Minimum Edge Dominating Set Problem

Vicente Costa 98515

Abstract **–This report presents algorithmic solutions for the Minimum Edge Dominating Set Problem, including a formal computational complexity analysis of each algorithm's efficiency, an analysis of the acquire results and predictions for large larger problem instances. This subject is addressed by the course "Advanced Algorithms" at the Aveiro's University. The suggested algorithms embrace one exhaustive search algorithm and one method using a greedy heuristic. The chosen coding language was Python(3.10).**

Resumo **–Este relatório apresenta soluções algorítmicas para o problema Minumum Edge Dominating Set, incluindo uma análise formal da complexidade computacional da eficiência de cada algoritmo, uma análise dos resultados obtidos e previsões para problemas de uma escala maior. Este tema é proposto pelo curso "Algoritmos Avançados"na Universidade da Aveiro. Os algorítmos sugeridos incluem um algoritmo de pesquisa exaustiva e um método com uma heuristica gulosa. A linguagem de programação escolhida foi Python(3.10).**

Keywords **–Graph, Edges, Edge Dominating Set, Vertices, Exhaustive Search, Greedy, Heuristics**

Palavras chave **–Grafo, Arestas, Edge Dominating Set, Vértices, pesquisa exaustiva, Algoritmos Gulosos, Heurísticas**

I. Introduction

This report discuss the first project of the course "Advanced Algorithms". The focus of this project is to design and test an exhaustive algorithm, in addition to another method using a greedy heuristic, to solve the Minimum Edge Dominating Set Problem. The subsequent sections illustrate the problem in detail, explain the algorithms behaviour, and analyze the computational complexity of each one. The subsequent sections illustrates the problem in detail, explain the algorithms behaviour and analyze the computational complexity of each one. For this analysis, I performed a formal and experimental computational complexity analysis of the algorithms. The later includes a series of experiments for progressively larger problem instances to register and study the number of basic operations carried out, the number of solutions tested, and the execution time. Finally, I compare the results of both the formal and experimental analyses. With the report, I submitted the code files and the results files. To run the code, first create the graphs by running the following command:

$ python CreateGraphs.py

Then, to run the algorithms, run the following command. The next lines represent the commands to run the algorithm with an exhaustive approach and the method with a greedy heuristic, respectively:

$ python ExhaustSerch.py
$ python GreedySerch.py

Finally, the results of the execution of the algorithms will appear in the folder "results" in a ".txt" file with the name of the algorithm in a table format. If a csv format is more appropriate, it is also generated in the same folder a file with a "csv" extension.

II. Minimum Edge Dominating Set Problem

This paper handles the Minimum Edge Dominating Set Problem, which it can be solved by finding the minimum edge dominating set of a given undirected graph G(V, E), with n vertices and m edges. In graph theory, the edge dominating set is a subset (let's call it D) such that every edge of the graph not in D is adjacent to at least one edge in D. The minimum edge dominating set is the smallest set of edges of a graph which fulfills this criteria. In the figure I, is illustrated examples of this concept.

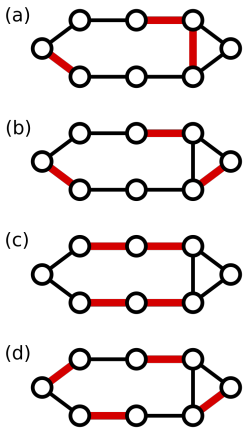

Fig. 1
Examples of edge dominating sets

In the context of the problem, it's important to characterize the graphs and their structure. The graph instances used in the computational experiments represent the following scenario:

- Graph vertices are 2D points on the XoY plane, with integer-valued coordinates between 1 and 100.
- Graph vertices should neither be coincident nor too close.
- The number of edges sharing a vertex is randomly determined.
- Use 12.5%, 25%, 50%, and 75% of the maximum number of edges for the number of vertices.
- Graph edges are unweighted and undirected.

