Maximum Independent Set Problem

Rafael Gomes de Sá, 104552, rafael.sa@ua.pt, MEI

***Resumo* - O presente relatório apresenta o algoritmo desenvolvido para resolver o problema do conjunto independente de vértices de cardinalidade máxima de um grafo não orientado, bem como a análise ao esforço computacional realizado pelo algoritmo.**

O algoritmo foi desenvolvido através de uma estratégia de pesquisa exaustiva, tendo disponíveis duas variantes para apresentar a solução para o problema: obtendo apenas o primeiro conjunto independente de cardinalidade máxima, ou obtendo todos os conjuntos independentes de cardinalidade máxima.

Para realizar a análise ao esforço computacional realizado pelo algoritmo foi analisada a sua complexidade, e foi realizada uma sequência de testes com instâncias sucessivamente maiores do problema, de forma a registar e analisar um conjunto de métricas. Por último, foi estimado o tempo de execução necessário para resolver instâncias do problema de muito maior dimensão com o algoritmo desenvolvido.

***Abstract* - This report presents the algorithm developed to solve the maximum independent set problem of an undirected graph, as well as the analysis of the computational effort performed by the algorithm.**

The algorithm was developed through an exhaustive search strategy, having two variants available to present the solution to the problem: obtaining only the first maximum independent set, or obtaining all the maximum independent sets.

To perform the analysis of the computational effort performed by the algorithm, its complexity was analyzed, and a sequence of tests was conducted with successively larger instances of the problem, in order to record and analyze a set of metrics. Finally, the execution time required to solve much larger instances of the problem with the developed algorithm was estimated.

I. INTRODUCTION

A graph is a pair $G = (V, E)$, where the elements of V are the vertices (or nodes, or points) of the graph G, and the elements of E are its edges (or lines). The usual way to picture a graph is by drawing a dot for each vertex and joining two of these dots by a line if the corresponding two vertices form an edge [1]. Figure 1 illustrates an example of a graph.

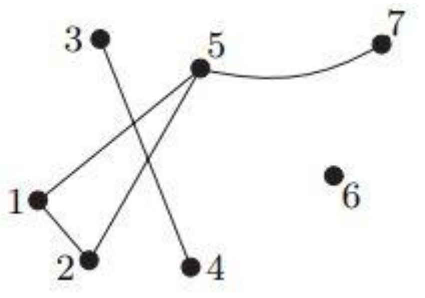

Fig. 1 - Example of graph with 7 vertices and 5 edges [1]

Graphs can be classified based on whether they are undirected or directed. An undirected graph simply represents edges as lines between the nodes, with no additional information about the relationship between the nodes. In a directed graph, the edges provide orientation in addition to connecting nodes [2]. In the case of the graph illustrated in figure 1, it is an undirected graph, as it has no representation of the orientation on the edges.

A lot of problems can be paraphrased to a graph problem or a near similar subproblem. One of the problems is the maximum independent set problem.

An independent set of an undirected graph is a subset of vertices in which no two vertices are adjacent. Given a set of vertices, the maximum independent set problem calls for finding the independent set of maximum cardinality [3].

The maximum independent set problem is a classic one in computer science and in graph theory, and is known to be NP-hard [3]. A problem p is NP-hard if a polynomial-time algorithm for p would imply a polynomial-time algorithm for every problem in NP [4]. The maximum independent set problem has many important applications, including combinatorial auctions, graph coloring, coding theory, geometric tiling, fault diagnosis, pattern recognition, molecular biology, and more recently its application in bioinformatics have become important [3].

To solve the problem, an exhaustive search algorithm was developed. An exhaustive search algorithm consists of an algorithm that tests all possibilities in order to obtain a solution to the problem.

II. ALGORITHM ANALYSIS

In order to analyze the computational effort performed by the algorithm, an analysis of the complexity of the algorithm must be made. In order to provide a better understanding, an explanation of the developed algorithm will be made first.

A. Graph Generation and Representation

The graphs are generated considering the number of vertices and the percentage of edges in relation to the total number of possible edges. The graph is initialized with the given number of vertices, without any edges. Then, for each vertex pair, a number between 0 and 1 is randomly generated, indicating whether or not there is an edge between them. This procedure is done while the graph does not reach the minimum percentage of edges required. For graphs with a small number of vertices it is not always possible to reach the exact percentage of edges. In this case, an approximation is made to the closest percentage of edges higher than the one required. The total number of possible edges in a graph with *n* vertices is given by the formula:

$$f(n) = \frac{n(n-1)}{2}$$

The representation of the graph is done through an adjacency matrix. Therefore, a two-dimensional vector V x V is maintained, where V represents the number of vertices of the graph. The matrix contains a Boolean representation, where the value zero indicates that there is no edge between the two vertices, and the value one indicates the existence of an edge [2]. Since these are undirected graphs, there are two entries in the matrix for each edge, changing only the order between the two vertices.

