Second 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. Já que se focou numa abordagem de força bruta e numa abordagem heurística gulosa da última vez, agora exploramos abordagem aleatória. 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.**

Observamos que uma abordagem aleatória tem uma complexidade computacional semelhante à da força bruta, reduzida por um fator do número de subconjuntos de k-vértices considerados em cada nível. É capaz de fornecer resultados precisos na maioria das vezes, às vezes até melhores do que a abordagem heurística gulosa. No entanto, dada a sua natureza não determinística, não somos capazes de obter sempre soluções iguais ou ótimas em cada execução.

***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. Since we focused on a brute-force and a greedy heuristic approach last time, now we delve into a randomized 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 a randomized approach has a similar computational complexity as the brute-force, dimmed down by a factor of the number of considered k-vertex subsets at each level. It is able to provide accurate results most of the time, sometimes even better than the greedy heuristic approach. However, given its non-deterministic nature, we aren't able to get the same or optimal solutions each time we execute it.

I. Introduction

In the context of the 2nd project of the Advanced Algorithms course (2023/2024), it was proposed the development and analysis of a randomized search algorithm over the same graph problem from the 1st project.

As previously decided, the considered graph problem was 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.

One randomized algorithm was developed, and its performance was compared to the previous two algorithms, developed in the scope of the 1st project, a brute force and a greedy heuristic approach.

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/
- benchmarks_internet/
- excel/
- graphs/
- graphs_internet/
- src/
    - benchmark.py
    - graph_generator.py
    - mvc_brute_force.py
    - mvc_greedy.py
    - mvc_randomized.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 previously generated graphs in JSON format. The graphs_internet folder contains the graphs provided by the professor. The benchmarks folder contains results from the randomized algorithm's performance on the generated graphs in CSV format. The benchmarks_internet folder contains results from all the algorithms' performance on the graphs provided by the professor in CSV format. 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 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 the implementation of a greedy heuristic algorithm, 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.

Mvc_randomized.py includes the implementation of a randomized algorithm, which goes through a pre-defined percentage of all k-vertex subsets, in a decreasing order of k. If a vertex cover is found, we decrement k and run the algorithm for that size. If the maximum number of tries for a given k is reached, we return the best solution found until that point.

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

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

III. ANALYSIS OF THE ALGORITHM

A. Formal Analysis (Worst Case)

Recall that, in the previous project, we considered two deterministic algorithms: a brute force approach, with an exponential complexity of $\boldsymbol{O(2^n)}$, where n was the number of vertices; and a greedy approach with a highest edge incidence heuristic, which had a quadratic complexity of $\boldsymbol{O(m \times n^2)}$, where n was the number of vertices and m the number of edges.

Randomized Decreasing: For each k-vertex subset of a n-vertex graph, where k goes from n to 1, we go through a percentage of all the n choose k combinations, pk. Once we find a vertex cover, we keep it, decrease k, and run the algorithm on the new conditions. Once we reach the maximum number of allowed solutions for a given k, which depends on pk, we return the best solution we found so far. As an optimization, we randomly choose a k-vertex subset and derive the next subsets from it, by changing one member at a time. Note that in the actual implementation of the algorithm, if $k < 20$, we go through <u>all</u> possible combinations instead of a percentage, since we usually reach these k values when we are faced with small graphs, which have a relatively small number of subsets to explore. Also when we have a very sparse graph, k can reach these values, but it's highly unlikely for the algorithm to reach this stage in those conditions. For simplification, we consider the usage of pk for all k. Therefore, for a given n-vertex graph and a percentage of considered k-vertex subsets pk, the number of operations is the following.

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

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].

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

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

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

$$= pk \times 2^n - pk$$

$\boldsymbol{O(pk \times 2^n)}$ **- Exponential**

We can conclude that this randomized algorithm has a similar complexity as the brute force approach, however it is reduced by the factor pk. When $pk = 1$, we are faced with the brute force algorithm, which goes through all the possible n choose k subsets.

