Decision problem: Edge dominating set of a graph with k edges

Bruna de Melo Simões
103453 - brunams21@ua.pt

Abstract – **This paper presents an overview of the edge dominating set problem of a graph, that is, the problem finding an edge dominating set with size k of a graph. An edge dominating set of a graph G consists of a set of edges D, such that every edge not in D is adjacent to at least one edge in D. This problem is addressed using a randomized approach, which generates a random solution and then verifies if that solution is the correct one or not. It is then compared to an exhaustive approach to check the result accuracy, and time comparison between graphs with different nodes and edges. Therefore, it is inferred whether the generated randomized algorithm is in fact less time consuming than other specific approaches.**

Resumo – **Este artigo apresenta uma visão geral do problema do conjunto dominante de arestas de um grafo, ou seja, o problema de encontrar um conjunto dominante de arestas com tamanho k de um grafo. Um conjunto dominante de arestas de um grafo G consiste num conjunto de arestas D, tal que cada aresta que não esteja em D seja adjacente a pelo menos uma aresta em D. Este problema é=foi abordado através de uma abordagem aleatória, que gera uma solução aleatória e depois verifica se essa solução é a correta ou não. De seguida, esta metodologia é comparada a uma abordagem exaustiva para verificar a precisão do resultado.**

Keywords – **k-edge dominating set, exhaustive approach, heuristic approach, randomized algorithms**

Palavras chave – **conjunto dominante de k arestas, algoritmos de força bruta, abordagem exaustiva, abordagem heurística, algoritmos vorazes, algoritmos aleatórios**

I. Introduction

Given a graph G, and a subset D, an edge dominating graph of G is one such that every edge not in D is adjacent to at least, one edge in D. In other words, every edge in the graph is either part of the edge dominating set or adjacent to an edge in it.

Using a more formal mathematical approach, we can define the problem as such: let $G = (V, E)$ be an undirected graph. A set $D \subseteq E$ is an edge dominating set if, for every edge e, either $e \in D$ or there exists an edge $f \in D$ such that e and f share a common vertex. [1]

Finding whether a graph has a dominating set with a certain size is an NP-hard problem, that is, a decision problem, which can be reduced to polynomial time. [2]

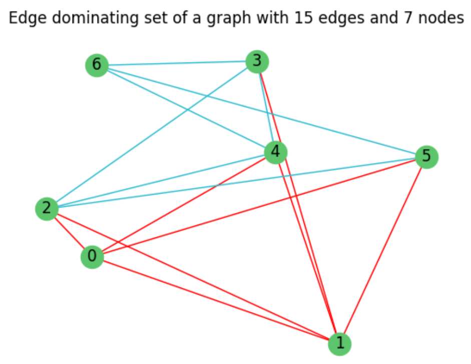

Fig. 1 - Edge dominating set of a graph with 7 nodes and 15 edges, where the edges in the dominating set, with $k = 8$, are shown with the red color

In this paper, we use a randomized approach to solve the k-edge dominating set, in which it tries to select a random subset of k-sized edges, and then check if the selected subset is an edge-dominating set of the generated graph. We then compare this approach to an exhaustive and greedy approach, in order to verify the computed solutions, and compare the computational time and operations needed [3].

II. Randomized Search

A. Algorithm Analysis

The first approach to this problem is trying to generate a subset of edges with size k and verify whether it corresponds to an edge dominating set or not. It is then defined a certain criteria in which the algorithm will keep searching for solutions or stop. Upon testing with time criteria and number of iterations criteria, it was defined that the best solution would be the number of iterations criteria, since the time taken into verifying one given candidate was proportional to its k-edge size, and therefore, bigger problems would take a lot longer to process than smaller ones, and therefore, larger problems would not have a similar amount of tries to find a correct solution, comparing to smaller graphs, which are quicker to check.

To better analyse the results and test their accuracy, the following algorithm was implemented in Python language:

```
def random_algorithm(graph, k,
    num_configurations):
```

```
    solution = None
    tested_solutions = set()

    for _ in range(num_configurations):

        candidate = random.sample(list(graph.
            edges()), k)
        if str(candidate) in tested_solutions:
            continue

        tested_solutions.add(str(candidate))

        if is_valid_solution(graph, candidate)
            :
            return candidate

    return solution


def is_valid_solution(graph, candidate):
    all_edges = set(graph.edges())
    covered_vertices = set()

    for edge in candidate:
        covered_vertices.update(edge)

    for edge in all_edges:
        if not set(edge).intersection(
            covered_vertices):
            return False

    return True
```

This algorithm starts by admitting a number of possible tries, and defaulting the tested solutions as an empty set. On each try, a candidate, that is, a completely random subset of the graph with size k, is generated. If the candidate, and possible solution, has already been generated before, then the cycle continues, while reducing the number of tries. If not, then it is added to the list of tested solutions, in order to check in the next possible iteration if this candidate has been generated before.

