2nd Project – Vertex Cover of size k

Filipe Silveira - 97981

***Resumo* –O problema da cobertura de vértices envolve encontrar um conjunto de vértices num grafo não direcionado de maneira a que cada aresta esteja incidente em, pelo menos, um vértice do conjunto. Este problema é conhecido por ser NP-completo [1], o que indica que encontrar um algoritmo eficiente é improvável. Este relatório explora e analisa duas abordagens aleatórias para resolver o problema do conjunto de vértices de cobertura.**

***Abstract* –The vertex cover problem involves finding a set of vertices in a given undirected graph such that each edge is incident to at least one vertex in the set. This problem is known to be NP-complete [1], indicating that finding an efficient algorithm is unlikely. This report explores and analyzes two randomized approaches to solve the vertex cover problem.**

***Keywords* –Vertex Cover, Randomized, Attempts, Algorithm**

I. Introduction

The vertex cover problem, a cornerstone in graph theory, involves determining the existence of a k-vertex cover in an undirected graph $G(V, E)$ with n vertices and m edges. Being NP-complete, this problem holds significance across domains like network design and optimization. This report investigates a randomized algorithm for solving the vertex cover problem. It provides a detailed examination of two distinct randomized approaches: the primary algorithm and an alternative method that includes a premature termination using a maximum number of attempts and the addition of not testing the same solution. Through formal complexity analysis, empirical experiments (on expanding problem instances) and a comparative study of the results, this report aims to unveil the algorithm's strengths and practical implications. The algorithms were developed in Python. The code is present in the **algorithms.py** script in the folder */code*. To run the program, you can access the root folder and run the batch file *run.bat* or the following commands:

```
> cd .\code\
> python main.py
```

II. Randomized Algorithm

This section describes a randomized algorithm designed to find a vertex cover of size k in an undirected graph $G(V, E)$. The algorithm employs a probabilistic approach to select vertices that could potentially form a vertex cover.

A. Algorithm Description

The algorithm begins by initializing an empty set vertex_cover and a copy of the original graph remaining_graph. It also maintains a set added_vertices to track vertices already added to the vertex cover. The algorithm runs until the size of vertex_cover reaches k or there are no more edges in remaining_graph.

In each iteration, the algorithm selects a random edge from remaining_graph and then randomly chooses one of its vertices. If this vertex is not already in added_vertices, it is added to both vertex_cover and added_vertices. Subsequently, all edges connected to this vertex are removed from remaining_graph.

The algorithm returns the vertex cover if it successfully reduces remaining_graph to have no edges and vertex_cover is of size k. Otherwise, it returns None. It also tracks the number of operations, the execution time, and the number of solutions tested.

B. Pseudocode

C. Formal Analysis

The time complexity of the randomized vertex cover algorithm can be approximated as $O(k \cdot m)$, where k is the size of the desired vertex cover, and m is the number of edges. In each iteration, the algorithm randomly selects an edge and then a vertex from this edge, removing all edges incident to this vertex. This process is repeated up to k times or until no more edges remain.

In terms of complexity analysis:

- **Best Case:** The best case occurs when the algorithm efficiently finds a vertex cover that covers all edges early in the process. This scenario's complexity can be better than $O(k \cdot m)$, depending on the graph's structure and the randomness of the selections.
- **Average Case:** The average case complexity remains at $O(k \cdot m)$. On average, the algorithm may require several iterations to find an effective vertex cover, with each iteration involving random edge and vertex selections and edge removals.
- **Worst Case:** The worst case occurs when the algorithm iterates k times, each time dealing with a significant number of edges. Thus, the time complexity is $O(k \cdot m)$.

Algorithm 1 Randomized FPT

```
1:  procedure RANDOMIZED_FPT($graph, k$)
2:      $vertex\_cover \leftarrow$ set()
3:      $remaining\_graph \leftarrow graph.copy()$
4:      $added\_vertices \leftarrow$ set()
5:      while len($vertex\_cover$) $< k$ do
6:          if len($remaining\_graph.edges()$) $= 0$ then
7:              break
8:          end if
9:          $graph\_edges \leftarrow$ list($remaining\_graph.edges()$)
10:         $edge \leftarrow$ random.choice($graph\_edges$)
11:         $vertex \leftarrow$ random.choice($edge$)
12:         if $vertex \in added\_vertices$ then
13:             continue
14:         end if
15:         $vertex\_cover.add(vertex)$
16:         $added\_vertices.add(vertex)$
17:         $edges\_to\_remove \leftarrow$ [$e$ for $e$ in $remaining\_graph.edges(vertex)$]
18:         $remaining\_graph.remove\_edges\_from(edges\_to\_$
19:     end while
20:     if len($remaining\_graph.edges()$) $= 0$ and len($vertex\_cover$) $= k$ then
21:         return $vertex\_cover$
22:     else
23:         return None
24:     end if
25: end procedure
```