The first step to constructing a graph is generating the list of vertices and edges. The list of vertices is a list of randomly generated coordinates between 1 and 100 with a radius of 7. This radius is important to fulfill the second requirement of the graphs creation ("Graph vertices should neither be coincident nor too close."). Since the graph is undirected in this context, the list of edges is a list of the tuples of the vertices involved (their coordinates). Consequently, (v1, v2) is the equivalent of (v2, v1). Following are the examples of generated vertices and edges:

- Vertices: [(4, 27), (91, 30), (86, 49), (34, 34), (34, 13)]
- Edges: [((34, 34), (86, 49)), ((4, 27), (91, 30)), ((4, 27), (86, 49)), ((4, 27), (34, 13)), ((91, 30), (34, 13))]

To enable the storage of graphs (in JSON files), I chose to identify the vertices by numeric id's and convert the graphs data in a node-link format suitable for JSON serialization using the built-in function of package the networkx, node_link_data. Then, to load the stored graphs, I used the function node_link_graph of the networkx package. The generated graphs are stored in the folder "graphs". An example, with the previous vertices and edges, would be:

- 'directed': False, 'multigraph': False, 'graph': , 'nodes': ['id': 0, 'id': 1, 'id': 2, 'id': 3, 'id': 4], 'links': ['source': 0, 'target': 1, 'source': 0, 'target': 2, 'source': 0, 'target': 4, 'source': 1, 'target': 4, 'source': 2, 'target': 3]

III. Analysis of Algorithm Efficiency

A. Evaluate the Efficiency

The evaluation of the efficiency of the algorithms was achieved by two different approaches: execution time (in seconds) and number of basic operations. The first approach is flawed, despite time being a unit of measurement more logical, it comes with disadvantages. The execution time of an algorithm can be affect by the hardware and the environment, the compiler and editor used, the programming language, and others. On the other hand, we can objectively measure the running time by identifying a basic operation and computing the number of times this operation is executed on inputs of size n. The operations that contributed to this number were: every comparison and every list concatenation/difference.

B. Best-Case, Worst-Case and Average-Case Efficiencies

The efficiency of the algorithms depends on the input given to that algorithm. It may vary not only due to size of the input but the structure of the input. Thus, it's important to evaluate the best-case, worst-case and average-case efficiencies when analyzing an algorithm. The best-case efficiency of an algorithm is the required amount of time it requires to consider all input values. The worst-case efficiency is the maximum amount of time an algorithm requires to consider all input values. The average-case efficiency evaluates the efficiency for a random input and it lies between the worst-case and best-case efficiency. This cases may vary depending on the algorithm.

IV. Exhaustive Search Algorithm

The Exhaustive search algorithm was the first algorithm developed and it is the most accurate. This occurs because this algorithm calculates all possibles solutions and return best one (smallest set that satisfies the dominant set condition). Since the returned solution need to satisfy two conditions I consider two basic condition: possible solution is an edge dominant set and is the smallest set that satisfies this condition. Also, since the algorithm always calculates all possible solutions there is no difference between the computational complexity the best, worst and average case scenario. In this way, to calculate all possible solutions the algorithm calculates all combination of edges, so, since the number of combinations grows with the expression 2**n, it is reasonable to classify this algorithm complexity as 2**n and assume that the rising of problem instances will result in a quickly rise the number of iterations.

V. Greedy Search Algorithm

The Greedy search algorithm was the second algorithm developed. To solve this report problem, this algorithm achieves a solution by creating a set of edges with the most number of neighbors. The algorithm ends when every edge of the graph that is not in the resulting set is neighbor of an edge from the set. This result in a faster algorithm, however it reduces the accuracy. This occurs because the algorithm returns the first possible solution that it finds, which may not be the best. In this algorithm is worth to evaluate the best, worst and average case scenario, since the algorithm can find a solution earlier. The best solution happens when all edges are neighbor of the a dominant edge. In this case the algorithm only runs all the edges one time, to find the one with the most neighbors, which means a complexity of $\Omega(n)$ = n . The worst case scenario is when none of the edges has a neighbors. In this case the algorithm will repeatedly run the queue and remove an edge in the end of each cycle with a complexity of $O(n)$ = n(n-1)= n**2 * n = n**2.

VI. Results

In this section i will talk about the results of the algorithms developed. It is important to note that we could only test 20 graphs using the exhaustive algorithm without taking to much time, in contrast to the greedy algorithm which run the 385 test, which means that the graphs used for this approach have a lack of data.

