments was to investigate the graph geodetic number for random graphs and real-world graphs. Since the most related paper to this work of Märtens *et al.* [60] contains results for the graph diameter (which is, similar to the geodetic number, also based on shortest paths) the results obtained for the diameter reported and compared to these values. The metrics used to measure the goodness of a formula are mean absolute error and mean relative error. In the following subsections, the graphs used for the training as well as for the validation are described.

3.3.1 Random Graphs

A set of 120 random graphs was created by using the three well-known generative models: ER model [35], WS model [88] and BA model [4]. Regarding the number of vertices and edges the following approach was used:

- the number of vertices were $n = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100$, and
- the number of edges followed the scheme as in [45]:
  - for each case one can have maximum $\frac{n(n-1)}{2}$ edges,
  - 20%, 40%, 60%, and 80% of this maximum number of edges were taken.

3.3.2 Real-World Graphs

A collection of 10 real-world graphs from the Network Repository [78] was employed for this study. To construct the training dataset, 120 connected subgraphs were generated from these networks, varying in size (with $14 \leq N \leq 140$). This was accomplished using a straightforward procedure. Given a real-world graph $G = (V, E)$, the process began by randomly selecting a set $W \subset V$ of vertices. Subsequently, the subgraph of G with vertex set W was extracted. This subgraph, denoted as $\hat{G}$, might not necessarily be connected. As a final step, the largest connected component of $\hat{G}$ was chosen.

3.3.3 CGP Parameters

CGP needs predefined parameters to work properly. Table 3.1 summarizes the values of the parameters used in the experiments. The details of the parameters used are the following.

Table 3.1: *Parameters of CGP*

Parameter	**Value**
Evolutionary Strategy	$(1+4)$-ES
Vertex Arity	2
Mutation Type	Probabilistic
Mutation Rate	0.05
Fitness Function	Supervised Learning
Target Fitness	0.1
Selection Scheme	Select Fittest
Reproduction scheme	Mutation Random Parent
Number of generations	$200,000$
Update frequency	100
Threads	1
Function Set	`add sub mul div sqrt sq cube`

Evolutionary Strategy The evolutionary strategy uses selection and mutation as search operators. The usual version used by CGP is the one that is applied in this chapter, which is called $(1+4)$-ES. Here, the procedure selects the fittest individual as the parent for the next generation, from the combination of the current parent and the four children.

Vertex Arity Each vertex is assumed to take as many inputs as the maximum vertex arity value, namely, the maximum number of inputs connected to a specific vertex.

Mutation Type The mutation, as the basic search operator of the evolutionary strategy, is performed by adding a random vector to the current solution. In this chapter, this is done probabilistically.

Mutation Rate The probability of applying mutation on a specific solution.

Fitness Function The supervised learning fitness function applies to each solution and assigns a fitness value to how closely the solution output matches the desired output. Based on that, the solutions with better fitness value will be chosen for the next generations.

Target Fitness The fitness function used in this work is the absolute differences (absolute error) between the generated and predefined outputs, where the best solution is the one with an absolute difference less than or equal to the given value.

Selection Scheme The applied fittest selection schemes select the best solutions based on the closest fitness obtained by the solution.

Reproduction scheme There are two ways in which new children can be created from their parents. In the first method, the child is simply a mutated copy of the parent. In the second method, the child is a combination of both parents with or without mutation. This latter method is referred to as recombination. Usually, CGP-Library uses the random parent reproduction scheme which simply creates each child as a mutated version of its parents.

Number of generations How many iterations CGP will apply before termination unless one of the solutions obtains the target fitness.

Update frequency The frequency at which the user is updated on progress, where the progress details are shown on the terminal.

Threads The number of threads the CGP library will use internally.

Function Set the arithmetic operators used by CGP to combine the inputs.

3.3.4 Training data parameters

The list of parameters used as input in the training data, separated into different sets as follows.
For random graphs:

1) $N, M, \lambda_N, \lambda_i \ (i = 1,2,3)$
2) $N, M, \mu_{N-1}, \mu_i \ (i = 1,2,3)$
3) $N, M, \lambda_i, \lambda_{N-i-1} \ (i = 1,\dots,5)$
4) $N, M, \mu_i, \mu_{N-i-1} \ (i = 1,\dots,5)$
5) $N, M, \lambda_i, \lambda_{N-i-1} \ (i = 1,\dots,5)$ and constants $1,2,3,4,5$
6) $N, M, \mu_i, \mu_{N-i-1} \ (i = 1,\dots,5)$ and constants $1,2,3,4,5$

where N is the number of vertices, M is number of edges, λ_i is the i-th eigenvalue of Adjacency matrix, μ_i is the i-th eigenvalue of Laplacian matrix.

For real-world graphs:

1) $N, M, \gamma, \sigma,$ and constants $1,2,3,4,5$
2) $N, M, \gamma, \sigma, \lambda_i, \lambda_{N-i-1} \ (i = 1,\dots,5)$

3) $N, M, \gamma, \sigma, \mu_i, \mu_{N-i-1}$ $(i = 1, \ldots, 5)$

4) $N, M, \gamma, \sigma, \lambda_i, \lambda_{N-i-1}$ $(i = 1, \ldots, 5)$ and constants $1, 2, 3, 4, 5$

5) $N, M, \gamma, \sigma, \mu_i, \mu_{N-i-1}$ $(i = 1, \ldots, 5)$ and constants $1, 2, 3, 4, 5$

where γ is the number of vertices with degree one in the graph, and σ is the number of simplicial vertices in the graph.

3.4 Results

To derive formulas for both random graphs and real-world graphs, CGP was executed a dozen times for each category. Among the generated formulas, the best ones were selected based on their absolute error and relative error compared to the exact values. Therefore, the best formulas exhibited the smallest errors. The complete list of chosen formulas can be found in the Appendix 6. In the following sections, the best formula for each case will be presented and discussed.

It is important to note that both the diameter and the geodetic number are integer values. However, the formulas obtained through Symbolic Regression often yield non-integer results. Therefore, in the tables reporting the results, the formula-derived values were rounded before calculating the errors.

The results will be presented in two types of tables. For random graphs, only a summary of the approximation errors will be displayed. For real-world graphs, comprehensive details will be provided, including the calculated values for both the diameter and the geodetic number using the best formulas.

3.4.1 Diameter

The *diameter* of a graph, as discussed in 1.1.2, represents the length of the longest shortest path among all the shortest paths within the graph. This measure can be readily calculated, allowing for a comparison between the exact diameter value and the approximations derived from the formulas obtained through Symbolic Regression.

Random graphs

For the diameter of random graphs, Table 3.2 summarizes the results obtained for the different models, where (6.5) gives the best approximation:

$$\frac{N + \lambda_{N-2} + 4}{\sqrt{M}}$$

In the context of the examined random graphs, the eigenvalue λ_{N-2} falls within the range of $[-7, -1]$. On average, this range is canceled by the constant factor of $+4$ in formula

(6.5). Additionally, for these types of graphs, we have $M = \mathcal{O}(N)$, implying that formula (6.5) is approximately $\mathcal{O}(\sqrt{N})$. The square root function, within the range considered in the experiments, is close to the logarithm function. It is a well-known fact that the diameter of (random) graphs can be approximated by $\log(N)$. Interestingly, despite not having the log function in the function set (as indicated in Table 3.1), Symbolic Regression found formula (6.5), which is close to the logarithm of the number of vertices in the graphs.

Table 3.2: *Diameter validations on random graphs*

	formula	(6.1)	(6.2)	(6.3)	(6.4)	(6.5)	(6.6)
ER	mean absolute error	1	1.5	0.6	6.05	**0.4**	0.9
	mean relative error	0.4	0.53	0.19	2.46	**0.1**	0.33
BA	mean absolute error	1.3	0.8	0.35	4.2	**0.1**	0.55
	mean relative error	0.52	0.28	0.14	1.86	**0.03**	0.2
WS	mean absolute error	1.7	1.7	0.5	6.15	**0.35**	1.15
	mean relative error	0.57	0.57	0.18	2.48	**0.1**	0.4

Real-world graphs

For the diameter of real-world graphs, as presented in Table 3.3, formula (6.14) was the best, yielding values very close to the exact diameter:

$$\frac{2M}{\lambda_1\lambda_2^2}+\frac{\lambda_5^2+2(\lambda_N-\lambda_3)+50}{\lambda_1}+\frac{2}{\lambda_1\lambda_2}$$

Upon closer examination, it becomes evident that the last term in the formula typically has very small values, often below 0.1. The other components of (6.14) contribute in roughly equal measure to the final result. This formula contains the first three, fifth, and last eigenvalues of the adjacency matrix, along with the number of edges. It serves as a clear example of the surprising capabilities of Symbolic Regression, as it can discover a non-trivial combination of graph features that effectively approximates a graph metric like diameter. However, it is essential to note that the computational complexity remains $\mathcal{O}(N^3)$ due to the eigenvalue calculations. This implies that it requires a similar computational cost as applying an exact algorithm like Floyd-Warshall to determine the diameter.

It is worth mentioning that formulas (6.8) and (6.9) yielded the same mean relative error as (6.14). However, they performed less effectively in terms of mean absolute error. The formula (6.9) includes some eigenvalues of the Laplacian matrix and constants, while formula (6.8) utilizes eigenvalues of the adjacency matrix, the number of vertices, and the number of simplicial vertices. Consequently, these formulas, while not providing as accurate approximations as (6.14), are constructed using different graph parameters compared to (6.14). In the 5th column of Table 3.3, results from [60] for the same set of graphs are included. Clearly, all the formulas show smaller errors than the best solution reported in [60].

Table 3.3: *Diameter validations on real-world graphs*

graph(N, M)	d	[60]	(6.7)	(6.8)	(6.9)	(6.10)	(6.11)	(6.12)	(6.13)	(6.14)
`ca-netscience(379,914)`	17	21	13	9	14	19	4	17	12	10
`bio-celegans(453,2025)`	7	7	5	4	8	12	3	8	6	4
`rt-twitter-copen(761,1029)`	14	16	13	14	14	19	12	17	20	11
`soc-wiki-vote(889,2914)`	13	10	11	8	11	15	7	11	12	6
`ia-email-univ(1133,5451)`	8	6	9	9	7	12	9	8	13	10
`ia-fb-messages(1266,6451)`	9	7	10	8	8	12	7	9	11	6
`bio-yeast(1458,1948)`	19	19	14	28	15	20	18	18	39	18
`socfb-nips-ego(2888,2981)`	9	52	14	14	16	23	3	20	21	7
`web-edu(3031,6474)`	11	36	14	11	15	22	13	19	16	8
`inf-power(4941,6594)`	46	98	14	38	17	24	71	20	53	48
mean absolute error:		13.3	5.6	4	5.2	6.9	6.2	5.2	6.4	**3.3**
mean relative error:		0.92	0.28	0.27	0.27	0.53	0.37	0.31	0.48	**0.27**

3.4.2 Geodetic number

To compare the approximations provided by the formulas found by Symbolic Regression, it was necessary to compute the exact geodetic number of the input graphs. For this purpose, the *BILP* proposed in [45] was employed.

Random graphs

The results for the geodetic number of random graphs are presented in Table 3.4. Formula (6.15) yielded the best approximations for the *ER* and *WS* graphs:

$$\sqrt{\frac{N^{3/2}}{\lambda_1} - \frac{\lambda_{N-4}N^{3/2}}{\lambda_1^2 + N^{3/2}}}$$

In the case of *BA* graphs, formula (6.16) resulted in the lowest error:

$$\frac{\mu_4^2}{\mu_2\mu_{N-3}} + \sqrt{N - \mu_3}$$

Both of these formulas require the computation of all eigenvalues, making their computational cost $\mathcal{O}(N^3)$. Note that while the exact computation of the geodetic number is NP-hard, formulas (6.15) and (6.16) can be evaluated in polynomial time. In general, formula (6.15) provides the best approximation for all three types of random graphs. Upon closer examination, it becomes obvious that the second part of the formula is approximately half of the first

Table 3.4: *Geodetic number validations on random graphs*

	formula	(6.15)	(6.16)	(6.17)	(6.18)
ER	mean absolute error	**0.92**	1.31	1	1.07
	mean relative error	**0.1**	0.16	0.16	0.13
BA	mean absolute error	2.15	**1**	1.775	2.92
	mean relative error	0.18	**0.08**	0.17	0.26
WS	mean absolute error	**0.54**	1.38	0.92	0.69
	mean relative error	**0.04**	0.19	0.12	0.08

part. Thus, a simplified formula could be:

$$\frac{3}{2}\sqrt{\frac{N^{3/2}}{\lambda_1}}$$

On average, this simpler formula yields slightly less optimistic approximations (mean absolute error = 1.89, mean relative error = 0.1), but it only requires the computation of the first dominant eigenvalue, resulting in a lower computational cost of $\mathcal{O}(N^2)$.

Real-world graphs

Table 3.5 shows the results for the real-world graphs. In this case, the best approximation was obtained by the surprisingly compact formula (6.26):

$$\gamma + \sigma + \sqrt{M} - 2$$

Formula (6.26) contains the number of degree-one vertices and the number of simplicial vertices because these vertices are proven to be essential components of the geodetic set, as demonstrated in previous research [2, 46]. Both of these factors are present in all the best formulas, as detailed in the Appendix. In networks like the `ca-netscience` and `bio-celegans`, simplicial vertices are much, while the number of degree-one vertices is relatively low. Conversely, in other graphs, the number of simplicial vertices is limited, usually below 10. The remaining part of the geodetic number is approximated by $\sqrt{M} - 2$, contributing at most one-third to the overall approximation for these graphs. Formula (6.26) has a computational complexity of $\mathcal{O}(NM)$.

