T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

MAXIMUM DOMATIC PARTITION

HİKMET METE VAROL

SUPERVISOR
PROF. DIDEM GÖZÜPEK

GEBZE
2024

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

MAXIMUM DOMATIC PARTITION

HİKMET METE VAROL

SUPERVISOR
PROF. DIDEM GÖZÜPEK

2024
GEBZE

	GRADUATION PROJECT JURY APPROVAL FORM

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 25/01/2024 by the following jury.

JURY

Member
(Supervisor) : Prof. Didem Gözüpek

Member : Prof. Didem Gözüpek

Member : Dr. Tülay Ayyıldız

ABSTRACT

This study addresses the development and implementation of both polynomial-time heuristic algorithms and exact solution algorithms for the generally NP-hard Maximum Domatic Partition problem. The primary goal is to design effective algorithms for the Maximum Domatic Partition problem and to evaluate their performance through extensive simulations and computational complexity analysis. The study begins with a comprehensive literature review to identify existing approaches, then focuses on an original heuristic algorithm and an algorithm called ”Allied Domination” found in the literature. Each algorithm has been tested across different node counts and network densities, and the results are presented comparatively. Additionally, a graphical user interface has been developed to visualize the algorithms and enhance user experience. The results demonstrate the effectiveness and feasibility of the proposed algorithms and offer new perspectives in solving such problems.

Keywords: Maximum Domatic Partition Problem, NP-Hard Problems, Heuristic Algorithms, Computational Complexity Analysis, Graphical User Interface ,Allied Domination Algorithm.

ÖZET

Bu çalışma, genel olarak NP-hard kabul edilen Maksimum Domatik Bölümleme problemine yönelik polinom zamanlı sezgisel algoritmalarla beraber kesin çözüm sunan algoritmaların da geliştirilmesi ve uygulanmasını ele almaktadır. Ana hedef, Maksimum Domatik Bölümleme problemi için etkili algoritmalar tasarlamak ve bu algoritmaların performansını kapsamlı simülasyonlar ve hesaplama karmaşıklığı analizi ile değerlendirmektir. Çalışma, kapsamlı bir literatür taraması ile başlamakta, mevcut yaklaşımları belirlemekte ve ardından özgün bir Heuristic algoritma ve literatürde bulunan "Allied Domination" adında bir algoritma üzerinde yoğunlaşmaktadır. Her algoritma, farklı düğüm sayıları ve ağ yoğunlukları üzerinde test edilmiş ve sonuçlar karşılaştırmalı olarak sunulmuştur. Ayrıca, algoritmaların görselleştirilmesi ve kullanıcı deneyimini iyileştirmek için bir grafiksel kullanıcı arayüzü geliştirilmiştir. Sonuçlar, önerilen algoritmaların etkinliğini ve uygulanabilirliğini göstermekte ve bu tür problemlerin çözümünde yeni perspektifler sunmaktadır.

Anahtar Kelimeler: Maksimum Domatik Bölümleme Problemi, NP-Zor Problemler, Sezgisel Algoritmalar, Hesaplama Karmaşıklığı Analizi, Grafiksel Kullanıcı Arayüzü, Müttefik Hakimiyet Algoritması.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor, Prof. Didem Gözüpek, for feedback throughout this project. Her expertise have significantly contributed to the success of this project.

Additionally, I extend my thanks to my presentation team colleagues for the valuable discussions we have shared during the course of our projects.

Hikmet Mete Varol

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	**: Explanation**
n	: Number of nodes in the graph.
k	: Number of possible colors for graph coloring.
$O(k^n)$	: Complexity of Exhaustive Search, exponential in n and k.
$O(d)$	: Time to select a color for a node in the Heuristic algorithm.
d	: Degree of a node in the graph.
$O(n \times d_{\max})$	: Total complexity of the Heuristic algorithm.
$d_{\max}$	: Maximum degree among all nodes in the graph.
`coloringProducts`	: Represents the set of all possible color combinations for each node in the graph.
`isDomaticPartition`	: A function that checks whether a given color combination satisfies the conditions for being a domatic partition.
`isEqualPartition`	: A function used to assess the fairness of the color assignments across the graph.
`identify_blocks`	: A function used in the Allied Domination algorithm to divide the graph into biconnected components or blocks.
`augment_blocks`	: Refers to the process of expanding each identified block by including adjacent nodes.
`find_domatic_partition`	: A algorithm used to find the domatic partition of each block in the Allied Domination algorithm.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Today's complex network structures and data processing requirements have made solving problems in Graph Theory more important. In this report, the Maximum Domatic Partition problem, one of the important issues of Graph Theory, is discussed. This problem, classified as NP-hard, has an important place both theoretically and practically. The Maximum Domatic Partition problem forms the basis for applications such as efficiently controlling a network and optimizing resource allocation.

The aim of this study is to examine existing approaches to the Maximum Domatic Partition problem, develop various applicable algorithms for this problem, and evaluate the performance of these algorithms through extensive simulations and computational complexity analysis. The study focuses on algorithms that appear in the literature and are originally developed. The first of these algorithms is the Exhaustive Search Algorithm, which tries all possible solutions and aims to find the best result; second, the Heuristic Algorithm, which requires less computational power and provides fast results; and finally, the Allied Domination Algorithm[1], which divides the graph into blocks, produces separate solutions for each block and combines them.