A. Basic Operations

As we can see in graph the figure II and III illustrate that the order of growth of the Greedy algorithm is 2**N. In the figure IV and V we can see an N**2 growth in both figure, however in the figure V the growth is more accentuated. We can also see in both figure an appearance of 4 branches. That may symbolizes the 4 difference type of number of edges generation per vertex (12.5%, 25%, 50%, and 75% of the maximum number of edges). This indicates that the number of basic operations is affected not only by the number of vertices but by the number of edges. This effect also is happens in the exhaustive search, however i chose to use one branch in the figure II and III to better understand the graph growth.

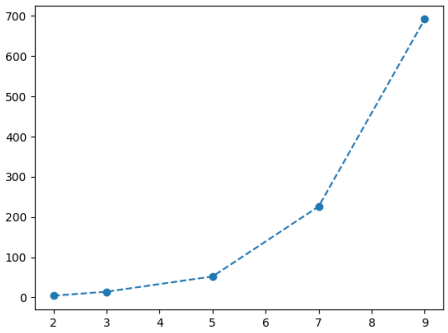

Fig. 2
Relation between edges and number of basic operations for the Exhaustive Search Algorithm

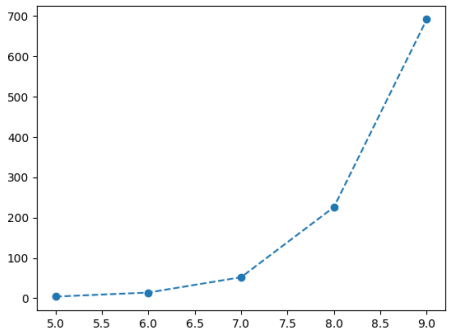

Fig. 3
Relation between vertices and number of basic operations for the Exhaustive Search Algorithm

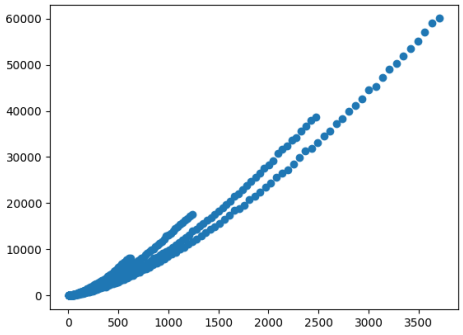

Fig. 4
Relation between edges and number of basic operations for the Greedy Search Algorithm

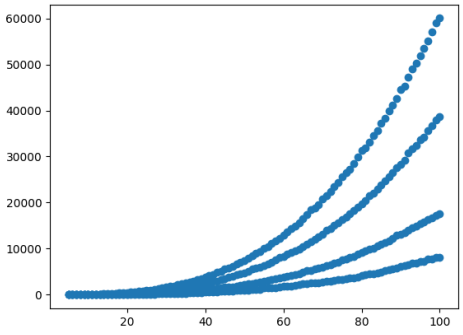

Fig. 5
Relation between vertices and number of basic operations for the Greedy Search Algorithm

B. Solutions Tested

In the figure VI to IX we can prove that the number of solutions tested also have the same growth for the exhaustive search algorithm and the Greedy al-

gorithm, except when compared with the number of vertices which looks linear.

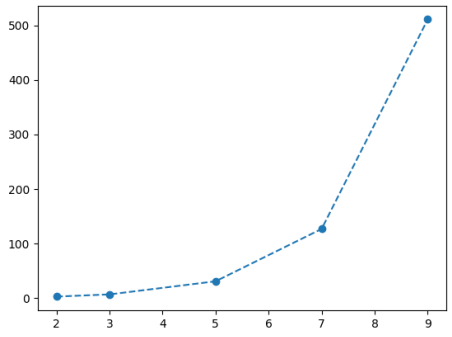

Fig. 6
Relation between edges and number of solutions tested for the Exhaustive Search Algorithm

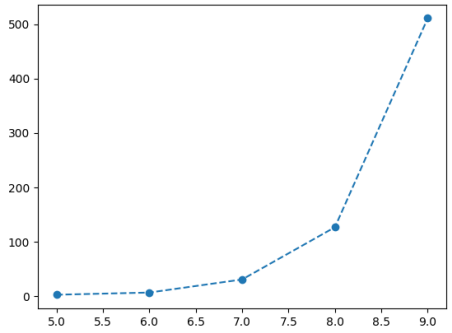

Fig. 7
Relation between vertices and number of solutions tested for the Exhaustive Search Algorithm

