Minimum Vertex Cover

Vicente Barros

Abstract **–This report presents a comprehensive analysis of the Minimum Vertex Cover problem, a classic problem in graph theory and combinatorial optimization. We explore two distinct algorithmic approaches — an exhaustive search method and a greedy heuristic — to determine a vertex cover of the smallest possible size in an undirected graph $G(V,E)$ with n vertices and m edges. The exhaustive search approach searches all possible vertex sets, while the greedy heuristic iteratively selects vertices based on a specific criterion. Through a series of experiments on various graph sizes and configurations, we evaluate the algorithms' performance in terms of computational complexity, execution time, and the quality of the solutions.**

Resumo **–Este relatório apresenta uma análise abrangente do problema do *Minimum Vertex Cover*, um problema clássico em teoria dos grafos e otimização combinatória. Exploramos duas abordagens algorítmicas distintas — um método de pesquisa exaustiva e uma heurística gulosa — para determinar um cobrimento de vértices do menor tamanho possível num grafo não direcionado $G(V,E)$ com n vértices e m arestas. A abordagem de busca exaustiva examina todos os possíveis conjuntos de vértices, enquanto a heurística gulosa seleciona iterativamente vértices com base num critério específico. Por meio de uma série de experiências com configurações de grafos, avaliamos o desempenho dos algoritmos em termos de complexidade computacional, tempo de execução e qualidade das soluções.**

Keywords **–Minimum Vertex Cover, Graph Theory, Brute Force Algorithm, Greedy Heuristic, Combinatorial Optimization, Algorithmic Analysis, Computational Complexity**

I. Introduction

Given an undirected graph $G=(V,E)$ where V is the set of vertices and E is the set of edges, the Minimum Vertex Cover problem seeks to find the smallest subset of vertices $C \subseteq V$ such that each edge in E is incident to at least one vertex in C.

Formally, the problem can be defined as follows:

Definition 1. *Given an undirected graph $G=(V,E)$, a vertex cover C is a subset of V such that for every edge $(u,v) \in E$, either $u \in C$ or $v \in C$ or both. The Minimum Vertex Cover problem aims to find a vertex cover of the smallest possible size.*

This report delves into the analysis of two algorithmic approaches for solving the Minimum Vertex Cover problem: the exhaustive search method and a greedy heuristic. The exhaustive search method examines every possible combination of vertices, ensuring the discovery of the optimal solution, albeit with significant computational complexity.

In contrast, the greedy heuristic rapidly constructs a solution by iteratively making local optimal choices, trading off potential optimality for efficiency. The subsequent sections will elaborate on these approaches, followed by a comprehensive experimental analysis to evaluate their performance in various scenarios.

II. Exhaustive Search

The Exhaustive Search approach to solving the Minimum Vertex Cover problem involves systematically exploring all possible combinations of the vertices in a given graph to identify the minimum vertex cover. This method guarantees the discovery of an optimal solution by considering every subset of vertices and checking if it forms a valid vertex cover - this analysis has a big computational cost that will be explained next.

A. Algorithm Overview

The exhaustive search algorithm operates by iterating through all subsets of the vertex set V of the graph $G=(V,E)$. For each subset $C \subseteq V$, the algorithm checks whether C covers all the edges in E. If it does, the algorithm compares its size to the current minimum vertex cover found and updates the minimum if C is smaller.

B. Pseudocode

The following pseudocode outlines the steps of the exhaustive search algorithm:

Algorithm 1 Exhaustive Search for Minimum Vertex Cover

```
1:  function ExhaustiveSearch($G(V, E)$)
2:      $min\_cover \leftarrow V$        ▷ Initialize with all vertices
3:      for each $C \subseteq V$ do
4:          if IsVertexCover($G, C$) then
5:              if $|C| < |min\_cover|$ then
6:                  $min\_cover \leftarrow C$
7:              end if
8:          end if
9:      end for
10:     return $min\_cover$
11: end function
12: function IsVertexCover($G(V, E), C$)
13:     for each $(u, v) \in E$ do
14:         if $u \notin C$ and $v \notin C$ then
15:             return False
16:         end if
17:     end for
18:     return True
19: end function
```

C. Computational Complexity

The exhaustive search for the Minimum Vertex Cover problem is computationally intensive due to its need to check every possible subset of vertices. Given a graph with n vertices, there are 2^n possible subsets to consider, leading to an exponential time complexity of $O(2^n)$, which leads to being impractical to use in bigger graphs. As the algorithm checks every possible solution, the best and worst cases have the same complexity.

Despite these limitations, the exhaustive search consistently identified the optimal minimum vertex cover, affirming its effectiveness for smaller graphs where the computational expense is not a critical factor.