B. Determination of the Maximum Independent Set

The determination of the maximum independent set is based on the use of the combinations method under the itertools module of Python. The combinations method generates all possible combinations of r elements, for an array of size n. This method was used instead of the permutations method, since these are undirected graphs, and the order of the vertices is not relevant. The number of possible combinations of *r* elements from an array of size *n* is given by the formula [5]:

$$\binom{n}{r} = \frac{n!}{r!\,(n-r)!}$$

known as the binomial coefficient.

All possible combinations of vertices are generated, starting with those with the largest size, since that is the purpose of the problem. For each of the generated combinations, the independence of the set must be verified. The verification of the independence of the set is also performed using the combinations method. For each possible pair of vertices of the set, it is verified that there is no edge between them. If there is, then the condition of independence is not verified. If after all combinations no edge is found, then the set checks the condition of independence.

The search for the set is interrupted when the first set that checks the condition of independence is found. This is done because the larger sets are generated first, and all sets that will be tested later either have the same size as the set already verified or have a smaller size.

A variation of the algorithm was also developed, where all the maximum independent sets are identified. To identify all the maximum independent sets the same logic is followed, with the difference that the program only interrupts the search when it starts to analyze sets with a smaller size than the sets that have already checked the condition of independence.

C. Algorithm Complexity Analysis

The algorithm starts by going through all possible sizes for the set, starting with the number of vertices, *n*, of the graph, which corresponds to the largest possible size for the set, down to the sets of size one vertex. For each of these sizes, *r*, the possible combinations, $\binom{n}{r}$, are generated, and for each combination, all possible combinations of size 2, $\binom{r}{2}$, are also generated.

Thus, the number of iterations that the algorithm has to perform is given by the formula:

$$\sum_{r=n}^{1}\left(\frac{n!}{r!\,(n-r)!}\times\frac{r(r-1)}{2}\right) = \sum_{r=n}^{1}\left(\binom{n}{r}\times\binom{r}{2}\right)$$

where the sum refers to the processing of all sizes for the sets, and the empty set is not checked.

Even if the algorithm does not have to go through all possible sizes, the formula for determining the number of iterations remains the same type.

The number of candidate vertex sets verified is given by the formula:

$$\sum_{r=n}^{1}\binom{n}{r} = \sum_{r=n}^{1}\frac{n!}{r!\,(n-r)!}$$

In the worst case, the algorithm will have to process all the possible sets of vertices, $2^n - 1$ sets, being therefore an algorithm with an exponential complexity order, $O(2^n)$.

III. TEST SEQUENCE ANALYSIS

To test the developed algorithm, a sequence of tests was performed with successively larger instances of the problem. For each test performed, the execution time, the number of basic operations performed, and the total number of maximum independent sets were registered. It was considered as a basic operation the verification of the independence of a set, causing the number of basic operations to be equal to the number of tested solutions/configurations.

Tests were performed for graphs with a size between 2 and 25 vertices. For each graph size, tests were performed with graphs with 0%, 25%, 50%, 75% and 100% of edges, in order to understand which are the best and worst configurations.

It should be noted that for graphs with a small size it is not possible to obtain the exact percentages of edges, so an approximation is made, as explained in section II.

The results registered are regarding the algorithm that identifies all the maximum independent sets.

A. Execution Time Analysis

The results regarding the execution time of the tests performed are shown in figure 2. The execution time is given in seconds.

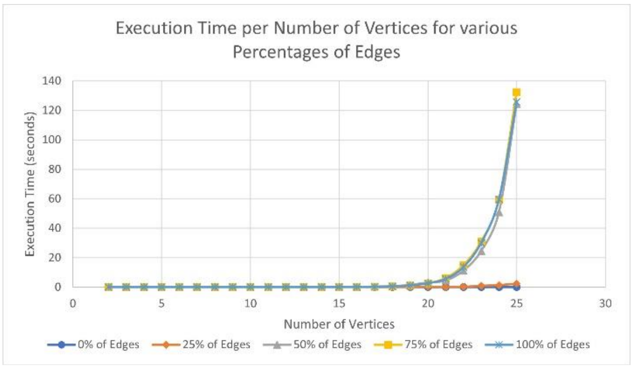

Fig. 2 - Test results regarding the execution time

By analyzing the graph, it is possible to conclude that that the growth rate of the execution time increases exponentially, even though for graphs with a reduced percentage of edges the runtime shows a much lower growth rate than graphs with a high percentage of edges. Thus, it is concluded that the higher the percentage of edges in a graph, the higher the growth rate of the execution time, being upperly limited by the exponential.