D. Experimental Analysis

To empirically assess the algorithm's performance, it was tested on randomly generated graphs with varying numbers of vertices and edges. The parameters were:

- n (number of vertices) Range: From 4 to 256, to cover a broad spectrum of graph sizes.
- m (number of edges) Proportions: Fixed percentages of the maximum possible number of edges, specifically {0.125, 0.25, 0.5, 0.75} of the maximum.
- k (vertex cover size) Proportions: Relative to the number of vertices, with proportions {0.125, 0.25, 0.5, 0.75}.

Two metrics were evaluated:

1. Number of Basic Operations: Operations for each random selection of edges and vertices, and for edge removals.
2. Number of Solutions Tested: Total number of vertex cover sets attempted until the algorithm terminates.

For the analysis, we focused on the first iteration, showing different comparisons across different k percentages.

D.1 Analysis of Operations

The number of operations required by the algorithm shows distinct growth patterns that vary with the edge density and vertex cover size. Figure 1 and 2 depict the operations for the first iteration.

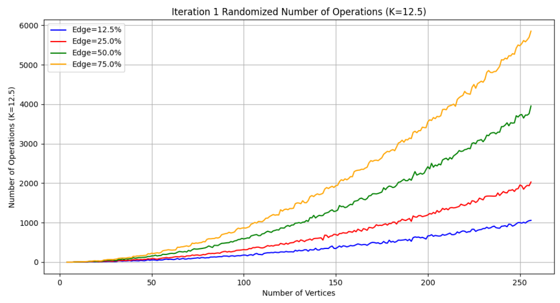

Fig. 1 - FPT | Number of operations for $K = 12.5$.

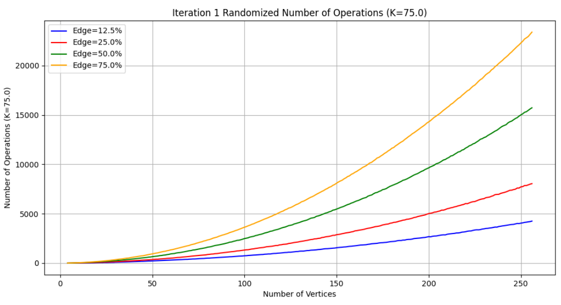

Fig. 2 - FPT | Number of operations for $K = 75.0$.

D.2 Analysis of Solutions Tested

Similarly, the number of solutions tested is depicted in Figures 3 and 4 for the first iteration.

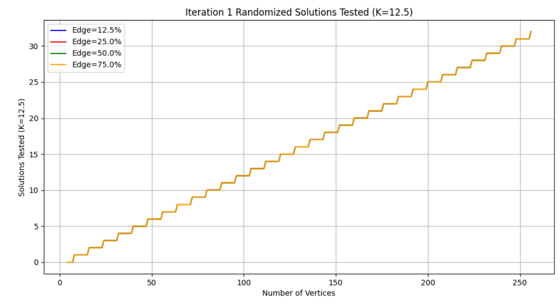

Fig. 3 - FPT | Number of solutions tested for $K = 12.5$.

D.3 Comparative Conclusions

Upon comparing the figures, it becomes evident that the algorithm's behavior maintains consistency in terms of the number of solutions tested, which appears to grow linearly as seen in Figures 3 to 4. However, a different pattern emerges when analyzing the number of operations. While the operations for $K = 12.5$

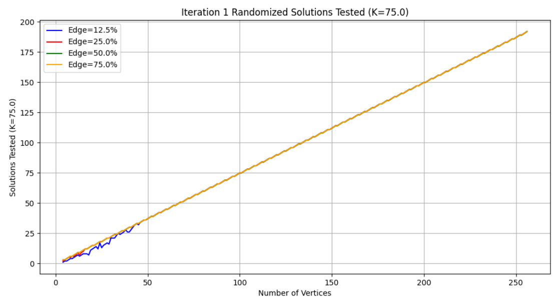

Fig. 4 - FPT | Number of solutions tested for $K = 75.0$.

exhibit a growth that could be interpreted as polynomial (Figure 1), the operations for $K = 75.0$ suggest a steeper, possibly exponential increase (Figure 2). This steepening trend for larger vertex covers implies that the algorithmic complexity is significantly affected by the vertex cover size sought.