Improvement

The goal is to derive a general formula for the geodetic number that can provide accurate approximations for any real-world graph. To achieve this, linear regression was employed to refine formula (6.26). The generalized formula, which includes multipliers as variables,

Table 3.5: *Geodetic number validation on real-world graphs.*

graph(N, M)	$g(G)$	(6.19)	(6.20)	(6.21)	(6.22)	(6.23)	(6.24)	(6.25)	(6.26)
`ca-netscience(379,914)`	253	208	151	190	198	194	206	195	200
`bio-celegans(453,2025)`	172	213	115	119	195	188	225	203	146
`rt-twitter-copen(761,1029)`	459	436	437	438	439	428	446	442	444
`soc-wiki-vote(889,2914)`	275	247	212	220	222	231	247	259	245
`ia-email-univ(1133,5451)`	244	225	182	194	181	192	208	196	233
`ia-fb-messages(1266,6451)`	318	266	254	264	276	280	296	313	311
`bio-yeast(1458,1948)`	784	763	761	766	761	751	775	762	773
	mean absolute error:	32.7	56.1	44.9	39.9	39.0	29.7	28.1	**21.9**
	mean relative error:	0.12	0.21	0.17	0.14	0.13	0.12	0.11	**0.08**

takes the form:

$$a \cdot \gamma + b \cdot \sigma + c \cdot \sqrt{M} - d$$

The initial values for these variables were set to $a = b = c = d = 1$. Linear regression was utilized to determine the optimal values of the variables a, b, c, and d that minimize the mean absolute error in the approximated values. The results of the linear regression indicated that the optimal values are $a = 0.99$, $b = 0.79$, $c = 0.97$, and $d = 0.99$. Therefore, the formula can be expressed as:

$$0.99 \cdot \gamma + 0.79 \cdot \sigma + 0.97 \cdot \sqrt{M} - 0.99 \tag{3.1}$$

Validation of improved formula

To evaluate the accuracy of formula (3.1), a validation process was conducted using 120 sub-graphs (with $31 \leq N \leq 485$) extracted from the real-world graphs listed in Table 3.5. These sub-graphs were generated following the procedure outlined in Section 3.3.2. The geodetic number was computed twice for each sub-graph: first, the exact value was determined using the *BILP* from [45], and second, an approximation was obtained using formula (3.1). Figure 3.1 presents a comparison between the two sets of values for these sub-graphs, illustrating that the approximations are close to the exact $g(G)$ values. Across all 120 graphs, the mean absolute error obtained using formula (3.1) was 12.27, and the mean relative error was 0.18. This represents a slight improvement over formula (6.26), which resulted in a mean absolute error of 12.37 and the same relative error as formula (3.1).

In Figure 3.1, two noticeable gaps are evident, representing that for certain graphs, the approximation is significantly lower than the exact value. This occurs when the number of simplicial vertices and/or degree-one vertices is zero for these specific graphs. Since formula (3.1) relies on the summation of the number of simplicial vertices, the number of degree-one vertices, and the number of edges, the absence of any of these values results in these gaps. For graphs where γ and/or σ is zero, it may be more beneficial to utilize one of the formulas

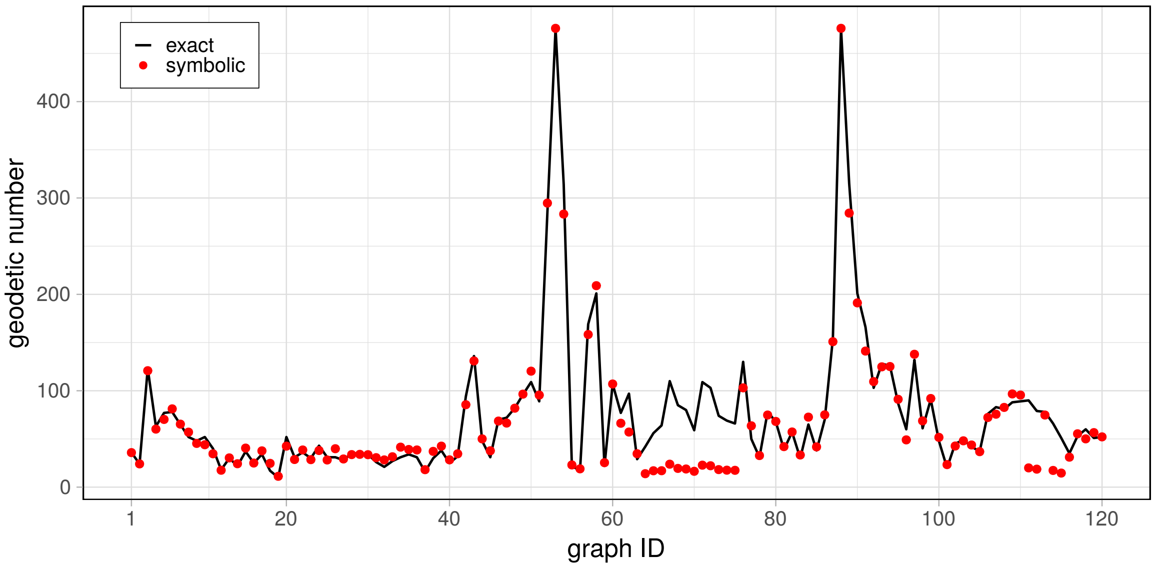


Figure 3.1: *Exact $g(G)$ and values given by the optimized formula* (3.1)

derived for random graphs. For instance, when applying formula (6.15) to these graphs, the mean absolute error obtained is 39.87, and the mean relative error is 0.57. In contrast, using formula (3.1) on the same graphs yields a mean absolute error of 40.87 and a mean relative error of 0.6.

3.5 Concluding remarks

Metaheuristics are viable options for handling complicated graph problems. This work demonstrated that Symbolic Regression and Cartesian Genetic Programming are successfully applicable to derive optimized formulas for graph geodetic number.

The author of this Ph.D. thesis is responsible for the following contributions presented in this chapter:

III/1. I have used Symbolic Regression together with Cartesian Genetic Programming to derive a general formula that can approximate the value of the geodetic number. The formula is simply the sum of the number of edges, the number of degree-one vertices, and the number of simplicial vertices. Thus, the approximation of the geodetic number can be obtained in a reasonable computational time, even for graphs with thousands of vertices and edges.

III/2. I have demonstrated how different training sets will lead to different formulas with different accuracy which validates that using parameters that are highly related to the graph property as training data will help Symbolic Regression to approximate in a better manner.

Chapter 4

Exact Methods for the Longest Induced Cycle Problem

4.1 Introduction

A significant part of combinatorial optimization is closely related to graphs. Within graph theory, the concept of graph cycles has fundamental importance. Identifying a simple cycle or a cycle with a specific structure within a graph forms the basis for numerous graph-theoretical problems that have been under investigation for many years.

Kumar *et al.* [53], introduced a heuristic algorithm for the longest simple cycle problem. The authors utilized both adjacency matrices and adjacency lists, achieving a time complexity for the proposed algorithm proportional to the number of vertices plus the number of edges of the graph. In [6], the authors investigated the longest cycle within a graph with a large minimal degree. For a graph $G = (V,E)$ with a vertex count of $|V| = n$, the parameter $\text{min\_deg}(G)$ denotes the smallest degree among all vertices in G, while $\text{c}(G)$ represents the size of the longest cycle within G. The authors demonstrated that for $n > k \geq 2$, with $\text{min\_deg}(G) \geq n/k$, the lower bound $\text{c}(G) \geq [n/(k-1)]$ holds. Broersma *et al.* [23] proposed exact algorithms for identifying the longest cycles in claw-free graphs. A claw, in this context, refers to a star graph including three edges. The authors introduced two algorithms for identifying the longest cycle within such graphs containing n vertices: one algorithm operates in $\mathcal{O}(1.6818^n)$ time with exponential space complexity, while the second algorithm functions in $\mathcal{O}(1.8878^n)$ time with polynomial space complexity.

In the work by Lokshtanov [58], the focus lies on the examination of the longest isometric cycle within a graph, which is defined as the longest cycle where the distance between any two vertices on the cycle remains consistent with their distances in the original graph. The author introduced a polynomial-time algorithm to address this specific problem.

The primary focus of this work, which is based on [10], is dedicated to addressing the challenge of identifying the longest induced (or chordless) cycle problem. For a graph $G = (V,E)$ and a subset $W \subseteq V$, the W-induced graph $G[W]$ comprises all the vertices from set W

and the edges from G that connect vertices exclusively within W. The objective of the longest induced cycle problem is to determine the largest possible subset W for which the graph $G[W]$ forms a cycle. While it may seem straightforward to obtain an induced cycle since every isometric cycle is an induced cycle, it has been shown that identifying the longest induced cycle within a graph is an NP-complete problem, as demonstrated by Garey *et al.* [40].

The longest induced path (P), discussed in [62], represents a sequence of vertices within graph G, where each consecutive pair of vertices is connected by an edge $e \in E$ and there is no edge between non-consecutive vertices within P. In the context of a general graph G, determining the existence of an induced path with a specific length is proven to be NP-complete, as detailed in [40]. Consequently, the longest induced cycle can be considered as a special case of the longest induced path.

Holes in a graph, defined as induced cycles with four or more vertices, play a significant role in various contexts. Perfect graphs, for instance, are characterized by the absence of odd holes or their complements [28]. Moreover, when addressing challenges like finding maximum independent sets in a graph [71], the existence of odd holes leads to the formulation of odd hole inequalities, strengthening approaches for these problems. Similarly, in other problem domains such as set packing and set partitioning [20], these odd hole inequalities serve as crucial components.

Several papers have explored the longest induced cycle problem in graphs with specific structures. In [39], the author investigated the longest induced cycle within the unit circulant graph. To define the unit circulant graph $X_n = Cay(\mathbb{Z}_n; \mathbb{Z}_n^*)$, where n is a positive integer, consider the following. The vertex set of X_n, denoted as $V(n)$, comprises the elements of $\mathbb{Z}_n$, the ring of integers modulo n. The edge set of X_n, represented as $E(n)$, for $x, y \in V(n)$, $(x, y) \in E(n)$ if and only if $x - y \in \mathbb{Z}_n^*$, with $\mathbb{Z}_n^*$ being the set of units within the ring $\mathbb{Z}_n$. The author demonstrates that if the positive integer n has r distinct prime divisors, then X_n contains an induced cycle of length $2^r + 2$. In a separate study by Wojciechowski *et al.* [90], the authors examine the longest induced cycles within hypercube graphs. If G represents a d-dimensional hypercube, they proved the existence of an induced cycle with a length $\geq (9/64) \cdot 2^d$.

Pereira *et al.* [74] dealt with the longest cordless cycle problem, which is equivalent to the longest induced cycle problem. They presented an *ILP* formulation along with additional valid inequalities to strengthen and refine the formulation, all of which were incorporated into a *B&C* algorithm. They applied a multi-start heuristic method for initial solution generation and then conducted performance evaluations of the algorithm on a range of randomly generated graphs, including those with up to 100 vertices.

This work proposes three *ILP* designed to handle the longest induced cycle problem within general graphs. Some of these models build upon the models applied by prior work focused on solving the longest induced path problem, as seen in the studies by Marzo *et al.* [61] and Bokler *et al.* [17]. Matsypura *et al.* [62] introduced three *ILP* formulations and an exact iterative algorithm based on these formulations for tackling the longest induced path problem. However, it is important to note that these methods do not extend in this work, as they were found to be less effective in comparison to models in [17, 61].

4.2 Models

4.2.1 Notations

Let $G = (V,E)$ be an undirected graph with vertex set V and edge set $E \subset V \times V$. An edge $e \in E$ can be given as (i, j) for some $i, j \in V$. However, the symmetric pair $\bar{e} = (j,i)$ is not included in E. Thus, the symmetric edge set $E^* = E \cup \{\bar{e} = (j,i) : e = (i, j) \in E\}$ is introduced. Throughout, an edge $e = (i, j)$, and $\bar{e} = (j,i)$ unless explicitly stated otherwise.

The notation δ is used for adjacent edges over vertices and edges as follows. The outgoing and incoming edges of vertex i denoted with $\delta^+(i) = \{(i,k) \in E^*\}$, and $\delta^-(i) = \{(k,i) \in E^*\}$, respectively. Additionally, $\delta(i) = \delta^+(i) \cup \delta^-(i)$ denote all the edges incident to vertex i.

For an edge $e = (i, j) \in E^*$, outgoing edges are $\delta^+(e) = \delta^+(i) \cup \delta^+(j) \setminus \{e,\bar{e}\}$, and similarly, incoming edges are $\delta^-(e) = \delta^-(i) \cup \delta^-(j) \setminus \{e,\bar{e}\}$. The neighbour edges of e are denoted by $\delta(e) = \delta^+(e) \cup \delta^-(e)$ for all $e \in E^*$. This notation can be extended to any subset of vertices $C \subset V$, where $\delta^+(C) := \{(k,l) \in E^* : k \in C, l \in V \setminus C\}$ and $\delta^-(C) := \{(k,l) \in E^* : l \in C, k \in V \setminus C\}$ denote the outgoing and incoming edges of C, respectively, and $\delta(C) = \delta^+(C) \cup \delta^-(C)$ all edges that connect C with $V \setminus C$.

