Advanced Algorithms
1st Project - Exhaustive Search

Margarida Silva Martins

Abstract **–This report presents the work done on the first project of the Advanced Algorithms course.**
The paper will analyse and compare two different algorithmic approaches developed, exhaustive search and greedy technique in order to solve a maximum weight clique problem.

I. Introduction

Exhaustive search, or brute-force search is a algorithmic technique for problem solving. Another technique is a greedy one using greedy heuristics.
Both this approaches can be used in order to solve graph problems. This paper aims to use these two approaches to solve the following maximum weight clique graph problem [1]:

> Find a maximum weight clique for a given undirected graph G(V, E), whose vertices carry positive weights, with n vertices and m edges. A clique of G is a subset of vertices, all adjacent to each other, i.e., defining a complete subgraph of G. The weight of a clique is the sum of its vertices' weights. A maximum weight clique is a clique whose total weight is as large as possible.

Exhaustive search is a brute-force approach used in order to solve combinatorial problems [2]. It generates every element of the domain and selects the optimal one, the maximum weighted clique in this case. Although very straightforward and always producing the right answer, exhaustive search algorithms have high computational complexities, making them too slow for larger problems.
The greedy approach consists in constructing a solution where each choice made is feasible, locally optimal and irrevocable [3]. For some problems this approach might not always reach the correct answer but can give good approximations while being faster than the exhaustive search.

II. Formal Computational Analysis

A. Exhaustive search

In order to do a formal computational analysis first is necessary to understand how the algorithm works. The exhaustive search algorithm developed will loop for each possible clique size [1, V] and generate all possible vertices combinations, where the order does not matter. Then for each combination it will sum all the vertices weights and compare with the maximum weight calculated, if the value is larger, the maximum weight variable is updated and the iteration continues. In the end, the maximum weight of the clique and the set of vertices that compose it are returned. Here is the pseudocode for the algorithm:

Algorithm 1 Exhaustive Search

```
1:  maximum_weight←0
2:  maximum_clique←{}
3:  for i = 1...V do
4:      maximum_weight←0
5:      maximum_clique←{}
6:      P←set of all vertices combinations of size i
7:      while P is not empty do
8:          p ← pop last value from P
9:          weight ← sum of p weights
10:         if weight ¿ maximum_weight then
11:             maximum_weight ← weight
12:             maximum_clique ← p
    return maximum_weight, maximum_clique
```

The computational complexity of going through all the vertices combinations is the number of k-combinations for all k, where k is the number of subsets of V, this is given by the following equation [4].

$$\sum_{k=1}^{n} \binom{n}{k} = 2^n - 1 \tag{1}$$

We can then conclude that the complexity of going through all the vertices combinations is $O(2^n)$, with n being the number of vertices in the graph. The complexity of removing the last value of a list is O(1) [5] while summing all weight vertices is O(n), this is done inside the combination loop.
With this we can conclude that the complexity of the algorithm is $O(n) * O(2^n) = O(n2^n)$

B. Greedy heuristic

The greedy algorithm developed first orders the graph vertices by their weights. Then it iterates removing one vertex at the time (by descending order of weight) and building a clique if possible, if not it discards the vertex. In the end it returns the clique built and the sum of it vertices weights.
Here is the pseudocode for the algorithm:

Algorithm 2 Greedy Heuristic

```
max_weight←0
max_clique←{}
V ← Order_by_weight(V)
while V is not empty do
    v ← remove last value from V
    if {v} U max_clique is a clique then
        max_clique ← {v} U max_clique
        max_weight ← max_weight + v.weight
return max_weight, max_clique
```

The computational complexity of sorting the vertices by their weight is O(n log n) [5], with n being the number of vertices. Iterating over the graph vertices is O(n) and removing the last value of a list is O(1) [5]. With this we can conclude that the complexity of the greedy algorithm is O(n) + O (n log n) = O(n log n).

Another greedy heuristic was implemented considering the number of edges a vertex had but it was discarded as the overall results were worse. However a combination of the results of both heuristics could lead to improved results.