Then, it is verified whether the candidate solution is in fact a solution to the k-edge dominating set problem or not, by defining the covered edges as the edges of the candidate, and then checking if the candidate is a set of edges where every edge not in the candidate, but in the graph, is connected to at least one edge in the candidate subset. If this criteria is met, then the candidate is an edge dominating set and therefore, a solution is found and returned, not needing to further inspect other possible solutions.

B. Time Complexity Analysis

The algorithm starts iterating through all predefined number of configurations, and trying to select one candidate with size k from a sample, in this case, the edges of the graph. Then, it checks if the candidate solution was already tested before, and if it was, then the algorithm is started over again, with most likely another different sample. After certifying that the candidate solution has not been tested before, it is added to the number of tested solutions, and then, it is checked whether or not the tested solution is in fact an edge dominating set of the graph or not. While testing this hypothesis, the algorithm iterates trough all edges in the candidate solution to add them to the covered edges set and then, iterates through all graph edges to check if there are any graph edges which have an intersection with covered vertices, and therefore, check if the candidate solution is an edge-dominating set or not.

Analysing these operations, the time complexity associated with the whole algorithm can be deduced.

The in the first loop, the algorithm iterates through the number of configurations. Admitting the number of configurations as n, the time complexity associated with this loop would be $O(n)$.

Then, a random subset sample is generated, that is, the randomized possible solution to be tested, it's checked whether it has been tested before and, then, check if it corresponds to a valid edge dominating set or not.

To do so, the algorithm iterates through every candidate edge, and then, iterates through all edges. Admitting the size of the candidate solution as c and the number of all graph edges as e, it is possible to conclude that these operations alone have a time complexity of $O(\max(c, e))$. However, this can be simplified due to the fact that it is impossible to have an edge dominating set with greater edge size than the number of the edges of the original graph. Therefore, the operation $\max(c, e)$ will always return e. Therefore, this time complexity can be reduced to $O(e)$.

Therefore, since the verification of the solution as an edge dominating set is performed within the loop which iterates through the number of configurations, the overall time complexity of the algorithm can be described as:

$$O(n \times e)$$

In which n is the number of configurations and e is the amount of graph edges.

It is possible to conclude, therefore, that the execution time depends mostly on the number of configurations and number of graph edges. Therefore, for larger solutions, it is expected that the time taken to find a solution with this randomized algorithm increases exponentially with the number of configurations and graph edge size.

III. RESULT ANALYSIS

A. Randomized Algorithm Results

Several tests were performed to test the efficiency of the randomized approach. In this section, a detailed analysis of the obtained results will be performed.

The randomized approach is, as mentioned above, a solution that tries to find a random solution with size k, and then checks if the candidate solution generated is an edge dominating one or not.

By analyzing Fig. 2, it is possible to infer about the time complexity of the given algorithm. Either for larger or smaller graph sizes, the linearity of the randomized algorithm is visible, and therefore, it is possible to verify the veracity of the time complexity inferred above, which, in this case, is $O(n \times e)$, n being

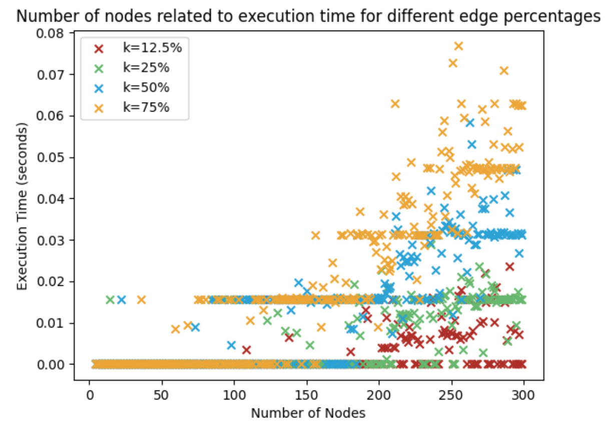

Fig. 2 - Number of nodes related to the amount of time needed to find an edge-dominating set of size k of a graph, using the randomized algorithm. The label, which denotes different colors, corresponds to different values of density of edges. For example, for larger k values, the number of edges for a certain node is also larger.

the number of configurations (number of tries) and e being the number of graph edges.