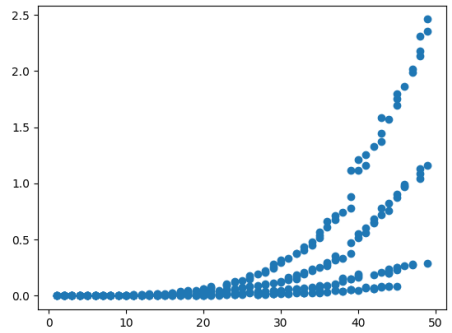

Fig. 8
Relation between edges and number of solutions tested for the Greedy Search Algorithm

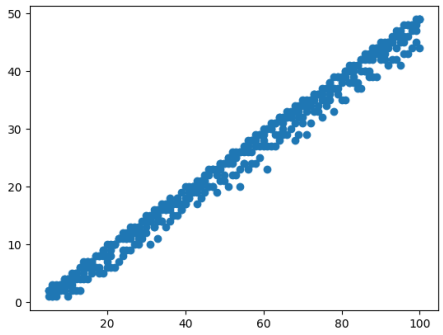

Fig. 9
Relation between vertices and number of solutions tested for the Greedy Search Algorithm

C. Execution Time

As we can see in figure in the next figure, we can prove that the exhaustive search algorithm has a exponential growth while the greedy algorithm has a n**2 growth. We can also notice that using execution time as unit of measurement of algorithm complexity is not ideal since its noticeable the presence of outliers. However the results given are acceptable.

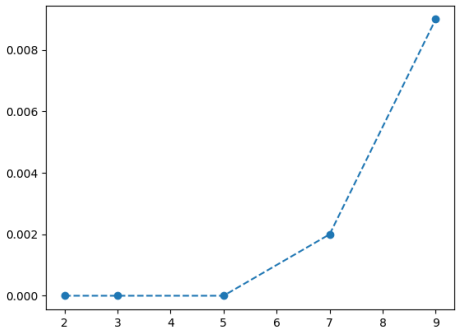

Fig. 10
Relation between edges and execution time for the Exhaustive Search Algorithm

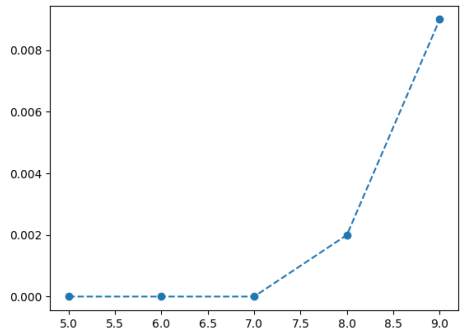

Fig. 11
Relation between vertices and execution time for the Exhaustive Search Algorithm

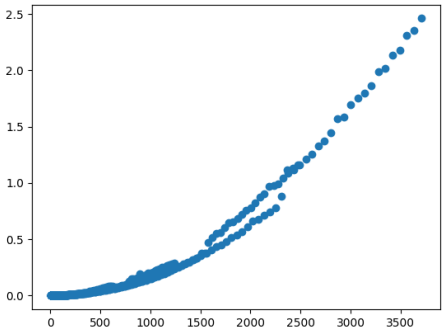

Fig. 12
Relation between edges and execution time for the Greedy Search Algorithm

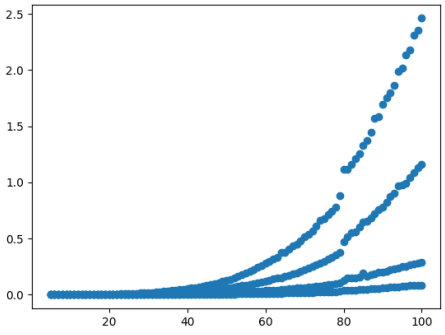

Fig. 13
Relation between vertices and execution time for the Greedy Search Algorithm

VII. Conclusion

In conclusion, despite an exhaustive search produces the optimal result can be very costly because it has an exponential growth. In contrast, the greedy search produces an faster result with a growth of n**2, but loses its accuracy.

References

Wikipedia (2023). *Edge Dominating Set* [Online]. Available: https://en.wikipedia.org/wiki/Edge_dominating_set

Alina Campan, Traian Marius Truta, and Matthew Beckerich *Fast Dominating Set Algorithms for Social Networks* [Online]. Available: https://ceur-ws.org/Vol-1353/paper_16.pdf