Minimim Edge Dominating Set

Francisco Ferreira - 124467
Algoritmos Avançados
Universidade de Aveiro
ftferreira@ua.pt

***Abstract*—This project explores the problem of finding minimum edge dominating sets in graphs by comparing two approaches: an exhaustive search and a greedy heuristic. The exhaustive search algorithm guarantees an optimal solution but is computationally intensive, especially as graph size and density increase. In contrast, the greedy algorithm provides faster, approximate solutions. Experimental results on various graph configurations demonstrate the trade-offs between the two methods, highlighting the efficiency of the greedy algorithm for larger graphs and the accuracy of the exhaustive approach for smaller instances.**

***Index Terms*—graph theory, edge dominating set, exhaustive search, greedy algorithm, computational complexity, heuristic methods**

I. Introdução

The objective of this project is to develop and evaluate two algorithms—an exhaustive search and a greedy heuristic—to address a specific problem in graph theory: identifying a minimum edge dominating set [1]. In graph-based applications, a minimum edge dominating set provides a way to control or influence the entire graph using the smallest subset of edges possible, which has practical applications in network design, resource allocation, and clustering.

In a graph $G = (V, E)$, an *edge dominating set* is a subset of edges $D \subseteq E$ such that every edge in E is either in D or adjacent to an edge in D. A *minimum edge dominating set* is an edge dominating set of the smallest possible size, making it a highly efficient structure for scenarios where complete edge coverage with minimal redundancy is critical. For example, in a communication network, a minimum edge dominating set can help optimize monitoring or control by minimizing the number of nodes actively engaged in surveillance or relay.

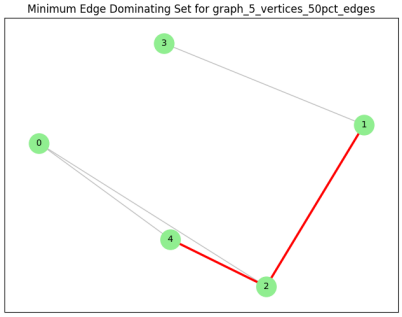

Fig. 1. Example of an edge dominating set

The problem of finding a minimum edge dominating set is NP-hard, meaning that the computation time required grows exponentially with the size of the graph.

To address this challenge, two types of algorithms are explored: an exhaustive search algorithm [2], which guarantees an optimal solution by evaluating all possible subsets of edges, and a greedy heuristic [3], which approximates a solution more quickly by selecting edges based on local, immediate benefits. While the exhaustive search provides the most accurate solution, it is computationally expensive, especially as graph size or density increases. The greedy heuristic, in contrast, sacrifices some accuracy for speed, making it a practical alternative for large graphs.

This report presents a comparative analysis of the two approaches across a variety of graph configurations, focusing on execution time, solution quality, and computational efficiency. By examining these trade-offs, we aim to provide insights into the practical applications and limitations of exhaustive and heuristic approaches for minimum edge dominating set problems.

II. Formal Analysis: Exhaustive Search Algorithm

The exhaustive search algorithm for finding a minimum edge dominating set in a graph operates by evaluating all possible subsets of edges to identify the smallest subset that covers the entire graph. Given a graph $G = (V, E)$, the algorithm checks each subset of edges to determine if it is a dominating set, meaning every edge in E is either included in the subset or adjacent to an edge in the subset.

A. Algorithm Description

The exhaustive search algorithm proceeds as follows:

1) **Subset Generation**: The algorithm generates all possible subsets of the edge set E. For each subset $S \subseteq E$, it tests whether S satisfies the conditions of an edge dominating set.
2) **Dominating Set Check**: For each subset S, the algorithm verifies that every edge $e \in E$ is either in S or adjacent to an edge in S. If S is a dominating set and smaller than previously identified sets, it is recorded as a candidate for the minimum edge dominating set.
3) **Optimal Solution Selection**: After evaluating all subsets, the algorithm outputs the smallest valid edge dominating set found, ensuring that the solution is optimal.

B. Computational Complexity

The exhaustive search algorithm is computationally expensive due to the need to examine all subsets of E:

- **Subset Generation**: The number of subsets of E is $2^{|E|}$, meaning the algorithm has to consider $2^{|E|}$ potential edge subsets. This leads to exponential growth in computation time with respect to the number of edges.
- **Dominating Set Verification**: For each subset S, the algorithm verifies whether every edge in E is covered by S, which can take $O(|E|)$ operations. This adds an additional factor of $|E|$ to the computation time for each subset verification.