In this case, larger k values translate in larger edge density, that is, for a graph with n nodes, a larger k value means that, for that amount of nodes, the amount of edges in the graph is larger.

Therefore, it is expected that the time to compute a solution is linearly greater with greater k values, since in response the number of edges is larger, and therefore, the number of edges needed to check to find an edge dominating set is also greater.

Since the number of nodes is also directly proportional to the number of edges, since the maximum number of edges corresponds to $\frac{n\times(n-1)}{2}$, n being the number of nodes, it is also expected that the time needed to find a randomized solution is directly proportional to the number of nodes, and therefore, with greater number of nodes, the greater the execution time of the algorithm.

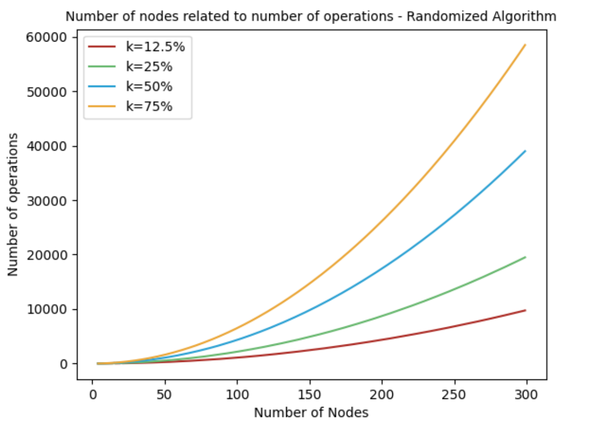

Fig. 3 - Number of nodes related to the number of operations based on edge density, given by the k percentage in graph labels.

By visualizing Fig. 3's graphic, we can denote a similarity between the relation of time and the number of basic operations.

As expected, for large k values, the number of basic operations is also larger, since the number of operations to check each edge of the candidate solution is directly proportional to the overall number of graph edges.

Therefore, by analysis Fig. 2's scatter plot and Fig. 3's line plot, it is possible to denote a similarity between the two, since it is to be expected that the number of operations has a somewhat linear relation to the execution time of the algorithm.

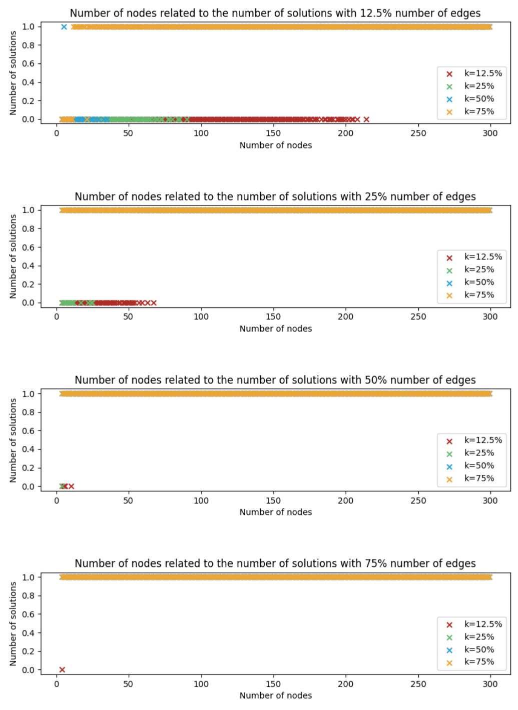

Fig. 4 - Number of nodes related to the number of solutions found, for different edge density, with solutions having size based on the k percentage in graph labels. For example, greater k percentage means the solution found is an edge dominating set with a greater amount of edges.

It is possible to infer through Fig. 4 whether the algorithm has found a solution or not. In the first subgraph, the amount of solutions found for graphs with a greater amount of edges is lesser. This might be explained through the fact that, when the percentages of edges and k values are low enough, the edge dominating set being searched has $k = 0$, and therefore, does not find a dominating set, because it cannot find one with size zero, counting the number of solutions found as none, even though it isn't possible to find one and erroneously leads to the false conclusion that the algorithm isn't functional for small graph sizes.

However, for the vast majority of tested solutions it is possible to infer through the remaining subplots that the algorithm was mostly successful in finding a solution to the problem.