B. Experimental Analysis

The benchmark of the algorithm took into consideration n-vertex graphs, with n ranging from 4 to 29, edge densities p of 12.5%, 25%, 50% and 75%, and percentages of all the n choose k combinations pk of 12.5%, 25%, 50% and 75%.

Randomized Decreasing: The time taken and the number of operations had a similar exponential growth (note that the y-scale is logarithmic). <u>The time taken and number of operations are inversely proportional to the edge density of the graph</u>, unlike the brute force algorithm. This can be explained by the fact that when the edge density increases, so does the MVC size. In the randomized approach we start from k=n and decrease k, whereas in the brute force approach we start from k=1 and increase k. Therefore we encounter larger MVC solutions when the edge density of the graph increases. <u>The MVC size is inversely proportional to pk</u>, since the higher the pk is, the more n choose k combinations are tested, and higher is the chance of encountering a smaller MVC.

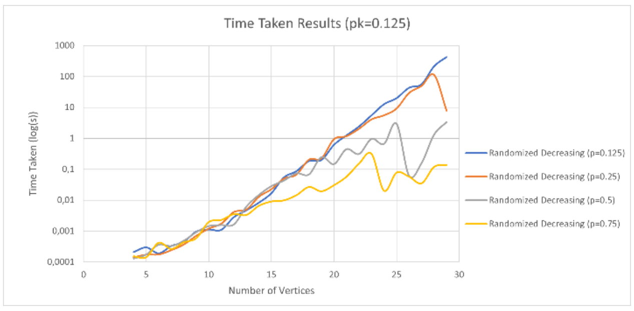

Fig. 1 - Time Taken Results (Randomized Decreasing, pk of 12.5%).

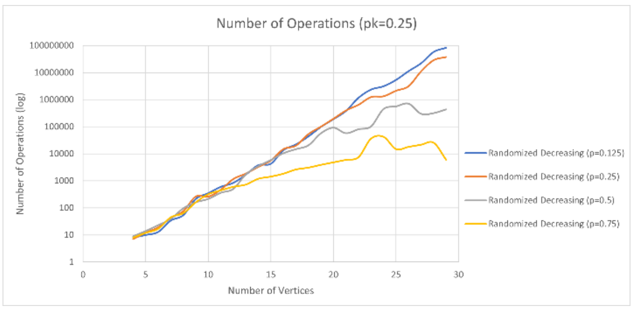

Fig. 6 - Number of Operations (Randomized Decreasing, pk of 25%).

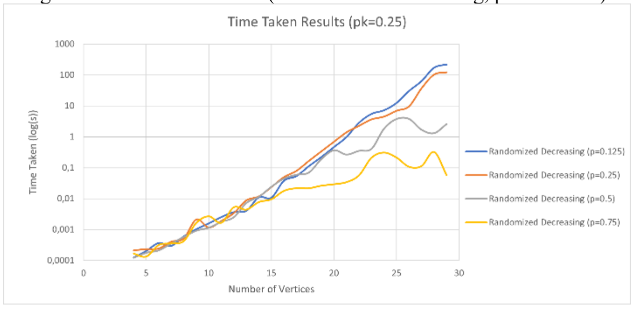

Fig. 2 - Time Taken Results (Randomized Decreasing, pk of 25%).

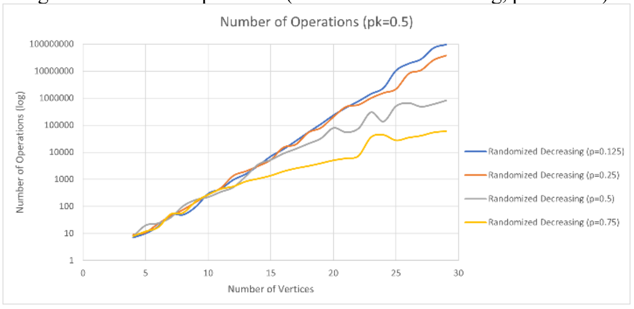

Fig. 7 - Number of Operations (Randomized Decreasing, pk of 50%).

