

Advanced Algorithms 1st Project

Pedro Figueiredo

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 matching 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 matching for a given undirected graph $G(V, E)$, with n vertices and m edges. A matching in G is a set of pairwise non-adjacent edges, i.e., no two edges share a common vertex.”

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

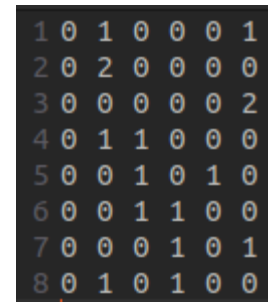
In order to do a formal computational analysis first is necessary to understand how the algorithm works.

The exhaustive search algorithm is represented by a function, which takes in 1 entry argument: a text **file**, which contains inside a representation of an **adjacency matrix**. In this matrix, the number of lines represents the

number of edges existent in the matrix, while the number of figures in a line represents the **number of vertices** in the matrix.

As can be seen in figure 1, every single digit on the matrix can be a 0, a 1, or a 2, depending on how many times a given edge touches a given vertex. As an example, line 1 column 2 of the matrix is a 1, indicating that an edge (edge 1 for ease of understanding), touches a certain vertex (vertex 2 for ease of understanding) one time.

If the number is a 2, it means the edge is a starting and ending on a given vertex.



1	0	1	0	0	0	0	1
2	0	2	0	0	0	0	0
3	0	0	0	0	0	0	2
4	0	1	1	0	0	0	0
5	0	0	1	0	1	0	0
6	0	0	1	1	0	0	0
7	0	0	0	1	0	1	0
8	0	1	0	1	0	0	0

Image 1 – Adjacency matrix example

The exhaustive research algorithm takes in every line and creates a list containing both vertices of every edge, as well as the number of edges, and flagging lone vertices (vertices not touched by an edge).

This info is fed into a second function which, for every edge, determines the number of edges that don't share a common vertex with it.

We then take the highest number we find and return as the maximum matching. In case the lone vertice flag is activated, we will add 1 to the result, as the comparison between edges always excludes an edge being compared to itself, so it does not account for a vertex with no edges.

The computational complexity of comparing the vertices of every edge with every other edge is E number of edges times $E-1$ (edges being compared to).

$$E * E-1$$

Image 2 – Computacional Complexity of edge comparing

Beyond this, we must also account for the time of translating the adjacency matrix into a list of edges and their vertices, which is E (number of edges and lines in the file) times $2V-1$ (the number of vertices times 2 to account for spaces between figures, removing 1, as the first figure has no space before).

$$E * 2V-1$$

Image 3 – Computacional
Complexity of file reading

Taking everything into account, the complexity of the algorithm is $O(E) * O(E^2) = O(E^3)$.

B. GREEDY HEURISTIC

In this algorithm, we again register the edges and their vertices, as well as the number of edges and the number of total vertices.

Afterwards, we register, for every edge, which vertices they touch.

The heuristic being used is that the vertex which the maximum matching's edge's do not touch is the vertex with the lowest number of connections.

As such we take the vertex with the lowest amount of connections and subtract that value from the lowest number of edges.

The computational complexity of the second part of the algorithm, is V (number of vertices).

When combining it with the complexity of the first part of the original algorithm (Image 3), the complexity becomes $O(V^2)$.

III. GRAPH GENERATION

In order to test the algorithms developed graphs were generated randomly using a defined seed. For each number of vertices 4 different graphs were generated with edge densities of 12,5%, 25%, 50%, and 75%.

Each graph is then stored in a file, which will contain the adjacency matrix for the given graph.

IV. EXPERIMENTS

Experiments were made for both algorithms using as number of vertices all numbers between 4 and 60. The solution found, the number of basic operations carried out, the execution time

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.

In the table I below we can analyze the differences between the maximum matchings found..

n	Edge %	Exhaustive Search	Greedy Search
5	25	3	3
10	25	10	11
25	12,5	37	38
57	75	1170	1169

Table 1 – Search results for
different experiments

As we can see, the results for the greedy search start becoming less accurate once we increase the number of vertices being used to distribute the edges.