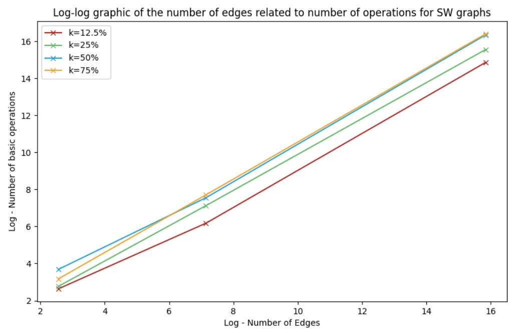

Fig. 5 - Number of edges compared to the number of basic operations for three different-sized graphs with increasing amount of edges and nodes, the first having 13 edges and the last having 7586063.

To further view the evolution of the presented algorithm for even larger graphs and predict the computational effort needed to find a k-edge dominating set, experiments with three increasingly sized graphs were performed.

The three graph characteristics are as follows:

- Smaller graph: 13 nodes and 13 edges.
- Medium sized graph: 250 nodes and 1273 edges.
- Larger graph: 1000000 nodes and 7586063 edges.

In Fig. 5, it is possible to visualize, through a log-log scatter plot, the relation between the number of operations and the number of edges. For each graph, the algorithm tried to find four edge dominating sets, each with a size equivalent to certain percentage of the number of edges, as described by the figure's label, and as was done in the previous experiments with smaller sized graphs.

B. Comparison with other approaches

In order to better test this specific algorithm efficiency, the results obtained of the time execution, solutions found and number of basic operations were compared to the ones obtain in an exhaustive search and an heuristic approach, with the same graphs used to test the randomized approach.

In Fig. 6 it is possible to denote the correct and incorrect solutions found. This graph is generated while comparing the results from an exhaustive algorithm with the randomized one.

Because the randomized algorithm always checks if the generated subset is an edge dominating one, the solutions are always edge dominating sets, and therefore, the notation of correct and incorrect does not define if the presented randomized solution is in fact an edge dominating set or not, but defines whether the algorithm was able or not to find a solution, when there is in fact one.

Therefore, it is possible to conclude that, for the experiment conducted, with a maximum of ten tries to generate a feasible solution, the algorithm was mostly able to find a k-edge dominating set within these

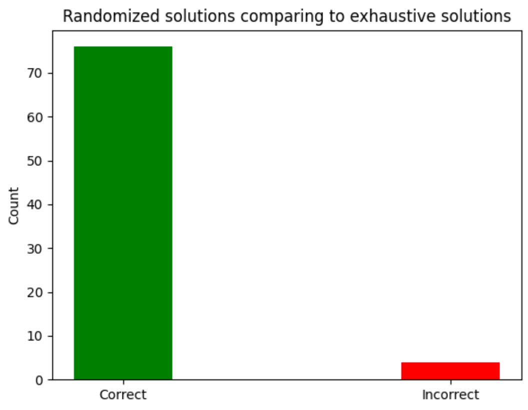

Fig. 6 - Number of correct solutions compared to an exhaustive approach

amount of tries. It is expected that with greater amount of times, the number of "incorrect" solutions would decrease, and potentially be reduced to zero.

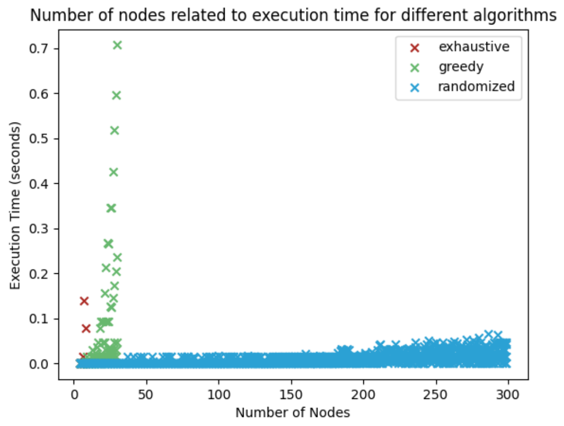

Fig. 7 - Execution time compared to the number of graph nodes, for different algorithms - exhaustive, greedy and randomized

In Fig. 7, the differences in execution time comparing to the number of nodes are more visible. The execution time of the exhaustive algorithm, which tries to find all solutions for a given graph, is clearly the most time consuming one, since the amount of operations needed to find all solutions is greater than any of the other one, even though there aren't many samples to visualize the data from, due to the fact that the time needed to test larger graphs is too great, and can only handle graphs with about eight nodes.

The greedy algorithm, which implements an heuristic, is less time demanding, but still has an exponential time complexity, and starts to be very demanding in larger (but still overall small) graphs.