4.2.2 Order-based model

The first model to discuss, called *LIC*, is an *MILP* model using order-based formulation to avoid subtours. The formalism of the model is as follows:

$$\max \sum_{i \in V} y_i \tag{4.1}$$

subject to

$$x_e + x_{\bar{e}} \leq 1 \qquad \forall\, e \in E \tag{4.2}$$

$$\sum_{g \in \delta^+(e)} x_g \leq 1 \qquad \forall\, e \in E^* \tag{4.3}$$

$$y_i = \sum_{g \in \delta^+(i)} x_g \qquad \forall i \in V \tag{4.4}$$

$$y_i = \sum_{g \in \delta^-(i)} x_g \qquad \forall i \in V \tag{4.5}$$

$$\sum_{i \in V} w_i = 1 \tag{4.6}$$

$$w_i \leq y_i \qquad \forall\, i \in V \tag{4.7}$$

$$u_i - u_j \leq n(1 - x_e) - 1 + nw_i \qquad \forall e \in E^* \tag{4.8}$$

$$\sum_{i \in V} iw_i \leq jy_j + n(1 - y_j) \qquad \forall\, j \in V \tag{4.9}$$

$$y_i, u_i \geq 0 \qquad \forall i \in V \tag{4.10}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E^* \tag{4.11}$$

$$w_i \in \{0, 1\} \qquad \forall i \in V \tag{4.12}$$

The variable y_i indicates whether vertex i is part of the longest induced cycle or not. Consequently, the objective in (4.1) aims to maximize the sum of these variables, which directly corresponds to the length of the cycle. The decision variable x_e is one if the edge e is included in the solution, and zero otherwise.

The constraints can be understood as follows. Given that E^* is symmetric, constraint (4.2) guarantees that only one of the edges e or $\bar{e}$ can exist in the cycle, preventing the formation of small cycles. Constraint (4.3) ensures that for any edge $e = (i, j) \in E^*$, only one outgoing edge from either vertex i or vertex j can be part of the cycle. Constraints (4.4) and (4.5) ensure that for a given vertex i, only one outgoing edge and one incoming edge can be chosen to be part of the cycle. The constraint (4.8) is a modified Miller-Tucker-Zemlin (MTZ) order-based formulation: if edge e is in the cycle, vertices i and j must be arranged in sequential order unless the binary variable w_i equals 1. This variable is introduced to handle the position of the last vertex in the cycle, facilitating the ordering process. Constraint (4.9) functions as a symmetry-breaking constraint, as described in [85]. It enforces that the last vertex in the cycle must have the smallest index among all vertices in the cycle.

For a variation of the above introduced *LIC* model consider the following constraint:

$$x_e + x_{\bar{e}} \geq y_i + y_j - 1 \qquad \forall\, e = (i, j) \in E \tag{4.13}$$

Constraint (4.13) guarantees that either edge e or $\bar{e}$ must be included in the solution if both endpoints i and j are part of the solution. Conversely, if an edge is not selected for the solution, neither of its endpoints can be included in the solution. By substituting constraint (4.3) in the original *LIC* model with constraint (4.13), we create a new model *LIC*2. This modification leads to improved runtime performance compared to *LIC*, as demonstrated in Section 4.4.

4.2.3 Subtour-elimimation model

The second model used to address the longest induced cycle problem is based on the model presented by Bokler *et al.* [17], which is referred to ILP_{cut} and was originally designed for identifying the longest induced path. E^* is the symmetric edge set, as defined previously.

Let $\mathscr{C}$ represent the set of cycles in G. The model is defined as follows:

$$\max \frac{1}{2} \sum_{e \in E^*} x_e \tag{4.14}$$

subject to

$$x_e = x_{\bar{e}} \qquad \forall e \in E \tag{4.15}$$

$$x_e \leq \sum_{g \in \delta^-(i)} x_g \qquad \forall\, e = (i,j) \in E^* \tag{4.16}$$

$$\sum_{g \in \delta^-(i)} x_g + \sum_{g \in \delta^+(j)} x_g \leq 2 \qquad \forall\, e = (i,j) \in E^* \tag{4.17}$$

$$\sum_{e \in \delta(i)} x_e \leq \sum_{g \in \delta(C)} x_g \qquad \forall C \in \mathscr{C}, i \in C \tag{4.18}$$

$$x_e \in \{0,1\} \qquad \forall e \in E^* \tag{4.19}$$

The binary decision variable x_e indicates whether edge e is a part of the longest induced cycle. In this case edge selection is symmetric, thus the objective is to maximize half of the sum of these variables, as defined in objective function (4.14). Symmetry of the solution is guaranteed by (4.15). Constraint (4.16) enforces that the solution forms a cycle or cycles, while constraint (4.17) specifies that for any edge e, precisely two of its adjacent edges must also be selected. This ensures the induced property of the solution. Constraint (4.18) is utilized to eliminate small cycles in the graph.

4.2.4 Cycle-elimination model

The third model, called *cec*, is a modified version of the *cec* model introduced in [61] to find the longest induced path. The formalism of the model is as follows:

$$\max \sum_{i \in V} y_i \tag{4.20}$$

subject to

$$\sum_{e \in \delta(i)} x_e = 2y_i \qquad \forall\, i \in V \tag{4.21}$$

$$x_e \leq y_i \qquad \forall\, i \in V, e \in \delta(i) \tag{4.22}$$

$$x_e \geq y_i + y_j - 1 \qquad \forall e = (i,j) \in E \tag{4.23}$$

$$\sum_{i \in C} y_i \leq |C| - 1 \qquad \forall C \in \mathscr{C} \tag{4.24}$$

$$y_i \in \{0,1\} \qquad \forall i \in V \tag{4.25}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{4.26}$$

The binary decision variable y_i maintains its previous interpretation, equal to one if vertex i is part of the solution. Additionally, variable x_e is set to one if edge e is included in the solution. However, in this context, the symmetric edge is not needed. The objective function (4.20) seeks to maximize the number of vertices within the induced cycle. Constraint (4.21) guarantees that each vertex within the solution is connected to precisely two vertices in the cycle. Constraints (4.22) and (4.23) are in place to ensure that the cycle is induced. To eliminate solutions composed of small cycles from consideration, constraint (4.24) is introduced. $\mathscr{C}$ represents the set of the cycles for the given graph. Constraint (4.24) is combined into the model to enforce the solution to consist of a single cycle. This means that multiple small cycles are not deemed valid solutions.

4.2.5 Cordless-cycle model

The CCP formulation was introduced by Pereira *et al.* [74] to deal with the problem at hand. The formulation is described as follows:

$$\max \sum_{i \in V} y_i \tag{4.27}$$

subject to

$$\sum_{e \in E} x_e = \sum_{i \in V} y_i \tag{4.28}$$

$$\sum_{i \in V} y_i \geq 4 \tag{4.29}$$

$$\sum_{e \in \delta(i)} x_e = 2y_i \qquad \forall\, i \in V \tag{4.30}$$

$$x_e \leq y_i \qquad \forall\, i \in V, e \in \delta(i) \tag{4.31}$$

$$x_e \geq y_i + y_j - 1 \qquad \forall\, e = (i,j) \in E \tag{4.32}$$

$$\sum_{g \in \delta(C)} x_g \geq 2(y_i + y_j - 1) \qquad C \subset V, i \in C, j \in V \setminus C \tag{4.33}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{4.34}$$

$$y_i \in \{0,1\} \qquad \forall i \in V \tag{4.35}$$

The formulation includes the usual sets of binary variables: y_i and x_e, indicating whether vertex i and edge e are in the cycle or not. Consequently, the number of selected vertices and edges is equal, as required by (4.28), and at least four vertices must be selected by (4.29). Each vertex within the solution is incident to precisely two edges, as guaranteed by (4.30). Furthermore, (4.31)–(4.32) ensures that any solution is an induced subgraph of G. More specifically, any edge of G with both its endpoints belonging to the solution must be part of the solution. Moreover, the subgraph defined by x and y remains connected, as guaranteed by the subtour elimination constraint (4.33).

The CCP formulation was employed by the authors of [74], along with various valid inequalities. They introduced nine *B&C* algorithms and subsequently chose the top three among them. The first one, labeled as $BC1$ contains constraints (4.28)-(4.35), and in addition the following constraint:

$$\sum_{g\in\delta(C)} x_g \geq 2x_e \quad C\subset V, e=(i,j)\in E, i\in C, j\in V\setminus C \tag{4.36}$$

This algorithm initiates by separating (4.33), and subsequently, the resulting inequality is enhanced to the more robust form of (4.36). This specific constraint ensures that if $x_e = 1$, then it is mandatory for $y_i = y_j = 1$ to hold true, due to the presence of inequalities (4.31)-(4.32).

For the $BC2$ and $BC3$ algorithms, both constraints (4.37) and (4.38) were included together with constraints (4.28)-(4.36).

$$\sum_{i\in Q} y_i \leq 2 \tag{4.37}$$

$$\sum_{e\in E(Q)} x_e \geq \sum_{i\in Q} y_i - 1 \tag{4.38}$$

For a clique $Q\subset V$, $|Q|\geq 3$. Constraint (4.37) ensures that within a clique Q at most two of its vertices can be part of the induced cycle. On the other hand, constraint (4.38) guarantees that for a clique Q the number of vertices that can be part of the induced cycle is limited to at most one more than the number of edges that can be included from the clique. Namely, only one of the edges from Q might be included in the solution.

For the $BC2$ algorithm, they implemented a rule that imposes no restrictions on the number of separation rounds. In other words, whenever a violated inequality is detected, it is included in the cut pool. Conversely, for $BC3$, a fixed number of separation rounds, specifically 30, was established, and inequalities were added to the cut pool if a clique did not share two or more vertices with a clique in a previously accepted inequality. The order of inequalities in the cut pool was determined by descending order of the absolute values of their corresponding linear programming relaxation dual variables. All three algorithms utilized the lower bounds obtained from the multi-start CCP heuristic, which is a constructive procedure that takes a predefined edge as input data. The algorithm then seeks to extend a tentative path, containing the selected vertices. Vertices are added to the path one at a time, accepted if they are adjacent to one of the path's current extremities and not adjacent to any internal vertices. The procedure terminates when the endpoints of the path meet, resulting in a chordless cycle of G, or when further expansion of the path becomes impossible.

4.3 Algorithms

Out of the three models only *LIC* (and *LIC*2) can be directly solved using any *MILP* solver. Both ILP_{cut} and *cec* rely on the set of small cycles, which are usually created as part of the solution process, either through an iterative cut generation approach or, more effectively, via *B&C* algorithm by employing separation. Note that subtour elimination inequalities (4.18) and (4.24), present in the ILP_{cut} and *cec* models respectively, exhibit exponential complexity. Consequently, attempting to enumerate all inequalities corresponding to each subtour within the graph and subsequently cutting them becomes impractical. Instead, these inequities can be combined into the ILP_{cut} and *cec* models upon encountering them. Hence, the cut generation approach is employed as follows: the method is initiated with a model relaxing all subtour elimination inequalities, and if subtours arise in integer solutions, violated inequalities are added, and this process is repeated until the optimal solution is reached. For that, callback functionality from Gurobi [43] was employed, which can be used to add these inequalities iteratively. The Depth-First Search (*DFS*) algorithm is employed on the induced subgraph of the integer solution to identify cycles, subsequently introducing a new inequality for each subtour discovered. The entire procedure, which combines the models and cut generations, is shown in Algorithms 5 and 6.

4.3.1 Initialization

In the initialization phase of the procedure, ILP_{cut} and *cec* models are created, encompassing the creation of their variables, constraints, and objective functions.

4.3.2 Cut generation

To tackle the ILP_{cut} and *cec* models, the cut generation mechanism combined with the *B&B* algorithm, as explained earlier. Consequently, each model was addressed using two distinct approaches, as outlined below.

First approach

The first approach involves cut generation as outlined in Algorithm 5. In each iteration, a subproblem from the *B&B* tree is solved. In line 4 the algorithm checks if the solution of the subproblem is an integer solution. Based on this, the *DFS* is employed to detect any subtours within the solution, as shown in line 5. If a subtour exists, and its length is less than or equal to the value of the variable `longest_induced_cycle`, a cut is appended for that cycle. If not, the value of the variable is updated to reflect the length of the cycle, and there is no need to introduce a cut because the cycle could potentially be the optimal solution. These details are clarified in lines 6 through 9. The cut generation terminates when there are no further subtours present in the solution, indicating the completion of the procedure.

Algorithm 5: Cut_Generation1

```
1  model Initialization()      // initializing the $ILP_{cut}$ or $cec$ model
2  longest_induced_cycle=0
3  Function Cut_Generation1()
4      if model.status==feasible_integer then   // has integer solution
5          C=DFS(feasible_integer)      // find subtour in the solution
6          if length(C) $\leq$ longest_induced_cycle then
7              model.addConstr(4.18,4.24)      // add cut (4.18) or (4.24)
8          else
9              longest_induced_cycle=length(C)   // update variable value
10 model.optimize(Cut_Generation1())             // solve the model
11 print(longest_induced_cycle)
```

Second approach

The second cut generation-based approach is detailed in Algorithm 6. In each iteration, a subproblem is solved, and if an integer solution is obtained, the algorithm verifies the presence of any subtours using *DFS* as described in lines 15 through 16. If any cycles are detected, a cut is integrated into the model (line 17), and the length of the cycle is updated if it exceeds the value of the variable `longest_induced_cycle` (line 19). It is important to note that to further improve the procedure, a constraint is added to the model in line 20. This constraint ensures that the objective value must be greater than the length of the longest induced cycle discovered so far. Using this cut generation leads to the problem becoming infeasible, yet the longest induced cycle length is recorded in the variable `longest_induced_cycle`.

