First Project of Advanced Algorithms - Minimum Vertex Cover

João Fonseca (103154)

Resumo **- Este projeto para a unidade curricular de Algoritmos Avançados aborda o desenvolvimento e análise - formal e experimental - de algoritmos de pesquisa para o problema do Minimum Vertex Cover (MVC) num grafo não direcionado. Focou-se numa abordagem de força bruta e numa abordagem heurística gulosa. Este problema consiste em encontrar o menor subconjunto de vértices de um grafo, de forma a que cada aresta seja incidente a pelo menos um vértice desse subconjunto.**

Observa-se que, embora o algoritmo de força bruta forneça sempre soluções ótimas, isto tem custos computacionais e de tempo significativos à medida que o número de vértices no grafo aumenta. Em contrapartida, a abordagem com heurística gulosa oferece soluções rápidas e quase ótimas, o que a torna uma escolha prática para instâncias de larga escala.

Abstract **- This project for the Advanced Algorithms course delves into the development and analysis – both formal and experimental - of search algorithms for the Minimum Vertex Cover (MVC) problem in an undirected graph. We focused on a brute-force and a greedy heuristic approach. This problem involves finding the smallest subset of vertices in a graph, such that each edge is incident to at least one vertex in that subset.**

We observe that, while the brute force algorithm always provides optimal solutions, it has significant time and computational costs as the number of vertices in the graph increases. In contrast, the greedy heuristic approach offers quick, near-optimal solutions, making it a practical choice for large-scale instances.

I. Introduction

In the context of the 1st project of the Advanced Algorithms course (2023/2024), it was proposed the development and analysis of distinct search algorithms over a graph problem.

The course professor randomly assigned each student a problem from the pool of graph problems. It was decided that I would get the 14th problem, which was about the Minimum Vertex Cover (MVC) of an undirected graph G(V, E), with n vertices and m edges. It consists on finding the smallest set C of vertices, such that each edge of the graph is incident to, at least, one vertex in C.

Two algorithms were developed, one following an exhaustive (or brute-force) approach, the other was based on a greedy heuristic.

II. Development environment

The graph generation and the search algorithms were developed using Python. The library Network X revealed essential in the entire process, since it provided an intuitive interface to generate and interact with graphs. The folder structure of the Python project is the following.

- benchmarks/
- excel/
- graphs/
- images/
- src/
    - benchmark.py
    - display_solution.py
    - graph_generator.py
    - mvc_brute_force.py
    - mvc_greedy.py
    - utils.py
- report.pdf
- requirements.txt

The requirements file contains essential Python libraries to run the scripts. The src folder contains all the Python scripts. The graphs folder contains the generated graphs in JSON format. The benchmarks folder contains results from the algorithms performance in CSV format. The images folder contains the graphic representation of some graph solutions, that were created to demonstrate the differences between the algorithms. The excel folder contains the analysis of the benchmarks and the charts.

Utils.py contains general functions that allowed the code in the other scripts to be more concise. These functions were used to visualize the graphs and to read and write graphs and write benchmark results to files.

Graph_generator.py is responsible for generating all the graphs used in the experiments, taking into account my student's number (103154) as the seed, the density of edges (12.5%, 25%, 50%, 75%) and the number of vertices (min=4, max=255).

Mvc_brute_force.py includes the implementation of a brute force approach, which in summary goes through all of the possible subsets of vertices in non-decreasing order, and stops on the first subset which is a minimum vertex cover of the graph.

Mvc_greedy.py includes greedy implementation of search algorithms. The one that was implemented uses an heuristic which sequentially chooses the node with the most uncovered incident edges at that moment. The node is then added to the MVC, and the search stops when the set of edges is finally covered.

Benchmark.py loads the generated graphs, runs the algorithms against them and registers the time taken, the number of operations and the resulting MVC and its size.

Display_solution.py generates images of algorithm solutions. The normal vertices are drawn using a blue colour, the MVC vertices are drawn using a red colour.