D. Experimental Analysis

The experimental evaluation of the Brute Force algorithm was conducted to measure its performance across varying graph sizes where the number of vertices is $n \in [4, 29]$ and edge densities were generated defining the probability of two vertices have an edge $p \in \{0.125, 0.25, 0.5, 0.75\}$. The metrics considered include the **number of solutions tested**, the **number of operations required**, and the **computational time taken**, which provide insight into the algorithm's scalability and resource requirements.

D.1 Number of Solutions Tested:

The number of solutions tested scales exponentially with graph size, particularly under higher edge densities Figure 1. This exponential growth aligns with the algorithm's combinatorial nature, where potential vertex covers increase as 2^n.

D.2 Number of Operations:

The operations count, including vertex subset checks, mirrors the solutions curve. For larger graphs, especially with dense edge configurations, the number of operations becomes prohibitively high, reinforcing the impracticality of this method for large-scale applications as is visible in Figure 2.

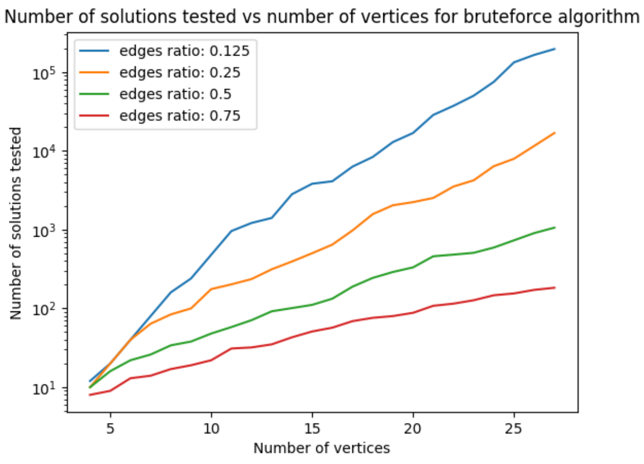

Fig. 1 - The Brute Force algorithm's number of solutions tested, varying with graph size and edge density.

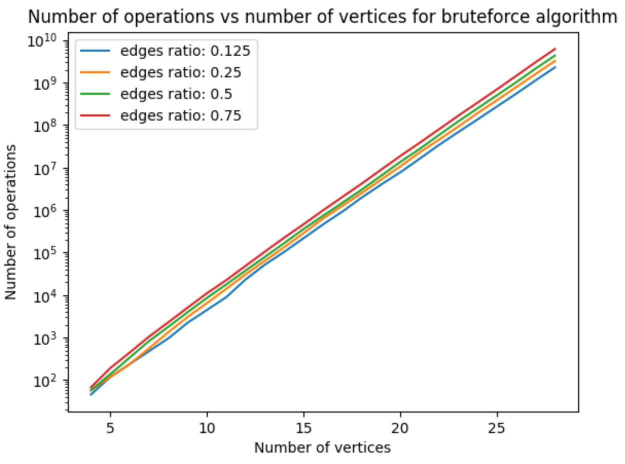

Fig. 2 - Number of operations versus number of vertices for the Brute Force algorithm.

D.3 Execution Time:

Execution time increases dramatically with graph size and edge density, demonstrating the algorithm's exhaustive nature that mandates checking all possible combinations Figure 3. This steep increase highlights the method's unsuitability for large, complex graphs.

The experimental results elucidate the high computational demand of the Brute Force algorithm as the graph grows in size and complexity. While it provides an optimal solution, its application is practically limited to small graphs due to the exponential growth in resource consumption.

III. Greedy Search

The Greedy algorithm for the Minimum Vertex Cover problem offers an efficient heuristic solution by making

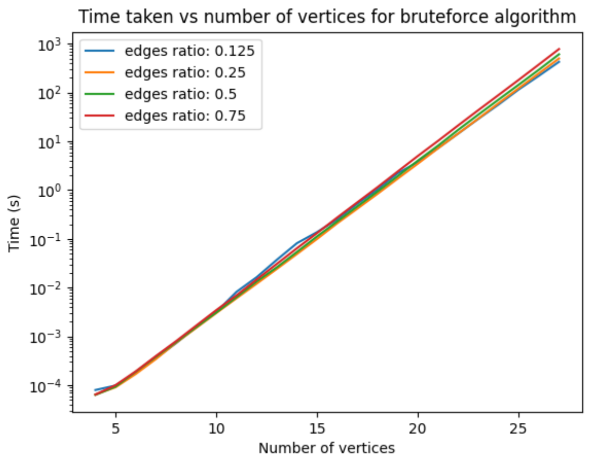

Fig. 3 - Execution time of the Brute Force algorithm as a function of the number of vertices and edge density.

locally optimal choices at each step. This section introduces the Greedy algorithm, followed by an in-depth look at its operation and characteristics.

