Maximum Independent Set: Exhaustive and Greedy Search

Joaquim Rosa, 109089

***Abstract*—The maximum independent set problem in graph theory involves finding the largest subset of vertices such that no two vertices are adjacent. In this report, two algorithmic approaches to solving this problem are implemented and analysed: a simple exhaustive (brute-force) approach, which guarantees optimal solutions, and a greedy heuristic approach, which prioritises computational efficiency at the cost of potential suboptimality. A thorough analysis of the results is conducted to evaluate the performance, accuracy, and trade-offs of these algorithms, offering insights into their applicability across different graph configurations.**

***Index Terms*—graph theory, exhaustive search, greedy algorithms, heuristic methods, algorithm analysis**

I. Introduction

The maximum independent set (MIS) problem in graph theory involves finding the largest subset of vertices in a graph such that no two vertices in the subset are adjacent. This problem is challenging due to its combinatorial nature, as the number of possible vertex subsets grows exponentially with the graph's size. The MIS problem has applications in network design, scheduling, and bioinformatics.

This study focuses on implementing and analysing two algorithmic approaches to solve the MIS problem: an exhaustive (brute-force) method that guarantees the optimal solution by exploring all possible subsets, and a greedy heuristic that focuses on computational efficiency by making locally optimal choices at each step. Both approaches were implemented in Python 3.12 and tested on a variety of graph configurations.

II. Study Environment

A. Graphs for the Computational Experiments

The experiments utilised 1,988 unique graphs, each representing a distinct combination of vertices and edge densities. The number of vertices ranged from 4 to 500, resulting in 497 vertex configurations, each tested with four distinct edge densities: 12.5%, 25%, 50%, and 75%.

B. Generating and Storing the Graphs

The graphs were generated using the NetworkX library's *fast_gnp_random_graph* method, which creates random graphs based on specified vertex counts and edge probabilities. A fixed seed (SEED = 109089) was used to ensure reproducibility. Each generated graph was stored in JSON format, capturing its adjacency list for future computations.

```python
import networkx as nx

def generate_graph(num_vertices, edge_density):
    return nx.fast_gnp_random_graph(
        num_vertices, edge_density, seed=109089)
```

Listing 1. Graph generation example

Each graph was saved with filenames indicating its characteristics, such as graph_5_50.json for a graph with 5 vertices and 50% edge density.

C. Conducting the Study

Both algorithmic approaches were applied to the same set of graphs. The results were stored in structured JSON files. For the exhaustive approach, each file contained details such as the number of vertices, edge density, execution time, operation count, and MIS size, as illustrated below:

```json
{
    "4_50": {
        "vertices": 4,
        "edge_density": 0.5,
        "execution_time": 3.37340002261044e-05,
        "operation_count": 17,
        "set_size": 2,
        "combinations": 16
    },
    ...
```

Listing 2. Exhaustive Results JSON Example

For the greedy heuristic, the structure was similar but excluded the combinations field:

```json
{
    "4_50": {
        "vertices": 4,
        "edge_density": 0.5,
        "execution_time": 4.5884999963163864e-05,
        "operation_count": 22,
        "set_size": 2
    },
    ...
```

Listing 3. Greedy Results JSON Example

III. Exhaustive Search

The exhaustive search approach to finding the Maximum Independent Set (MIS) leverages a combinatorial strategy. It systematically explores all subsets of vertices, evaluating each to determine if it constitutes an independent set. Among these, the largest independent set is identified as the MIS.

The algorithm begins by generating all subsets of the graph's vertices, ordered from largest to smallest to facilitate early termination upon finding a valid independent set. Each subset is checked for independence by ensuring that no two vertices in the subset share an edge. If such a subset is found, it is returned as the solution.

The pseudocode of the algorithm is as follows:

```
1. Initialize max_set as an empty set
2. Let n be the number of vertices in the
     graph
3. For each subset size k from n down to 1:
     a. For each subset of vertices of size k:
          i. Check if the subset is independent
                (no two vertices are adjacent)
          ii. If the subset is independent:
              - Return the subset as the
                  maximum independent set
4. Return max_set
```

Listing 4. Pseudocode for Exhaustive Search

A. Formal Analysis

The exhaustive search method is computationally expensive due to its combinatorial nature. Given a graph with n vertices, the number of subsets to evaluate is 2^n, leading to a worst-case time complexity of $O(2^n)$. The independence check for each subset involves pairwise comparisons, adding an additional factor of $O(k^2)$, where k is the size of the subset. Consequently, this approach is only feasible for small graphs.

