Max Cut Problem

Rafael Gil

Abstract –This report presents a study of two algorithms, developed with the aim of providing an approximation to the Max Cut problem. One of the algorithms is based on an exhaustive approach, while the other algorithm is based on a greedy heuristic approach. The article begins with a presentation of the problem to be addressed, followed by an explanation of the algorithms developed. After this, a formal and experimental analysis will be carried out for each of the algorithms, where the results obtained will be compared. Next, by obtaining the largest graph for which it is possible to obtain results in good time, an estimate will be made of the time needed to obtain results for graphs with significantly larger dimensions. Finally, the conclusions are presented.

Resumo –Este relatório apresenta um estudo de dois algoritmos, desenvolvidos com o objetivo de fornecer uma aproximação do problema Max Cut. Um dos algoritmos baseia-se numa abordagem exaustiva, enquanto que o outro algoritmo baseia-se numa abordagem de heuristicas vorazes. O artigo começa por fazer uma apresentação ao problema a ser abordado, sendo seguida por uma explicação dos algoritmos desenvolvidos. Após isso, será feita uma análise formal e experimental para cada um dos algoritmos, onde se irão comparar os resultados obtidos. De seguida, obtendo o maior grafo para o qual seja possivel obter resultados em tempo útil, irá ser feita uma estimativa do tempo necessário para a obtenção de resultados de grafos com dimensões significativamente maiores. Finalmente, são apresentadas as conclusões alcançadas.

Keywords –Max-Cut, Cut, Algorithm, Time Complexity, Greedy Heuristics, Exhaustive Search

Palavras chave –Max-Cut, Corte, Algoritmo, Complexidade Temporal, Heuristicas Vorazes, Procura Exaustiva

I. Introduction

A. Problem Definition

Find a maximum cut for a given undirected graph $G(V, E)$, with n vertices and m edges. A maximum cut of G is a partition of the graph's vertices into two complementary sets S and T, such that the number of edges between the set S and the set T is as large as possible.
This is a NP-hard problem, meaning that there is no known solution to solve it in polynomial time. The best approximation, known to date, for the maximum cut problem, was proposed in 1994, by Goemans and Williamson, where a semidefinite programming relaxation was used to obtain a consistent approximation of .87856 times of the optimal solution.

B. Exhaustive Search

This is an algorithmic technique based on a brute-force approach, where every possible iteration of a solution is analyzed. Generally, every candidate solution, for a problem, is listed and tested, in order to check if it meets the solution requirements.
When dealing with large amounts of data, this approach tends to prove useless, as it grows exponentially, in terms of execution time.

C. Greedy Heuristic

This is an algorithmic approach that searches to select the optimal solution of an iteration of a problem, without worrying about the overall optimal solution of the given problem.
This technique might not always provide the best result for every problem, but it, usually, provides a good enough approximation to the optimal solution, in return of being much less time consuming than other algorithmic approaches.

II. Analysis of the Algorithms Implemented

A. Exhaustive Search

As previously introduced, the exhaustive search algorithm consists of generating every possible iteration of the solution and getting the best one. This approach, when applied to the maximum cut problem, consists of every possible combination for grouping the vertices of the graph into two distinct sets, as to maximize the cut between the two sets, there are $2^n - 2$ possible solutions, where n is the number of vertices and -2 because, if the objective is to maximize the number of edges between the two sets, we cannot have an empty set, so we do not count those iterations of the solution.

For example, given a graph with 5 vertices, shown in Fig. 1, if we want to group the vertices in two sets, S and T, as to maximize the number of edges between the two sets, there are $2^5 = 32 - 2 = 30$ possible combinations, represented in the table I, where:

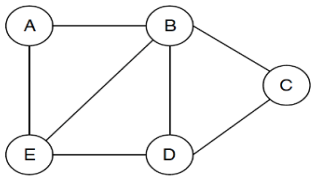

Fig. 1 - Undirected graph with 5 vertices

$$x_i \in S, \text{ if } x_i = 1, \text{ for } i = \{A, B, C, D, E\}$$

and:

$$x_i \in T, \text{ if } x_i = 0, \text{ for } i = \{A, B, C, D, E\}$$

A	B	C	D	E	Cut
0	0	0	0	1	3
0	0	0	1	0	3
0	0	0	1	1	4
0	0	1	0	0	2
0	0	1	0	1	5
0	0	1	1	0	3
0	0	1	1	1	4
0	1	0	0	0	4
0	1	0	0	1	5
0	1	0	1	0	5
0	1	0	1	1	4
0	1	1	0	0	4
0	1	1	0	1	5
0	1	1	1	0	3
0	1	1	1	1	2
1	0	0	0	0	2
1	0	0	0	1	3
1	0	0	1	0	5
1	0	0	1	1	4
1	0	1	0	0	4
1	0	1	0	1	5
1	0	1	1	0	5
1	0	1	1	1	4
1	1	0	0	0	4
1	1	0	0	1	3
1	1	0	1	0	5
1	1	0	1	1	2
1	1	1	0	0	4
1	1	1	0	1	3
1	1	1	1	0	3

