Minimum Vertex Cover Problem

Manuel Diaz

***Resumo* - Este relatório apresenta uma análise abrangente de algoritmos aleatórios para resolver o problema do Minimum Vertex Cover, um desafio clássico na teoria dos grafos e na otimização combinatória. Com base nas abordagens exaustivas e gulosas exploradas no primeiro projeto, foram implementados e avaliados dois métodos aleatórios: os algoritmos Randomized Vertex Cover e Randomized Vertex Cover Min. Estes métodos utilizam a aleatoriedade para aproximar soluções, equilibrando a eficiência computacional e a qualidade da solução através de contagens de iteração ajustáveis.**

Através de uma série de experiências em grafos gerados aleatoriamente com tamanhos e densidades variáveis, analisamos o desempenho destes algoritmos em termos de precisão, complexidade computacional e tempo de execução. Os resultados demonstram que, embora os algoritmos aleatórios nem sempre superem a pesquisa gulosa em termos de exatidão, constituem uma alternativa escalável e flexível, particularmente para grafos densos ou grandes, em que os métodos exaustivos são impraticáveis. Este estudo destaca os compromissos entre a qualidade da solução e a eficiência, oferecendo uma visão sobre a aplicabilidade de abordagens aleatórias para problemas de otimização NP-difíceis.

***Abstract* - This report presents a comprehensive analysis of randomized algorithms for solving the Minimum Vertex Cover problem, a classic challenge in graph theory and combinatorial optimization. Building on the exhaustive and greedy approaches explored in the first project, it was implemented and evaluated two randomized methods: the Randomized Vertex Cover and Randomized Vertex Cover Min algorithms. These methods leverage randomness to approximate solutions while balancing computational efficiency and solution quality through adjustable iteration counts.**

Through a series of experiments on randomly generated graphs with varying sizes and densities, we analyze the performance of these algorithms in terms of accuracy, computational complexity, and execution time. The results demonstrate that while randomized algorithms do not always outperform greedy search in accuracy, they provide a scalable and flexible alternative, particularly for dense or large graphs where exhaustive methods are impractical. This study highlights the trade-offs between solution quality and efficiency, offering insights into the applicability of randomized approaches for NP-hard optimization problems.

I. INTRODUCTION

The Minimum Vertex Cover problem is a well-known challenge in graph theory and combinatorial optimization, with applications across a variety of fields, including network security, bioinformatics, scheduling and resource management [1]. Formally, given an undirected graph, $G = (V, E)$, the objective is to identify the smallest subset of vertices $C \subseteq V$ such that every edge in E has at least one endpoint in C. This subset is called a vertex cover and indeed covers all edges, so the problem is a typical problem of a complete covering of edges using the minimum amount of resources.

Building upon the methods explored in the first project – exhaustive search and greedy heuristic approaches – we now extend the investigation to include randomized algorithms. Randomized approaches introduce an element of stochasticity, generating solutions either purely at random or by combining randomness with heuristic strategies [2]. These methods aim to balance solution quality and computational efficiency, especially for large or complex graphs where deterministic methods become computationally prohibitive.

In this report, we focus on two randomized algorithms for solving the MVC problem. The first algorithm (**randomized vertex cover**) starts with large subsets of vertices, progressively reducing their size over iterations, while the second algorithm (**randomized vertex cover min**) begins with small subsets and incrementally increases their size. Both methods incorporate randomness to explore the solution space, avoiding redundancy by ensuring that no subset is tested more than once.

In the subsequent sections we will comparatively analyse both algorithms for computational complexity, execution time, and the quality of the obtained solution by a series of experiments on randomly generated graphs of various sizes and edge densities. The results are further compared with the exhaustive and greedy approaches developed in the first project to evaluate the effectiveness and scalability of randomized methods.

II. RANDOMIZED VERTEX COVER

The randomized vertex cover algorithm is a randomized approach to approximating solutions for the Minimum Vertex Cover (MVC) problem. This algorithm generates candidate vertex subsets by incorporating randomness into the selection process, progressively reducing the size of the subsets over successive iterations. The goal is to balance exploration of the solution space with computational efficiency, identifying a vertex cover that is near-optimal within a practical time frame.