B. Experimental Analysis

To confirm the theoretical predictions, the exhaustive search algorithm was tested on randomly generated graphs with varying numbers of vertices and edge densities. For each number of vertices, four edge density levels were used: 12.5%, 25%, 50%, and 75%. The graph properties are as follows:

- $n \in [4, 28]$, as 28 vertices was the maximum computationally feasible for exhaustive search.
- e as one of four edge density values: $\{0.125, 0.25, 0.5, 0.75\}$.

The experiment evaluated several metrics: execution time, number of basic operations performed, and the number of combinations tested.

1) Number of Combinations: The number of combinations evaluated by the exhaustive algorithm is shown in Figure 1. The plot uses a logarithmic scale for the number of combinations, showing a linear growth pattern as expected. Since the exhaustive algorithm evaluates all subsets, this growth is exponential, following 2^n, regardless of the graph's edge density.

2) Execution Time: Figures 2 and 3 illustrate the execution time of the exhaustive search. Figure 2 shows the raw execution time, which grows exponentially, especially beyond 25 vertices. The linearity in Figure 3 (logarithmic scale) confirms the exponential growth rate.

As expected, higher edge densities tend to increase execution time since more edges require additional independence checks per subset. This trend is further evident in Figure 4,

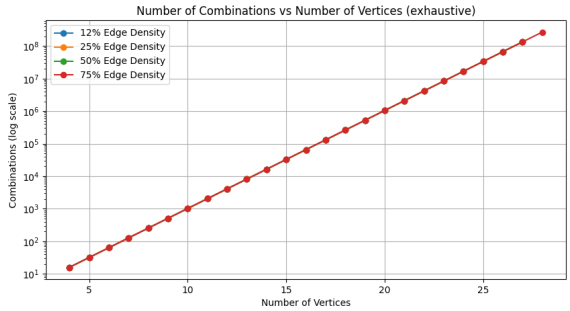

Fig. 1. Number of Combinations vs. Number of Vertices (log scale) for Exhaustive Search

which shows that execution time generally rises with edge density.

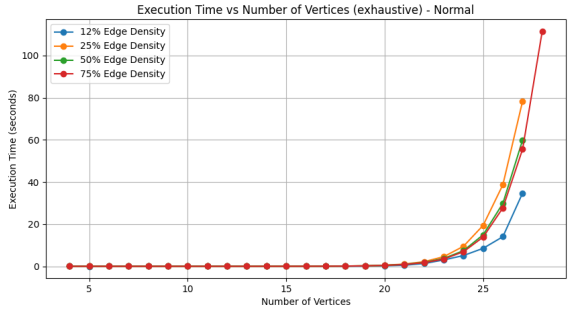

Fig. 2. Execution Time vs. Number of Vertices for Exhaustive Search (Normal Scale)

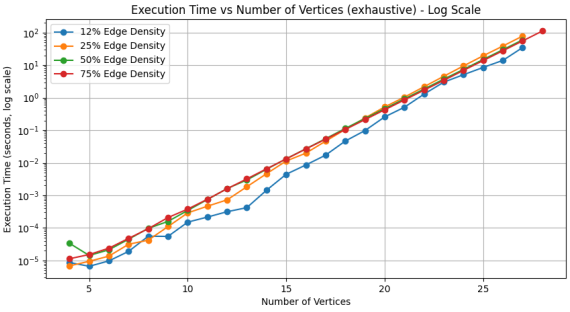

Fig. 3. Execution Time vs. Number of Vertices for Exhaustive Search (Log Scale)

3) Execution Time Prediction: Using an exponential fit model, we extended the execution time predictions to larger graphs, from 28 to 38 vertices, as shown in Figure 5. For a graph with 38 vertices and 75% edge density, the predicted execution time reaches nearly 120,000 seconds (approximately 33 hours). Further extrapolation estimates a 40-vertex graph would require around 477,906 seconds (approximately 5.5 days) to process, highlighting the exponential limitations of exhaustive search for larger graphs.