Excel was used to analyse the benchmark data and create the charts, since it provided an easy to use interface.

III. ANALYSIS OF THE ALGORITHMS

A. Formal Analysis (Worst Case)

Brute Force: For each k-vertex subset of a n-vertex graph, where k goes from 1 to n, we go through all the n choose k combinations. Therefore, the number of operations given a n-vertex graph is the following.

$$n_{ops}(n) = \sum_{k=1}^{n} \binom{n}{k} \text{ [i]}$$

Considering the binomial formula, which states that

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k} \text{ [ii]}$$

We can transform [i] using [ii].

$$n_{ops}(n) = \sum_{k=1}^{n} \binom{n}{k} 1^k 1^{n-k}, since\ 1^a = 1, \forall a \in \mathbb{R}$$

$$= \left[\sum_{k=0}^{n} \binom{n}{k} 1^k 1^{n-k}\right] - \binom{n}{0} 1^0 1^{n-0}$$

$$= (1+1)^n - 1 \times 1 \times 1$$

$$= 2^n - 1$$

$$\boldsymbol{O(2^n)}\textbf{ - Exponential}$$

Greedy – Highest Incidence: For each vertex of a n-vertex graph, we check all the m-edges to see which are incident to each vertex. We choose the vertex with the highest incidence of uncovered vertices in that moment, and we consider that vertex for the MVC. We repeat the process to the following n-1 vertices, until all edges have been covered.

In the worst case, where all vertices are part of the MVC, we run this operation n times, and in each instance k, we compare the incidence of n-k vertices against m edges. Therefore, the number of operations given a n-vertex graph is the following.

$$n_{ops}(n) = \sum_{k=0}^{n} (n-k) \times m$$

$$= m \sum_{k=0}^{n} k$$

$$= m \times \left(\frac{n(n+1)}{2}\right)$$

$$= \frac{1}{2} mn(n+1)$$

$$= \frac{1}{2} mn^2 + \frac{1}{2} mn$$

$$\boldsymbol{O(m \times n^2)}\textbf{ - Quadratic}$$

B. Experimental Analysis

The benchmark of the algorithms took into consideration graphs with n-vertices, with n ranging from 4 to 255, and edge densities p of 12.5%, 25%, 50% and 75%.

Brute Force: The time taken and the number of operations had a similar exponential growth (note that the y-scale is logarithmic). The time taken, number of operations and MVC is directly proportional to the edge density of the graph. This goes in line with the expected behaviour, since a random and uniform increase of edges on the graph increases the number of vertices necessary to cover them.

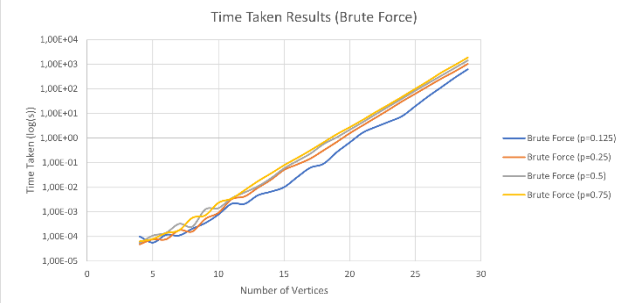

Fig. 1 - Time Taken Results (Brute Force).

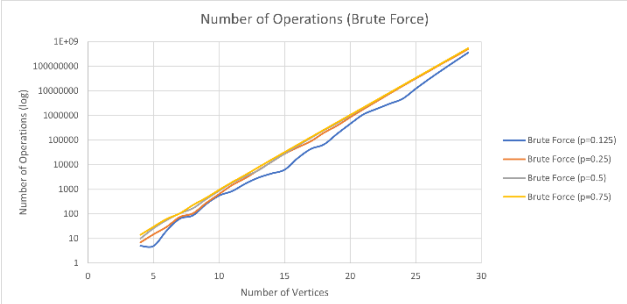

Fig. 2 - Number of Operations (Brute Force).