TABLE I
Every iteration of the solution and its corresponding cut

For this graph, there have been found many cuts which produce the maximum amount of edges between the two sets, S and T. One possible maximum cut has a value of 5, where

$$S = \{B, E\}$$

and:

$$T = \{A, C, D\}$$

Whit this we can conclude that, despite it always finding the maximum cut, it becomes useless, for larger graphs, as it is of exponential time complexity, $O(2^n)$.

B. Greedy Heuristic

There exist many different greedy heuristics that can be employed to get an approximate solution of the maximum cut problem.
The algorithm being analysed here, consists of a randomized greedy heuristic [1]. Given a graph, randomly attribute each vertex n to either set S or set T, with:

$$P(n \in S) = P(n \in T) = \frac{1}{2}$$

This makes it so that, for any edge (n, m), the probability it belongs to a cut C, is the same as the probability of the vertices belonging to different sets:

$$P((n, m) \in C) = \frac{1}{2}$$

The process of attributing vertices to sets, seen in algorithm 1, consists on iterating through every vertex in the graph and attributing it to one of the sets. This process runs in linear time, $O(n)$, where n is the number of vertices in the graph.

Algorithm 1 Generate random partition of sets

```
function Random_Partition(graph)
    partition ← {}
    for node in graph.nodes() do
        partition ← {node, random(S, T)}
    end for
    return partition
end function
```

The next process in this algorithm, described in algorithm 2, is the calculation of the cut' size of the random partition that has been generated, which consists in iterating through every edge in the graph and checking, in the random partition, if the vertices of the edge are in different sets. This runs in linear time, $O(e)$, where e is the number of edges in the graph.

Algorithm 2 Calculate Cut Size

```
function Calculate_Cut_Size(graph, partition)
    cut_size ← 0
    for edge in graph.edges() do
        if partition[edge[0]] ≠ partition[edge[1]] then   ▷ if the two vertices of the edge are in different sets
            cut_size ← cut_size + 1
        end if
    end for
    return cut_size
end function
```

In order to make this useful, the algorithm has to perform both this processes a given number of times N, demonstrated in algorithm 3, where the bigger that number the better the approximation, with an increase in computational cost. This makes it so that the overall algorithm runs in polynomial time, $O(N * (n + e))$, where N is the number of iterations, n is the number of vertices in the graph and e is the number of edges in the graph.

Algorithm 3 Random Greedy Heuristic for Max Cut

```
function Greedy_Max_Cut(graph, num_iterations)
    best_partition ← {}
    best_cut ← 0
    for i in range(num_iterations) do
        partition ← Random_Partition()
        cut ← Calculate_Cut_Size()
        if cut > best_cut then
            best_partition ← partition
            best_cut ← cut
        end if
    end for
    return best_partition, best_cut
end function
```

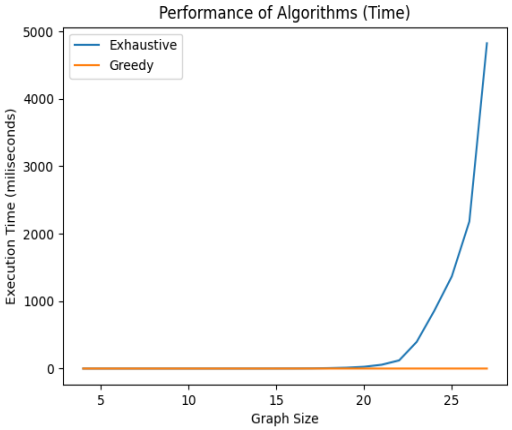

Fig. 3 - Impact of number of vertices in execution time per Algorithm

III. Results

A. Impact of the vertices on the algorithms

Experiments were carried out, to determine the number of basic operations and the execution time for each algorithm. In order to do this, several graphs were used, with a successively larger number of vertices, starting as low as 4 vertices and ending at a highest of 25 vertices, and a constant probability of 30% for edge creation. For the greedy algorithm, a constant of 1000 iterations was used, when generating the random partitions.
After running these experiments,it is clearly noticeable the difference in performance between the two algorithms, the larger the graph is. We can make a comparison of the growth between the algorithms, in Fig. 2 and Fig. 3, and see, after a certain point, the impracticability of the exhaustive algorithm.