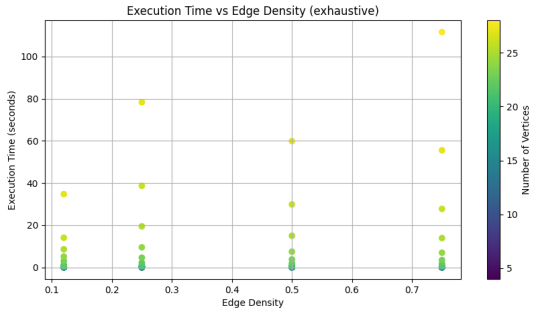

Fig. 4. Execution Time vs. Edge Density for Exhaustive Search

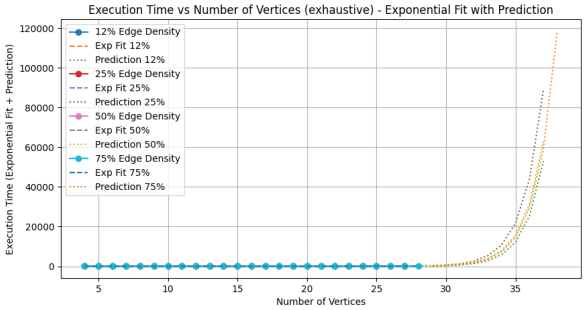

Fig. 5. Execution Time with Exponential Fit and Prediction for Exhaustive Search

4) Operation Count: Figures 6 and 7 display the number of operations performed. As with execution time, the operation count grows exponentially, becoming particularly substantial beyond 25 vertices. The logarithmic representation in Figure 7 exhibits a near-linear trend, confirming exponential growth. Higher edge densities generally require more operations due to the increase in independence checks, as seen in Figure 8.

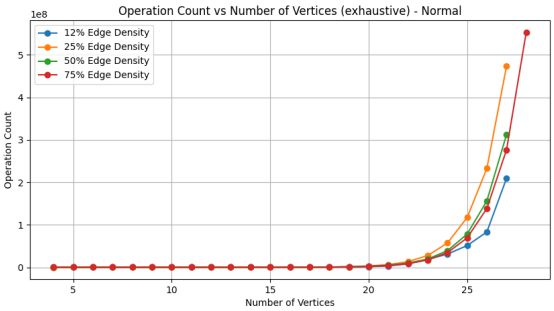

Fig. 6. Operation Count vs. Number of Vertices for Exhaustive Search (Normal Scale)

5) Operation Count Prediction: Using an exponential fit, the operation count was also predicted for graph sizes up to 38 vertices (Figure 8). For a graph with 38 vertices and high edge density, the algorithm would perform more than 5×10^{11} operations, reaffirming the practical infeasibility of exhaustive search for larger graphs.

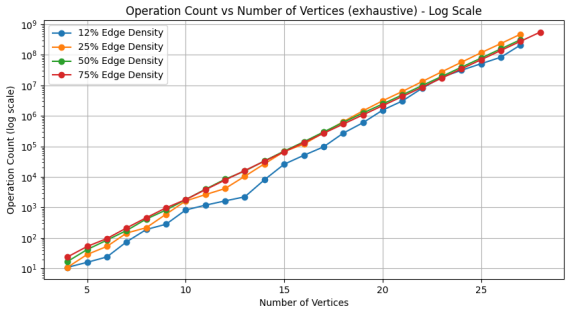

Fig. 7. Operation Count vs. Number of Vertices for Exhaustive Search (Log Scale)

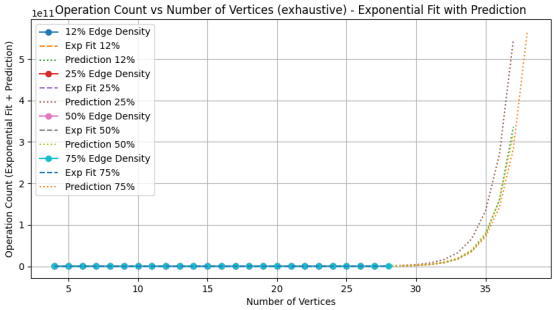

Fig. 8. Operation Count with Exponential Fit and Prediction for Exhaustive Search

IV. Greedy Search

In contrast to the exhaustive search, which evaluates all possible subsets of vertices to find the Maximum Independent Set (MIS), the greedy search algorithm operates based on a heuristic that reduces the search space by making locally optimal choices. This approach, while not guaranteed to yield a globally optimal solution, is significantly faster and can provide approximate solutions within a reasonable time.

A. Greedy Heuristic Approach

The greedy algorithm employed here follows a simple heuristic: it iteratively adds vertices with the lowest degree (fewest connections) to the independent set, aiming to maximize the MIS while minimizing conflicts with neighboring vertices. The algorithm steps are as follows:

1) Initialize an empty independent set.
2) Sort vertices by their degree in ascending order.
3) Iteratively select vertices:
   - For each vertex in the sorted list, if it has not been "skipped" (i.e., if it's not adjacent to any previously selected vertex), add it to the independent set and mark all its neighbors as skipped.
4) Return the independent set once all vertices have been processed.