The edge density also plays a crucial role. Higher edge densities lead to more complex graphs, which in turn require more operations, highlighting the non-linear impact of graph density on the computational workload.

Overall, the data suggests that while the number of solutions tested remains relatively stable and linear, the operations required by the algorithm for larger vertex covers and denser graphs could be subject to exponential increases in complexity. This underlines the importance of considering both the vertex cover size and edge density when assessing the performance and scalability of such randomized algorithms, however this algorithm very rarely finds a solution. This problem will be talked about more in the following sections.

III. Randomized Algorithm with Attempts

The second algorithm, *randomized_fpt_attempts*, was conceived to improve the solution-finding strategy utilized by the first algorithm. In particular, this algorithm was designed to mitigate the first algorithm's shortfall in discovering a sufficient number of viable solutions by performing multiple attempts and ensuring each set of vertices selected to form a potential vertex cover was unique, expanding the search space without revisiting the previously tested configurations. Although this proved to be more time-consuming, as explained further ahead, it did provide more solutions.

A. Pseudocode

Algorithm 2 Randomized FPT Attempts

```
procedure RANDOMIZED_FPT_ATTEMPTS($graph, k, max\_attempts = 10000$)
    $vertex\_cover \leftarrow$ set()
    $tested\_combinations \leftarrow$ set()
    for _ $\in$ range($max\_attempts$) do
        $vertex\_cover \leftarrow$ set()
        $remaining\_graph \leftarrow graph.copy()$
        $added\_vertices \leftarrow$ set()
        while len($vertex\_cover$) $< k$ do
            if len($remaining\_graph.edges()$) $= 0$ then
                break
            end if
            $edge \leftarrow$ random.choice(list($remaining\_graph.edges()$))
            $vertex \leftarrow$ random.choice($edge$)
            if $vertex \in added\_vertices$ then
                continue
            end if
            $vertex\_cover.add(vertex)$
            $added\_vertices.add(vertex)$
            $edges\_to\_remove \leftarrow$ [$e$ for $e$ in $remaining\_graph.edges(vertex)$]
            $remaining\_graph.remove\_edges\_from$ (edges_to_remove)
        end while
        $combination \leftarrow$ frozenset($vertex\_cover$)
        if $combination \in tested\_combinations$ then
            continue
        end if
        $tested\_combinations.add(combination)$
        if is_vertex_cover($graph, vertex\_cover$) then
            break
        end if
    end for
    if len($remaining\_graph.edges()$) $= 0$ and len($vertex\_cover$) $= k$ then
        return $vertex\_cover$
    else
        return None
    end if
end procedure
```

B. Formal Analysis

The time complexity of the *randomized_fpt_attempts* algorithm is influenced by introducing a maximum number of attempts to find a vertex cover. Each attempt randomly selects edges and vertices and removes all incident edges to the selected vertex, similar to the first algorithm.

In terms of complexity analysis:

- **Best Case:** The best case occurs when a vertex cover of size k is found without reaching the maximum number of attempts and without revisiting previously tested configurations. This scenario benefits from the non-redundant solution space exploration and can have a complexity better than $O(k \cdot m)$. However, this is highly dependent on the randomness of the selection process and the graph structure.
- **Average Case:** The average case considers the potential revisiting of certain vertex sets due to random selection despite the attempt to avoid redundancy. This could result in an average time complexity still approximately $O(k \cdot m)$, with the

added overhead of managing the tested combinations.

- **Worst Case:** The worst-case scenario occurs when the maximum number of attempts is reached without finding a satisfactory vertex cover. In this case, the algorithm's complexity is affected by the cap on attempts and the overhead of tracking tested combinations, potentially leading to a complexity of $O(max\_attempts \cdot k \cdot m)$, assuming the worst case happens at the last attempt.