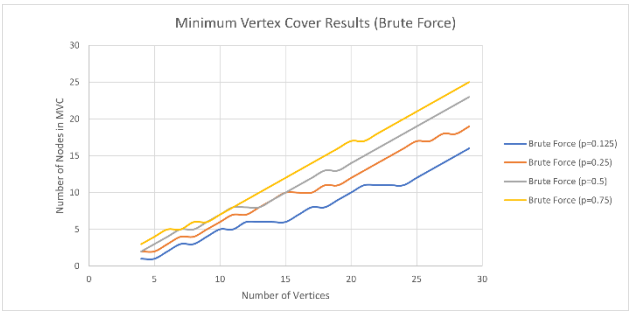

Fig. 3 - MVC Results (Brute Force).

Greedy – Highest Incidence: The time taken and the number of operations had a similar quadratic growth. As in the brute force approach, the time taken, number of operations and MVC is directly proportional to the edge density of the graph.

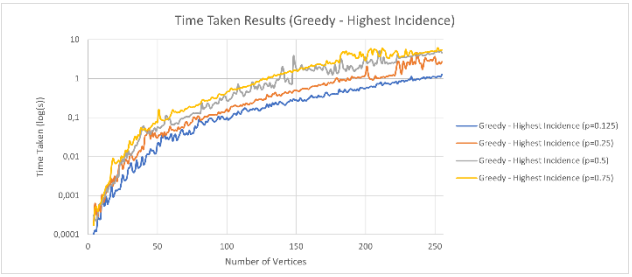

Fig. 4 - Time Taken Results (Greedy – Highest Incidence).

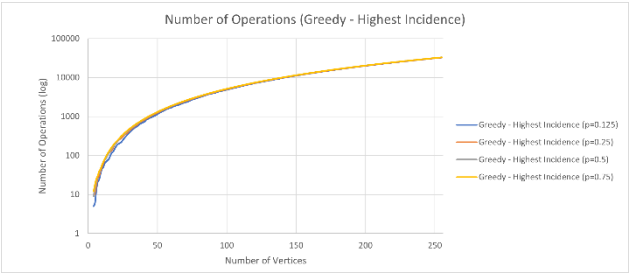

Fig. 5 - Number of Operations (Greedy – Highest Incidence).

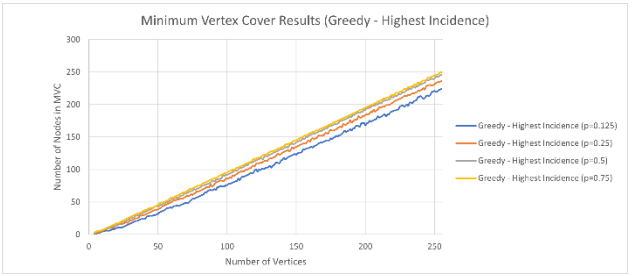

Fig. 6 - MVC Results (Greedy – Highest Incidence).

Brute Force vs Greedy – Highest Incidence: Since we considered the same graphs for both algorithms, we can compare the performance between the two algorithms. Despite the brute force always outputting the best solution, the time taken for that computation grows exponentially.

The algorithm with the greedy heuristic, that considered the vertex with the highest incidence of uncovered edges in each iteration out-performed the brute force algorithm in terms of the time taken and the number of operations. However, this approach doesn't guarantee the best solution every time, only an estimate by excess. Depending on the problem at hand, it could be good enough to compute an approximation if it means we save computing time and power.

For graphs with edge densities of 12.5%, 25%, 50% and 75%, the time taken and number of operations was exponential for the brute force approach, and it was quadratic for the greedy heuristic algorithm. The MVC in the brute force approach is always the best solution, so it is always lower or equal to any solution from the greedy approach.