The pseudocode of the algorithm is as follows:

```python
def greedy_heuristic_independent_set(graph):
    sorted_vertices = sort_vertices_by_degree
        (graph)
    independent_set = set()
    neighbors_to_skip = set()

    for v in sorted_vertices:
        if v not in neighbors_to_skip:
            independent_set.add(v)
            neighbors_to_skip.update(graph.
                neighbors(v))

    return independent_set
```

Listing 5. Greedy Heuristic for MIS

This pseudocode abstracts the main steps of the greedy algorithm. The `sort_vertices_by_degree` function sorts vertices in ascending order of their degree, and each selected vertex adds its neighbors to the `neighbors_to_skip` set to avoid future conflicts.

B. Formal Analysis

The greedy algorithm avoids the exponential complexity of exhaustive search by making immediate decisions at each step. The complexity can be analyzed as follows:

- **Sorting Step:** Sorting vertices by degree has a time complexity of $O(n \log n)$, where n is the number of vertices.
- **Main Loop:** After sorting, the algorithm iterates through each vertex, and for each selected vertex, it marks its neighbors as skipped. In the worst case, this marking process could require iterating through all edges connected to that vertex, yielding a worst-case complexity of $O(m)$ per iteration, where m is the number of edges.

Combining these steps, the time complexity of the greedy algorithm can be approximated as $O(n \log n + m)$, which is a significant improvement over the exponential complexity of exhaustive search. This polynomial complexity enables the algorithm to handle larger graphs efficiently, as demonstrated by tests with up to 500 vertices.

C. Experimental Analysis

To empirically assess the performance of the greedy algorithm, tests were conducted across graphs of varying vertex counts (up to 500 vertices) and edge densities (12.5%, 25%, 50%, and 75%). Three key metrics were evaluated: execution time, number of operations, and execution time versus edge density.

1) Execution Time: The execution time of the greedy algorithm shows near-linear growth with the number of vertices, as illustrated in Figure 9. This contrasts sharply with the exponential growth observed in exhaustive search, indicating that the greedy algorithm can handle larger graphs with feasible execution times. For example, even for graphs with 500 vertices, the execution time remained well within practical limits.

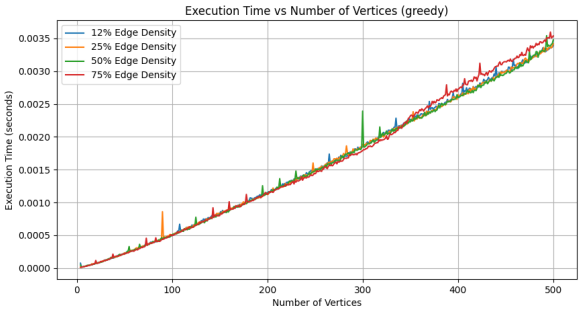

Fig. 9. Execution Time vs. Number of Vertices for Greedy Search

2) Number of Operations: Figure 10 shows the number of operations performed by the greedy algorithm relative to the number of vertices. Similar to the execution time, the operation count exhibits a near-linear relationship, underscoring the efficiency of the greedy approach. As expected, the number of operations is significantly lower than for exhaustive search, demonstrating the advantage of heuristic-based optimization in reducing computational load.

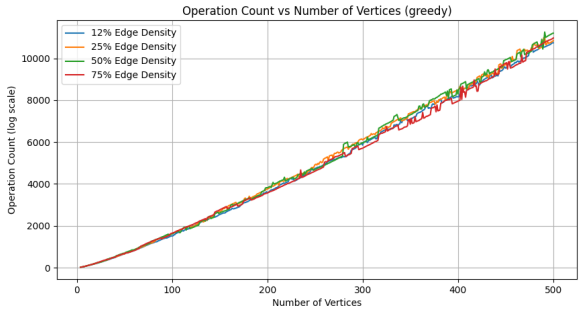

Fig. 10. Operation Count vs. Number of Vertices for Greedy Search

3) Execution Time vs. Edge Density: Figure 11 highlights the impact of edge density on execution time. While denser graphs (with higher edge densities) tend to slightly increase execution time due to more neighbors being marked as skipped, this effect is relatively modest. The greedy algorithm is less sensitive to edge density variations than the exhaustive search, which is beneficial for practical applications where edge density can vary across instances.

D. Effectiveness Analysis of Greedy Search

To evaluate the effectiveness of the greedy algorithm in finding the maximum independent set, we compared the results of the greedy search to those from exhaustive search on smaller graphs. The comparison highlights the frequency of matches between solutions provided by both algorithms.

- **Overall Comparison:** As shown in Fig. 12, the greedy algorithm matched the solution found by exhaustive search in approximately 81.4% of cases. In the remaining 18.6%, the greedy solution was suboptimal.