C. *Experimental Analysis*

To empirically assess the algorithm's performance, it was tested on randomly generated graphs with varying numbers of vertices and edges. The parameters were:

- n (number of vertices) Range: From 4 to 256, to cover a broad spectrum of graph sizes.
- m (number of edges) Proportions: Fixed percentages of the maximum possible number of edges, specifically {0.125, 0.25, 0.5, 0.75} of the maximum.
- k (vertex cover size) Proportions: Relative to the number of vertices, with proportions {0.125, 0.25, 0.5, 0.75}.

Two metrics were evaluated:

1. Number of Basic Operations: Operations for each random selection of edges and vertices, and for edge removals.
2. Number of Solutions Tested: Total number of vertex cover sets attempted until the algorithm terminates.

For the analysis, we focused on the first iteration, as we did in the *randomized_fpt* algorithm analysis.

C.1 *Analysis of Operations*

The number of operations indicates the algorithm's computational effort for each vertex cover size and edge density. As evidenced by Figures 5 and **??**, the algorithm demonstrates a growth pattern in the number of operations, which appears to be influenced by the edge density. Higher edge densities correspond to more operations, suggesting a non-linear relationship between the edge density and computational complexity.

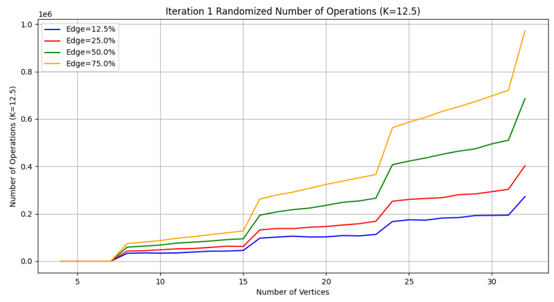

Fig. 5 - FPT Attempts | Number of operations for $K = 12.5$.

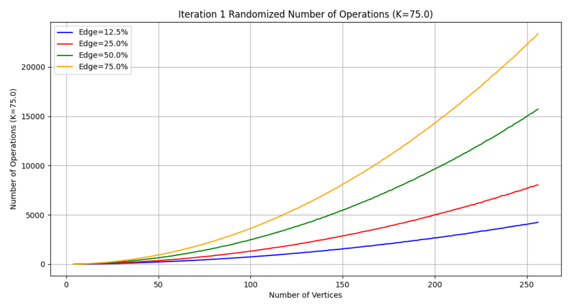

Fig. 6 - FPT Attempts | Number of operations for $K = 75.0$.

C.2 *Analysis of Solutions Tested*

The number of solutions tested reflects the algorithm's exploration depth in the search for a valid vertex cover. Figures 7 and 8 display the total number of vertex cover sets attempted. The variability observed, particularly the spikes in the number of solutions tested, suggests instances where the algorithm may be inefficiently cycling through combinations, especially at higher edge densities.

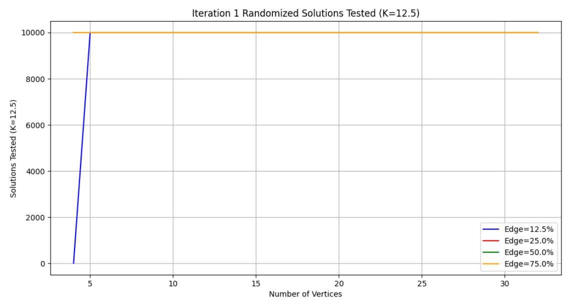

Fig. 7 - FPT Attempts | Number of solutions tested for $K = 12.5$.

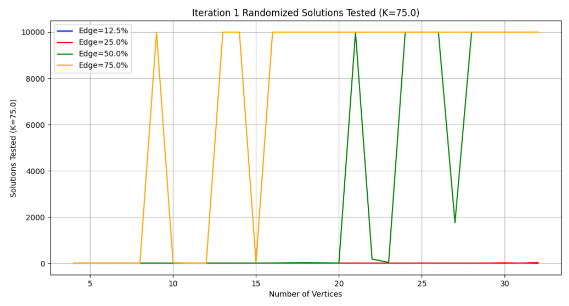

Fig. 8 - FPT Attempts | Number of solutions tested for $K = 75.0$.