Therefore, the **time complexity** of the exhaustive search is $O(|E| \cdot 2^{|E|})$, which makes it infeasible for large graphs. As graph size and density increase, the number of edges $|E|$ grows, leading to a rapid increase in execution time. This exponential growth restricts the practical use of the exhaustive search to small or sparsely connected graphs.

C. Limitations and Scalability

Due to its exponential complexity, the exhaustive search algorithm becomes impractical for graphs with even a moderate number of edges. To address this, a timeout was implemented to handle cases where the algorithm takes excessively long to execute, allowing a balance between obtaining optimal solutions and maintaining feasible runtime. Despite this, the exhaustive search remains valuable as a baseline for evaluating the accuracy of heuristic approaches, as it guarantees an optimal solution.

III. Formal Analysis: Greedy Algorithm

The greedy algorithm provides a heuristic approach to finding an edge dominating set in a graph. Unlike the exhaustive search, which guarantees an optimal solution by evaluating all possible subsets of edges, the greedy algorithm builds a solution incrementally. By selecting edges based on immediate coverage benefits, the algorithm sacrifices guaranteed optimality for computational efficiency.

A. Algorithm Description

The greedy algorithm proceeds as follows:

1) **Initialization**: Start with an empty edge dominating set D.
2) **Edge Selection**: While there are uncovered edges in the graph, select an edge $e \in E$ that maximizes the number of uncovered edges adjacent to e. This edge e is added to D.
3) **Coverage Update**: After selecting edge e, mark all edges adjacent to e as covered.
4) **Termination**: The algorithm terminates when all edges in the graph are either in D or adjacent to an edge in D.

The resulting set D serves as an approximate solution to the minimum edge dominating set. Although this solution is not guaranteed to be minimal, the greedy algorithm typically performs well in practice for large graphs where exhaustive search is infeasible.

B. Computational Complexity

The time complexity of the greedy algorithm can be analyzed as follows:

- **Edge Selection**: For each iteration, the algorithm selects the edge $e \in E$ that maximizes coverage. This selection process requires evaluating the uncovered neighbors of each edge, which takes $O(|E|)$ time in the worst case.
- **Number of Iterations**: The algorithm adds edges to D until all edges are covered, with at most $|E|$ iterations in the worst case.

Therefore, the **overall time complexity** of the greedy algorithm is $O(|E|^2)$, which is significantly faster than the exponential complexity of the exhaustive search. This efficiency makes the greedy algorithm practical for larger graphs.

C. Limitations and Approximation Quality

The greedy algorithm does not guarantee an optimal solution. By making selections based solely on immediate coverage benefits, it may overlook choices that could lead to a smaller edge dominating set. Thus, while the greedy solution is often close to optimal, it can sometimes be larger than the minimum edge dominating set.

The quality of the approximation depends on the structure of the graph. For sparse or randomly structured graphs, the greedy approach tends to produce near-optimal solutions. However, in dense or highly structured graphs, its performance may degrade, as locally optimal choices can lead to a larger-than-minimal edge dominating set.

IV. Experimental Setup

A. Graph Generation

To evaluate the performance of the exhaustive search and greedy algorithms, synthetic graph instances were generated using Python's NetworkX package. Each graph was constructed with vertices positioned in a two-dimensional space, ensuring a minimum distance between any pair of vertices to prevent overlapping. This setup provided a realistic spatial distribution for vertices and allowed for variability in graph density.

The graphs were generated with specific edge densities—12.5%, 25%, 50%, and 75%—to analyze algorithm performance across different levels of connectivity. For a graph with n vertices, each density corresponds to a particular number of edges, representing the fraction of all possible edges included in the graph.

To ensure reproducibility, a fixed random seed based on the student number was used in each experiment. This seed guarantees that the same graph structures are generated across multiple runs, allowing consistent evaluation of the algorithms' performance.

As to be able to run the graphs through the algorithms they were stored in a .txt file in this manner (Figure 2).