2. LITERATURE REVIEW: KEY CONCEPTS IN DOMATIC PARTITION

In this section, we will delve into the fundamental concepts that are important, focusing specifically on Dominant Set, Domatic Number, and Domatic Partition. These concepts form the theoretical framework of algorithms for the Maximum Domatic Partition problem.[2]

Dominating Set

A dominating set in a graph G is a subset of vertices such that every vertex not in the subset is adjacent to at least one vertex in the subset. The significance of dominating sets is their ability to provide a compact representation of a network, where the vertices in the set can monitor or control the entire network.2.1

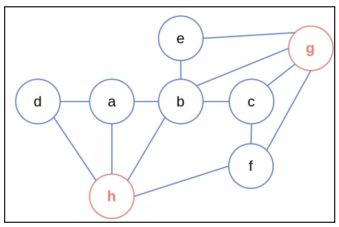

Figure 2.1: {g,h} is dominating set of the graph.[3]

Domatic Number

The domatic number of a graph is the maximum number of dominating sets into which the vertex set of the graph can be partitioned, such that each set is a dominating set.2.2

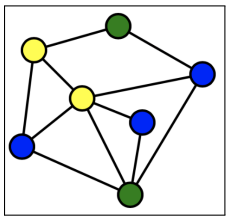

Figure 2.2: The graph has domatic number 3.[4]

Domatic Partition

A domatic partition of a graph is a partition of its vertices into dominating sets. The main goal in domatic partition, especially in larger and more complex graph structures, is to find the maximum number of dominant clusters that completely partition the vertex set of the graph.2.3

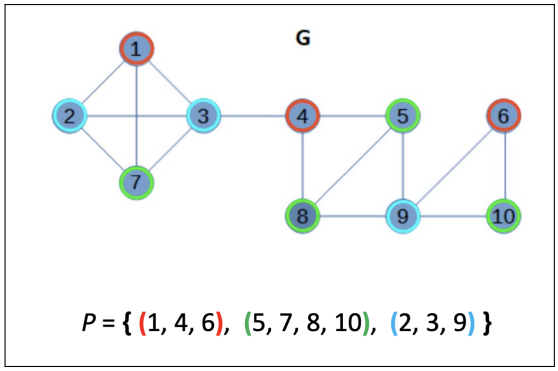

Figure 2.3: Domatic Partition of G.[1]

In the subsequent sections, we will examine how these basic concepts are implemented and extended by different algorithmic approaches, especially in the context of the Maximum Domatic Partition problem.

3. OVERVIEW OF ALGORITHMS

3.1. Algorithm 1: Exhaustive Search

Exhaustive Search is an approach that aims to find the optimal solution for the Maximum Domatic Segmentation problem by trying all possible color assignments. This algorithm assigns colors to all nodes of the graph, checking whether each color combination meets the domatic partition conditions.

3.1.1. Method and Implementation

The algorithm produces a combination of possible colors for each node `coloringProducts`. These combinations contain all possible color assignments for each node of the graph. Then, it is checked whether each combination meets the domatic partition conditions of the graph `isDomaticPartition`. If a combination meets these conditions, it is considered a valid domatic partition and is added to the list of results. Additionally, the fairness of color assignments (using approximately equal numbers of each color) can be checked with the `isEqualPartition` parameter.

3.1.2. Advantages and Disadvantages

The main advantage of the Exhaustive Search method is that it guarantees the best solution because it evaluates all possible combinations. However, the main disadvantage of this method is the high computational complexity on large graphs. With the growing number of nodes, the number of combinations attempted increases exponentially, which limits the scalability of the algorithm.

3.2. Algorithm 2: Heuristic

Heuristic algorithm uses a degree-based approach to solve the Maximum Domatic Partition problem. This method works by sorting nodes by their degree and assigning each node a color that is different from the colors of its neighbors. The main goal of this approach is to find the maximum domatic partition in a graph using the least possible number of colors.

3.2.1. Method and Implementation

The algorithm first ranks all nodes according to their degree. It then assigns for each node one of the colors that has not yet been used by that node's neighbors. If all available colors have been used by neighbors, it creates a new color. This process continues until all nodes are colored. A second pass is then made to optimize color distribution. For each node, the lowest available color is chosen, which helps reduce the overall number of colors.

3.2.2. Advantages and Disadvantages

The most important advantage of this heuristic algorithm is that it is low in complexity and easy to implement. Additionally, it can deliver fast results even on large graphics. However, the disadvantage of this approach is that it does not always guarantee the best result. In some cases, the algorithm may produce non-ideal color distributions.

3.3. Algorithm 3: Allied Domination

The "Allied Domination" algorithm offers a special solution approach for the Maximum Domatic Partition problem, taken from the literature article "The Domatic Partition Problem in Separable Graphs"[1]. This method divides graph into blocks (biconnected components), solves each block independently, and then combines the domatic partitions of these blocks to create the entire graph domatic partition.

3.3.1. Method and Implementation

The first step of the algorithm is to divide the graph into blocks `identify_blocks`. Each block is a connected and non-separable sub-block of graph. These blocks are then expanded to include neighboring nodes `augment_blocks`. For each augmented block, domatic partition is found `find_domatic_partition` using MILP (Mixed Integer Linear Programming). This approach improves the overall solution by optimizing the domatic partition of each block. In the final stage, the domatic partition of all graph is obtained by combining the domatic partition of all blocks.

3.3.2. Advantages and Disadvantages

The most important advantage of this algorithm is that its complexity is low and easy to implement compared to Exhaustive Search. It can deliver fast results even on large graphics as it works by dividing the graph into sub-blocks. However, the disadvantage of this algorithm is that it does not always guarantee the best result in our implementation.

4. ALGORITHM 1: EXHAUSTIVE SEARCH

4.1. Definition and Pseudocode of the Algorithm

Exhaustive Search is an approach that tries all possible color assignments to provide a solution to the Maximum Domatic Partition problem. This method generates all possible color combinations for each node and checks each of these combinations to see if they provide the domatic partition conditions.1

Algorithm 1 Exhaustive Search for Maximum Domatic Partition

Require: Graph G
Ensure: Maximum Domatic Partition of G

```
Initialize the best solution variable
for each possible color assignment coloring do
    if coloring satisfies domatic partition conditions then
        if coloring is better than the current best solution then
            Update the best solution with coloring
        end if
    end if
end for
return best solution
```

The Exhaustive Search algorithm was implemented in Python using the NetworkX library. The algorithm makes all possible color assignments to all nodes of graph and checks whether each assignment creates domatic partition.