A. Algorithm Overview

The algorithm operates by dynamically adjusting the size of randomly selected vertex subsets as iterations progress. Initially, larger subsets are generated, ensuring broader edge coverage, and as iterations continue, the subset size decreases to fine-tune the solution. Each randomly generated subset is tested to determine if it constitutes a valid vertex cover, and redundant subsets are avoided by maintaining a record of those already evaluated. The best solution is updated whenever a valid vertex cover smaller than the current best is found.

B. Complexity analysis

The computational complexity of the randomized vertex cover algorithm depends on the number of iterations and the operations performed during each iteration:

- **Subset Generation:** Generating a random subset requires $O(n)$ time, where n is the number of vertices.

- **Validation:** Checking whether the subset is a valid vertex cover requires $O(m)$, where m is the number of edges.

- **Redundancy Check:** Avoiding duplicate subsets involves a hash set lookup, which is $O(1)$ on average.

Overall, the complexity per iteration is $O(n+m)$, and for k iterations, the total complexity is $O(k \cdot (n+m))$. This linear growth with respect to k and the graph size demonstrates significantly better scalability than exhaustive search.

C. Implementation

The code present in the *Algorithm 1* outlines the steps of the randomized vertex cover algorithm.

The *randomized_vertex_cover* function generates random vertex subsets and evaluates each one through the helper function *is_vertex_cover*. It progressively reduces the size of the subsets over the iterations to refine the solutions space.

D. Experimental Analysis

To evaluate the performance of the randomized vertex cover algorithm in solving the Minimum Vertex Cover problem, it was conducted a series of experiments using randomly generated graphs with varying sizes where the number of vertices is $n \in [4, 256[$ and edge densities were

```
def is_vertex_cover(graph, vertex_set):
  for u, v in graph.edges():
    if u not in vertex_set and v not in vertex_set:
      return False
  return True

def randomized_vertex_cover(graph, iterations=1000):
   best_cover = None
   total_vertices = len(graph.nodes())
   checked_subsets = set()

   for i in range(iterations):
      subset_size = max(1, ceil(total_vertices * (1 - i / iterations)))
      random_subset = random.sample(graph.nodes(), subset_size)

      subset_key = frozenset(random_subset)
      if subset_key not in checked_subsets:
         checked_subsets.add(subset_key)

         if is_vertex_cover(graph, random_subset) and (best_cover is None or len(random_subset) < len(best_cover)):
            best_cover = set(random_subset)

   return best_cover
```

Algorithm 1: Randomized Vertex Cover for Minimim Vertex Cover.

generated defining the probability of two vertices have an edge $p \in \{0.125, 0.25, 0.5, 0.75\}$. The random seed was fixed to ensure reproducibility.

Preliminary analyses were performed with different iteration limits (100, 500, 1000, 5000 and 10000) to observe the effect of the number of iterations on the algorithms's performance. The results showed a clear trend: increasing the number of iterations improved the quality of the solutions, but the gains diminished significantly after 5000 iterations. For simplicity and clarity, the results presented in this report focus on the maximum of 10000 iterations, which represents the full exploration of the solution space within the iteration constraints.

For each graph configuration, we recorded the following metrics: **number of solutions tested**, **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 corresponds directly to the iteration limit imposed on the algorithm, as the algorithm ensures that each tested subset is unique. As shown in *Figure 1*, the number of solutions tested quickly reaches the maximum allowed (10,000 iterations) regardless of the graph's edge density (p). This behavior indicates that the iteration cap is the dominant factor controlling the number of solutions tested, independent of the graph's structure or connectivity.

For smaller graphs (fewer vertices), the number of solutions tested grows rapidly as the algorithm explores subsets, but it quickly plateaus as the iteration cap is reached. This consistency across different edge densities suggests that the randomized nature of the algorithm ensures uniform exploration of the solution space, unaffected by the connectivity of the graph.

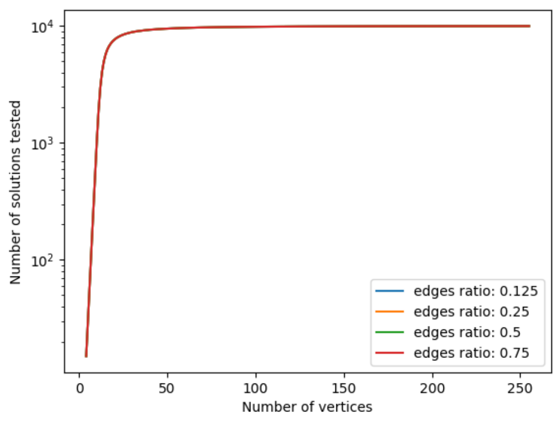

Figure 1: The Randomized Vertex Cover algorithm's number of solutions tested in function of the number of vertices and edge density.