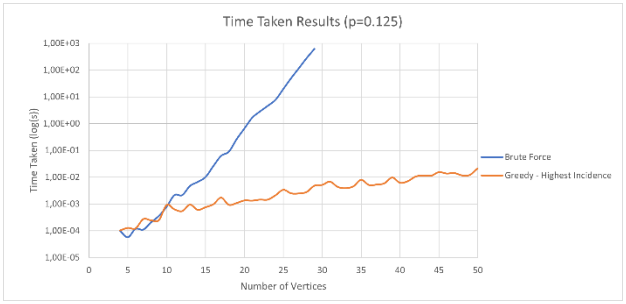

Fig. 7 - Time Taken Results (Edge Density of 12.5%).

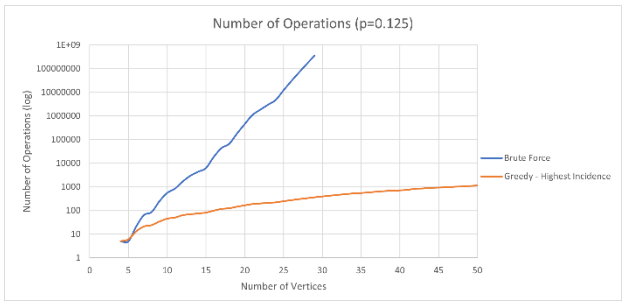

Fig. 8 - Number of Operations (Edge Density of 12.5%).

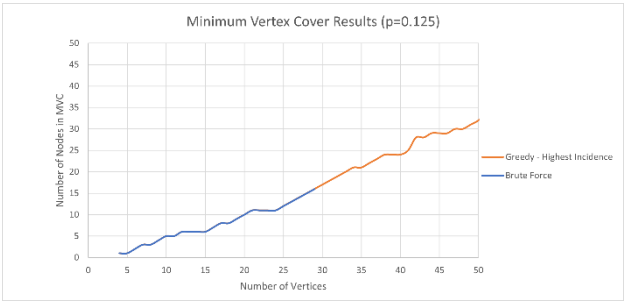

Fig. 9 - MVC Results (Edge Density of 12.5%).

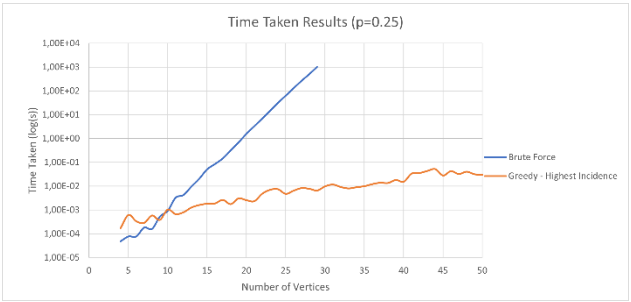

Fig. 10 - Time Taken Results (Edge Density of 25%).

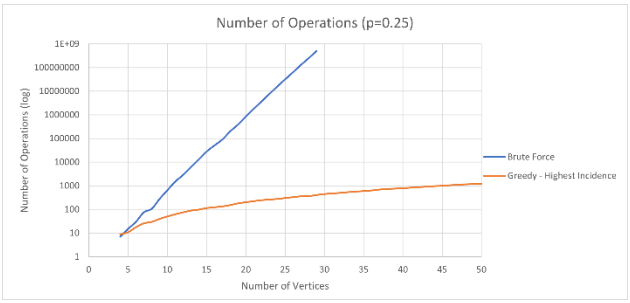

Fig. 11 - Number of Operations (Edge Density of 25%).

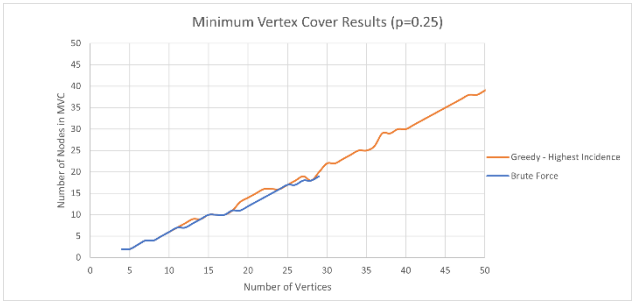

Fig. 11 - MVC Results (Edge Density of 25%).