B. Number of Basic Operations Analysis

The results of the tests performed regarding the number of basic operations performed, or the number of independence checks of a set, are shown in figure 3.

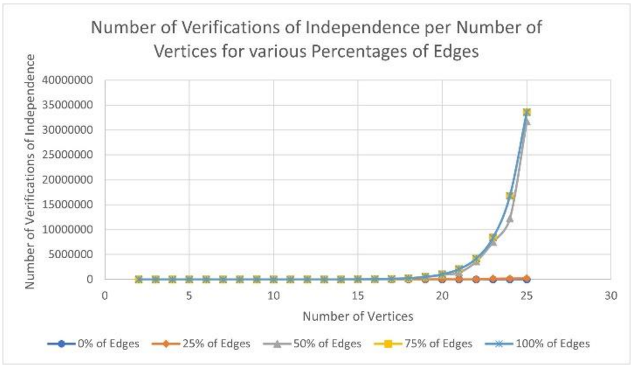

Fig. 3 - Test results regarding the number of basic operations

It is possible to observe that the results concerning the number of basic operations performed show a behavior very similar to the one observed in the results regarding the execution time. It can be concluded that the higher the percentage of edges, the greater the number of basic operations carried out by the algorithm, being upperly limited by the exponential.

C. Number of Maximum Independent Sets Analysis

The results of the tests performed regarding the total number of maximum independent sets are shown in figure 4.

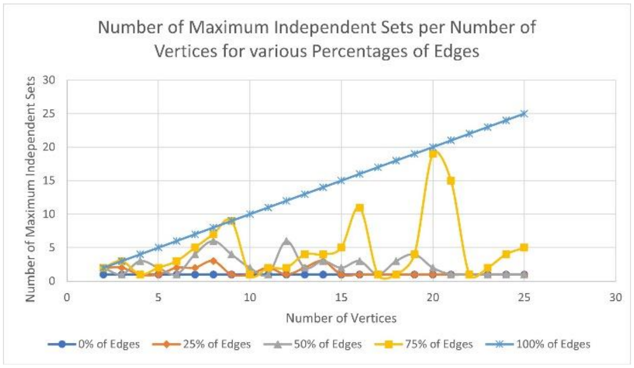

Fig. 4 - Test results regarding the number of Maximum Independent Sets

By analyzing the graph, it is possible to verify that, in general, the higher the percentage of edges in a graph, the greater the total number of maximum independent sets. This number is upperly limited by the number of vertices of the graph, as it is possible to confirm on the chart with the results for the complete graph, that is, with 100% of edges.

D. Worst Case and Best Case

Through the analyses performed previously, it is possible to identify the best and worst types of graphs, regarding the number of iterations that the algorithm has to perform.

As mentioned in subsection B, the higher the percentage of edges in the graph, the greater the number of basic operations the algorithm will have to perform. Therefore, it is possible to conclude that the best case is the lowest possible percentage of edges, in this case 0% of edges, and the worst case is the largest number of edges possible, in this case 100% of edges.

To obtain an expression, even if an approximate one, that fits the obtained data, the LOGEST function of the Microsoft Excel tool was used. In regression analysis, the LOGEST function calculates an exponential curve that fits the data and returns an array of values that describes the curve. The array that the function returns contains the values m and b, for an equation of type: $y = b \times m^x$, which describes the exponential that approximates the experimental results [6].

According to the experimental results, the equation obtained for the best case is $y = 2$, having a constant complexity order, $O(1)$. As for the equation obtained for the worst case is $y = 0.91308 \times 2.010108^x$, having an exponential complexity order, $O(2^n)$.

By establishing a comparison with the algorithm complexity analysis performed in section II, it is possible to conclude that the experimental results confirm the theoretical formula. The complexity order obtained through the experimental results for the worst case proved to be equal to the theoretical complexity order, and the equations obtained for both cases reveal a high degree of similarity.

It can be said that the more experimental results are considered to derive the equation, the greater the approximation to the theoretical equation.