III. Graph Generation

In order to test the algorithms developed graphs were generated randomly using a defined seed. For each number of vertices 3 different graphs were generated with edge densities of 25%, 50%, and 75%. The vertices have integer coordinates between 1 and 9 and integer weights between 1 and 100 inclusive.

Each graph is stored in a file where the first line is the number of vertices, n, the next n lines represent each vertices ,the x and y coordinates and the weight of the vertices. The last n lines represent the adjacency matrix of the graph. In the figure 1 below we can see an example of a graph generated.

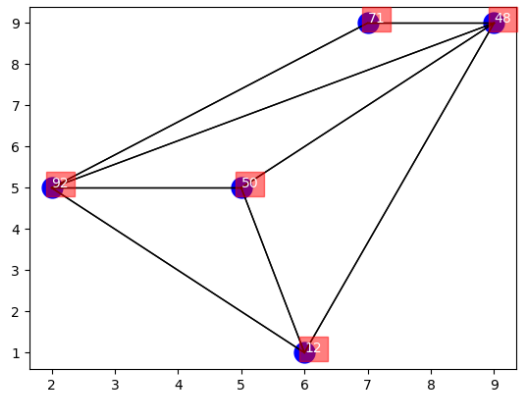

Fig. 1 - Graph with 5 vertices and edge density of 75%

IV. Experiments

Experiments were made for both algorithms using [2,3,4,5,7,9,11,13,15,17,20] as number of vertices. The solution found, the number of basic operations carried out, the execution time and the number of configurations tested were registered for each experiment.

A. Solutions found

The exhaustive search algorithm always finds the maximum weight clique for a graph. However this is not true for the greedy approach. Of the 33 graphs tested the solutions found in 5 of them using the greedy approach were wrong.

In the tables I and II below we can analyze the differences between the maximum weights cliques found.

n	edge %	exhaustive search weight	greedy weight
5	25	115	108
7	25	100	89
9	25	134	122
11	50	234	230
17	25	224	216

TABLE I
Different maximum clique weights found for exhaustive and greedy search

n	edge %	exhaustive search clique	greedy clique
5	25	{3,4}	{0,2}
7	25	{0,1,3}	{2}
9	25	{0,2,6}	{8,6}
11	50	{1,2,6,10}	{3,8,10}
17	25	{2,3,10}	{8,9,16}

TABLE II
Different maximum weight cliques found for exhaustive and greedy search

The greedy approach works better for graphs with higher edge density since the probability of the highest weighted vertex belonging to the clique is higher. As expected the number of vertices of the cliques found by the greedy approach is always equal or less than the ones in the exhaustive search.

In figure 2 below we can visualize the different solutions found by the exhaustive search algorithm and the greedy approach for the graph with 11 vertices and 50% of edge density.

In this case the greedy approach fails to reach the maximum weight clique because it chooses the node with weight 98 which does not belong to the maximum weight clique. Because the choice made is irrevocable it will be impossible to get the correct answer.