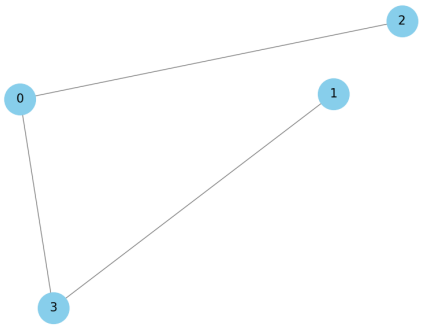

Fig. 2. Graph 4-vertices 50pct edges

```
Graph: graph_4_vertices_50pct_edges
Vertices: 4, Density: 0.5
Nodes:
0 268 551
1 543 559
2 603 676
3 297 216
Edges:
0 2
0 3
1 3
```

Fig. 3. Graph in Text File

B. Experimental Metrics

Several metrics were collected in the experiments to assess the performance and accuracy of each algorithm. These metrics include:

- **Basic Operations**: The number of fundamental operations executed by each algorithm, providing insight into computational effort. This count includes operations related to edge selection, coverage checking, and subset evaluations.
- **Execution Time**: The total time taken to complete each algorithm, measured in seconds. Execution time reflects the practical feasibility of each algorithm, especially in the context of larger or denser graphs.
- **Configurations Tested**: For the exhaustive search, the total number of edge subset configurations explored. This metric helps highlight the computational cost of guaranteeing an optimal solution, as the exhaustive approach evaluates multiple configurations to find the minimum edge dominating set.
- **Precision of the Greedy Heuristic**: The accuracy of the greedy heuristic relative to the exhaustive search. Precision is measured by comparing the size of the edge dominating set produced by the greedy algorithm to the optimal set size obtained through exhaustive search. This metric provides insight into how closely the greedy algorithm approximates the optimal solution.

V. Results and Analysis

This section presents the experimental results for the exhaustive search and greedy algorithms, analyzing their performance in terms of execution time, basic operations, precision, and scalability across various graph sizes and densities.

```
Graph: graph_13_vertices_75pct_edges
Edge Dominating Set: [(0, 2), (1, 9), (3, 4), (6, 12), (8, 10)]
Basic Operations: 461547
Time Taken: 8.5784 seconds
```

Fig. 4. Configurations saved after running an algorithm

A. Comparison of Execution Time

The execution times for both algorithms were recorded for graphs of varying sizes and densities. Results indicate an exponential increase in the execution time for the exhaustive search as graph size and density increase, making it impractical for larger instances. In contrast, the greedy algorithm demonstrates consistently lower execution times across all graph configurations, making it suitable for larger graphs where time efficiency is critical.

To effectively visualize the difference in execution times, a logarithmic scale was applied. Figure 4 shows the execution times on a log scale, illustrating the sharp increase for the exhaustive search compared to the more stable performance of the greedy algorithm. In cases where a single extreme value dominated, separate plots were used to ensure clarity.