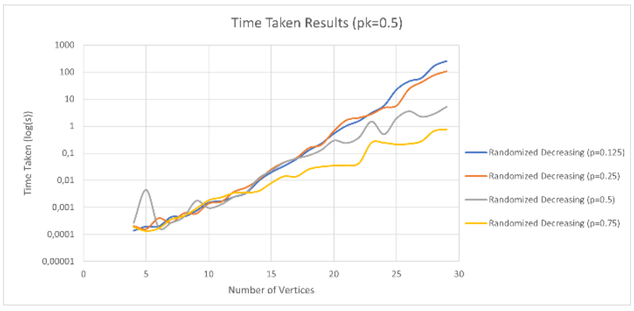

Fig. 3 - Time Taken Results (Randomized Decreasing, pk of 50%).

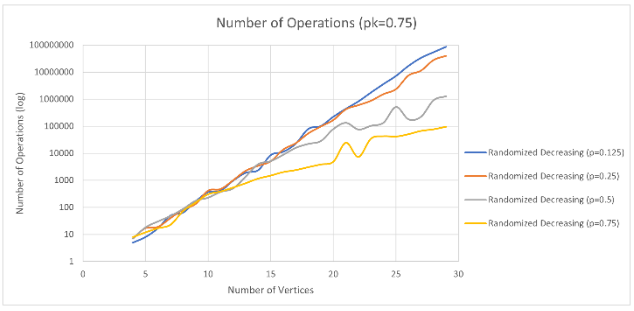

Fig. 8 - Number of Operations (Randomized Decreasing, pk of 75%).

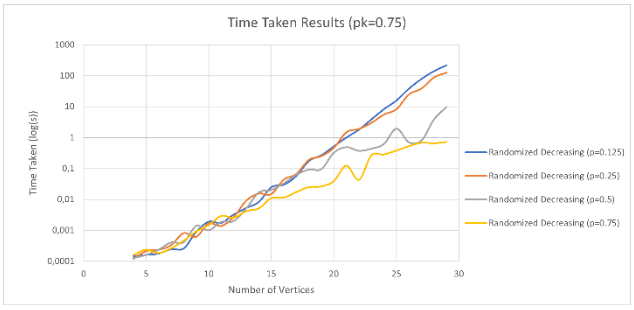

Fig. 4 - Time Taken Results (Randomized Decreasing, pk of 75%).

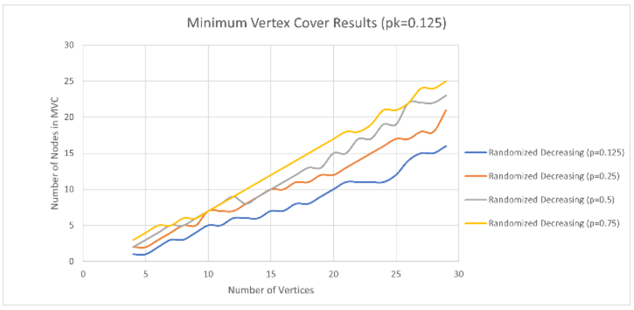

Fig. 9 - MVC Results (Randomized Decreasing, pk of 12.5%).

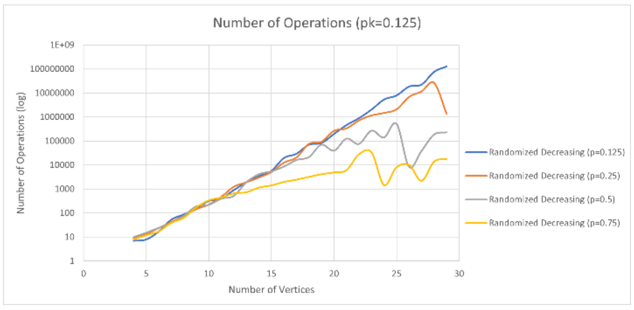

Fig. 5 - Number of Operations (Randomized Decreasing, pk of 12.5%).