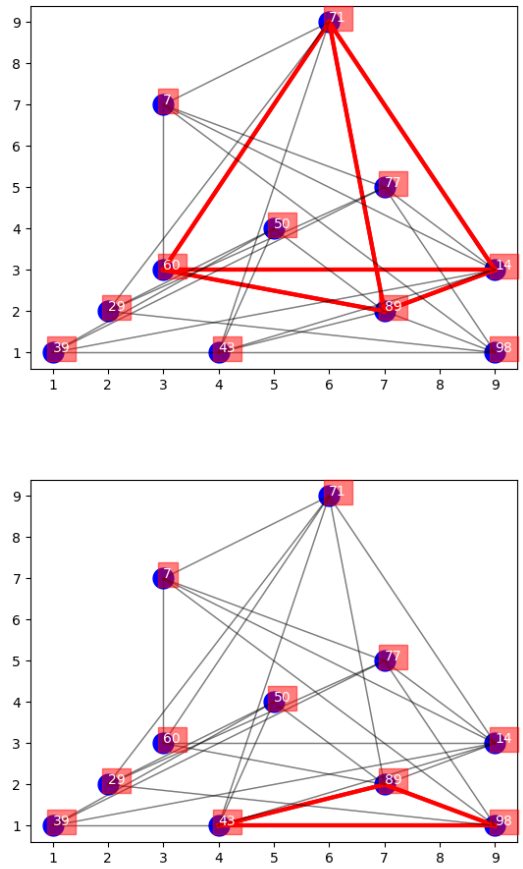

Fig. 2 - Maximum weight clique found for graph with 11 vertices and density of 50% using the exhaustive search algorithm (above) and the greedy approach (below)

B. Results

B.1 Number of basic operations

Table III shows the number of basic operations done by the two algorithms for graphs with different number of vertices and edges densities.

n	exhaustive search			greedy		
	25%	50 %	75 %	25%	50 %	75 %
2	19	19	19	9	9	9
3	40	42	57	14	15	18
4	109	111	127	18	18	23
5	290	300	319	21	23	27
7	1951	1972	2046	23	28	46
9	12073	12127	12331	45	55	69
11	69680	69778	70195	57	65	78
13	380995	381171	382293	68	83	100
15	1998957	1999276	2001819	90	92	122
17	10158197	10158641	10164035	99	110	146
20	111149244	111150060	111162802	113	147	170

TABLE III

NUMBER OF BASIC OPERATIONS FOR DIFFERENT NUMBER OF VERTICES AND EDGES DENSITIES

For both algorithms as the number of vertices increases the number of basic operations also increases. However the growth rate is much larger for the exhaustive search approach.

We can also observe that although in the same order of magnitude a higher number of edges leads to a higher number of basic operations. For the exhaustive search this can be explained due to the fact that the sum of weights of a set of vertices will not be computed if it was proved that they do not form a clique, there are more cliques in high edge density graphs. For the greedy approach this can be explained because the current weight and clique variables will not be updated if the vertex does not form a clique with the already constructed one.

B.2 Execution times

Table IV displays the execution time by the two algorithms for graphs with different number of vertices and edges densities.

n	exhaustive search			greedy		
	25%	50 %	75 %	25%	50 %	75 %
2	$1.2 * 10^{-5}$	$1.3 * 10^{-5}$	$1.7 * 10^{-5}$	$5.0 * 10^{-6}$	$6.2 * 10^{-6}$	$1.1 * 10^{-5}$
3	$2.0 * 10^{-5}$	$2.1 * 10^{-5}$	$2.4 * 10^{-5}$	$5.7 * 10^{-6}$	$6.4 * 10^{-6}$	$6.4 * 10^{-6}$
4	$3.9 * 10^{-5}$	$4.1 * 10^{-5}$	$4.5 * 10^{-5}$	$6.2 * 10^{-6}$	$6.2 * 10^{-6}$	$6.9 * 10^{-6}$
5	$8.4 * 10^{-5}$	$8.7 * 10^{-5}$	$9.2 * 10^{-5}$	$7.2 * 10^{-6}$	$7.2 * 10^{-6}$	$7.6 * 10^{-6}$
7	$4.3 * 10^{-4}$	$4.3 * 10^{-4}$	$4.4 * 10^{-4}$	$7.8 * 10^{-6}$	$8.1 * 10^{-6}$	$1.1 * 10^{-5}$
9	$2.2 * 10^{-3}$	$2.2 * 10^{-3}$	$2.3 * 10^{-3}$	$9.1 * 10^{-6}$	$9.8 * 10^{-6}$	$1.1 * 10^{-5}$
11	$9.9 * 10^{-3}$	$9.6 * 10^{-3}$	$1.0 * 10^{-2}$	$9.3 * 10^{-6}$	$9.5 * 10^{-6}$	$1.1 * 10^{-5}$
13	$5.2 * 10^{-2}$	$5.0 * 10^{-2}$	$4.7 * 10^{-2}$	$1 * 1^{-5}$	$1.1 * 10^{-5}$	$1.3 * 10^{-5}$
15	0.267	0.268	0.24	$1.5 * 10^{-5}$	$1.6 * 10^{-5}$	$1.7 * 10^{-5}$
17	1.4	1.36s	1.3	$1.6 * 10^{-5}$	$1.7 * 10^{-5}$	$2.0 * 10^{-5}$
20	19.6	17.6	16.6	$3.5 * 10^{-5}$	$2.0 * 10^{-5}$	$2.2 * 10^{-5}$