Algorithm 6: Cut_Generation2

```
12 model Initialization()      // initializing the $ILP_{cut}$ or $cec$ model
13 longest_induced_cycle=0
14 Function Cut_Generation2()
15     if model.status==feasible_integer then   // has integer solution
16         C=DFS(feasible_integer)      // find subtour in the solution
17         model.addConstr(4.18,4.24)        // add cut (4.18) or (4.24)
18         if length(C) $>$ longest_induced_cycle then
19             longest_induced_cycle=length(C)
20         model.addConstr(model.ObjVal $\geq$ longest_induced_cycle+1)
21 model.optimize(Cut_Generation2())             // solve the model
22 print(longest_induced_cycle)
```

4.3.3 Longest Isometric Cycle

Lokshtanov's algorithm, as described in [58], aims to identify the longest isometric cycle within a graph. By the definition of an isometric cycle, as discussed in Section 4.1, if a given graph G contains an isometric cycle with a length of ℓ, then there must also exist an induced cycle within the graph with a length of m where $m \geq \ell$. Consequently, the longest isometric cycle serves as a benchmark for the longest induced cycle. The algorithm's objective is to verify the existence of an isometric cycle with a length of k in a given graph $G = (V,E)$. If such a cycle exists, the graph G can be employed to construct a new graph G_k with vertices as vertex pairs of G. Namely, $V(G_k) = \{(u,v) \in V : d_{uv} = \lfloor k/2 \rfloor\}$, where d_{uv} is the length of the shortest path between u and v, and its edge set given by $E(G_k) = \{((u,v),(w,x)) : (u,w) \in E(G) \wedge (v,x) \in E(G)\}$. The method is outlined in Algorithm 7. For a given value of k, the algorithm computes the graph G_k and examines whether there exists a pair of vertices (u,v) and (v,x) within $V(G_k)$ such that (v,x) belongs to the set $M_k(u,v) := \{(u,x) : (u,x) \in V(G_k) \wedge (v,x) \in E(G)\}$ and $d_{G_k}[(u,v),(v,x)] = \lfloor k/2 \rfloor$. If such a pair is found, it indicates the presence of an isometric cycle with a length of k.

Algorithm 7: Longest Isometric Cycle

```
LISC=0
for $\forall l \in V, i \in V, j \in V$ do                          // distance calculation
    $d_{ij} = \min\{d_{ij}, d_{il} + d_{lj}\}$                        // by Floyd algorithm
if $G$ is a tree then                                    // no cycles in tree graph
    return LISC
for $k = 3 \to n$ do
    $V_k = \emptyset$                                               // vertices of $G_k$
    for $u \wedge v \in V$ do
        if $d_{uv} = \lfloor k/2 \rfloor$ then
            $V_k = V_k \cup \{(u,v)\}$
    $E_k = \emptyset$                                               // edges of $G_k$
    for $(u,v) \wedge (w,x) \in V_k$ do
        if $(u,w) \in E \wedge (v,x) \in E$ then
            $E_k = E_k \cup \{((u,v),(w,x))\}$
    $G_k = (V_k, E_k)$
    for $(u,v,x) \in V$ do
        if $(u,v) \in V_k \wedge (v,x) \in M_k(v,u) \wedge d_{G_k}[(u,v),(v,x)] = \lfloor k/2 \rfloor$ then
            LISC:= $k$
print(LISC)
```

4.4 Numerical experiments

To demonstrate and evaluate the effectiveness of the proposed methods, numerical results were presented for three models: the *LIC* model, the ILP_{cut} model, and the *cec* model. Furthermore, a comparison was conducted between the *cec* model and models from [74] on randomly generated graphs to highlight the efficiency of the proposed approach in comparison to existing methods.

4.4.1 Computational environment and dataset

The algorithms detailed in Section 4.3 were implemented in Julia 1.7.0, utilizing the JuMP package version 0.22.1. Gurobi 9.5.0 was used as the solver for all experiments. Each run was constrained to a one-hour time limit and a single thread. For the longest isometric cycle algorithm, the implementation is done using Python 3.8 with a 24-hour time limit. These computations were performed on a computer with an Intel Core i7-4600U CPU, 8GB of RAM, and running the Windows 10 operating system.

To verify the efficacy of the proposed methods, two sets of network datasets were used. The first is the RWC set, comprising 19 real-world networks that encompass communication and social networks within companies, networks of book characters, as well as transportation, biological, and engineering networks, as described in [62]. Additionally, the Movie Galaxy (MG) set, consisting of 773 graphs that represent social networks among movie characters, as detailed in [50]. For further information about these instances, refer to the following link: `http://tcs.uos.de/research/lip`.

To perform a comparison with the results presented in [74], the experiments were conducted on random graphs with varying values of n ranging from 50 to 100, considering both 10% and 30% density. For every case, 10 graphs were generated. Every run was restricted to a maximum duration of one hour, with no restrictions on the number of threads, and with an initial solution set to 4, as described in [74]. Regarding the hardware comparison, the information available in the following link (`https://www.cpubenchmark.net/compare`) was utilized to collect the details of the CPU used in all experiments, as outlined in Table 4.1. To ensure a fair comparison, the execution times were normalized in all cases. The ratio between the single-thread ratings gives a good approximation of the relative speed. Therefore, the percentage value in the last row of Table 4.1 was calculated. The run time was then modified by multiplying it by the corresponding percentage factor.

4.4.2 Computational results

Table 4.2 presents the computational experiments conducted on the RWC instances. The second column displays the optimal solutions for each instance (opt). The third column shows the length of the longest isometric cycle (*LISC*). The fourth and fifth columns respectively indicate the number of vertices (N) and edges (M) within the corresponding graph. Columns

Table 4.1: *CPU performance comparison between the CPU used in this work and in [74].*

Benchmarks	Intel Core i7-4600U	Intel Xeon W-3223
Clock Speed (GHz)	2.1	3.5
Turbo Speed (GHz)	Up to 3.3	Up to 4.0 GHz
Number of Physical Cores	2 (Threads: 4)	8 (Threads: 16)
Single Thread Rating	1641	2480
Percent	0.66	1

six through eleven show the time in seconds required to identify the optimal solution using the various methods employed in this study. Specifically, $ILP_{cut}2$ and $cec2$ refer to the methods outlined in Algorithm 6. For all these methods, the search was initiated using the *LISC* value as an initial solution, incorporating a constraint: `ObjVal` $\geq$ `LISC`. These results are indicated in every second row corresponding to each graph. Instances that resulted in timeouts are denoted by the symbol ⏲.

Table 4.2: *Running times on RWC instances, time is given in seconds.*

graph	opt	LISC	N	M	LIC	$LIC2$	ILP_{cut}	$ILP_{cut}2$	cec	$cec2$
high-tech	10	5	33	91	1.22	0.63	0.95	0.33	0.38	0.14
					0.99	0.42	1.15	1.57	1.02	0.77
karate	6	5	34	78	0.63	0.58	0.21	0.24	0.20	0.19
					0.66	0.53	0.23	0.29	0.24	0.17
mexican	13	7	35	117	0.92	0.82	0.66	0.88	0.24	0.20
					0.83	0.78	0.69	0.74	0.20	0.20
sawmill	6	5	36	62	0.54	0.43	**0.18**	0.37	**0.15**	0.10
					0.33	0.30	**0.36**	0.16	**0.13**	0.11
tailorS1	12	7	39	158	2.93	1.11	1.37	1.45	0.34	0.33
					1.38	0.89	1.76	1.76	0.37	0.44
chesapeake	15	5	39	170	1.01	0.69	0.56	0.81	0.24	0.22
					1.05	0.72	0.97	0.87	0.28	0.31
tailorS2	12	5	39	223	3.11	2.05	3.49	4.46	0.74	0.65
					3.25	3.09	3.74	4.62	0.83	0.75
attiro	28	9	59	128	0.93	1.24	0.55	0.53	0.18	0.24
					0.67	0.71	0.52	0.68	0.28	0.31
krebs	8	7	62	153	10.91	7.23	**1.19**	0.94	**0.94**	0.48

Table 4.2: *(continued) Running times on RWC instances, time is given in seconds.*

graph	opt	LISC	N	M	LIC	$LIC2$	ILP_{cut}	$ILP_{cut}2$	cec	$cec2$
					10.39	5.56	**0.86**	1.02	**0.57**	0.38
dolphins	20	7	62	159	14.75	23.70	1.81	2.35	1.70	1.02
					10.66	13.83	2.80	2.98	0.74	1.50
prison	28	9	67	142	5.90	10.22	**0.83**	1.56	**0.62**	0.61
					6.93	10.23	**4.81**	1.03	**0.66**	0.48
huck	5	5	69	297	519.95	299.12	**19.53**	17.79	**4.31**	4.51
					447.72	493.74	**18.22**	19.71	**3.34**	3.31
sanjuansur	35	11	75	144	6.16	5.85	0.68	0.71	0.37	0.49
					7.70	3.61	0.82	1.36	0.44	0.38
jean	7	5	77	254	276.98	147.77	15.93	14.57	2.45	2.41
					150	147.85	13.97	14.80	2.32	2.41
david	15	8	87	406	544.99	308.06	46.23	54.23	5.22	4.36
					219.14	323.96	45.24	38.33	3.31	3.47
ieeebus	32	13	118	179	2.52	5.14	0.76	0.94	0.43	0.62
					7.82	5.82	0.79	1.35	0.33	0.3
sfi	3	3	118	200	6.98	6.51	**0.74**	0.90	**0.3**	0.31
					6.40	2.80	**0.84**	1.32	**0.34**	0.31
anna	15	🕓	138	493	90.75	52.11	10.60	23.65	1.37	1.71
					-	-	-	-	-	-
494bus	116	🕓	494	586	108.48	126.77	27.13	33.09	2.73	2.10
					-	-	-	-	-	-
average					84.19	52.63	7.02	8.41	1.21	1.09
					51.52	59.70	5.75	5.45	0.9	0.92

The various methods exhibit diverse performance characteristics in terms of execution time and the number of instances solved optimally. Key observations from Table 4.2 are as follows:

- $cec2$ outperforms cec in 13 cases, ILP_{cut}, $ILP_{cut}2$, LIC and $LIC2$ in all the cases.
- $ILP_{cut}2$ was faster than $LIC2$ for 15 cases and LIC for all instances.

- *LIC*2 outperforms *LIC* in 14 cases.
- For some instances in ILP_{cut} and *cec*, the graphs and results are indicated by boldface and underlined in Table 4.2. This is to emphasize that these graphs contain multiple longest induced cycles of the same length, and the procedures described in Algorithm 5 cut the cycle if its length is less than or equal to the longest induced cycle found so far. Thus, for these graphs, all the longest cycles are found by the method.
- Using *LISC* as an initial solution does not contribute significantly to improving the execution time in the majority of cases.
- The results emphasize the correlation between graph density and execution time. Graph density is defined as the ratio of the edges present in a graph to the maximum number of edges it can hold. This relationship is particularly evident for dense graphs like huck, jean, and david, especially in the case of *LIC* and *LIC*2 models. However, it is not the case for *cec* and *cec*2 models as their running times show less sensitivity to the graph's density.

The results for the MG instances, organized into groups based on the number of edges, are presented in Table 4.3. Unlike *LIC* and *LIC*2, where the running times increase proportionally with the instance size, the results indicate that *cec* and *cec*2 are more reliable, with running times showing less sensitivity to the graph's size.

Table 4.3: *Running times on MG instances, time is given in seconds.*

nr. of edges	nr. of instances	*LIC*	*LIC*2	ILP_{cut}	$ILP_{cut}2$	*cec*	*cec*2
1–49	107	0.14	0.14	0.12	0.18	0.09	0.08
50–74	135	0.37	0.29	0.2	0.3	0.17	0.12
75–99	151	0.7	0.58	0.32	0.39	0.22	0.17
100–124	121	1.74	1.32	0.51	0.65	0.29	0.24
125–149	90	4.5	3.67	0.82	0.99	0.37	0.34
150–199	89	10.45	7.48	1.49	1.7	0.56	0.49
200–629	80	169.38	110.49	13.28	16.39	2.41	1.9
average		157.23	126.35	44.73	57.28	26.84	22.01

The results for the random graphs are presented in Table 4.4, where *cec*2 compared against the top three algorithms introduced in [74]. The runtime represents the average duration on the ten graphs in each case. Notably, *cec*2 outperforms these algorithms in all cases, even before normalizing the execution times with the ratio listed in Table 4.1. Moreover, *cec*2 successfully solved the instance with 100 vertices and 30% density, a scenario where none of the algorithms in [74] succeeded.

Table 4.4: *Running times on random instances for* cec2, BC*1,* BC2*, and* BC*3, time is given in seconds. Values in brackets show the original execution time of* cec2*.*

	Randomly generated graphs: 10% density			
n	$cec2$	$BC1$	$BC2$	$BC3$
50	0.32 (0.49)	0.33	0.3	0.39
60	0.79 (1.2)	1.19	1.2	1.34
70	4.17 (6.32)	5.57	4.93	5.58
80	20.5 (31.1)	37.15	26.9	27.34
90	93.93 (142.32)	160.1	155.82	168.02
100	518.75 (785.99)	1321.41	1129.47	1094.8
average	106.41	254.29	219.77	216.25
	Randomly generated graphs: 30% density			
n	$cec2$	$BC1$	$BC2$	$BC3$
50	4.15 (6.28)	8.21	9.6	8.78
60	26.14 (39.6)	39.82	46.18	51.9
70	90.11 (136.52)	234.45	206.36	283.28
80	203.81 (308.8)	935.78	676.52	1139.55
90	544.88 (825.58)	2072.16	1874.91	3011.17
100	1810.49 (2743.17)	⏲	⏲	⏲
average	446.6	658.08	562.71	898.94

4.5 Concluding remarks

Considering the NP-hardness of the longest induced cycle problem, it is crucial to find an efficient method that can provide optimal solutions within a reasonable time. Standard methods have been introduced and evaluated against existing methods from the literature to address this problem.

The author of this PhD thesis is responsible for the following contributions presented in this chapter:

IIII/1. I proposed three integer linear programs, some of which are extensions of models created for solving the longest induced path problem. The proposed programs had varying execution times and the number of instances they were able to solve optimally.