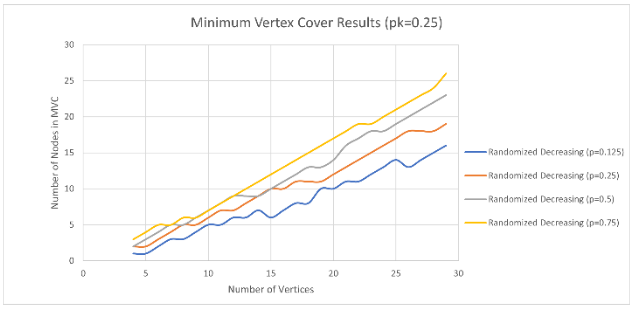

Fig. 10 - MVC Results (Randomized Decreasing, pk of 25%).

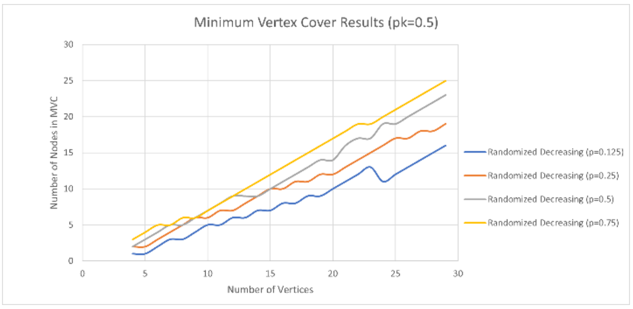

Fig. 11 - MVC Results (Randomized Decreasing, pk of 50%).

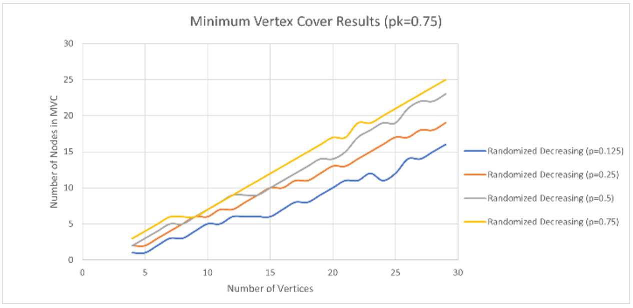

Fig. 12 - MVC Results (Randomized Decreasing, pk of 75%).

Accuracy and relative error: We conducted an analysis on the accuracy and the relative error of the randomized algorithm. The following table presents the average results for the accuracy and the relative error, based on the edge density of the graph (p) and the percentage of considered k-vertex subsets (pk). The complete analysis can be found in the provided Excel spreadsheet. The cells in green and red represent the most and least accurate pk choices for each edge density respectively. Note that these results only take into consideration graphs from 4 to 29 vertices, since those were the graphs on which the brute force algorithm was able to output the solution in a reasonable amount of time. The brute force algorithm is the only developed algorithm capable of providing the optimal solution, so we face this limitation on the results.

Accuracy Ratio / Relative Error		Percentage of considered k-vertex subsets (pk)			
		12.5%	25%	50%	75%
Edge Density (p)	12.5%	1.01212 0.01212	1.02758 0.02758	1.03361 0.03361	1.00646 0.00646
	25%	1.02742 0.02742	1.01572 0.01572	1.02465 0.02465	1.02786 0.02786
	50%	1.01777 0.01777	1.01685 0.01685	1.01968 0.01968	1.02313 0.02313
	75%	1.00586 0.00586	1.00594 0.00594	1.00440 0.00440	1.00983 0.00983

Table 1 – Accuracy and relative error of the randomized algorithm.

In general, to achieve accurate results, the lower the edge density p, the higher should be the percentage of considered k-vertex subsets pk. The choice of pk is more accurate in the range between 25% and 50%. The quality of the results may not be precise, given the small number of tested graphs and the single run of the randomized algorithm.