4.2. Computational Complexity Analysis

The computational complexity of the Exhaustive Search algorithm increases exponentially with the number of nodes and the number of possible colors. When the number of nodes is n and k possible colors for each node, the complexity of the algorithm is $O(k^n)$. This limits the applicability of the algorithm on large graphs.

4.3. Simulations and Numerical Evaluation

4.3.1. Simulations

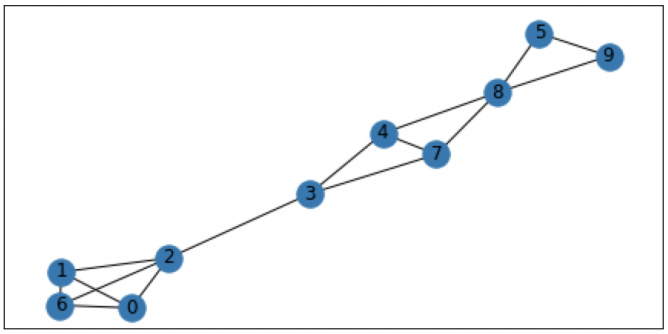

Figure 4.1: Sample Graph

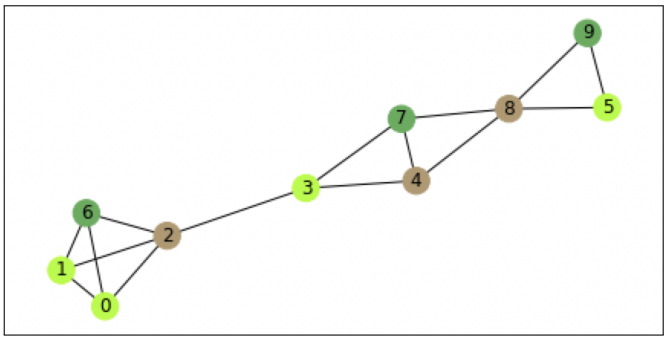

Figure 4.2: Domatic Partition of Sample Graph

This simulation shows the execution of the Exhaustive Search algorithm on an example graph. First of all, a sample graph was created with a certain number of nodes and density, and the best domatic partition of this graph was found using the Exhaustive Search algorithm.

4.3.2. Numerical Evaluation

4.3.2.1. Exhaustive Search Execution Time vs Node Count

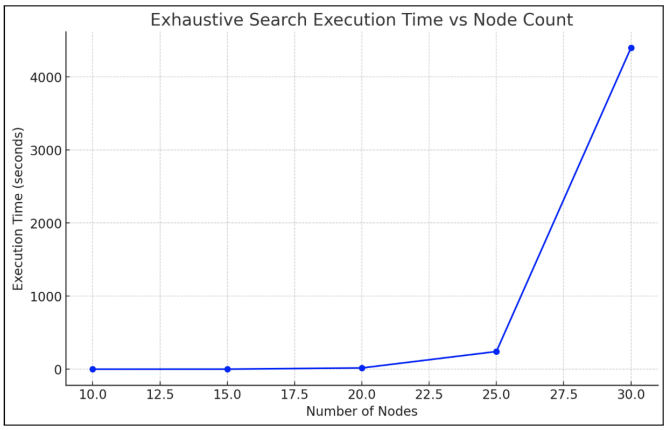

Figure 4.3: Exhaustive Search Execution Time vs Node Count

The graph showing the variation of the execution time of the Exhaustive Search algorithm, created using the given values, according to the number of nodes. Each execution time value was calculated as the average of 50 different experiments performed under the same conditions.

- The x-axis shows the number of nodes.
- The y-axis represents the execution time (in seconds) of the algorithm.
- The blue line and circle markers show the execution time for each number of nodes.

The graph shows that execution time increases exponentially as the number of nodes increases. This is an important observation that shows that the Exhaustive Search algorithm has limited applicability on large graphs.

Node Count	Execution Time (s)
10	0.049
15	0.48
20	16.28
25	241.26
30	4397.62

Table 4.1: Exhaustive Search: Execution Time by Node Counts (Density: 0.4)

4.3.2.2. Exhaustive Search Execution Time vs Density for 15 Nodes

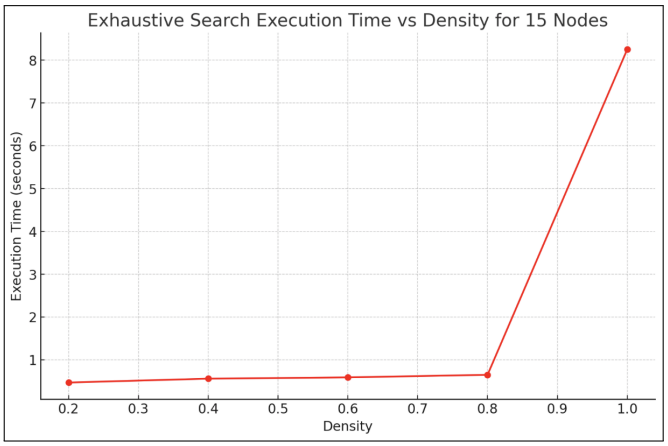

Figure 4.4: Exhaustive Search Execution Time vs Node Count

The graph shows the execution time of the Exhaustive Search algorithm for different density values on a 15-node graph. Each execution time value was calculated as the average of 50 different experiments performed under the same conditions.

- The x-axis shows density values (from 0.2 to 1.0).
- The y-axis shows the execution time of the algorithm (in seconds).
- The red line and circle markers indicate the practice time at each density level.

This graph shows how the execution time of the algorithm changes exponentially with increasing density. In particular, a significant increase in application time is observed at the point where the density increases from 0.8 to 1.0.