IIII/2. I conducted a comparison between my proposed methods and the methods proposed previously in the literature, and the results demonstrated that the newly introduced methods consistently outperformed those presented in the literature.

Chapter 5

Maximizing the Smallest Eigenvalue of the Grounded Laplacian Matrix

5.1 Introduction

The most basic approach to describing graphs involves their topological representation, which defines a graph as a set of vertices and edges. Nevertheless, employing spectral representations, like the Adjacency matrix or Laplacian matrix, can greatly enhance the ability to describe the structural and functional characteristics of the graph. Laplacian matrix (L) is a symmetric, positive semidefinite matrix. A widely recognized fact is that it comprises n eigenvalues, which are real numbers and are always non-negative. These eigenvalues are constrained within a range, bounded by twice the maximum vertex degree, as clarified in [11]. Consequently, $0 \leq \mu_1 \leq \mu_2 \leq \ldots \leq \mu_n \leq 2 \cdot \max_{i \in V} deg_i$, with μ_i representing the i-th eigenvalue of L.

According to Mohar *et al.* [70], the eigenvalues of the Laplacian matrix find utility across a wide range of disciplines. One of their primary applications lies within the domain of graph theory, where they have significant implications. Specifically, the count of spanning trees in a graph G can be demonstrated by the product of all non-zero eigenvalues of L, as noted in [56]. Furthermore, the aggregate resistance distances between all pairs of vertices can also be computed using the eigenvalues of L, as highlighted in [51]. Another notable aspect of L is the Fiedler value, introduced by Fiedler [37], which corresponds to the second smallest eigenvalue. The Fiedler value plays a pivotal role in determining the graph's connectivity, as a graph is considered connected when its Fiedler value is greater than zero. Lastly, the number of components within G is equivalent to the multiplicity of the zero eigenvalues of L.

Consider the graph $G = (V, E)$ along with its Laplacian matrix. The grounded Laplacian matrix, denoted as $L(S)$ and introduced in [65], constitutes a submatrix of dimensions $(n-k) \times (n-k)$. It is derived by removing k rows and their corresponding columns from L, where $S \subset V$, $|S| = k$, and $0 < k \ll n$. The minimum eigenvalue of $L(S)$ is represented as

$\mu(S)$. It is important to note that $L(S)$ is a symmetric positive definite matrix, which implies that all of its eigenvalues are strictly positive real numbers. Consequently, $\mu(S) > 0$. The smallest eigenvalue within $L(S)$ plays a crucial role in determining the convergence rate of leader-follower networked dynamical systems, as discussed in [77]. It also influences the effectiveness of the pinning control scheme for complex dynamical networks, as explored in [57]. A bigger $\mu(S)$ corresponds to a faster convergence rate and improved performance in pinning control.

It is worth mentioning that identifying the $L(S)$ with the maximum possible $\mu(S)$ has been proven to be a computationally challenging problem, classified as NP-hard, as outlined in [87]. Wang *et al.* [87] introduced two greedy-type algorithms for this purpose. The first, known as the NAÏVE algorithm, entails a k iterations. During each iteration, a candidate vertex is selected if its inclusion in set S maximizes $\mu(S)$. The second algorithm referred to as the FAST algorithm, assesses candidate vertices based on the sum of eigenvalues of their adjacent vertices. The optimal candidate is determined by identifying the maximum sum value. To compute these eigenvalues, the eigenvector corresponding to the smallest eigenvalue of $L(S)$ is employed, and this is approximated using the SDDM solver [31]. Importantly, this approach avoids the need to calculate the entire eigensystem. In this work, which is published in [7], two algorithms have been proposed to deal with the problem at hand. The first algorithm relies on the centrality of the vertices to select elements of set S, while the second algorithm is based on the vertex cover problem.

5.2 Methodologies

5.2.1 First approach

The first approach, referred to as DEGREE-G, draws inspiration from the well-known Gerschgorin circle theorem [41]. The theorem provides bounds on the eigenvalues of a square matrix. It claims that each eigenvalue of a square matrix resides within at least one Gerschgorin circle, with each circle corresponding to a row in the matrix. In this representation, the center of the circle corresponds to the diagonal element of the matrix, while the radius is determined by the sum of the absolute values of the off-diagonal elements. Figure 5.1 visually illustrates the Gerschgorin circles for a Laplacian matrix.

The key idea for maximizing the smallest eigenvalue is to move the Gerschgorin circles further away from the origin. To achieve this, the proposed method ranks the vertices based on a specific centrality measure and subsequently removes the corresponding row and column from the Laplacian matrix. Algorithm 8 outlines this approach, this approach utilizing degree centrality for this purpose.

To illustrate this method, consider applying it to the L shown in Figure 5.1 with $k = 2$, resulting in $\mu(S) = 1.27$ with $S = \{2,5\}$, as demonstrated in Figure 5.2. In contrast, the NAÏVE algorithm from [87] yields the set $S = \{2,4\}$ and $\mu(S) = 1.2$. Note that neither of

$$L = \begin{pmatrix} 3 & -1 & -1 & -1 & 0 \\ -1 & 2 & 0 & -1 & 0 \\ -1 & 0 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

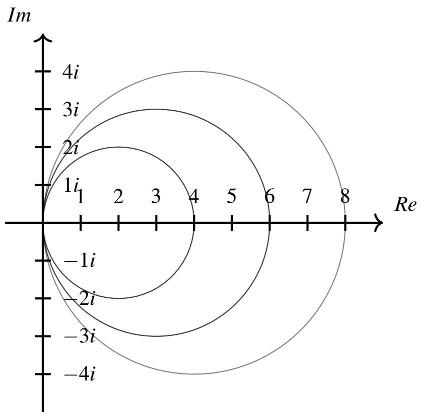


Figure 5.1: *A Laplacian matrix (left) and its Gershgorin circles (right)*

Algorithm 8: Degree-G Algorithm

```
1 $vertex\_cen = sort(centrality(V))$
2 for $i = 1 \to k$ do
3     $remove(L(vertex\_cen[i]))$
4 $compute(\mu(S))$
```

these methods could achieve the optimal solution, which, in this simple example, is $\mu(S) = 1.47$ with $S = \{1,5\}$

$$L = \begin{pmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 4 \end{pmatrix}$$

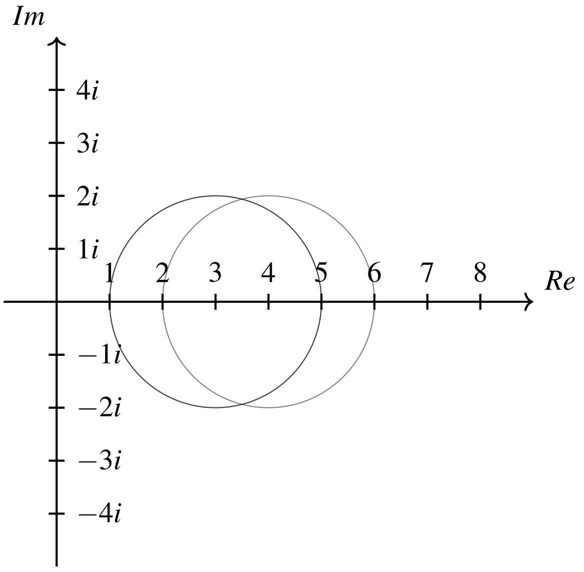


Figure 5.2: *Illustration of the result of Algorithm 8 using degree centrality.*

5.2.2 Second approach

The second method known as COVER, takes advantage of the Gerschgorin circles, but it utilizes the concept of the maximum k vertex cover problem. This problem is rooted in the classic vertex cover problem [27], with the key distinction being that in the k vertex cover problem, the objective is to identify a set of k vertices that are incident to the maximum number of edges in the graph. This differs from the vertex cover problem, which aims to find the minimum number of vertices such that each edge in the graph is incident to at least one of these vertices. The *BILP* for the vertex cover is formulated as follows:

$$\min \sum_{i \in V} x_i \tag{5.1}$$

subject to

$$x_i + x_j \geq 1 \qquad \forall (i, j) \in E \tag{5.2}$$

$$x_i \in \{0, 1\} \qquad \forall i \in V \tag{5.3}$$

Note that the binary variables x_i represent the minimum number of vertices that incident to all the edges in the graph. The *BILP* of k vertex cover is defined as follows:

$$\max \sum_{j \in V} y_j \tag{5.4}$$

subject to

$$\sum_{i \in V} x_i = k \tag{5.5}$$

$$y_j \leq \sum_{\forall i \in V : (j,i) \in E} x_i \qquad \forall j \in V \tag{5.6}$$

$$k \in \mathbb{N} \tag{5.7}$$

$$x_i, y_i \in \{0, 1\} \qquad \forall i \in V \tag{5.8}$$

Once more, in this *BILP*, the variables x_i represent vertices that cover the maximum number of vertices within the graph, while y_i denotes vertices that have been covered. The constraints are designed to guarantee that only k vertices can be chosen and that the value of y_i is equal to 1 if and only if at least one of its adjacent vertices, represented by x_i, is selected. After successfully solving this *BILP*, the algorithm proceeds to remove the rows and columns in L corresponding to the solution. Subsequently, it determines the smallest eigenvalue of the resulting modified matrix.

Additionally, combining vertex cover and degree centrality concepts has the potential to enhance the outcomes. Consequently, the objective function in the *BILP* has the following modifications:

$$\max \sum_{j \in V} y_j - \varepsilon \sum_{j \in V} deg_j x_j, \tag{5.9}$$

where ε is a small number to not change the main objective. For instance $\varepsilon = \frac{1}{\sum_{j \in V} deg_j}$ can be chosen. This modification aims to select k vertices with the lowest degree to maximize the objective. This approach is defined as COVER1.

The maximum k vertex cover problem often yields multiple solutions, starting to employ a method to explore the possibility of obtaining an improved value for $\mu(S)$ through alternate solutions. The concept involves an iterative process of solving the *BILP* while including a constraint that prevents the solution from resembling previous ones. By examining various solutions, different values for $\mu(S)$ can be obtained, and select the one that yields the best result. This additional constraint is formulated as follows for a given previously obtained solution $S \subset V$:

$$\sum_{i \in S} x_i \leq k - 1. \tag{5.10}$$

During each iteration, the algorithm ensures that the solution covers the maximum possible number of vertices, denoted as *nc*, which means that $\sum_{i \in V} y_i = nc$ must hold; otherwise, the algorithm terminates the iteration. This approach is referred to as COVER2. The maximum number of iterations, and consequently the maximum number of alternative solutions, is fixed at 100.

5.3 Numerical results

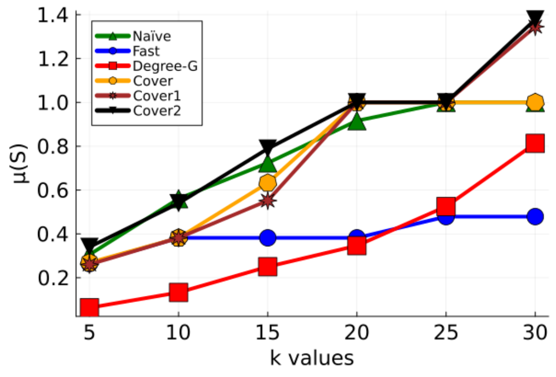
To evaluate and compare the effectiveness of the proposed methods, a comparative analysis against the two algorithms introduced by Wang *et al.* [87] was conducted. All these algorithms were implemented using Julia 1.7.0 along with the JuMP 0.22.1 package. The Gurobi 9.5.0 solver was utilized on a computer with an Intel Core i7-4600U CPU and 8GB RAM, running the Windows 10 operating system. The experiments were carried out on real-world networks, all of which are publicly available through KONECT [54], SNAP [55], and RWC[1].

Results for different *k* values. Figure 5.3 displays the outcomes obtained by applying the proposed approaches to a range of real-world graphs. The figure presents the smallest eigenvalue, denoted as $\mu(S)$, for six distinct values of k.

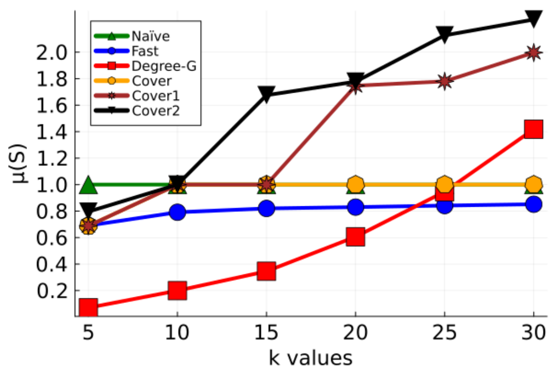
The displayed results illustrate that the effectiveness of these methods varies across different graphs, yet certain patterns arise. It is evident that the DEGREE-G and FAST methods

[1]`http://tcs.uos.de/research/lip`

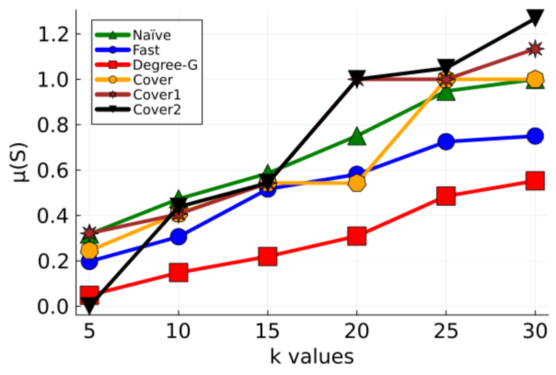
exhibit inferior performance compared to the NAÏVE and COVER methods. Notably, the NAÏVE method consistently outperforms the other techniques, although the COVER method performs equally well or even better than the NAÏVE method at specific values of k.