TABLE IV

EXECUTION TIMES (S) FOR DIFFERENT NUMBER OF VERTICES AND EDGES DENSITIES

As expected the execution time growth rate is similar to the number of basic operations. It is much larger for the exhaustive search compared with the greedy approach.

We can also observe that the density of the edges does not interfere with the execution time.

It is also important to notice that in all cases (even the smaller number of vertices) the greedy approach is faster than the exhaustive search.

B.3 Configurations tested

Table V shows the number of configurations tested by the two algorithms for graphs with different number of vertices and edges densities.

n	exhaustive search			greedy		
	25%	50 %	75 %	25%	50 %	75 %
2	3	3	3	2	2	2
3	7	7	7	3	3	3
4	15	15	15	4	4	4
5	31	31	31	5	5	5
7	127	127	127	7	7	7
9	511	511	511	9	9	9
11	2047	2047	2047	11	11	11
13	8191	8191	8191	13	13	13
15	32767	32767	32767	15	15	15
17	131071	131071	131071	17	17	17
20	1048575	1048575	1048575	20	20	20

TABLE V

NUMBER CONFIGURATIONS TESTED FOR DIFFERENT NUMBER OF VERTICES AND EDGES DENSITIES

The number of configurations tested is not affected by the edge density.

For the greedy approach the number of configurations tested will be equal to the number of vertices as the algorithm iterates over them only one time.

Regarding the exhaustive search algorithm the number of combinations tested is the number of all possible combinations as explained in section II. This value grows exponentially.

V. Experimental and formal analysis comparison

Figures 3, 4 and 5 show the number of basic operations, the number of solutions tested and the execution time in a logarithmic scale for the greedy and exhaustive search approach compared with the formal analysis complexity.

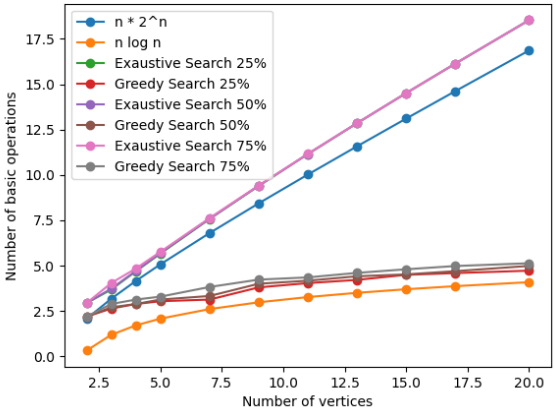

Fig. 3 - Comparison of experiment number of basic operations with formal analysis

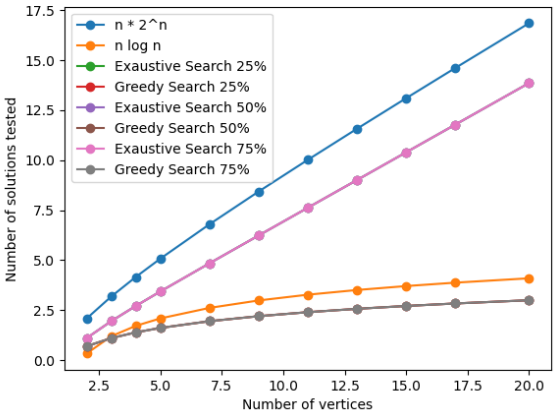

Fig. 4 - Comparison of experiment number of configurations tested time with formal analysis