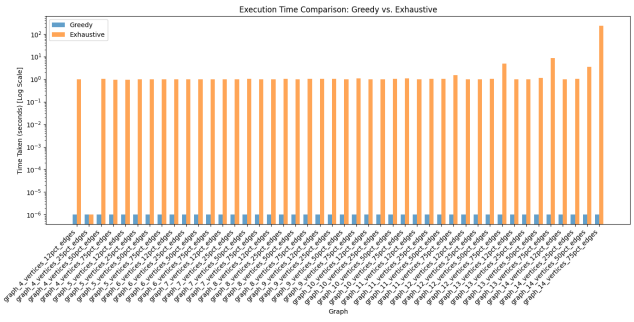

Fig. 5. Execution times

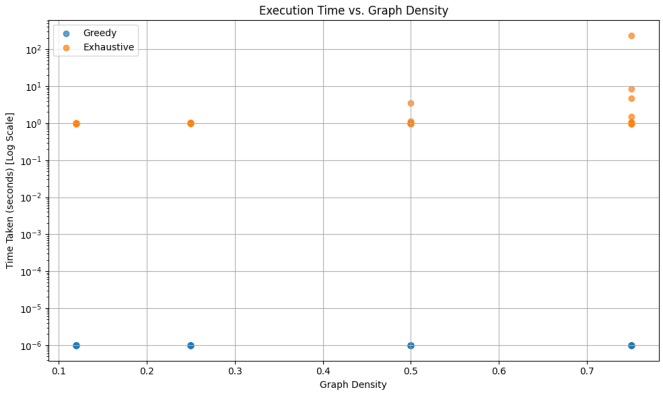

Fig. 6. Execution time vs Graph Density

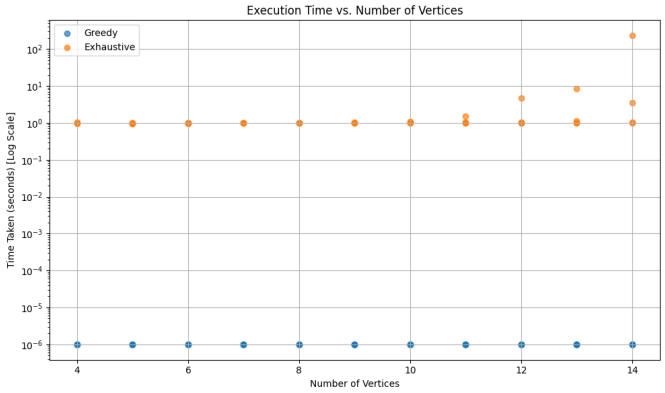

Fig. 7. Execution time vs Vertices numbers

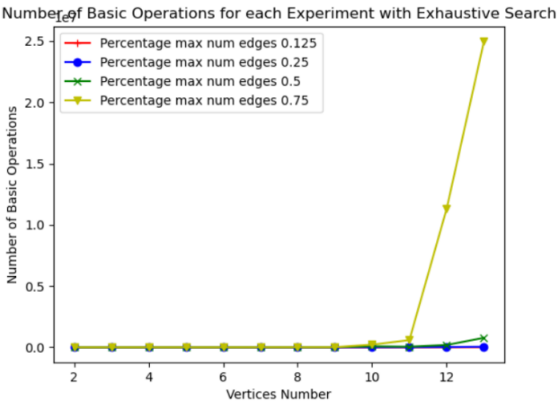

Fig. 9. Baisc Operations vs vertices for exhaustive algorithm

B. Comparison of Basic Operations

The number of basic operations required by each algorithm highlights the computational efficiency of the greedy algorithm relative to the exhaustive search. Figure 8 and 9 illustrates the operation count for each algorithm across different graph configurations. While the greedy algorithm's operations grow linearly with graph size and density, the exhaustive search exhibits exponential growth, further illustrating its computational cost for large graphs.

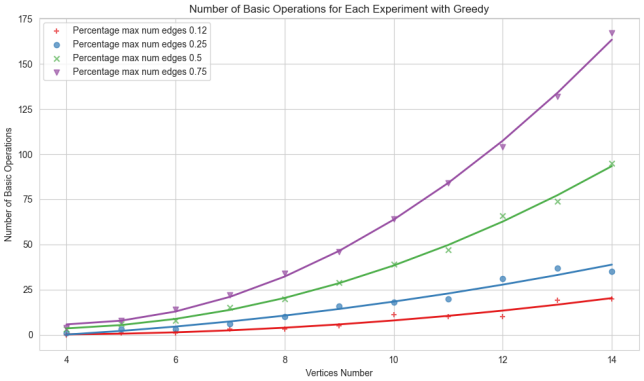

Fig. 8. Baisc Operations vs vertices for greedy algorithm

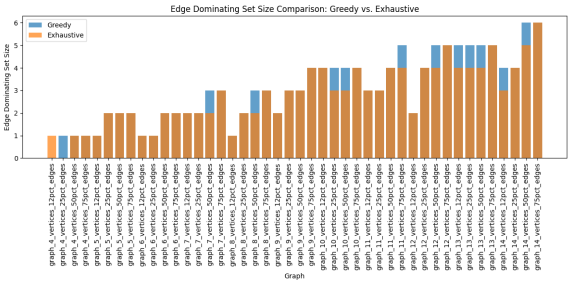

Fig. 10. Solution Quality Comparison

C. Precision of the Greedy Heuristic

The precision of the greedy heuristic was evaluated by comparing the edge dominating set sizes generated by the greedy algorithm to the optimal solutions produced by the exhaustive search. Precision was calculated as the ratio of the greedy result size to the exhaustive result size, with a value close to 1 indicating high precision. Figure 10 shows the precision ratios across various graph configurations. Results reveal that while the greedy algorithm generally produces near-optimal solutions, deviations increase in denser graphs, where locally optimal choices may lead to suboptimal sets.

Figures 11 and 12 show that for the same graph that the algorithms showed different solutions, the result from the exhaustive being the correct one.

D. Largest Graph Processed

The largest graph that each algorithm could process within reasonable time limits was identified. The exhaustive search was limited to small graphs with lower densities, as it quickly reached the timeout threshold on larger or denser graphs (16 vertices, 75pct was more than 45 minutes). In contrast, the greedy algorithm successfully processed larger graphs, demonstrating its scalability and practical applicability for more extensive datasets.