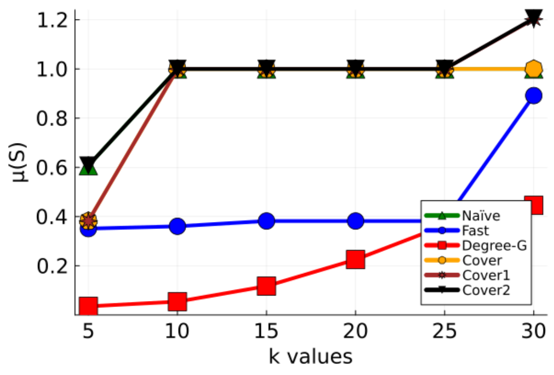
(a) *Prison*



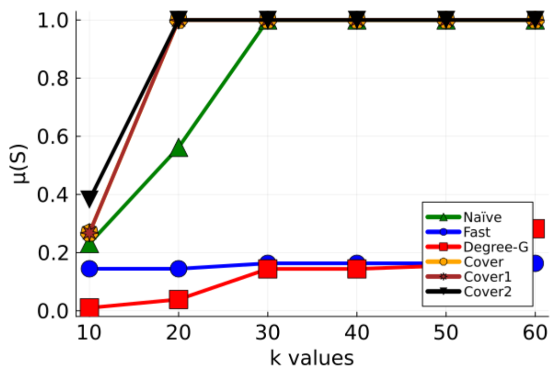
(b) *Huck*



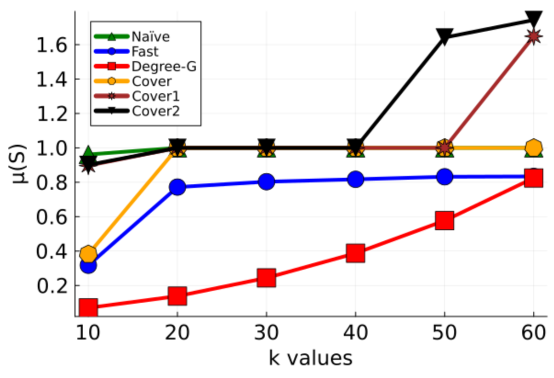
(c) *Sanjuansur*



(d) *Jean*



(e) *Sfi*



(f) *Anna*

Figure 5.3: *Values of $\mu(S)$ obtained by the algorithms for different k values.*

Results for specific k values. Instead of employing arbitrary values for k a more systematic approach is used by utilizing the vertex cover *BILP* to identify the smallest value of k that guarantees a substantial increase in the value of $\mu(S)$. Table 5.1 presents the corresponding values of k and $\mu(S)$ for various real-world graphs using the diverse methods discussed. It is evident that the NAÏVE and COVER methods outperform the DEGREE-G and FAST methods. Interestingly, the COVER methods consistently match or even outperform the performance of the NAÏVE method across all cases.

Table 5.1: *Values of $\mu(S)$ for k value obtained from vertex cover.*

graph	N	M	k	NAÏVE	FAST	DEGREE-G	COVER	COVER1	COVER2
Prison	67	142	41	1	0.48	1.97	1.59	1.63	2.38
Huck	69	297	44	1	1	1.7	1	2.48	3.5
Sanjuansur	75	144	40	1.59	0.84	0.95	1.29	1.27	1.59
Jean	77	254	42	1	0.37	0.67	1	1.24	1.24
David	87	406	51	1	1	2.45	3.44	2.65	2.77
ieeebus	118	179	61	1	0.59	0.73	1	1.06	1.2
Sfi	118	200	53	1	0.27	0.24	1	1	1
Anna	138	493	58	1	0.83	0.87	1	1.65	1.65
Usair	332	2126	149	1	0.74	1	1	1.59	1.59
494bus	494	586	216	0.38	0.07	0.14	1	1	1
average				0.99	0.62	1.07	1.33	1.56	1.79

Furthermore, a time comparison among the different methods was conducted. Table 5.2 displays the runtimes for $k = 5$ using each method, with a time limit of one hour. The results indicate that the DEGREE-G and FAST methods exhibit significantly shorter execution times compared to the NAÏVE and COVER methods. Notably, the NAÏVE and COVER algorithms exceeded the time limit for graphs with thousands of vertices and edges, emphasizing that the COVER methods generally outperform NAÏVE in terms of task completion speed.

Table 5.2: *The time (in seconds) to compute $\mu(S)$ for $k = 5$.*

graph	N	M	NAÏVE	FAST	DEGREE-G	COVER	COVER1	COVER2
Prison	67	142	0.63	0.005	0.002	0.031	0.028	1.14
Huck	69	297	0.78	0.008	0.003	0.036	0.038	5.69
Sanjuansur	75	144	1.00	0.008	0.003	0.031	0.036	7.71
Jean	77	254	1.02	0.008	0.004	0.035	0.034	13.62
David	87	406	1.42	0.014	0.008	0.041	0.041	2.21
ieeebus	118	179	2.87	0.011	0.003	0.043	0.042	9.13
Sfi	118	200	2.71	0.011	0.008	0.041	0.041	7.72
Anna	138	493	3.88	0.016	0.014	0.067	0.066	14.54
Usair	332	2126	29.77	0.049	0.034	0.651	0.513	21.86
494bus	494	586	102.98	0.057	0.051	0.382	0.323	15.71
Email-Univ	1133	5451	1589.44	0.550	0.531	13.632	17.497	104.34
Routers-RF	2113	6632	>3600	1.448	1.515	55.288	57.114	273.0
US-Grid	4941	6594	>3600	14.255	15.551	312.706	315.643	1947.85
WHOIS	7476	56943	>3600	50.343	50.425	>3600	>3600	>3600
PGP	10680	24340	>3600	140.943	143.747	>3600	>3600	>3600
average			1075.79	13.852	14.126	505.532	506.094	641.63

5.4 Concluding remarks

Given the fact that maximizing the smallest eigenvalue of the grounded Laplacian matrix is a complicated problem, it is desirable to develop efficient algorithms that can provide solutions of acceptable quality within a reasonable time. Two approaches were introduced and demonstrated experimentally.

The author of this PhD thesis is responsible for the following contributions presented in this chapter:

IV/1. I proposed two algorithms to address the problem: the first one, named DEGREE-G relies on vertex centrality, while the second one, called COVER is based on the maximum vertex cover problem.

IV/2. Both algorithms, as evidenced by experimental results, consistently produced promising solutions within a reasonable time, emphasizing their competitiveness when compared to existing algorithms in the literature.

Chapter 6

Appendix

The best formulas, obtained by *SR* and *CGP* in chapter 3, are listed for the different graph types and graph properties, concerning mean absolute deviation.

Formulas for random graphs diameter. The results obtained by these formulas are shown in Table 3.2.

$$\sqrt{\frac{n}{\lambda_1}} + \sqrt{\frac{2n}{\lambda_1} - \lambda_3 + 1} \tag{6.1}$$

$$\frac{N(M+\mu_{N-2})^3(2N^3 + (M+\mu_{N-2})^3(N+\mu_1))}{(N^3 + \mu_1(M+\mu_{N-2})^3)^2} \tag{6.2}$$

$$\frac{N + 2\sqrt{\lambda_1} - 2\lambda_3}{\lambda_1} \tag{6.3}$$

$$\frac{N(M+N-\mu_{N-2})}{2(N+\mu_1\mu_4)} \tag{6.4}$$

$$\frac{N + \lambda_{N-2} + 4}{\sqrt{M}} \tag{6.5}$$

$$\frac{\mu_{N-2}(3N-\mu_1)}{N\mu_N + \mu_1\mu_{N-2}} \tag{6.6}$$

Formulas for real-world graphs diameter. The results obtained by these formulas are shown in Table 3.3.

$$\mu_{N-3} + \mu_{N-4} - 2 + 4(\mu_{N-4} - 2)^2 - \frac{\mu_{N-1}(\mu_{N-3} + \mu_{N-4} + \sigma)}{5} \tag{6.7}$$

$$\sqrt{\frac{2\lambda_4^3}{(\lambda_1+\lambda_{N-4})^3}+\frac{(\lambda_1+N+\sigma)}{(\lambda_1+\lambda_{N-4})}+\lambda_N} \tag{6.8}$$

$$\left((\mu_{N-4}-2)^2+\frac{(\mu_{N-4}^2+1)}{\frac{\mu_2^2}{125}+\mu_5-(\mu_{N-4}-2)^2}\right)^2 \tag{6.9}$$

$$((\sqrt{(3-\sqrt{\mu_{N-4}})})^2((3-\sqrt{\mu_{N-4}})^2-2)) \tag{6.10}$$

$$\frac{N+2\mu_{N-3}\gamma+\mu_2+\mu_5+\sigma-2(\mu_{N-4}-5)^3}{\mu_{N-3}\gamma+\mu_2+\mu_3+\mu_{N-4}+\sigma} \tag{6.11}$$

$$\mu_{N-3}-2\mu_{N-4}^2-((\mu_{N-4}-1)^3-\sqrt{3})^3 \tag{6.12}$$

$$\sqrt{2}\sqrt{\lambda_N+\frac{\lambda_1^3}{(\lambda_1+\lambda_{N-4})^3}+\frac{\lambda_1+N+\sigma}{\lambda_1+\lambda_{N-4}}} \tag{6.13}$$

$$\frac{2M}{\lambda_1\lambda_2^2}+\frac{\lambda_5^2+2(\lambda_N-\lambda_3)+50}{\lambda_1}+\frac{2}{\lambda_1\lambda_2} \tag{6.14}$$

Formulas for random graphs geodetic number. The results obtained by these formulas are shown in Table 3.4.

$$\sqrt{\frac{N^{3/2}}{\lambda_1}-\frac{\lambda_{N-4}N^{3/2}}{\lambda_1^2+N^{3/2}}}. \tag{6.15}$$

$$\frac{\mu_4^2}{\mu_2\mu_{N-3}}+\sqrt{N-\mu_3} \tag{6.16}$$

$$\frac{N\lambda_4}{3\lambda_1}+\sqrt{5} \tag{6.17}$$

$$\frac{5(N\mu_4+5\mu_1-5\mu_{N-3})}{\mu_4^2+10\mu_4+25} \tag{6.18}$$

Formulas for real-world graphs geodetic number. The results obtained by these formulas are shown in Table 3.5.

$$\gamma+\sigma+\sqrt{\gamma}-\lambda_2-\lambda_3\lambda_{N-1}-\lambda_{N-2}\lambda_{N-4}+\frac{N}{\gamma} \tag{6.19}$$

$$\frac{\gamma^2(\gamma+\sigma)+}{(\gamma^2+N)}+\sqrt{\gamma+\sigma+\lambda_N+\frac{N}{\gamma}+\frac{N}{\gamma}+\frac{N}{\gamma^2}}+1 \tag{6.20}$$

$$\gamma+\sigma+\lambda_{N-4}+\sqrt{N}+\frac{\lambda_N}{\gamma}+\frac{N(\gamma+1)}{\gamma^2+\gamma\sigma} \tag{6.21}$$

$$\gamma+\sigma+\sqrt{\gamma}-\frac{\mu_2\mu_{N-3}}{\mu_{N-4}}+\mu_2+\frac{N}{\gamma}-\sqrt{\frac{\gamma^2}{\mu_2 N}+\frac{\gamma\sigma}{\mu_2 N}} \tag{6.22}$$

$$\gamma+\sigma-\frac{\gamma}{\sqrt{N}}-\mu_2\mu_{N-2}+\mu_2\mu_{N-4}+2\mu_4-2\mu_5-\mu_{N-2}+\mu_{N-4}+\sqrt{N} \tag{6.23}$$

$$\gamma+\sigma-2\mu_{N-2}+\sqrt{-2\mu_5+N}-8+\frac{\mu_2}{\mu_{N-2}+3}+\frac{N}{\gamma} \tag{6.24}$$

$$\gamma+\sigma+\frac{\mu_1}{\gamma+\sigma}-\mu_2\mu_{N-2}+\mu_4+9\mu_{N-2}^2\mu_{N-3}^2+\sqrt{\mu_{N-4}}(\mu_2-\mu_5) \tag{6.25}$$

$$\gamma+\sigma+\sqrt{M}-2 \tag{6.26}$$

Bibliography

[1] Mohamed Abdel-Basset, Laila Abdel-Fatah, and Arun Kumar Sangaiah. Metaheuristic algorithms: A comprehensive review. *Computational intelligence for multimedia big data on the cloud with engineering applications*, pages 185–231, 2018.

[2] H A Ahangar, F Fujie-Okamoto, and V Samodivkin. On the forcing connected geodetic number and the connected geodetic number of a graph. *Ars Combinatoria*, 126:323–335, 2016.

[3] Takanori Akiyama, Takao Nishizeki, and Nobuji Saito. NP-completeness of the Hamiltonian cycle problem for bipartite graphs. *Journal of Information processing*, 3(2):73–76, 1980.

[4] R Albert and A-L Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, 2002.

[5] Bushra Alhijawi and Arafat Awajan. Genetic algorithms: Theory, genetic operators, solutions, and applications. *Evolutionary Intelligence*, pages 1–12, 2023.

[6] Noga Alon. The longest cycle of a graph with a large minimal degree. *Journal of graph theory*, 10(1):123–127, 1986.

[7] Ahmad T Anaqreh, G Boglárka, and Tamás Vinkó. New methods for maximizing the smallest eigenvalue of the grounded laplacian matrix. In *Proceedings of the 12th International Conference on Applied Informatics (ICAI 2023)*, pages 1–9, 2023.

[8] Ahmad T Anaqreh, G Boglárka, Tamás Vinkó, et al. Symbolic regression for approximating graph geodetic number. *Acta Cybernetica*, 25(2):151–169, 2021.