D.2 Number of Operations:

The operation count, which includes checks for each vertex subset, grows as a function of both the number of vertices (n) and the graphs's edge density (p). As shown in *Figure 2*, the number of operations increase significantly with higher edge densities, reflecting the additional computational effort required to handle the larger number of edges in denser graphs.

For sparse graphs, the growth is slower, with fewer operations required even as the number of vertices increases. In contrast, for denser graphs, the growth is much steeper, as a greater number of edges must be considered in each iteration of the algorithm.

D.3 Execution Time:

The execution time of the randomized vertex cover algorithm exhibits a growth trend directly proportional to the number of vertices and the graph's edge density. As shown in *Figure 3*, denser graphs require significantly more time to process compared to sparser ones, due to the increased number of edges that influence the computation.

Despite the general trend, fluctuations and periodic spikes in the execution time are observed, reflecting the algorithm's randomization and the varying complexity of the input graphs. These fluctuations are more pronounced for denser graphs, where the algorithm has to handle more connections.

The results indicate that while the algorithm performs efficiently for smaller graphs and lower edge densities, its execution time scales less favorably with increasing graph size and density. This underlines the trade-off between accuracy and computational efficiency in randomized algorithms for NP-hard problems like Minimum Vertex Cover.

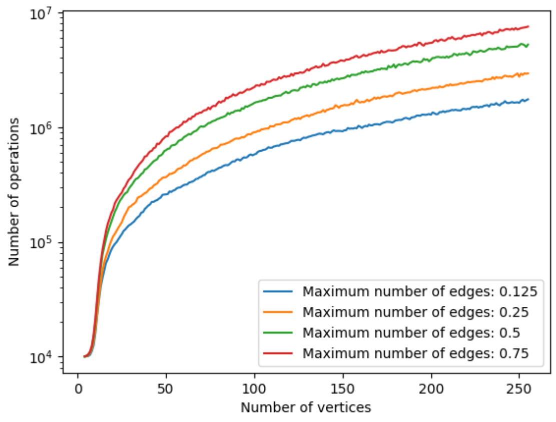

Figure 2: The Randomized Vertex Cover algorithm's number of operations in function of the number of vertices and edge density.

III. RANDOMIZED VERTEX COVER MIN

The randomized vertex cover min algorithm builds upon the principles established by the randomized vertex cover algorithm. While the first algorithm prioritizes larger initial subsets that shrink over time, the second inversely begins with smaller subsets that grow incrementally. This adjustment introduces a different dynamic in exploring the solution space, emphasizing finer-grained exploration in earlier iterations.

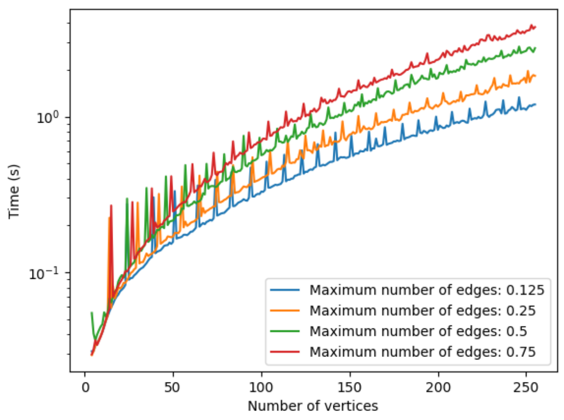

Figure 3: The Randomized Vertex Algorithm algorithm's time execution in function of the number of vertices and edge density.

A. *Algorithm Overview*

The algorithm modifies the subset-selection strategy from its predecessor:

- **Subset Selection Dynamics:** Instead of starting with larger subsets (as in the first algorithm), it begins with smaller subsets. Over iterations, the subset size increases, allowing for broader coverage in later stages.
- **Subset Uniqueness:** As in the first algorithm, previously tested subsets are tracked to avoid redundant evaluations.
- **Best Solution Tracking:** The smallest valid vertex cover found during the iterations is stored.

B. *Complexity Analysis*

The computational complexity of the randomized vertex cover min is the same of the randomized vertex cover algorithm, but first's runtime behaviour benefits from the more incremental growth in subset sizes, especially for dense graphs.