A. *Algorithm Overview*

The Greedy algorithm incrementally constructs a vertex cover by iteratively selecting the vertex with the highest degree—that is, the most adjacent edges. At each step, the selected vertex and all edges incident to it are removed from the graph, and the vertex is added to the vertex cover set. This process is repeated until there are no remaining edges.

The Greedy heuristic is founded on the principle that by prioritising vertices with higher degrees, the algorithm can cover more edges with fewer vertices. Although this does not guarantee an optimal solution, it often results in a near-optimal vertex cover, especially in graphs where the vertex degrees are not uniformly distributed.

B. *Pseudocode*

The pseudocode for the Greedy algorithm is as follows:

Algorithm 2 Greedy Algorithm for Minimum Vertex Cover

```
1: function GreedyVertexCover($G(V, E)$)
2:     $vertex\_cover \leftarrow \emptyset$
3:     while $E \neq \emptyset$ do
4:         $v \leftarrow$ vertex in $V$ with maximum degree in $E$
5:         $vertex\_cover \leftarrow vertex\_cover \cup \{v\}$
6:         Remove all edges incident to $v$ from $E$
7:     end while
8:     return $vertex\_cover$
9: end function
```

C. *Computational Complexity*

The computational complexity of the Greedy algorithm for the Minimum Vertex Cover problem can be analysed by considering the two primary operations it performs during each iteration: **selecting the vertex with the maximum degree** and **removing the incident edges**.

- **Vertex Selection**: The algorithm selects the vertex with the maximum degree by examining all vertices, which takes $O(n)$ time in each iteration, where n is the number of vertices. Since this selection is made in every iteration, and each iteration removes at least one edge, the selection step contributes $O(n \cdot m)$ over the entire run of the algorithm.
- **Edge Removal**: Once the vertex with the maximum degree is chosen, the algorithm removes this vertex and all edges incident to it. If the graph is represented using an adjacency list, removing all edges incident to a single vertex can be done in $O(\deg(v))$ time, where $\deg(v)$ is the degree of the chosen vertex. The total time for removing edges across all iterations is $O(m)$ since each edge is removed exactly once.

Given that in each iteration at least one edge is removed from the graph, the algorithm will perform at most m iterations. Combining the vertex selection and edge removal operations, the overall time complexity of the Greedy algorithm is $O(m \log n + m)$, which simplifies to $O(m \log n)$, assuming that $m \geq n$.

It is important to note that the complexity could vary depending on the graph's structure. In the best case, where the graph is sparse, the complexity may be lower. Conversely, in dense graphs where m is close to n^2, the $\log n$ factor becomes less significant, and the complexity is dominated by the m factor.

D. *Experimental Analysis*

The Greedy algorithm's performance was empirically evaluated across a range of graphs with varying sizes and edge densities. The experimental results were analysed using the same metrics used previously on the exhaustive implementation as well as the configuration parameters from the previous algorithm. These metrics were plotted against the number of vertices, providing insights into the scalability and efficiency of the Greedy approach.

D.1 *Number of Solutions Tested:*

The Greedy algorithm demonstrates a more controlled increase in solutions tested as graph size and edge density rise Figure 4. While the rate of increase is notable in dense graphs, it remains significantly lower than the Exhaustive Search, reflecting its heuristic efficiency.

D.2 *Number of Operations:*

In comparison, the Greedy algorithm shows a less steep, though still considerable, increase in operations with graph size which is visible in figure 5. This trend suggests better scalability, despite the relationship between graph size and operations count being more than

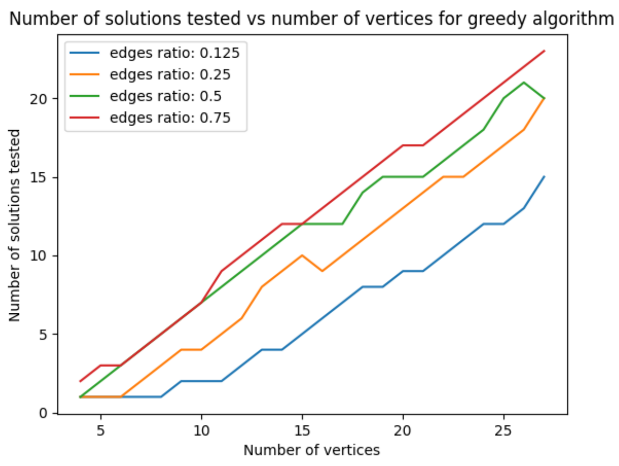

Fig. 4 - Number of solutions tested by the Greedy algorithm for different graph sizes and edge densities.

linear, particularly in denser graphs.