Density	Execution Time (s)
0.2	0.47
0.4	0.56
0.6	0.59
0.8	0.65
1.0	8.25

Table 4.2: Exhaustive Search: Execution Time by Density (Node Count: 15)

5. ALGORITHM 2: HEURISTIC

5.1. Definition and Pseudocode of the Algorithm

The Heuristic Domatic Partition algorithm provides a heuristic solution to the Maximum Domatic Partition problem. This method sorts nodes by their rank and assigns each node a color that is different from the colors of its neighbors. The goal is to create effective domatic division using the least number of colors possible.2

Algorithm 2 Heuristic Algorithm for Maximum Domatic Partition

Require: Graph G
Ensure: Domatic Partition of G
1: Sort nodes of G by degree in descending order
2: Initialize node_colors as an empty dictionary
3: **for** each node n in sorted nodes **do**
4: Determine colors used by neighbors of n
5: Assign the smallest unused color to n
6: **end for**
7: **return** node_colors as the domatic partition

5.2. Computational Complexity Analysis

Heuristic algorithm has polynomial time complexity. This algorithm involves sorting nodes according to their degree and assigning an appropriate color to each node. For each node, it takes $O(d)$ time to select a suitable color from the available colors, where d is the degree of the node. When we repeat this process for all nodes, the total complexity becomes $O(n \times d_{\max})$, where is the number of nodes and $d_{\max}$ is the maximum node degree. This can be expressed as a polynomial function of the number of nodes, indicating that the algorithm is polynomial time.

This complexity ensures that the algorithm can work effectively even on large graphs, but it does not always guarantee the best solution.

5.3. Simulations and Numerical Evaluation

5.3.1. Simulations

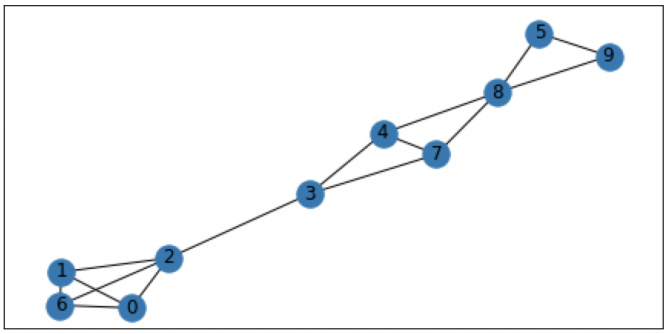

Figure 5.1: Sample Graph

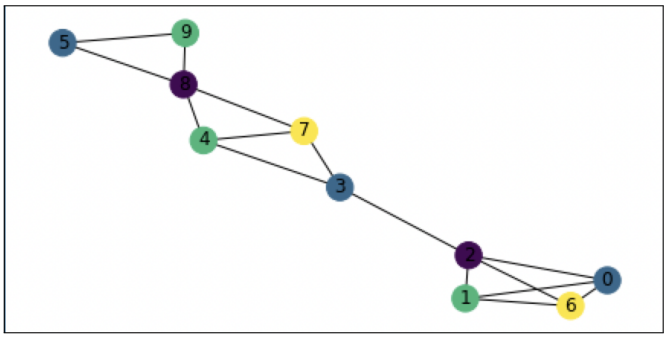

Figure 5.2: Approximate Domatic Partition of Sample Graph

This simulation shows the execution of the Heuristic algorithm on an example graph. First of all, a sample graph was created with a certain number of nodes and density, and the approximate domatic partition of this graph was found using the Heuristic algorithm.

5.3.2. Numerical Evaluation

5.3.2.1. Heuristic Algorithm Execution Time vs Node Count

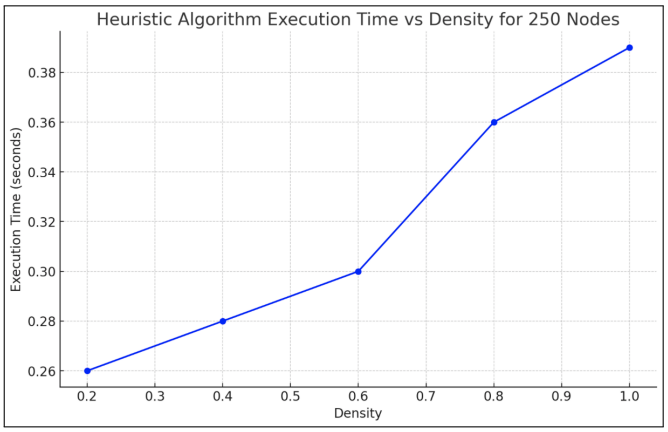

Figure 5.3: Heuristic Algorithm Execution Time vs Node Count

The graph shows the execution time of the Heuristic algorithm for different numbers of nodes. Each execution time value was calculated as the average of 50 different experiments performed under the same conditions.

- The x-axis shows the node numbers (from 50 to 250).
- The y-axis represents the execution time (in seconds) of the algorithm.
- The green line and circle markers indicate the execution time for each number of nodes.

The graph shows that the execution time of the algorithm increases slightly as the number of nodes increases. The rate of this increase indicates a linear relationship with the number of nodes, showing that the algorithm can work effectively even on large graphs.

Node Count	Execution Time (s)
50	0.05
100	0.09
150	0.18
200	0.21
250	0.3

Table 5.1: Heuristic Algorithm: Execution Time by Node Counts (Density: 0.8)

5.3.2.2. Heuristic Algorithm Execution Time vs Density for 250 Nodes

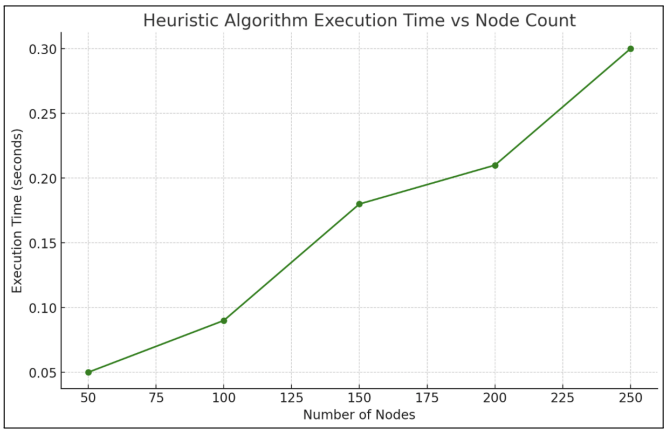

Figure 5.4: Heuristic Algorithm Execution Time vs Density for 250 Nodes

The graph shows the execution time of the Heuristic algorithm for different density values on a 250-node graph. Each execution time value was calculated as the average of 50 different experiments performed under the same conditions.