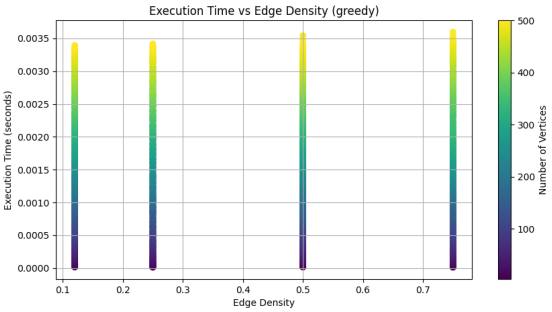

Fig. 11. Execution Time vs. Edge Density for Greedy Search

- **Errors by Vertex Count:** Fig. 13 illustrates the distribution of errors (instances where the greedy solution differed from the exhaustive solution) by the number of vertices. The errors are spread across various graph sizes but are slightly more frequent in graphs with higher vertex counts, which may indicate that the heuristic loses accuracy as graph complexity increases.
- **Errors by Edge Density:** Fig. 14 shows the relationship between edge density and the number of errors made by the greedy algorithm. The highest number of errors occurred in graphs with 50% edge density, followed by 25%, while no errors were observed in graphs with 75% edge density. This trend may be attributed to the nature of the greedy heuristic, which tends to perform better in extremely sparse or dense graphs, where the structural constraints on the independent set are more clearly defined. In contrast, intermediate densities like 50% may create ambiguous scenarios where the heuristic struggles to make optimal choices.

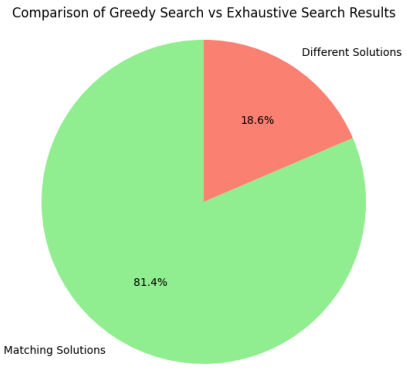

Fig. 12. Comparison of Greedy Search vs. Exhaustive Search Results

V. Conclusion

This report compared an exhaustive search and a greedy heuristic for solving the Maximum Independent Set (MIS) problem. The exhaustive search guarantees optimal solutions

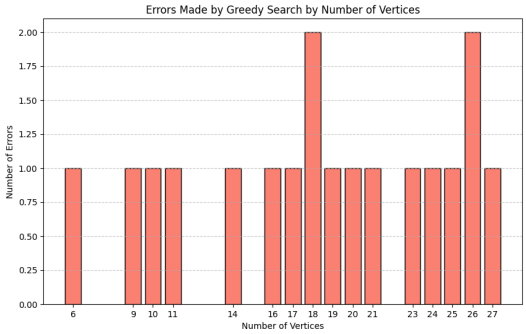

Fig. 13. Errors Made by Greedy Search by Number of Vertices

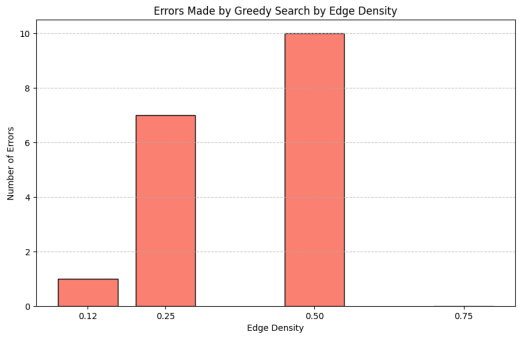

Fig. 14. Errors Made by Greedy Search by Edge Density

but is computationally infeasible for larger graphs due to its exponential complexity. In contrast, the greedy algorithm demonstrated near-linear efficiency, making it suitable for larger graphs, though it was suboptimal in 18.6% of cases. Notably, it performed best in sparse or dense graphs. These results highlight the trade-offs between accuracy and efficiency, emphasizing the importance of selecting algorithms based on graph characteristics and application needs.

References

[1] Wikipedia contributors, "Maximal independent set," *Wikipedia, The Free Encyclopedia*. [Online]. Available: https://en.wikipedia.org/wiki/Maximal$_i ndependent_s et$.

[2] Wikipedia contributors, "Independent set (graph theory)," *Wikipedia, The Free Encyclopedia*. [Online]. Available: https://en.wikipedia.org/wiki/Independent$_s et_(graph_t heory)$.

[3] J. E. Kim, "The Maximum Independent Set Problem," University of Maryland, Baltimore County (UMBC). [Online]. Available: https://userpages.cs.umbc.edu/jekkin1/JEKKIN/MIS.pdf.