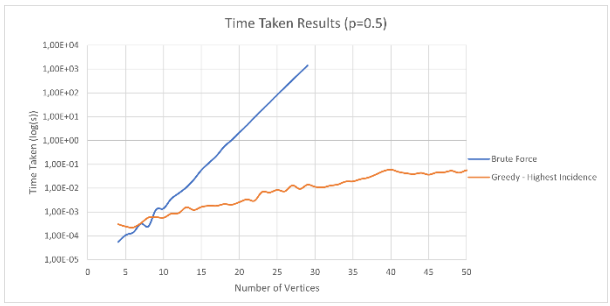

Fig. 12 - Time Taken Results (Edge Density of 50%).

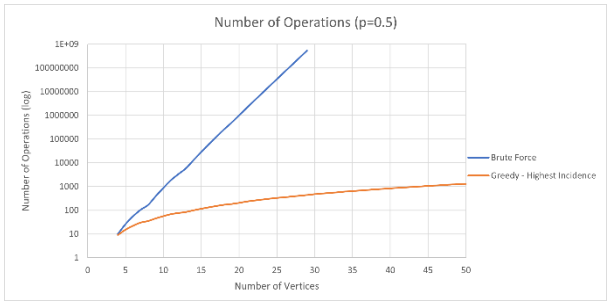

Fig. 13 - Number of Operations (Edge Density of 50%).

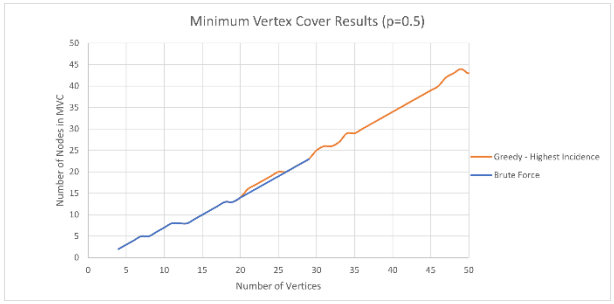

Fig. 14 - MVC Results (Edge Density of 50%).

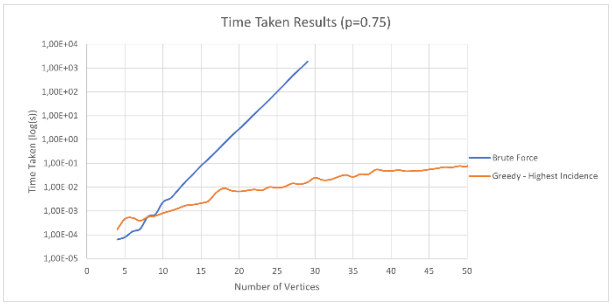

Fig. 15 - Time Taken Results (Edge Density of 75%).

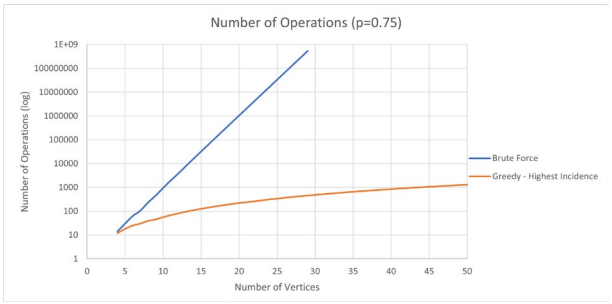

Fig. 16 - Number of Operations (Edge Density of 75%).

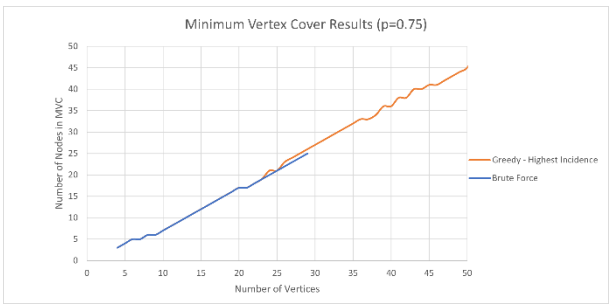

Fig. 17 - MVC Results (Edge Density of 75%).