- The x-axis represents density values (from 0.2 to 1.0).
- The y-axis shows the execution time of the algorithm (in seconds).
- Blue line and circle markers indicate practice time at each density level.

The graph shows that the execution time of the algorithm increases slightly with increasing density. This shows how the algorithm's performance is affected depending on the density level, and shows that the algorithm can work effectively on graphs of different densities.

Density	Execution Time (s)
0.2	0.26
0.4	0.28
0.6	0.3
0.8	0.36
1.0	0.39

Table 5.2: Heuristic Algorithm: Execution Time by Density (Node Count: 250)

6. ALGORITHM 3: ALLIED DOMINATION

6.1. Definition and Pseudocode of the Algorithm

The ”Allied Domination” algorithm provides a unique solution to the Maximum Domatic Partition problem. This algorithm creates the domatic partition of the entire graph by separating the graphe into blocks and calculating the domatic partitioning of each block separately.[1]

Algorithm 3 Allied Domination for Maximum Domatic Partition

Require: Graph G
Ensure: Domatic Partition of G
1: Identify blocks in G using biconnected components
2: Augment each block by including adjacent nodes
3: **for** each augmented block B **do**
4: Find domatic partition of B using MILP
5: **end for**
6: Combine domatic partitions of all blocks to form the final solution
7: **return** final solution
[1]

6.1.1. Block Decomposition

It divides the graph into connected and non-separable sub-blocks. In this step, the biconnected components function of the NetworkX library is used. Each block represents a sub-block of the graph.[1]

6.1.2. Augmented Blocks

It expands each block to include the nodes adjacent to that block. This includes all nodes adjacent to the outer edges of the block. This expansion is done to enable independent domatic partition within each block.[1]

6.1.3. Domatic Partition of Augmented Blocks

Domatic partition is calculated separately for each augmented block. This process is done using the Mixed Integer Linear Programming (MILP) method. In this step, the

maximum domatic partition of each block is found.[1]

6.1.4. Merging Partitions

The domatic partitions found for all blocks are combined to form the entire graph domatic partition. This assembly process provides a comprehensive domatic partition of the entire graph.3[1]

6.2. Computational Complexity Analysis

The complexity of the ”Allied Domination” algorithm depends on the number and size of blocks. Finding domatic partition for each block is an NP-hard problem since MILP is used. However, dividing the graph into blocks in advance and solving each one separately can speed up the overall solution process. For large and complex graphs, this approach can reduce the complexity of the overall problem.