IV. Estimates for Larger Problem Instances

To obtain the estimates of the execution time required for much larger problem instances, the equation obtained through the LOGEST function, referred to in section III, could be used, but Microsoft Excel provides the GROWTH function that is exactly for that purpose. The GROWTH function calculates predicted exponential growth by using existing data [7].

Since it is not possible to approximate exponentials with values equal to zero in the data, only data where the execution time was greater than zero were used. Execution times were estimated for graphs with 25%, 50%, 75% and 100% of edges and with sizes equal to 50, 100, 250, 500 and 900 vertices. It was not possible to estimate execution times for graphs with a size much higher than 900 vertices because the Microsoft Excel tool considers the estimated values too large.

The obtained estimates are presented in figure 5. For a better representation of the obtained data, a logarithmic scale of base 10 on the y axis was used.

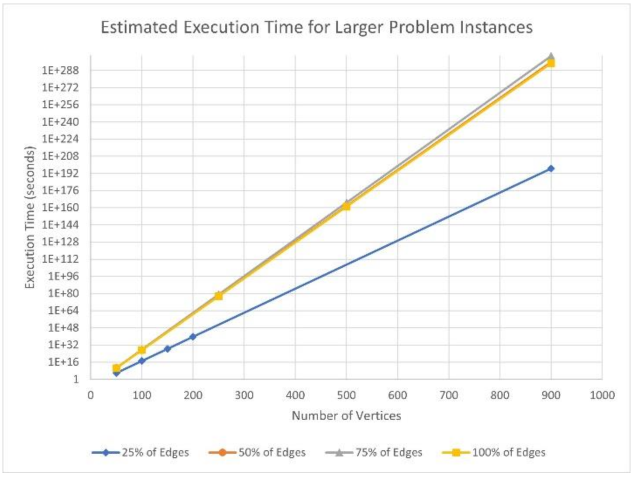

Fig. 5 - Estimated execution time for larger problem instances

By analyzing the graph, it is possible to confirm that for graphs with a lower percentage of edges, the execution time increases slower than for graphs with a higher percentage of edges. However, growth is exponential in all cases.

Analyzing the data in more detail, for graphs with 75% edges, for example, the estimated execution time is given by the equation:

$$y = (3.94212 \times 10^{-07}) \times 2.198065705^x$$

The estimates for the execution times obtained for this example are shown in table 1.

Table 1 - Execution time estimates for graphs with 75% of edges

Number of Vertices	**Estimated Execution Time (seconds)**
50	49861271941
100	6.30662E+27
250	1.27614E+79
500	4.1311E+164
900	2.706E+301

For graphs with 50 edges, the estimated execution time is 49861271941 seconds, which is equivalent to more than 1500 years. For graphs with a size of, for example, 900 vertices, the estimated execution time is so high that it is not even possible to find a time scale to have a better notion of the value.

Based on this analysis, it is possible to conclude that, as the execution time grows exponentially, it is not possible to obtain results for much larger instances of the problem through the exhaustive search algorithm developed.

V. Conclusions

In short, through the analysis of various types of graph configurations, it is concluded that the percentage of edges has a great influence on the execution time and the number of basic operations. It was found that in the best case, the algorithm solves the problem with a constant complexity order, but this configuration is unrealistic.

In the worst case, the algorithm solves the problem with an exponential complexity order, which limits the size of the graphs with which results can be obtained in a timely manner.

It is concluded that, through the exhaustive search algorithm developed in this project, it is not possible to obtain results for graphs with a relatively high size, due to the complexity order of the algorithm.

References

[1] Diestel, R. (2005). Graph Theory. (3rd ed.). Springer. https://doi.org/10.1007/978-3-662-53622-3

[2] Baka, B. (2017). Graphs and Other Algorithms. In B. Baka, Python Data Structures and Algorithms (pp. 168 - 190). Packt. ISBN 9781786467355

[3] Balaji, S. & Venkatasubramanian, Swaminathan & Kannan, K.. (2010). A Simple Algorithm to Optimize Maximum Independent Set. 12.

[4] Erickson, J. (2019). NP-Hardness. In J. Erickson, Algorithms (pp. 379 - 428). ISBN 9781792644832

[5] Simmons, B. (2017, July 19). Combination Formula. Mathwords. https://www.mathwords.com/c/combination_formula.htm

[6] Microsoft. (2020). LOGEST function. Microsoft. https://support.microsoft.com/en-us/office/logest-function-f27462d8-3657-4030-866b-a272c1d18b4b

[7] Microsoft. (2020). GROWTH function. Microsoft. https://support.microsoft.com/en-us/office/growth-function-541a91dc-3d5e-437d-b156-21324e68b80d