Comparison with previous algorithms: Since we considered the same graphs in all three algorithms, we can compare the performance between them. The randomized approach performed worse for graphs with an edge density of 12.5%, having a similar computational complexity as the brute force approach. However, an increase in edge density lead to a better performance of the randomized approach, which became closer to the greedy heuristic approach. An increase in the number of vertices in the graph also benefits the randomized approach over the brute force, which is more evident for larger graph instances.

In terms of the MVC, the results of the randomized approach are satisfactory, only differing from the brute force approach (which always outputs the best solution) by at most 3 vertices each time. Sometimes it outputs better solutions than the greedy heuristic approach, but those results are not deterministic, so we are not guaranteed it can happen all the time.

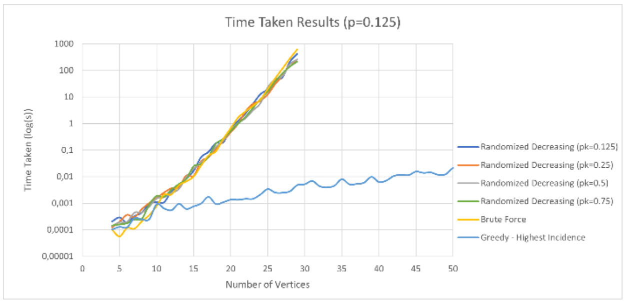

Fig. 13 – Time Taken Results (Edge Density of 12.5%).

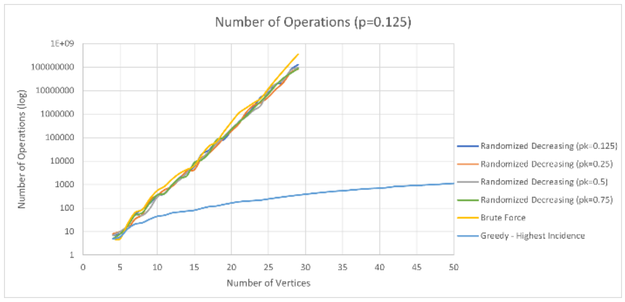

Fig. 14 – Number of Operations (Edge Density of 12.5%).

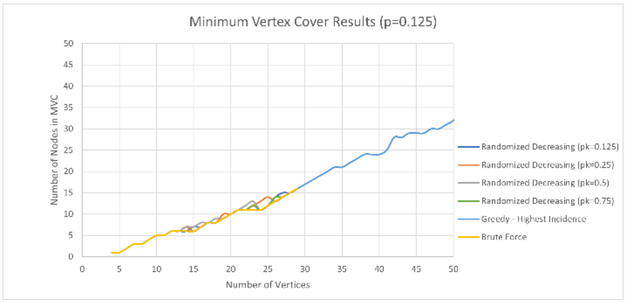

Fig. 15 – MVC Results (Edge Density of 12.5%).

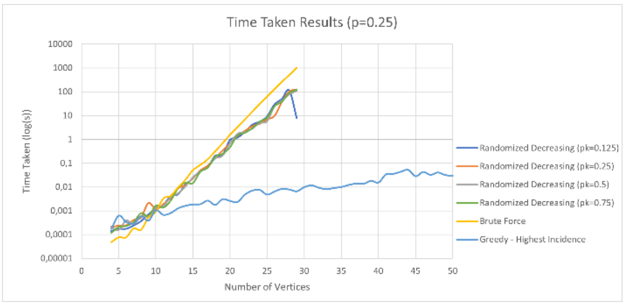

Fig. 16 – Time Taken Results (Edge Density of 25%).