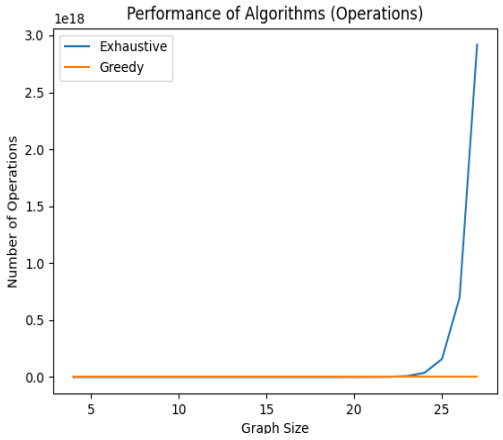

Fig. 2 - Impact of number of vertices in operations per Algorithm

An interesting aspect to observe is, that for the smaller graphs, the exhaustive algorithm actually performs better than the greedy algorithm, taking less time to execute, seen in table II, and performing less operations, seen in table III. This quickly gets overturned as the graph slightly grows.

N. Vertex	Exhaustive Time(s)	Greedy Time(s)
4	0.0001389	0.0045344
5	0.0002294	0.0081493
6	0.0002922	0.0075021
7	0.00068979	0.009448
8	0.0015438	0.0097511
9	0.0049507	0.0120359
10	0.0095575	0.0109243
11	0.0229472	0.0134678
12	0.0453573	0.0148012
13	0.1089745	0.0153767
14	0.2192243	0.0192595
15	0.7063999	0.0276222
16	1.1780833	0.0295291
17	2.5152729	0.0236615
18	6.2236802	0.0450665
19	12.5995052	0.0322919
20	26.0108515	0.0360631
21	56.2371403	0.0327581
22	120.9686704	0.0364963
23	395.7680206	0.1048222
24	855.8324934	0.0706066
25	1366.1357973	0.0385439

TABLE II
Results for execution time

N. Vertex	Exhaustive Operations	Greedy Operations
4	1184	8522000
5	6240	11037000
6	37056	14922000
7	172800	17532000
8	886528	21411000
9	4329472	25524000
10	20457472	29437000
11	100683776	34893000
12	528531456	43505000
13	2516680704	50645000
14	11274584064	55882000
15	51540099072	62958000
16	244814249984	73026000
17	1108103790592	81585000
18	5153964687360	93107000
19	22677435187200	101445000
20	102254601306112	113003000
21	442003724697600	121451000
22	1899956206043136	130030000
23	8233143320444928	140091000
24	36732484947279872	154520000
25	158751888005660672	165918000

TABLE III
Results for operations carried out

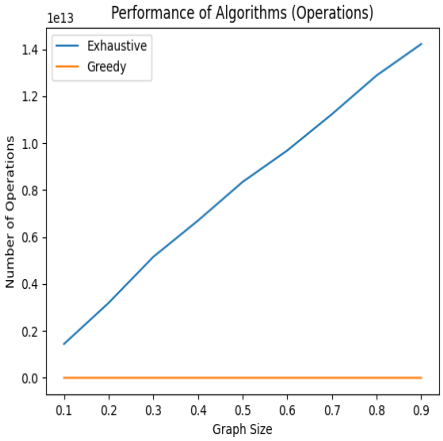

Fig. 4 - Impact of number of edges in operations per Algorithm

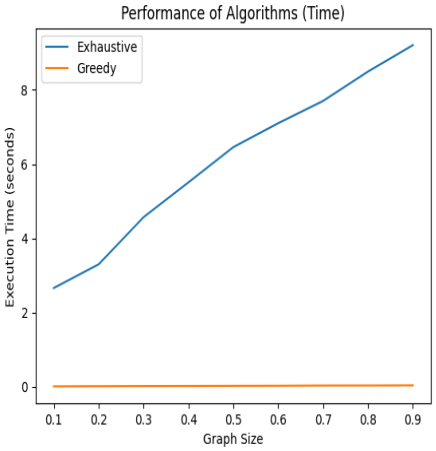

Fig. 5 - Impact of number of edges in execution time per Algorithm

The experimental results do not properly follow the formal analysis, as a close look to the results shows that, for the exhaustive approach, the results grow at an exponential rate of base 5 and not of base 2, as previously stated.