The randomized algorithm, however, for a larger number of nodes, is almost always able to find a solution, and in less time than it would take for a exhaustive or even greedy approach.

In Fig. 8, we can observe the relation between the number of operations and number of nodes. As ex-

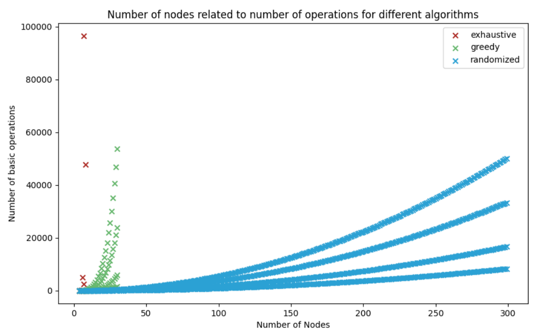

Fig. 8 - Number of basic operations related to the number of nodes for different algorithms - exhaustive, greedy and randomized

pected, the number of operations necessary to compute relatively small graphs is greater than any of the other two algorithms.

Even though the greedy approach consists of an improvement from the exhaustive approach, it is still very inefficient for larger graphs.

Therefore, the randomized approach could consist in a better solution to find k-sized edge dominating sets in larger graphs, since the computational effort in this instance is comparatively smaller than the two previous approaches.

We can also denote a similarity between the execution time and the number of operations in these graphs, which is not surprising due to the fact that both are intrinsically connected, as mentioned above.

IV. Results discussion

The conducted research was implemented through a series of trials, with the purpose of implementing an algorithm capable of returning an edge dominating set which would be able to be as efficient as possible, and therefore, be able to process even larger graphs and greater edge dominating sets, comparing to the previous heuristic and brute-force approaches.

As mentioned before, the number of incorrect solutions might be due to the fact that the stopping criteria, which in this case, was stopping after reaching a number of tries, was insufficient in those cases, and therefore, the number of solutions not found might decrease with a greater allowed number of tries to find an edge dominating set.

Therefore, the randomized approach should be tested differently according to the defined necessities, that is, for example, trying to increase the number of maximum iterations to make sure that one solution is almost always found, or even implementing another methodology to stop the algorithm from trying to search for a solution.

Additionally, in order to better test the time complexity, further testing is needed, with even larger graphs. However, even the most efficient algorithms will take a significant amount of time whenever the problem size increases, with this randomized approach not being an exception. Even though it is a mostly reliable method to search for k edge dominating sets in larger graphs, increasing the number of nodes and number of edges will result in a larger processing time needed to find a solution, even though it is more reliable than the previous exhaustive and greedy implementations.

V. Conclusion

This project allowed for greater graph comprehension, as well as learning how implementing randomized algorithms is useful in certain conditions, but might not be in others, since the requirements change based on certain criteria, which can make even the time-inefficient brute-force approach the best solution for a given problem.

Therefore, since the use of an algorithm is problem-specific, depending on whether the result should rely on having all possible solutions, or trying to find one in the least amount of time possible.

By analysing the presented results of the conducted experiments, we can conclude that the randomized approach has the most positive results in terms of execution time and number of basic operations needed to find a solution. However, as mentioned above, for larger graphs, it might be necessary to increase the maximum number of iterations, which in turn will make the algorithm slower when it does not immediately find a solution and when the graph a great amount of nodes it needs to check, to verify whether the randomized candidate solution generated is an edge dominating set or not.

Additionally, it is possible to infer the reliability of the presented solutions, since the algorithm thoroughly checks the presence of an edge dominating set upon generating the candidate, which is not accepted if this condition is not met.

This paper also demonstrates how heavily graph characteristics affect the results, their operations and the time it takes to find edge dominating sets. This is even more evident in graphs where some characteristics are the same. For example, in this paper, results in graphs with the same number of nodes are variant depending on edge density within the nodes.

References

[1] Diep N. Nguyen The Trung Tran M. Nguyen, Minh Hoàng Hà, “Solving the k-dominating set problem on very large-scale networks”, *Computational Social Networks*, 2020.

[2] M. Yannakakis and F. Gavril, “Edge dominating sets in graphs”, *SIAM Journal on Applied Mathematics*, vol. 38, no. 3, pp. 364–372, 1980.
URL: `http://www.jstor.org/stable/2100648`

[3] GeeksForGeeks, “Randomized algorithms”, 2023.
URL: `https://www.geeksforgeeks.org/randomized-algorithms/`