[9] Ahmad T Anaqreh, Boglárka G.-Tóth, and Tamás Vinkó. Algorithmic upper bounds for graph geodetic number. *Central European Journal of Operations Research*, 30(4):1221–1237, 2022.

[10] Ahmad T. Anaqreh, Boglárka G. Tóth, and Tamás Vinkó. Exact methods for the longest induced cycle problem. *arXiv preprint arXiv:2311.15899*, 2023.

[11] W Anderson and T Morley. Eigenvalues of the laplacian of a graph. *Linear and Multilinear Algebra*, 18(2):141–145, 1985.

[12] M Atici. Computational complexity of geodetic set. *International journal of computer mathematics*, 79(5):587–591, 2002.

[13] ML Balinski and Albert W Tucker. Duality theory of linear programs: A constructive approach with applications. *Siam Review*, 11(3):347–377, 1969.

[14] Marc Barthelemy. Betweenness centrality in large complex networks. *The European physical journal B*, 38(2):163–168, 2004.

[15] Mindaugas Bloznelis and Valentas Kurauskas. Clustering function: another view on clustering coefficient. *Journal of Complex Networks*, 4(1):61–86, 2016.

[16] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

[17] Fritz Bökler, Markus Chimani, Mirko H Wagner, and Tilo Wiedera. An experimental study of ILP formulations for the longest induced path problem. In *International Symposium on Combinatorial Optimization*, pages 89–101. Springer, 2020.

[18] Phillip Bonacich. Some unique properties of eigenvector centrality. *Social networks*, 29(4):555–564, 2007.

[19] John Adrian Bondy. Basic graph theory: paths and circuits. *Handbook of combinatorics*, pages 1–2, 1995.

[20] Ralf Borndörfer. *Aspects of set packing, partitioning, and covering*. PhD thesis, 1998.

[21] B Brešar, S Klavžar, and A T Horvat. on the geodetic number and related metric sets in cartesian product graphs. *Discrete Mathematics*, 308(23):5555–5561, 2008.

[22] Boštjan Brešar, Matjaž Kovše, and Aleksandra Tepeh. Geodetic sets in graphs. In *Structural Analysis of Complex Networks*, pages 197–218. Springer, 2011.

[23] Hajo Broersma, Fedor V Fomin, Pim van't Hof, and Daniël Paulusma. Exact algorithms for finding longest cycles in claw-free graphs. *Algorithmica*, 65(1):129–145, 2013.

[24] F Buckley and F Harary. Geodetic games for graphs. *Quaestiones Mathematicae*, 8(4):321–334, 1985.

[25] D Chakraborty, F Foucaud, H Gahlawat, S K Ghosh, and B Roy. Hardness and approximation for the geodetic set problem in some graph classes. *Conference on Algorithms and Discrete Applied Mathematics*, 12016:102–115, 2020.

[26] G Chartrand, F Harary, and P Zhang. On the geodetic number of a graph. *Networks: An International Journal*, 39(1):1–6, 2002.

[27] Jianer Chen, Iyad A Kanj, and Weijia Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001.

[28] Yijia Chen and Jorg Flum. On parameterized path and chordless path problems. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*, pages 250–263. IEEE, 2007.

[29] Sathiyaraj Chinnasamy, M Ramachandran, M Amudha, and Kurinjimalar Ramu. A review on hill climbing optimization methodology. *Recent Trends in Management and Commerce*, 3(1), 2022.

[30] Jens Clausen. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30, 1999.

[31] Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. Solving sdd linear systems in nearly m log1/2 n time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 343–352, 2014.

[32] George B Dantzig. Origins of the simplex method. In *A history of scientific computing*, pages 141–151. 1990.

[33] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[34] T Ekim, A Erey, P Heggernes, P van 't Hof, and D Meister. Computing minimum geodetic sets of proper interval graphs. *LATIN 2012: Theoretical Informatics*, 7256:279–290, 2012.

[35] P Erdős and A Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[36] Martin G Everett and Stephen B Seidman. The hull number of a graph. *Discrete Mathematics*, 57(3):217–223, 1985.

[37] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

[38] Kernighan B. Fourer, R. A modeling language for mathematical programming. *Duxbury Press*, 2002.

[39] Elena D Fuchs. Longest induced cycles in circulant graphs. *the electronic journal of combinatorics*, pages R52–R52, 2005.

[40] Michael R Garey. Computers and intractability: A guide to the theory of np-completeness, freeman. *Fundamental*, 1997.

[41] Gene H Golub and Charles F Van Loan. *Matrix computations*. Johns Hopkins University Press, 2013.

[42] Ralph Edward Gomory. *An algorithm for the mixed integer problem*. Rand Corporation California, 1960.

[43] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[44] Adriana Hansberg and Lutz Volkmann. On the geodetic and geodetic domination numbers of a graph. *Discrete mathematics*, 310(15-16):2140–2146, 2010.

[45] P Hansen and N van Omme. On pitfalls in computing the geodetic number of a graph. *Optimization Letters*, 1(3):299–307, 2007.

[46] F Harary, E Loukakis, and C Tsouros. The geodetic number of a graph. *Mathematical and Computer Modeling*, 17(11):89–95, 1993.

[47] F Harary and J Nieminen. Convexity in graphs. *J. Differential Geom*, 16(2):185–190, 1981.

[48] Adeel Javaid. Understanding dijkstra's algorithm. *Available at SSRN 2340905*, 2013.

[49] Michael Jünger, Thomas M Liebling, Denis Naddef, George L Nemhauser, William R Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A Wolsey. *50 Years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media, 2009.

[50] Jermain Kaminski, Michael Schober, Raymond Albaladejo, Oleksandr Zastupailo, and César Hidalgo. Moviegalaxies-social networks in movies. 2018.

[51] Douglas J Klein and Milan Randić. Resistance distance. *Journal of mathematical chemistry*, 12:81–95, 1993.

[52] J R Koza. Genetic programming: On the programming of computers by means of natural selection. *MIT Press, Cambridge, USA*, 1992.

[53] Parveen Kumar and Nitin Gupta. A heuristic algorithm for longest simple cycle problem. In *Proceedings of the International Conference on Wireless Networks (ICWN)*, 2014.

[54] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*, pages 1343–1350, 2013.

[55] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1–20, 2016.

[56] Huan Li, Stacy Patterson, Yuhao Yi, and Zhongzhi Zhang. Maximizing the number of spanning trees in a connected graph. *IEEE Transactions on Information Theory*, 66(2):1248–1260, 2019.

[57] Hui Liu, Xuanhong Xu, Jun-An Lu, Guanrong Chen, and Zhigang Zeng. Optimizing pinning control of complex dynamical networks based on spectral properties of grounded laplacian matrices. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(2):786–796, 2018.

[58] Daniel Lokshtanov. Finding the longest isometric cycle in a graph. *Discrete Applied Mathematics*, 157(12):2670–2674, 2009.

[59] Klavžar S. Xavier A. Arokiaraj A. Thomas E. Manuel, P. geodetic problem in networks. *Discussiones Mathematicae Graph Theory*, 2018.

[60] M Märtens, F Kuipers, and P Van Mieghem. Symbolic regression on network properties. *Genetic Programming*, pages 131–146, 2017.

[61] Ruslán G Marzo, Rafael A Melo, Celso C Ribeiro, and Marcio C Santos. New formulations and branch-and-cut procedures for the longest induced path problem. *Computers & Operations Research*, 139:105627, 2022.

[62] Dmytro Matsypura, Alexander Veremyev, Oleg A Prokopyev, and Eduardo L Pasiliao. On exact solution approaches for the longest induced path problem. *European Journal of Operational Research*, 278(2):546–562, 2019.

[63] T McConaghy. Ffx: Fast, scalable, deterministic symbolic regression technology. *Genetic Programming Theory and Practice IX*, pages 235–260, 2011.

[64] Russell Merris. Laplacian matrices of graphs: a survey. *Linear algebra and its applications*, 197:143–176, 1994.

[65] Ulla Miekkala. Graph properties for splitting with grounded laplacian matrices. *BIT Numerical Mathematics*, 33(3):485–495, 1993.

[66] J Miller. Cartesian genetic programming. *Springer*, 2011.

[67] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in engineering software*, 69:46–61, 2014.

[68] John E Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, 1(1):65–77, 2002.

[69] S L Mitchell. Another characterization of the centroid of a tree. *Discrete Mathematics*, 24(3):277–280, 1978.

[70] Bojan Mohar. *Some applications of Laplace eigenvalues of graphs*. Springer, 1997.

[71] George L Nemhauser and Gabriele Sigismondi. A strong cutting plane/branch-and-bound algorithm for node packing. *Journal of the Operational Research Society*, 43(5):443–457, 1992.

[72] Kazuya Okamoto, Wei Chen, and Xiang-Yang Li. Ranking of closeness centrality for large-scale social networks. In *International workshop on frontiers in algorithmics*, pages 186–195. Springer, 2008.

[73] Ferdinand Peper. *Efficient network topologies for extensible massively parallel computers*. PhD thesis, TU Delft, 1989.

[74] Dilson Lucas Pereira, Abilio Lucena, Alexandre Salles da Cunha, and Luidi Simonetti. Exact solution algorithms for the chordless cycle problem. *INFORMS Journal on Computing*, 34(4):1970–1986, 2022.

[75] Veeraraghavan Prakash. An efficient g-centroid location algorithm for cographs. *International Journal of Mathematics and Mathematical Sciences*, 2005(9):1405–1413, 2005.

[76] Veeraraghavan Prakash. Application of g-convexity in mobile ad hoc networks. In *6th International Conference on Information Technology in Asia*, pages 33–38, 2009.

[77] Amirreza Rahmani, Meng Ji, Mehran Mesbahi, and Magnus Egerstedt. Controllability of multi-agent systems from a graph-theoretic perspective. *SIAM Journal on Control and Optimization*, 48(1):162–186, 2009.

[78] R A Rossi and N K Ahmed. An interactive data repository with visual analytics. *SIGKDD Explor*, 17(2):37–41, 2016.

[79] M Schmidt and H Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[80] M Schmidt and H Lipson. Solving iterated functions using genetic programming. *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 2149–2154, 2009.

[81] Douglas R Shier. A computational study of floyd's algorithm. *Computers & Operations Research*, 8(4):275–293, 1981.

[82] Harmanjit Singh and Richa Sharma. Role of adjacency matrix & adjacency list in graph theory. *International Journal of Computers & Technology*, 3(1):179–183, 2012.

[83] J A Soloff, R A Márquez, and L M Friedler. Products of geodesic graphs and the geodetic number of product. *Discussiones Mathematicae Graph Theory*, 35(1):35–42, 2015.

[84] Andrew Vince. A framework for the greedy algorithm. *Discrete Applied Mathematics*, 121(1-3):247–260, 2002.

[85] Toby Walsh. Symmetry breaking constraints: Recent results. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 2192–2198, 2012.

[86] F H Wang, Y L Wang, and J M Chang. The lower and upper forcing geodetic numbers of block–cactus graphs. *European Journal of Operational Research*, 175(1):238–245, 2006.

[87] Run Wang, Xiaotian Zhou, Wei Li, and Zhongzhi Zhang. Maximizing the smallest eigenvalue of grounded laplacian matrix. *arXiv preprint arXiv:2110.12576*, 2021.

[88] D J Watts and S H Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[89] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[90] Jerzy Marian Wojciechowski. *Long induced cycles in the hypercube and colourings of graphs*. PhD thesis, Citeseer, 1990.

[91] Stelios H Zanakis and James R Evans. Heuristic "optimization": Why, when, and how to use it. *Interfaces*, 11(5):84–91, 1981.

[92] Junlong Zhang and Yu Luo. Degree centrality, betweenness centrality, and closeness centrality in social network. In *2017 2nd international conference on modelling, simulation and applied mathematics (MSAM2017)*, pages 300–303. Atlantis press, 2017.

Summary

This PhD thesis introduces optimization methods for graph problems classified as NP-hard. These are problems for which no deterministic algorithm is capable of solving them in polynomial time. More specifically, three graph problems were addressed, and for each, different optimization methods were used. These methods include standard methods, metaheuristics, and heuristics. In all cases, the performance of these methods was compared with those proposed in the literature, considering factors such as execution time and the quality of the solutions achieved. This comparative analysis aims to demonstrate the effectiveness of the proposed optimization methods.

The thesis was structured into four major parts. In the following sections, I will provide an overview of the findings from Chapter 2 to Chapter 5. Chapter 1 is an introductory chapter, that introduces the reader to the fundamentals of optimization and graph theory, and clarifies the interrelationship between these two domains.

Thesis I.

Chapter 2 dealt with a graph problem known as graph geodetic number, which is a global measure for simple connected graphs and it belongs to the path covering problems, to find the minimal-cardinality set of vertices, such that all shortest paths between its elements cover every vertex of the graph. Two greedy algorithms were proposed to obtain upper bounds for the geodetic number in an algorithmic way. The efficiency of these algorithms is demonstrated on real-world graphs and randomly generated graphs.

Thesis II.

In Chapter 3, Symbolic Regression with an evolutionary algorithm called Cartesian Genetic Programming has been used to derive formulas capable of approximating the graph geodetic number. The obtained formulas are evaluated on random and real-world graphs. It demonstrated how various graph properties as training data can lead to diverse formulas with different accuracy. It also investigated which training data are closely related to the graph property.

Thesis III.

In Chapter 4, the work aimed to deal with the longest induced (chordless) cycle problem, which is a graph problem involves the task of determining the largest possible subset of vertices within a graph in such a way that the induced subgraph forms a cycle. Within this work, three integer linear programs have been proposed to yield optimal solutions for this problem. To demonstrate the efficiency of these methods, experiments were conducted on a range of real-world graphs as well as random graphs. Additionally, a comparative analysis against approaches previously proposed in the literature was performed.