C.3 *Conclusions from the Experimental Analysis*

The single iteration analysis shows that the *randomized_fpt_attempts* algorithm exhibits considerable variability in its performance, which is heavily influenced by the graph's edge density. While the algorithm can explore a wide range of solutions, the spikes in the number of solutions tested indicate potential inefficiencies. The increased number of operations with higher edge densities also suggests that the algorithm's computational load may grow significantly with more complex graphs. These insights underscore the necessity for further optimization to improve the algorithm's consistency and efficiency.

D. Performance of the Second Algorithm

The algorithm introduces a cap of 10000 on the number of solution attempts, aiming to overcome the limitations of the first algorithm in finding sufficient feasible solutions. By tracking previously tested vertex combinations, the algorithm avoids redundant computations. However, the performance, as demonstrated in Figures 7 and 8, showed substantial variability in the number of solutions tested, which sometimes led to high peaks. These peaks indicate inefficiency and a potential oversimplification in the solution space exploration.

E. Comparative Analysis with the First Algorithm

Compared to the first algorithm, the second algorithm's objective was to enrich the set of solutions rather than minimize the computational time. Despite this, the second algorithm did not consistently yield a broader spectrum of solutions. The erratic nature of the results, especially in the number of solutions tested, highlights the challenges in balancing exploration depth with computational efficiency. In contrast, despite its limitations in discovering diverse solutions, the first algorithm exhibited a more stable and predictable pattern.

The findings suggest that the second algorithm's strategy of limiting attempts may not be sufficient to address the complexities of the vertex cover problem. Future enhancements could focus on more sophisticated methods of solution space exploration to reliably extend the range of solutions discovered.

IV. Results

For this section, the results presented demonstrate the total number of solutions, found between the algorithms *randomized_fpt* and *randomized_fpt_attempts*, of graphs ranging from 4 to 32 vertices. To maintain consistent and even results, the table I shows the results

During the analysis of the results, we found that for graphs with more edge percentage, although the *randomized_fpt_attempts* algorithm performed more poorly in terms of time, it actually found the double, or more, of solutions for the same percentage of edges.

V. Conclusion

This report has presented the development and evaluation of a randomized algorithm aimed at solving the vertex cover problem. The main algorithm uses a straightforward approach to try out different possible solutions, making sure to try and find a vertex of size k by randomly choosing an edge and its' vertice. Alongside this main method, the report introduced a secondary strategy that introduced stopping conditions and solution "tracking".

	FPT				FPT Att			
Edge % / It	12.5	25	50	75	12.5	25	50	75
It1	1	6	11	11	1	6	27	33
It2	1	4	11	16	1	6	28	34
It3	1	4	7	12	1	6	31	31
It4	1	4	7	12	1	6	27	34
It5	1	5	16	11	1	6	29	33
It6	1	5	5	16	1	6	26	35
It7	1	3	6	12	1	6	27	30
It8	1	5	8	16	1	6	29	36
It9	1	3	8	12	1	6	26	38
It10	1	5	8	18	1	6	24	34

TABLE I
Solutions found across the different algorithms and the iterations.

Both methods were put to the test with a thorough analysis of how complex they are to run and a series of experiments to see how they perform in action, leading to the "Results" section. The complexity analysis gave us a basic idea of the effort needed by the first algorithm, $O(k \cdot m)$, while the experiments, shown in Figures 1 to 4, gave us a more detailed look. They showed that as we increase the number of vertices, the number of solutions tested grows in a straightforward, linear way, but the number of operations needed can grow much faster, which is more noticeable when we're looking for larger vertex covers or dealing with graphs that have many connections.

The results from the first trials compared to later ones showed that the main algorithm was consistent in how many solutions it tested. However, the second algorithm made it clear that more complex graphs require a lot more work. For small graphs, the algorithm did its job well, but as the graphs got bigger, it struggled because it needed to do a lot more work.

In the end, this study matches what the project set out to do by giving us useful information on how random algorithms act when facing hard problems like the vertex cover. It also highlights the importance of finding a good balance between exploring enough possible solutions and not using up too much computer power, which is something important to consider for future improvements in this area.

References

[1] Wikipedia contributors, "Vertex cover — Wikipedia, the free encyclopedia", 2023, [Online; accessed 11-November-2023].
URL: `https://en.wikipedia.org/w/index.php?title=Vertex_cover&oldid=1174729149`