6.3. Simulations and Numerical Evaluation

6.3.1. Simulations

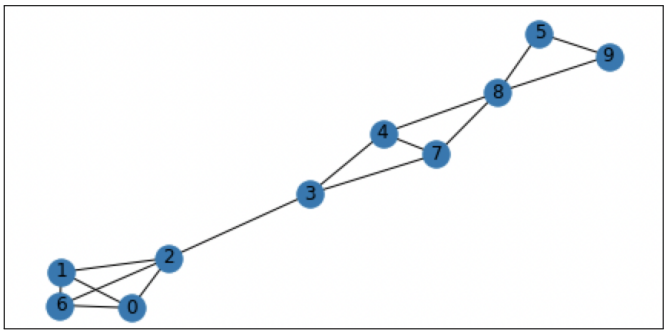

Figure 6.1: Sample Graph

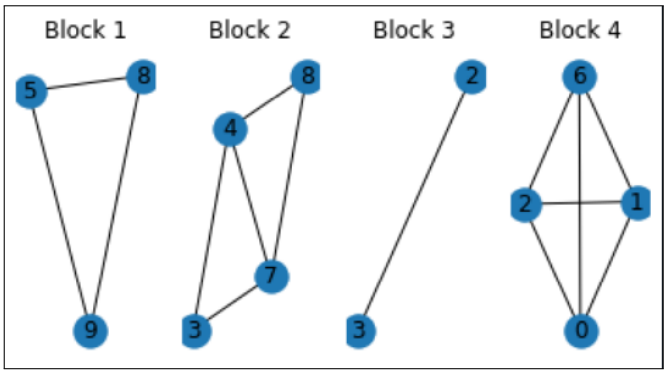

Figure 6.2: Block Decomposition

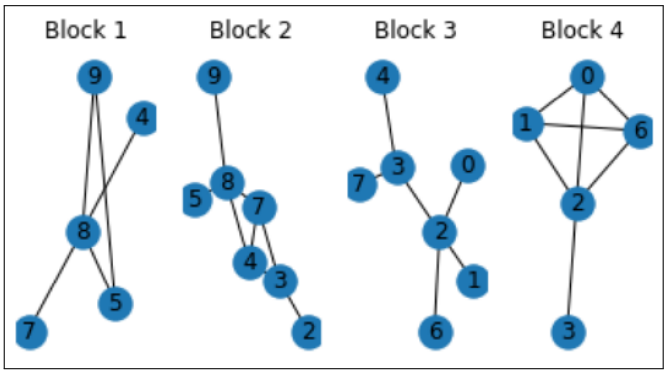

Figure 6.3: Augmented Blocks

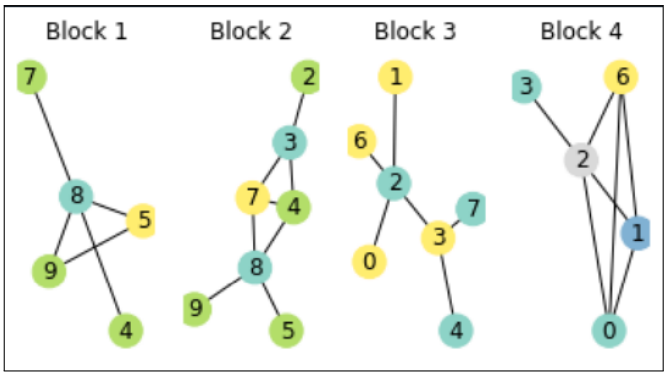

Figure 6.4: Domatic Partitions of Augmented Blocks

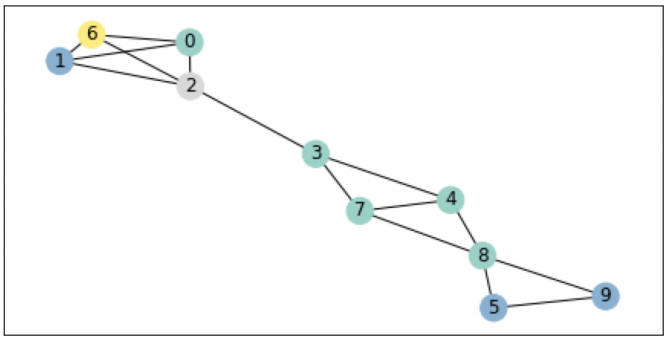

Figure 6.5: Approximate Domatic Partition of Sample Graph

This simulation shows the step-by-step execution of the Allied Domination algorithm on an example graph. First of all, a sample graph was created with a certain number of nodes and density, and the approximate domatic partition of this graph was found using the Allied Domination algorithm.

6.3.2. Numerical Evaluation

6.3.2.1. Allied Domination Algorithm Execution Time vs Node Count

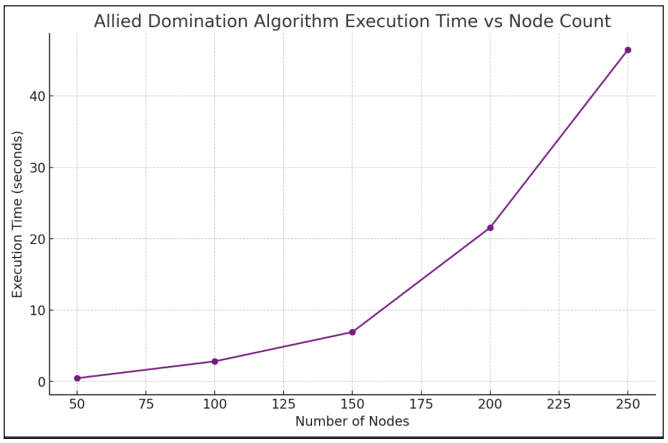

Figure 6.6: Allied Domination Algorithm Execution Time vs Node Count

The graph shows the execution time of the Allied Domination algorithm for different numbers of nodes. Each execution time value was calculated as the average of 50 different experiments performed under the same conditions.

- The x-axis represents the node numbers (from 50 to 250).
- The y-axis shows the execution time of the algorithm (in seconds).
- The Purple line and circle markers indicate execution time for each number of nodes.

The graph shows that the execution time of the algorithm increases significantly as the number of nodes increases. This offers important information about the applicability and performance of the algorithm on large graphs.

Node Count	Execution Time (s)
50	0.45
100	2.81
150	6.92
200	21.55
250	46.45

Table 6.1: Allied Domination: Execution Time by Node Counts (Density: 0.8)

6.3.2.2. Allied Domination Algorithm Execution Time vs Density for 100 Nodes

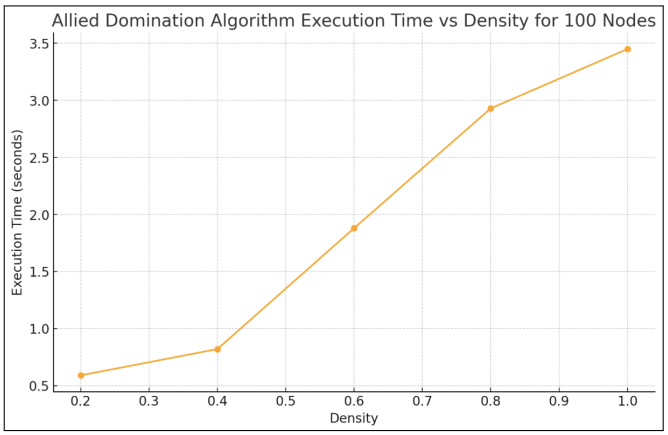

Figure 6.7: Allied Domination Algorithm Execution Time vs Density for 100 Nodes

The graph shows the execution time of the Allied Domination algorithm for different density values on a 100-node graph. Each execution time value was calculated as the average of 50 different experiments performed under the same conditions.

- The x-axis represents density values (from 0.2 to 1.0).
- The y-axis shows the execution time of the algorithm (in seconds).
- Orange line and circle markers indicate practice time at each density level.

The graph shows how the execution time of the algorithm changes with increasing density. As the density increases, there is also an increase in the execution time of the algorithm.

Density	Execution Time (s)
0.2	0.59
0.4	0.82
0.6	1.88
0.8	2.93
1.0	3.45

Table 6.2: Allied Domination: Execution Time by Density (Node Count: 100)

7. GRAPHICAL USER INTERFACE (GUI)

7.1. GUI Design and Functionality

The developed GUI was created using Python's Tkinter and NetworkX libraries. This interface allows users to graphically create graphs, apply algorithms, and observe the results. The main screen contains buttons where you can select algorithms, input fields where you can enter parameters such as number of nodes and density, and buttons for running algorithms.

- **Graph Creation:** Users can create random graphs by specifying the number of nodes and their density. Additionally, predefined sample graph can also be selected.
- **Algorithm Selection and Operation:** Users can choose from Exhaustive Search, Heuristic and Allied Domination algorithms. The selected algorithm is applied to the graph according to the parameters specified by the user.
- **Visualization of Outputs:** After the implementation of the algorithms, the results are presented to the user graphically. This allows the user to easily understand the impact of the algorithm on the graph.