VI. Discussion and Interpretation

A. Comparing Formal and Experimental Analysis

The experimental results align closely with the theoretical complexity analysis for both algorithms. The exhaustive search exhibits exponential growth in execution time and basic operations, confirming its $O(|E| \cdot 2^{|E|})$ complexity. This exponential increase was evident in the experimental data, where the exhaustive algorithm quickly became infeasible as graph size and density increased. In contrast, the greedy algorithm's performance followed its $O(|E|^2)$ complexity, remaining practical even for larger graphs. Discrepancies between theoretical and actual performance were minimal, suggesting that the theoretical analysis provides an accurate representation of each algorithm's computational demands.

B. Implications of Results

The results of this study highlight the strengths and limitations of each algorithm:

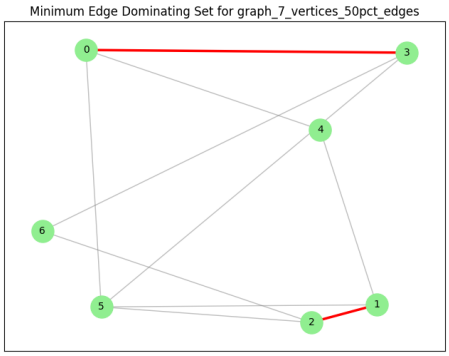

Fig. 11. Solution Found by exhaustive algorithm

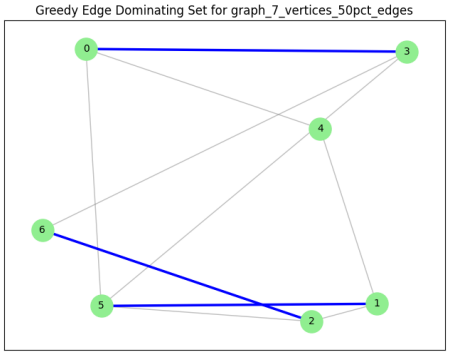

Fig. 12. Solution Found by greedy algorithm

- The **exhaustive search** is suitable for small graphs or applications where achieving the minimum edge dominating set is critical. Its ability to guarantee an optimal solution makes it valuable in scenarios where accuracy outweighs computational cost.
- The **greedy algorithm**, while not guaranteed to produce an optimal solution, is highly efficient and scalable. This makes it well-suited for larger graphs or time-sensitive applications where an approximate solution is sufficient. Its performance remains practical across different graph sizes and densities, offering a reliable alternative to exhaustive search.

The study's findings suggest that algorithm selection depends largely on graph size, density, and the need for solution accuracy. For most large-scale applications, the greedy algorithm provides a reasonable balance between speed and precision, while the exhaustive search is more applicable in cases where the graph is small, and exact solutions are required.

C. Limitations and Considerations

Several limitations were encountered during experimentation:

- **Dependence on Graph Structure**: The performance of the greedy algorithm varied with graph density and structure. While it generally produced near-optimal solutions, its precision decreased in denser graphs, where local choices could lead to suboptimal overall results.

These limitations highlight that while the greedy algorithm is scalable, it may lack precision in certain graph structures. Conversely, the exhaustive search, although accurate, is impractical for larger graphs, suggesting that a hybrid or alternative heuristic approach may offer better performance for specific graph types.

VII. Conclusion

This study examined two approaches for finding a minimum edge dominating set: an exhaustive search algorithm, which guarantees an optimal solution, and a greedy heuristic, which approximates the solution more efficiently. Key findings from the analysis include:

- The **exhaustive search** provides optimal solutions but incurs exponential time complexity, making it feasible only for small or sparse graphs.
- The **greedy algorithm** is computationally efficient with $O(|E|^2)$ complexity, enabling it to handle larger graphs, though it may produce slightly larger-than-optimal dominating sets.
- The **trade-off between accuracy and execution time** is evident: the exhaustive search is suited to applications where exact solutions are necessary, while the greedy algorithm offers practical scalability at the cost of some precision.

In practical terms, the greedy algorithm is recommended for applications with large graphs where approximate solutions are acceptable, such as in network design and resource allocation. The exhaustive search, though limited in scalability, is valuable for smaller graphs where guaranteed optimality is required.

A. Future Work

Future research could explore hybrid approaches that combine the accuracy of exhaustive search with the speed of heuristic methods. Additionally, investigating alternative heuristics may yield solutions that are both efficient and closer to optimal. Optimization techniques, such as branch-and-bound or dynamic programming, could also be applied to improve the scalability of the exhaustive search for moderate-sized graphs.

References

[1] https://en.wikipedia.org/wiki/Edge_dominating_set
[2] https://www.freecodecamp.org/news/greedy-algorithms/
[3] https://codestandard.net/articles/exhaustive-search/