B. Impact of the edges on the algorithms

Now, to carry out these experiments, several graphs were used, with a successively larger probability of edge creation, starting at 10% and going all the way up to 90%, and a constant number of vertices of 18. For the greedy algorithm, a constant of 1000 iterations was used, when generating the random partitions.
Similarly to the previous results, the exhaustive algorithm requires a bigger computational effort than the greedy algorithm. But there is an interesting observation to make: in contrast to the number of vertices, the exhaustive search algorithm never produces a result that is less costly than the greedy algorithm, no matter how low the number of edges is, as exemplified by Fig. 4 and Fig. 5.

C. Cut

It is known already that the exhaustive search algorithm always provides the best cut (or one of the best cuts) and that the greedy algorithm provides an approximation to the best cut, but how good is that approximation? Using the results obtained from the tests carried out in subsection *A*, if we analyse the calculated cuts, seen in table IV, we can see that the approximations are not that far off the actual cut values. It is noticeable that, as the graph grows larger, the approximations become less accurate but still giving an overall good enough value for the cut.

N. Vertex	Exhaustive Cut	Greedy Cut
4	3	3
5	4	4
6	5	5
7	6	6
8	8	8
9	10	10
10	12	11
11	15	15
12	18	17
13	21	21
14	24	22
15	27	24
16	30	27
17	33	31
18	38	36
19	42	40
20	46	43
21	50	46
22	53	49
23	58	51
24	64	59
25	69	62

TABLE IV
Calculated Cut per Algorithm

IV. Estimating the cost for larger graphs

Trying to solve this problem, with the implemented algorithms, presents a great expense of computational power and time. Following the experimental results done previously, we can make a rough estimate of the execution time, for each of the algorithms, by making a spatial extrapolation of our results.
The larger graph we managed to used, in the experimental analysis, had 25 vertices, which took, approximately, 1366 seconds, or 23 minutes, for the exhaustive algorithm and 0.04 seconds for the greedy algorithm. Considering the same conditions used for the experimental analysis of the impact of the vertices on the algorithms, we can fit the obtained results to a mathematical model.

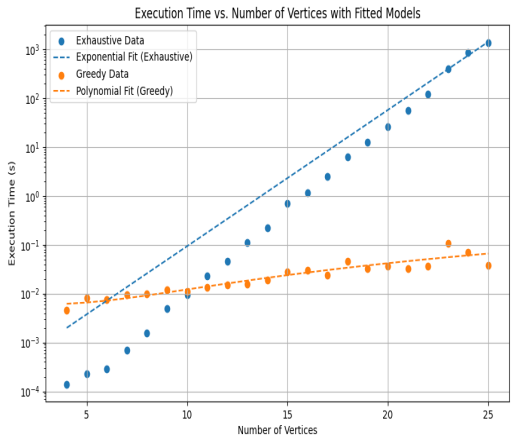

Fig. 6 - Fitting mathematical models

Based on the fits, seen in Fig. 6, we can obtain the following functions for the Exhaustive algorithm and for the Greedy algorithm, respectively:

$$f(N) = 0.000153 \times e^{0.641501 \times N}$$

$$f(N) = 0.000123 \times N^2 - 0.000734 \times N + 0.007150$$

Now, we can used these to make our estimates on the execution time, for a given graph with N vertices. In table V, it is evident how much more efficient the Greedy algorithm is when compared to the Exhaustive Algorithm, even though it does not give an accurate solution.

N. Vertex	Exhaustive Cut ($\approx s$)	Greedy Cut ($\approx s$)
30	34890	0.09583
35	862390	0.1321
40	21315975	0.17459
45	526874197	0.2232
50	13022928393	0.27795

TABLE V
Approximate estimations on execution time

V. Real life applications

The Max-Cut problem has many applications in various fields such as network design, statistical physics and VLSI design, circuit layout design [2], and data clustering [3].

VI. Conclusion

To conclude this report, it is essential to say how beneficial making it was to complementing the understandings of the Exhaustive and Greedy algorithms and how to apply them to a given problem. It was also extremely efficient for practicing the formal analysis of algorithms and understanding the various concepts it involves, such as analysing the time complexity of an algorithm.

References

[1] Mudathir Mahgoub, "Maxcut problem", Lecture Notes CS:5350 Introduction to approximation algorithms, 2019.

[2] M. Junger F. Barahona, M. Grotschel and G. Reinelt, "An application of combinatorial optimization to statistical physics and circuit layout design", Operational Research, 1988.

[3] T. Zeugmann J. Poland, "Clustering pairwise distances with missing data: Maximum cuts versus normalized cuts", Lecture Notes in Computer Science, 2006.