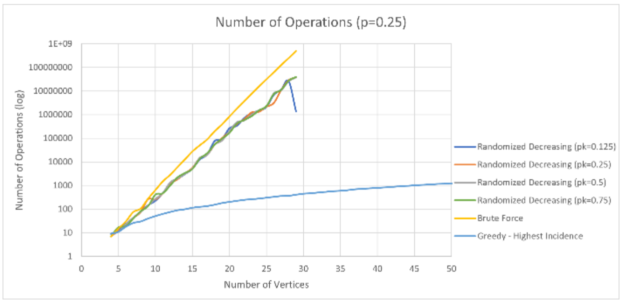

Fig. 17 – Number of Operations (Edge Density of 25%).

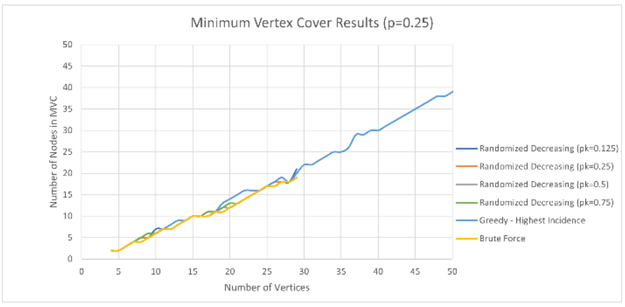

Fig. 18 – MVC Results (Edge Density of 25%).

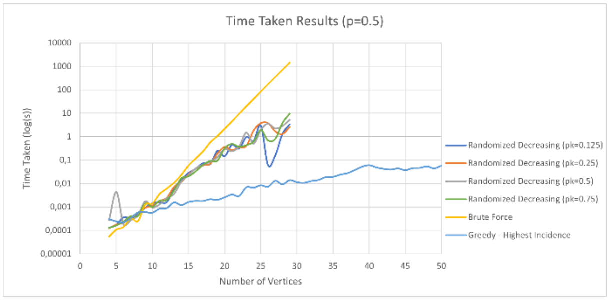

Fig. 19 – Time Taken Results (Edge Density of 50%).

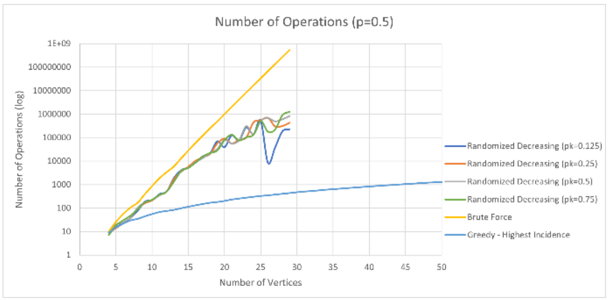

Fig. 20 – Number of Operations (Edge Density of 50%).

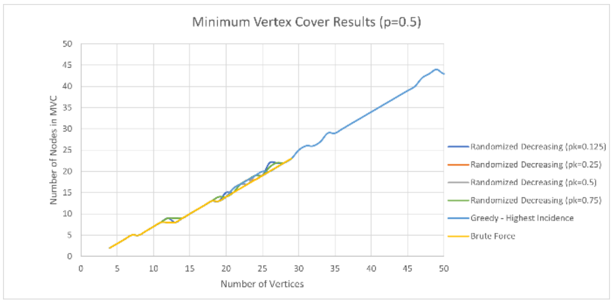

Fig. 21 – MVC Results (Edge Density of 50%).

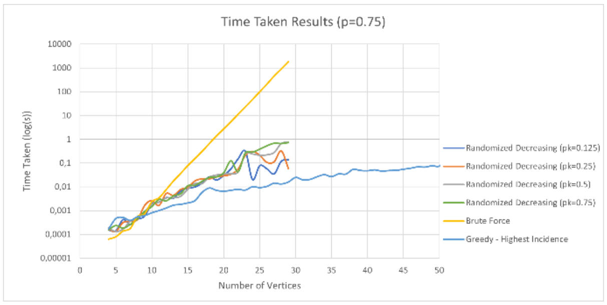

Fig. 22 – Time Taken Results (Edge Density of 75%).