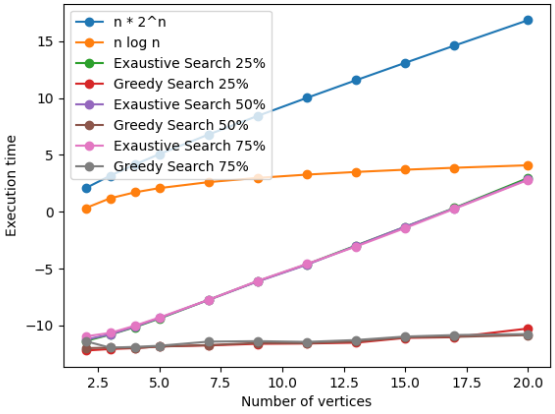

Fig. 5 - Comparison of experiment execution time with formal analysis

In all the three graphs the curves of the formal analysis and the experimental results are similar which indicates that the complexities calculated in the formal analysis describe the complexities of the algorithms developed and can be used make estimations for other number of vertices.

We can also observe that different edge densities does not have a relevant impact in the number of basic operations done or the execution time.

VI. Larger Problem Instances

With 20 vertices the exhaustive search algorithms takes around 17 seconds to find the solution while the greedy approach takes only $2.2 * 10^{-5}$ seconds, which is almost 1 million times faster.

Because of its computational complexity knowing the exact amount of times it would take for the exhaustive search algorithm to find the solution for larger number of vertices might be impossible in a lifetime, however it is possible to approximately predict it using the expression calculated in section II.

For the exhaustive search approach if 20 vertices take 17s, n vertices execution time can be approximated by the following equation:

$$execution\_time(n) = \frac{n2^n}{20 * 2^{20}} * 17 \qquad (2)$$

Similarly for the greedy approach if 20 vertices take $2.2 * 10^{-5}$s, n vertices execution time can be approximated by the following equation:

$$execution\_time(n) = \frac{nlog(n)}{20 * log(20)} * 2.2 * 10^{-5} \qquad (3)$$

Table VI shows the approximated values of execution times for large number of vertices.

n	exhaustive search	greedy
20	17 s	$2.2 * 10^{-5}$s
50	$4,46 * 10^{10}$s	$7 * 10^{-5}$s
100	$1,03 * 10^{26}$s	$1,69 * 10^{-4}$s
500	$1,03 * 10^{147}$s	$1,14 * 10^{-3}$s
1000		$2,53 * 10^{-3}$s
10000		$3,39 * 10^{-2}$s
100000		0,42s
1000000		5,07s

TABLE VI
ESTIMATED EXECUTION TIMES FOR LARGE NUMBER OF VERTICES

For only 50 vertices the exhaustive search algorithm would take approximately 1400 years to finish. Meanwhile the greedy algorithm can arrive to a solution in less than a second for 100000 vertices, making it the viable choice for larger problems.

VII. Conclusion

For smaller computational problems the exhaustive search algorithm is a simple way of achieving the correct answer. However the execution time increases exponentially, which means that they are not a viable approach for large problems. Meanwhile, while not always giving a correct answer, greedy approaches are useful when we only need an approximated answer quickly. Due to their significantly lower computational complexity they escalate much better for bigger problems.

References

[1] J.Madeira, "Ideas for the 1st project".
URL: `https://elearning.ua.pt/mod/resource/view.php?id=1003230`

[2] Anany Levitin, *Introduction to the design analysis of algorithms / Anany Levitin. — 3rd ed*, 2012.

[3] Anany Levitin, *Introduction to the design analysis of algorithms / Anany Levitin. — 3rd ed*, 2012.

[4] "Combinations".
URL: `https://en.wikipedia.org/wiki/Combination#Number_of_k-combinations_for_all_k`

[5] "Time complexity".
URL: `https://wiki.python.org/moin/TimeComplexity`