C. Conclusion

The results from both the formal and experimental analysis are coherent and conclusive. We confirmed the brute force algorithm has an exponential computational growth, and that the greedy heuristic based on the highest uncovered incident vertices approach has a quadratic computational growth.

IV. LARGER PROBLEM INSTANCES

An limitation on the size of the considered graphs wasn't due to the algorithms themselves, but to the time it took to generate and store the graphs for later experiments. Therefore, it was decided that the largest graphs contain 255 vertices.

Considering the brute force algorithm, the largest graph my computer could handle had 29 vertices and 75% edge density. The solution took around 30 minutes to compute. Since the observed growth of the time taken (available in the Excel sheet) between graphs with n and n+1 vertices was around 2, it would mean that for a graph with 30 vertices the expected time taken would be of 1 hour.

Considering now the greedy heuristic algorithm for the same 29 vertex and 75% edge density graph, the time taken was around 15 milliseconds. However, it didn't provide the smallest MVC, that in this case should be of 25 vertices, not 26 vertices. The largest graph in which this algorithm was tested had 255 vertices and an edge density of 75%. The solution took around 5.39 seconds to compute, giving an MVC of 250 vertices, which is probably not the smallest amount, but it was fast! The growth rate of the taken time was not clear analysing the time taken. However, we know that a quadratic has a growth rate of 4 between n and 2n. Considering the time taken for a graph with 255 vertices was of 5.39 seconds, for a graph with 510 vertices it would take 21.56 seconds. That's less time than what it took to compute the solution for 23 vertices using the brute force approach (which took around 23.22 seconds).

V. Visualization of the Minimum Vertex Cover

Let's visualize the differences between the brute force approach, which provides the best solution, and the greedy heuristic approach, which provides a fast computation of a not optimal but close enough solution. We will consider the graph with 12 vertices and an edge density of 25%, since it is small enough to display properly, and it was the graph with the least vertices that produced different MVCs among the algorithms. The brute force approach found the best MVC with 7 vertices, the greedy heuristic approach computed a 8-vertex MVC.

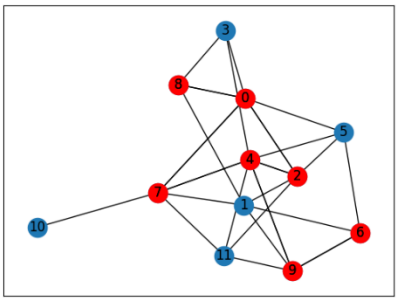

Fig. 18 - MVC (in red) of the Brute Force algorithm for 12 vertices and an edge density of 25%.

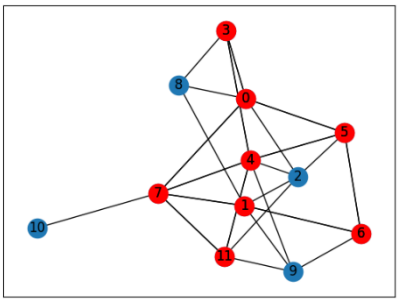

Fig. 19 - MVC (in red) of the Greedy – Highest Incidence algorithm for 12 vertices and an edge density of 25%.

VI. Conclusion

We conclude that a brute force algorithm will always find the best solution to a problem, in this case a graph problem, at the expense of computational power and time. However, an approach which considers a greedy heuristic can make a good enough approximation of the best solution in a much shorter amount of time. Sometimes it is preferable to find a quick and sub-optimal solution instead of the best solution in an absurd amount to time, but it will always depend on the purpose of the search and if we are willing to make that compromise.

References

[1] Wikipedia, "Vertex cover", available at: https://en.wikipedia.org/wiki/vertex_cover, Oct 2023.

[2] Network X, "Documentation", available at: https://networkx.org/documentation/stable/reference/, Oct 2023.

[3] OpenAI, "ChatGPT", available at: https://chat.openai.com/, Oct 2023.