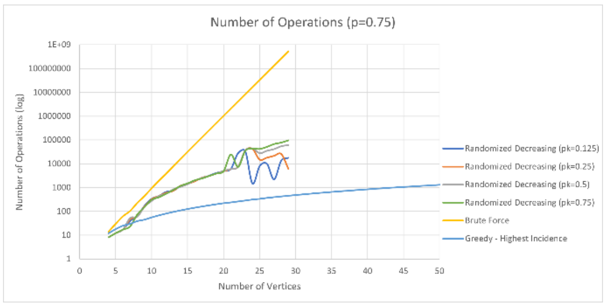

Fig. 23 – Number of Operations (Edge Density of 75%).

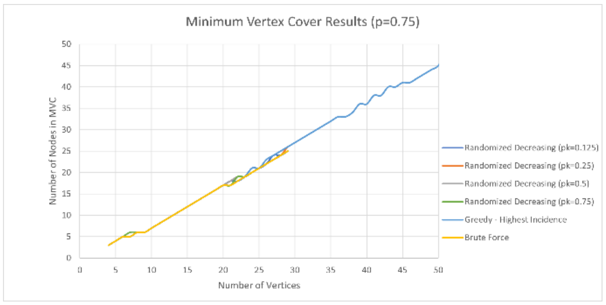

Fig. 24 – MVC Results (Edge Density of 75%).

Running the algorithms on instances provided by the professor: After inspecting the provided graph instances, there were only 3 that applied to this problem: the non-oriented graphs SWtinyG, SWmediumG and SWlargeG. However, SWmediumG was too large to run in a reasonable amount of time, and SWlarge was even worse. The only graph in which all algorithms were able to run was SWtinyG, which had 13 vertices and 13 edges, The results of those runs, for each algorithm, are presented in the following table.

Algorithm	MVC	N. Ops.	Time Taken (s)
B. F.	6	2454	0.00340
Greedy	6	69	0.00055
Rand. (12.5%)	8	1769	0.00668
Rand. (25%)	7	3458	0.01261
Rand. (50%)	7	2309	0.00745
Rand. (75%)	7	2938	0.00769

Table 2 – Results of the algorithms on the SWtinyG graph.

C. Conclusion

The results from both the formal and experimental analysis are coherent and conclusive. We confirmed the randomized algorithm has an exponential computational growth, like the brute force approach, but with a reduced factor. It outperforms the brute force algorithm for higher edge density graphs, by providing a significant speed improvement without much loss in accuracy. The chosen percentage of considered k-vertex subsets didn't have much impact in the resulting MVC size, but in terms of time taken and number of operations, a lower percentage lead to faster results.

IV. Larger problem instances

The largest graph on which the randomized algorithm was tested had 29 vertices, the same amount the largest graph my computer could handle on the brute-force algorithm. The graphs with the lowest edge density (12.5%) dictated this limit, since they took a similar time as the brute force approach. The randomized algorithm could certainly run well for larger graph instances, but it wouldn't be possible to analyse the accuracy of those results, since we didn't have the optimal solution for those.

The observed growth of the time taken (available in the Excel sheet) between graphs with n and n+1 vertices was around 1.88, 1.85, 1.99 and 1.58, for edge densities 12.5%, 25%, 50% and 75% respectively. These growth values are lower than the expected growth of the brute force algorithm, which was of around 2. This means that every time n is incremented, we should expect the time taken to be almost the double.

VI. Conclusion

We conclude that a randomized algorithm will be able to outperform a brute force approach in terms of time taken, but never in the size of the MVC, which will be always equal or larger. However, these two approaches share similar computational complexity, being the randomized dimmed down by a factor dependent of the percentage of considered k-vertex subsets at each level. This approach can provide better results than the greedy heuristic algorithm, but this isn't guaranteed to happen in a deterministic way. In the end, the randomized approach sits in between of the brute force and the greedy heuristic approach, in terms of time taken and number of operations.

References

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

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

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