C. Implementation

The code for the randomized vertex cover min algorithm is as the follows in *Algorithm 2*.

```
def is_vertex_cover(graph, vertex_set):
  for u, v in graph.edges():
    if u not in vertex_set and v not in vertex_set:
      return False
  return True

def randomized_vertex_cover(graph, iterations=1000):
   best_cover = None
   total_vertices = len(graph.nodes())
   checked_subsets = set()

   for i in range(iterations):
      subset_size = min(total_vertices, ceil((i + 1) /
iterations * total_vertices))
      random_subset = random.sample(graph.nodes(),
subset_size)

      subset_key = frozenset(random_subset)
      if subset_key not in checked_subsets:
         checked_subsets.add(subset_key)

         if is_vertex_cover(graph, random_subset) and
(best_cover is None or len(random_subset) <
len(best_cover)):
            best_cover = set(random_subset)

   return best_cover
```

Algorithm 2: Randomized Vertex Cover Min for Minimum Vertex Cover

D. *Experimental Analysis*

To evaluate the performance of this algorithm, it was conducted experiments similar to those in the first analysis, using randomly generated graphs with varying sizes and edge densities. It is used the same metrics used previously too, allowing us to compare the efficiency and accuracy of the two approaches.

D.1 Number of Solutions Tested:

The number of unique subsets evaluated by the algorithm remains consistent across different iterations. This behaviour aligns with the subset exploration mechanism and indicates that the avoidance of duplicate evaluations through the *checked_subsets* (*Algorithm 2*) structure works effectively in both algorithms. The gradual increase in subset size in randomized vertex cover min does not drastically affect the total solutions explored compared to the baseline algorithm, as we can see in the *Figure 4*.

D.2 Number of Operations:

The number of operations required scales similarly with the number of vertices and edges in the graph. While both algorithms exhibit comparable growth patterns, the randomized vertex cover min might demonstrate slight variations in operations count due to differences in subset generation and validation dynamics. These differences, however, are marginal, as reflected in the overlapping trends seen in the operations count graph (*Figure 5*).

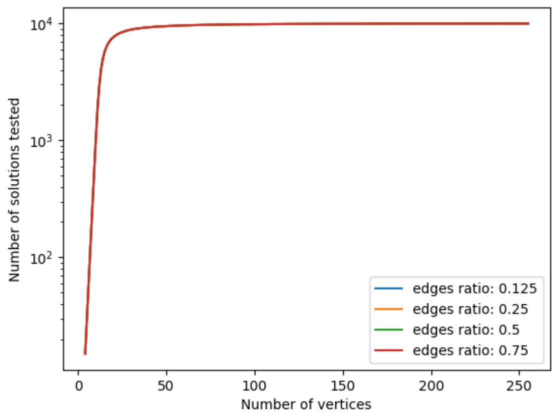

Figure 4: The Randomized Vertex Cover Min algorithm's number of solutions tested in function of the number of vertices and edge density.

D.3 *Execution Time:*

The execution time of the randomized vertex cover min algorithm closely mirrors that of the randomized vertex cover algorithm. Both algorithms demonstrate logarithmic to linear growth in time complexity with respect to the size of the input graph. The similarity in execution time underscores the comparable computational overhead introduced by both approaches, with no significant trade-off in runtime efficiency for the modified exploration strategy.

As we can see in *figure 6*, we can only notice fewer variations in time in early phases, especially in dense graphs ($p = 0.75$), perhaps due to the greater initial efficiency of this algorithm due to smaller subsets.

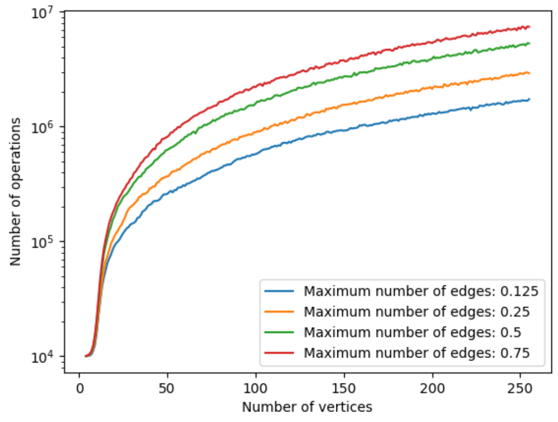

Figure 5: The Randomized Vertex Cover Min algorithm's number of operations in function of the number of vertices and edge density.