Thesis IV.

In Chapter 5, the purpose is to maximize the smallest eigenvalue of the grounded Laplacian matrix which is the Laplacian matrix's $(n-k) \times (n-k)$ submatrix after k rows and associated columns have been removed. Motivated by the Gershgorin circle theorem, the degree centrality is used to select k nodes that would maximize the smallest eigenvalue. In addition, the vertex cover problem was employed as an additional method of solving the problem. The efficiency of these approaches is demonstrated on real-world graphs and compared to the methods proposed in the literature.

Contributions of the thesis

In the **first thesis group**, the contributions are related to obtaining upper bounds of the graph geodetic number in an algorithmic way which was capable of providing solutions of acceptable quality within a reasonable time. Detailed discussion can be found in Chapter 2.

I/1. I introduced two greedy-type algorithms. The first, known as the greedy algorithm, relies on Floyd's algorithm, while the local greedy algorithm is based on Dijkstra's algorithm.

I/2. I have empirically demonstrated that the proposed algorithms can efficiently obtain upper bounds that closely approximate the optimal solution obtained from the binary integer linear programming. Meanwhile, their computational time remains a small fraction of that needed to obtain the exact geodetic number.

In the **second thesis group**, the contributions are related to using Symbolic Regression and Cartesian Genetic Programming to derive optimized formulas for graph geodetic number. Detailed discussion can be found in Chapter 3.

II/1. I have used Symbolic Regression together with Cartesian Genetic Programming to derive a general formula that approximates the graph geodetic number. The formula is simply the sum of the number of edges, the number of degree-one vertices, and the number of simplicial vertices. Thus, the approximation of the geodetic number can be obtained in a reasonable computational time, even for graphs with thousands of vertices and edges.

II/2. I demonstrated how different training sets will lead to different formulas with different accuracy which validates that using parameters that are highly related to the graph property as training data will help in approximation in a better manner.

In the **third thesis group**, the contributions are related to finding an efficient approach that can deliver optimal solutions within a reasonable time for the longest induced cycle problem. Exact methods have been introduced and evaluated against existing methods from the literature to address this problem. Detailed discussion can be found in Chapter 4.

III/1. I proposed three integer linear programs, some of which are extensions of models created for solving the longest induced path problem. The proposed programs had varying execution times and the number of instances they were able to solve optimally.

III/2. I conducted a comparison between my methods and the methods proposed previously in the literature, and the results demonstrated that the newly introduced methods consistently outperformed those presented in the literature.

In the **fourth thesis group**, the contributions are related to proposing new methods to maximize the smallest eigenvalue of the grounded Laplacian matrix, which can deliver solutions of acceptable quality within a reasonable time. Two approaches were introduced and demonstrated through experiments and compared to the methods proposed in the literature. A detailed discussion can be found in Chapter 5.

IV/1. I proposed two algorithms to address the problem: the first one, named DEGREE-G relies on vertex centrality, while the second one, called COVER is based on the vertex cover problem.

IV/2. Both algorithms, as evidenced by experimental results, consistently produced promising solutions within a reasonable time, emphasizing their competitiveness when compared to existing algorithms in the literature.

Összefoglalás

Jelen PhD-értekezés optimalizációs módszereket mutat be, elemez és tárgyal NP-nehéz, gráfokon értelmezett problémákhoz. Ezek olyan problémák, amelyekre jelen tudásunk szerint nincs polinomiális futási idejű algoritmus. Konkrétan három gráfproblémát vizsgáltunk, és mindegyikhez különböző optimalizációs módszereket alkalmaztunk. Ezek a módszerek alkalmazzák a standard eljárásokat, metaheurisztikákat és heurisztikákat. Minden esetben ezeknek a módszereknek a teljesítményét összehasonlítottuk a szakirodalomban javasolt módszerekkel, figyelembe véve olyan tényezőket, mint az végrehajtási idő és az elért megoldások minősége. Az összehasonlító elemzésnek a célja a javasolt optimalizációs módszerek hatékonyságának bemutatása.

Az értekezés négy fő részből áll. A következő szakaszokban áttekintést nyújtok a 2. és 5. fejezetek eredményeiről. Az 1. fejezet bevezető fejezetként szolgál, és megismerteti az olvasót a gráfelmélet ide vonatkozó alapjaival és a jelen tézisben releváns, optimalizálással kapcsolatos fogalmakkal. Tisztázzuk továbbá e két terület közötti kapcsolatot.

1. tézis

A 2. fejezetben foglalkoztam a geodetikus szám meghatározásával, amely egy NP-nehéz gráfprobléma. Ez egy globális mérőszám egyszerű összefüggő gráfokhoz, és az útvonal lefedési problémák közé tartozik. Célja a minimális csúcshalmaz megtalálása, amelynek minden eleme közötti legrövidebb utak fedik a gráf összes csúcsát. Két mohó algoritmust javasoltam a geodéziai szám felső korlátjainak algoritmikus meghatározására. Ezeknek az algoritmusoknak hatékonyságát valós világbeli gráfokon és véletlenszerűen generált gráfokon mutattuk be.

2. tézis

A 3. fejezetben a geodetikus szám közelítésére használható képletek előállításával foglalkoztunk, ahol a szimbolikus regressziót egy Genetikus Programozás nevű evolúciós algoritmussal együtt alkalmaztuk. Az előállított képletek pontosságát valós világbeli és véletlen gráfokon teszteltük. Ezzel bemutattuk, hogy különböző gráf tulajdonságok használata tréning adatként különböző pontosságú képletekhez vezethet. Továbbá vizsgáltuk, mely tréning adatok kapcsolódnak valóban a különböző tulajdonságokhoz.

3. tézis

A 4. fejezetben a leghosszabb indukált kör nevű gráfproblémával foglalkoztunk, ami egy NP-teljes feladat. Itt meghatározandó a gráfban található lehető legnagyobb csúcshalmaz kiválasztása oly módon, hogy az indukált részgráf kört alkosson. Három vegyes egészértékű lineáris programot javasoltunk az optimális megoldások meghatározására. A módszerek számítási hatékonyságának bemutatására és összehasonlítására kísérleteket végeztünk különböző valós gráfokon, valamint véletlen gráfokon egyaránt. Emellett összehasonlító elemzést végeztünk a szakirodalomban korábban javasolt megközelítésekkel.

4. tézis

A 5. fejezet témája gráfok Dirichlet-Laplace (DL) mátrixával[1] kapcsolatos probléma. A DL mátrix a gráf Laplace mátrixának egy $(n-k)\times(n-k)$ részmátrixa, miután k sor és az azokhoz tartozó oszlopokat eltávolítjuk. A feladatban olyan DL mátrixot kell előállítani, amelynek a legkisebb sajátértéke maximális. Ez a probléma is NP-nehéz. A Gershgorin-kör tétel motiválta a fokszám alapú módszerünket. Emellett kidolgoztunk olyan módszert is, amely a csúcsfedési feladat megoldását használja fel az eredeti probléma megoldására. Ezeknek a megközelítéseknek az hatékonyságát valós gráfokon demonstráltuk, és összehasonlítottuk az irodalomban javasolt módszerekkel.

A disszertáció tézisei

Az **első tézis-csoportban** a gráfok geodetikus számára vonatkozó, algoritmikusan meghatározható felső korlátok előállítására alkalmas eredményeimet mutattam be. Ezek a módszerek megfelelő minőségű megoldásokat kínáltak elfogadható időkereten belül. A részletes tárgyalás megtalálható a 2. fejezetben.

I/1. Két mohó típusú megközelítést mutattam be. Az első, amit egyszerűen csak mohó algoritmusnak nevezünk, a Floyd algoritmusra támaszkodik; míg a lokálisan mohó algoritmus a Dijkstra algoritmuson alapul.

I/2. Numerikus kísérletekkel igazoltam, hogy a javasolt algoritmusok hatékonyan képesek olyan felső korlátokat előállítani, amelyek közelítik a bináris egészértékű lineáris

[1]az angol nyelvű szakirodalomban inkább szokás a *grounded Laplacian* elnevezés, azonban erre a változatra nincsen szerencsés magyar fordítás

programozással kapott optimális megoldást. Eközben a számítási idejük csak kis töredéke annak, amire szükség van a geodetikus szám pontos meghatározásához.

A **második tézis-csoportban** az eredmények a szimbolikus regresszió és a Genetikus Programozás használatához kapcsolódnak annak érdekében, hogy optimalizált képleteket kapjunk egy gráf geodetikus számának meghatátozására. A részletes bemutatás a 3. fejezetben található.

II/1. Szimbolikus regressziót használtam Genetikus Programozással együtt, hogy olyan általános képleteket állítsak elő, amelyek közelítik egy gráf geodetikus számát. A legjobb képlet egyszerűen az élek számának, az 1-fokszámú csúcsok számának és a szimpliciális csúcsok számának összege. Így a geodetikus szám közelítése elérhető elfogadható számítási idő alatt, még az ezer csúcsú gráfok esetén is.

II/2. Kimutattam, hogy különböző tanítóhalmazok különböző képletekhez vezetnek különböző pontossággal, amely igazolja, hogy a gráf tulajdonságához erősen kapcsolódó paraméterek használata tanítóadatként segíti a szimbolikus regressziót a jobb közelítés elérésében.

A **harmadik téziscsoportban** a leghosszabb indukált kör megkereséséhez használható modellek kidolgozásával foglalkoztam. A feladat egy NP-teljes probléma. Új, egzakt modelleket vezettünk be és vetettük össze a szakirodalomban található modellekkel. Részletes bemutatás a 4. fejezetben található.

III/1. Három új vegyes egész értékű lineáris programot javasoltam, amelyek között van olyan is, amely a leghosszabb indukált út probléma megoldására készített modellek kiterjesztése. A javasolt programok különböző végrehajtási idővel és az általuk optimálisan megoldható példák számával rendelkeztek.

III/2. Összehasonlítást végeztem a saját javasolt módszereim és az irodalomban korábban javasolt módszerek között, és az eredmények azt mutatták, hogy az újonnan bemutatott módszerek folyamatosan jobban teljesítettek, mint az irodalomban bemutatottak.

A **negyedik téziscsoportban** található hozzájárulásaim a Dirichlet-Laplace mátrixok legkisebb sajátértékének maximalizálásával kapcsolatosak. Két megközelítést vezettünk be, amelyekre a numerikus kísérletekkel összehasonlítást végeztünk az irodalomban javasolt módszerekkel. Részletes bemutatás a 5. fejezetben található.

IV/1. Két algoritmust javasoltam a probléma kezelésére: az elsőt DEGREE-G-nek neveztem, és a fokszám központosságára támaszkodik, míg a másodikat COVER-nek neveztem, és a csúcsfedés problémára épül.

IV/2. Mindkét algoritmus, a kísérleti eredmények szerint, állandóan ígéretes megoldásokat hozott létre elfogadható időkereten belül, kiemelve versenyképességüket az irodalomban meglévő algoritmusokhoz viszonyítva.

Publications

Table 6.1: *Relation between the chapters and publications*

no.	Publication	Chapter2	Chapter3	Chapter4	Chapter5
1	J1	●			
2	J2		●		
3	J3			●	
4	C1				●

Journal publications

[J1] **Anaqreh, Ahmad T** and G.-Tóth, Boglárka and Vinkó, Tamás. Algorithmic Upper Bounds for Graph Geodetic Number. *Central European Journal of Operations Research* **30**, 1221–1237, 2022.

[J2] **Anaqreh, Ahmad T** and G.-Tóth, Boglárka, G and Vinkó, Tamás. Symbolic Regression for Approximating Graph Geodetic Number. *Acta Cybernetica* **25**, 151–169, 2021.

[J3] **Anaqreh, Ahmad T** and G.-Tóth, Boglárka and Vinkó, Tamás. Exact Methods for the Longest Induced Cycle Problem. *Discrete Applied Mathematics (under review).*

Full papers in conference proceedings

[C1] **Anaqreh, Ahmad T** and G.-Tóth, Boglárka and Vinkó, Tamás. New Methods for Maximizing the Smallest Eigenvalue of the Grounded Laplacian Matrix. *In Proceedings of the 12th International Conference on Applied Informatics. ICAI 2023*, 1–9, 2023.

Further publications

[J4] Baniata, Hamza and **Anaqreh, Ahmad** and Kertesz, Attila. PF-BTS: A Privacy-Aware Fog-enhanced Blockchain-assisted task scheduling. *Information Processing & Management*, **58**, 102393, 2021.

[J5] Baniata, Hamza and **Anaqreh, Ahmad** and Kertesz, Attila. DONS: Dynamic Optimized Neighbor Selection for smart blockchain networks. *Future Generation Computer Systems*, **130**, 75–90, 2022.

[J6] Baniata, Hamza and **Anaqreh, Ahmad** and Kertesz, Attila. Distributed Scalability Tuning for Evolutionary Sharding Optimization with Random-equivalent Security in Permessionless Blockchain. *Internet of Thing*, **24**, 100955, 2023.

Acknowledgments

To my mother, who surrounds me with love and always being a haven when I am in need. To my father, the best friend ever, our endless conversations shape me in every aspect. To my sisters and brothers, for their boundless love and support. To my friends, who consistently stand by me in times of need and never disappoint.

To my supervisors, Dr. Boglárka G.-Tóth and Dr. Tamás Vinkó, not only for introducing me to the topic that I am passionate about but for guiding and motivating me along the way. To the University of Szeged, for this incredible experience, and to Hungary, for hosting me over the years. To my beloved country, Jordan. Finally, to that person who once told me, ”Take what you can take”, those words catalyzed me to move forward during challenging times.