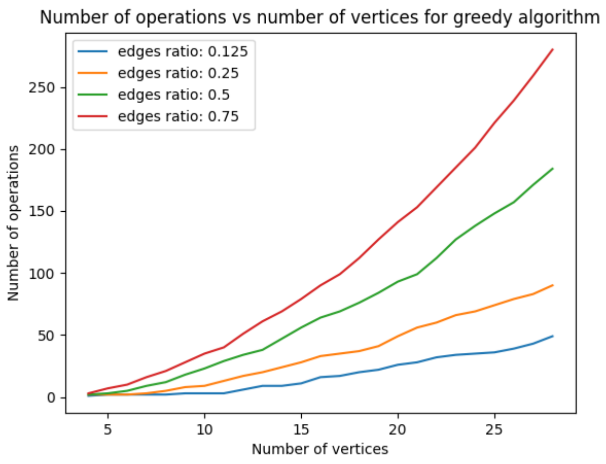

Fig. 5 - Number of operations versus number of vertices for the Greedy algorithm.

D.3 Execution Time:

The execution time for the Greedy algorithm, while also rising with graph size and density, shows a more gradual curve Figure 6. This indicates a relatively more efficient handling of larger and denser graphs, though the increase in computational workload is still evident.

The analysis of the Greedy algorithm demonstrates a predictable increase in the number of solutions tested, the number of operations, and execution time as the graph size grows and becomes denser. While it maintains a performance advantage over exhaustive methods, particularly for larger and denser graphs, the results underscore the importance of considering graph density alongside size when evaluating the Greedy algorithm's efficiency.

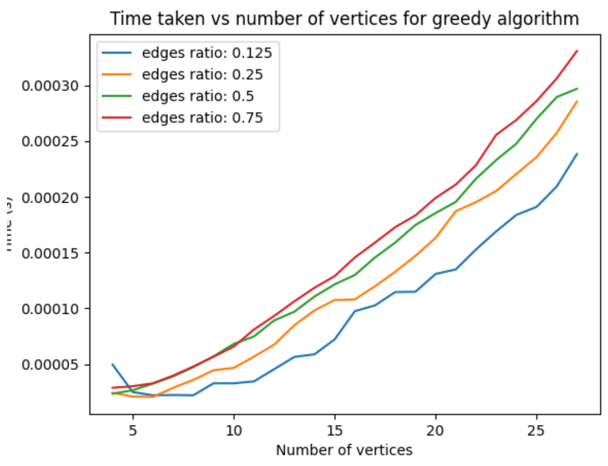

Fig. 6 - Execution time of the Greedy algorithm as a function of the number of vertices.

IV. CONCLUSION

The Brute Force algorithm, while guaranteed to find an optimal solution, was shown to have exponential time complexity. Experimental results confirmed the theoretical predictions, as the number of operations and the execution time increased exponentially with the size of the graph. The practical use of this algorithm is therefore limited to small graphs, where the exhaustive nature of the search does not pose a computational barrier.

On the other hand, the Greedy algorithm emerged as a more practical solution for larger graphs. Despite not guaranteeing an optimal solution, the algorithm's performance in experiments revealed a significant reduction in execution time compared to the Brute Force approach, especially as the graphs became larger and denser.

The Greedy algorithm's time complexity, primarily influenced by vertex selection and edge removal, was observed to be $O(m \log n)$, making it a more scalable option for the Minimum Vertex Cover problem.

In conclusion, the choice of algorithm depends on the size and density of the graph, as well as the requirement for an optimal solution. For small graphs or when optimality cannot be compromised, the Brute Force algorithm is suitable. For larger graphs or when near-optimal solutions are acceptable within practical time constraints, the Greedy algorithm is more appropriate.

REFERENCES

[1] GeeksforGeeks, "Introduction and approximate solution for vertex cover problem", 2023.
URL: `https://www.geeksforgeeks.org/introduction-and-approximate-solution-for-vertex-cover-problem/`

[2] Author(s) not provided, "Title not provided", 2023.
URL: `https://link.springer.com/article/10.1007/s11227-023-05397-8`

[3] Eric W. Weisstein, "Vertex cover", 2023.
URL: `https://mathworld.wolfram.com/VertexCover.html`

[4] Wikipedia contributors, “Vertex cover”, 2023.
URL: `https://en.wikipedia.org/wiki/Vertex_cover`