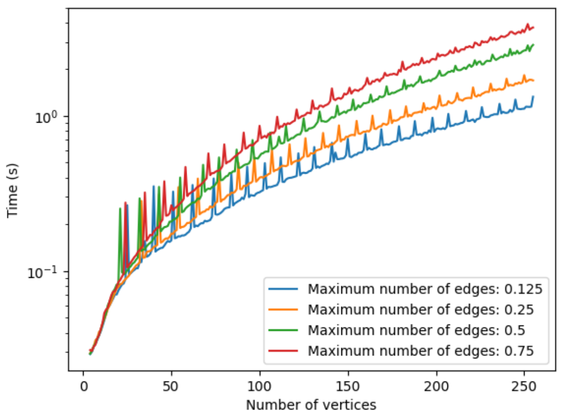

Figure 6: The Randomized Vertex Cover Min algorithm's time execution in function of the number of vertices and edge density.

IV. COMPARISON WITH PREVIOUS ALGORITHMS

To evaluate the accuracy of the randomized algorithms and the greedy approach, it was compared their solutions with the optimal solutions produced by the brute force method. The number of matching solutions (**hits**), the number of non-matching solutions (**misses**), and the proportion of hits (**accuracy**) were calculated. The results provide insights into the trade-offs between solution quality and computational efficiency for each approach.

The **brute force algorithm** achieved a perfect accuracy of 100% (104/104 hits), as expected, since it exhaustively evaluates all possible vertex covers to find the optimal solution. However, this comes at the cost of exponential computational complexity, making it infeasible for larger graphs [1].

The **greedy search** algorithm demonstrated a hit rate of approximately 55% (58/104 hits). While it is computationally efficient, its heuristic approach often leads to suboptimal solutions, especially in denser graphs where local decisions (e.g., choosing the vertex with the highest degree) do not guarantee global optimality [3].

The **randomized algorithms** showed varying levels of accuracy, depending on the number of iterations and the specific strategy used:

- **Randomized vertex cover:** Achieved a maximum accuracy of ~24% (25/104 hits) with 10,000 iterations. This indicates that higher iteration counts improve solution quality, though it remains significantly below greedy search.

- **Randomized vertex cover min:** Reached an accuracy of ~18.3% (19/104 hits) with 10,000 iterations. This slight drop in accuracy compared to the standard randomized approach suggests that starting with smaller subsets may trade off initial broad exploration for later refinement.

Algorithm	**Hits**	**Misses**	**Accuracy (%)**
Brute Force	104	0	100.0
Greedy Search	58	46	55.0
R. V. C (10000)	25	79	24.0
R. V. C. M. (10000)	19	85	18.3
R. V. C (100)	11	93	10.5

TABLE I
SUMMARIZING HITS, MISSES, AND ACCURACY FOR EACH ALGORITHM

As we can see in *Table I*, the randomized algorithms demonstrated lower accuracy compared to the greedy search, with hit rates ranging from 10.5% (100 iterations) to 24.0% (10,000 iterations), highlighting the importance of iteration count in improving solution quality. While the greedy approach achieved a higher accuracy of 55.0%, the flexibility of randomized methods allows them to balance computational effort and solution quality, making them a viable alternative in scenarios where greedy search may produce significantly suboptimal solutions, such as in dense graphs.

V. CONCLUSION

This project explored the application of randomized algorithms for approximating solutions to the Minimum Vertex Cover problem, building upon the exhaustive and heuristic approaches analyzed in the first project. The randomized methods introduced flexibility in balancing computational effort and solution quality, particularly through the adjustable iteration count.

The experimental results demonstrated that while the randomized algorithms did not consistently outperform the greedy search in terms of accuracy, they provided a competitive alternative, especially for denser graphs where greedy heuristics may fall short.

In terms of scalability, the randomized methods offered significant advantages over brute force, handling graphs of up to 256 vertices with linear growth in execution time. However, the trade-off between accuracy and computational cost remains evident, as higher iteration counts improved results but increased runtime.

Overall, the randomized algorithms provide a versatile framework for tackling the Minimum Vertex Cover problem in scenarios where exhaustive methods are infeasible, and greedy heuristics may lack precision [2].

.

REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[2] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[3] GeeksforGeeks, "Introduction and Approximate Solution for Vertex Cover Problem." [Online]. Available: https://www.geeksforgeeks.org/introduction-and-approximate-solution-for-vertex-cover-problem/