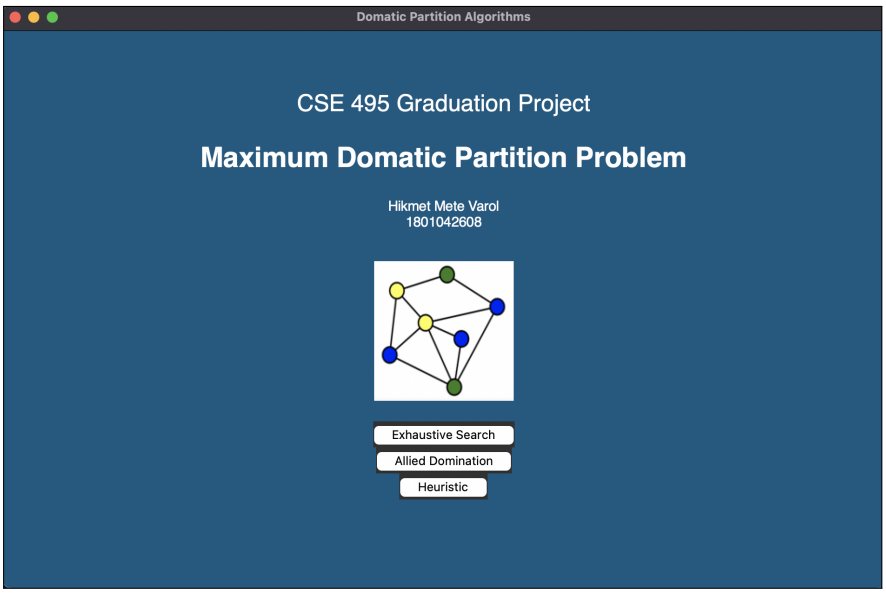

Figure 7.1: GUI Main Screen

7.2. Demonstration of Algorithms with GUI

Users can test all 3 algorithms in real time and see graphical representations of the results. In this section, the application steps of each algorithm on the GUI and the visual outputs of the obtained results are presented.

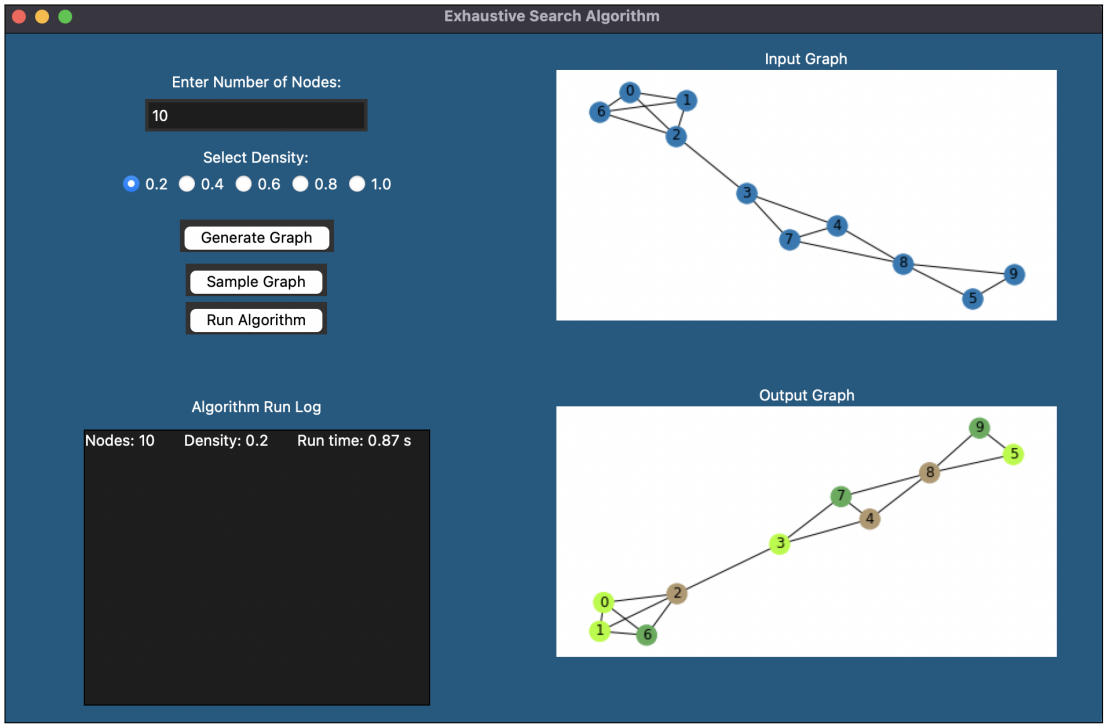

Figure 7.2: Exhaustive Search Algorithm Screen

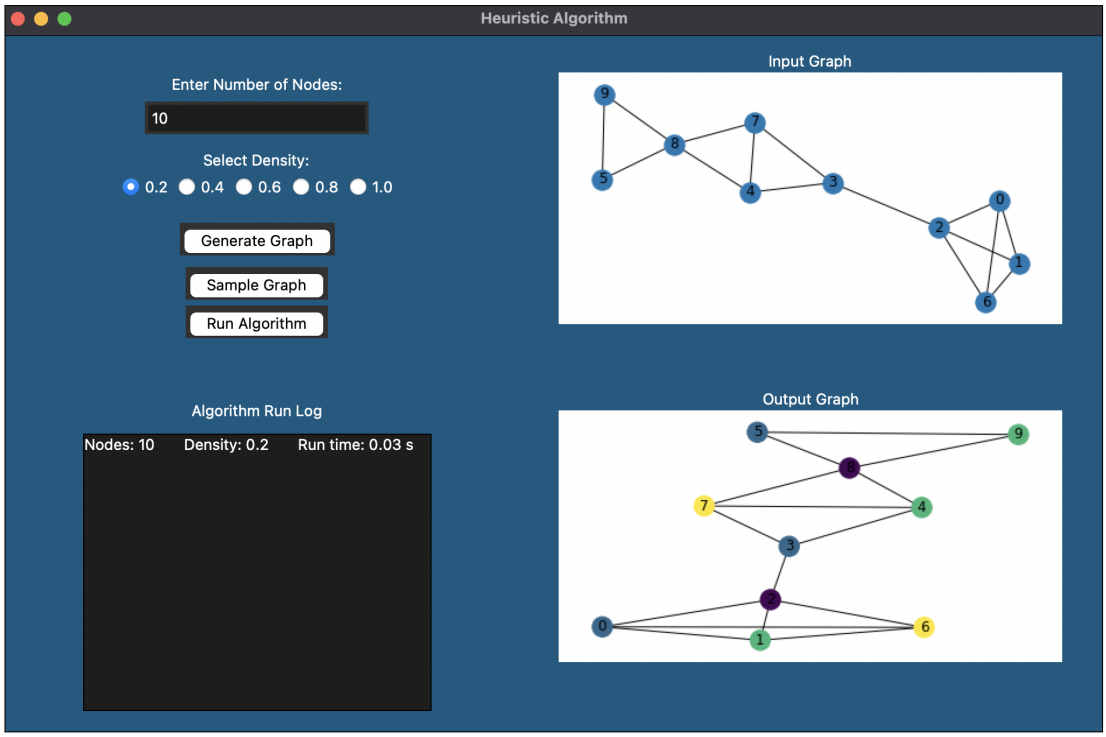

Figure 7.3: Heuristic Algorithm Screen

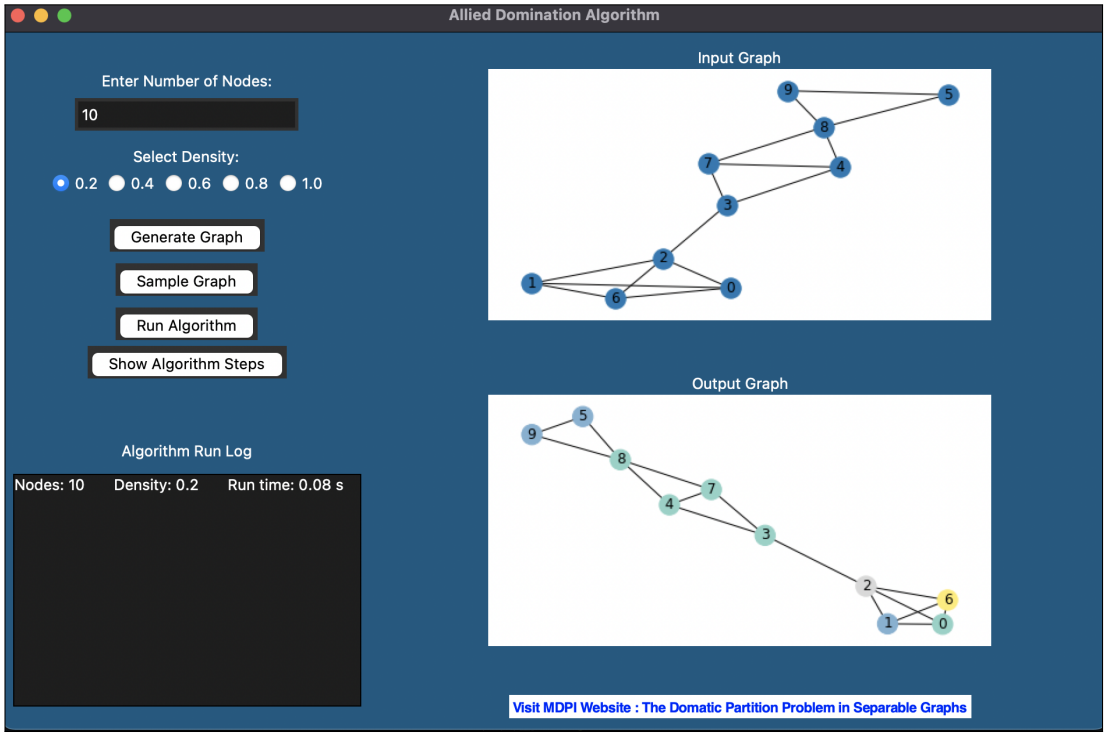

Figure 7.4: Allied Domination Algorithm Screen

8. CONCLUSIONS

In this report, the development and evaluation of three different algorithms for the Maximum Domatic Partition Problem, an NP-hard problem, are discussed. In the first chapter, the problem is introduced and the scope of the study is determined. In the second chapter, basic concepts and literature on the subject were examined.

In the third section, an overview of three main algorithms is presented: Exhaustive Search, Heuristic and Allied Domination. The basic features, application methods, advantages and disadvantages of each algorithm are summarized.

Finally, the Graphical User Interface (GUI) developed to visualize the algorithms and increase the user experience was introduced. The GUI allows users to interactively experience the algorithms and visually examine the results.

This work provides different solutions for the Maximum Domatic Partition Problem and comprehensively evaluates the performance of these algorithms.

BIBLIOGRAPHY

[1] M. Landete and J. Sainz-Pardo, "The domatic partition problem in separable graphs," *Mathematics*, vol. 10, p. 640, Feb. 2022. DOI: 10.3390/math10040640.

[2] U. Feige, M. M. Halldórsson, G. Kortsarz, and A. Srinivasan, "Approximating thedomatic number," *SIAM Journal on Computing*, vol. 32, no. 1, pp. 172–195, 2002. DOI: 10.1137/S0097539700380754. [Online]. Available: https://doi.org/10.1137/S0097539700380754.

[3] M. Simic, *Dominating set*, Image, Accessed November 6, 2023, 2023. [Online]. Available: https://www.baeldung.com/cs/dominating-sets-domination-numbers.

[4] Wikipedia, *Domatic partition*, Image, Accessed November 6, 2023, 2008. [Online]. Available: https://en.wikipedia.org/wiki/Domatic_number.