However, the error level is very small even for large numbers of edges.

The discrepancies on the greedy algorithm happens both by over and underestimating the size of the maximum matching. That can be attributed to the discrepancies in the heuristic being used.

B. RESULTS

B.1 NUMBER OF BASIC OPERATIONS

The table below shows the number of basic operations done by the algorithms for differently sized graphs (N means number of vertices).

N	Exhaustive search				Greedy Search			
	12,5	25	50	75	12,5	25	50	75
4	2	6	12	30	1	2	3	5
5	6	12	30	72	2	3	5	8
6	6	20	72	156	2	4	8	12
8	20	56	210	462	4	7	14	21
10	42	156	552	1190	6	12	23	34
15	210	756	2862	6320	14	27	53	79
20	600	2352	9120	20592	24	48	95	143

Table 1 – Number of basic
operations for different instances

For both algorithms the number of basic operations increases with the number of vertices, and, although on a smaller scale, it also increases with the number of edges.

The increases is also much more elevated for the exhaustive search approach. This is justified by the fact that searching through every single edge leads to an exponential increase in the number of operations required.

In the greedy approach, the increase can be justified by the higher number of vertices and edges present.

Both approaches also make it clear there is a sharp increase in the number of operations required when there is an increase in density.

B.2 EXECUTION TIMES

Table 2 shows the execution time of the pair of algorithms for differently sized graphs (N means number of vertices, time is in seconds).

N	Exhaustive search				Greedy Search			
	12,5	25	50	75	12,5	25	50	75
4	$2.4 \cdot 10^{-5}$	$2.6 \cdot 10^{-5}$	$2.8 \cdot 10^{-5}$	$3.7 \cdot 10^{-5}$	$3.0 \cdot 10^{-5}$	$2.2 \cdot 10^{-5}$	$2.3 \cdot 10^{-5}$	$2.5 \cdot 10^{-5}$
5	$3.5 \cdot 10^{-5}$	$3.1 \cdot 10^{-5}$	$4.3 \cdot 10^{-5}$	$5.1 \cdot 10^{-5}$	$2.3 \cdot 10^{-5}$	$2.5 \cdot 10^{-5}$	$2.9 \cdot 10^{-5}$	$3 \cdot 10^{-5}$
6	$3 \cdot 10^{-5}$	$3.4 \cdot 10^{-5}$	$6.9 \cdot 10^{-5}$	$7.2 \cdot 10^{-5}$	$2.4 \cdot 10^{-5}$	$2.6 \cdot 10^{-5}$	$3.2 \cdot 10^{-5}$	$3.8 \cdot 10^{-5}$
8	$3.9 \cdot 10^{-5}$	$6.5 \cdot 10^{-5}$	$9.1 \cdot 10^{-5}$	$1.1 \cdot 10^{-4}$	$2.8 \cdot 10^{-5}$	$5.1 \cdot 10^{-5}$	$4.6 \cdot 10^{-5}$	$5.4 \cdot 10^{-5}$
10	$4.4 \cdot 10^{-5}$	$8.2 \cdot 10^{-5}$	$2.1 \cdot 10^{-4}$	$3.5 \cdot 10^{-4}$	$3.0 \cdot 10^{-5}$	$4.1 \cdot 10^{-5}$	$6.0 \cdot 10^{-5}$	$8.2 \cdot 10^{-5}$
15	$1.1 \cdot 10^{-4}$	$2.6 \cdot 10^{-4}$	$8.3 \cdot 10^{-4}$	$1.7 \cdot 10^{-3}$	$5.4 \cdot 10^{-5}$	$8.2 \cdot 10^{-5}$	$1.4 \cdot 10^{-4}$	$2.0 \cdot 10^{-4}$
20	$2.4 \cdot 10^{-4}$	$7.5 \cdot 10^{-4}$	$2.6 \cdot 10^{-3}$	$5.4 \cdot 10^{-3}$	$9.3 \cdot 10^{-5}$	$1.6 \cdot 10^{-4}$	$2.9 \cdot 10^{-4}$	$4.3 \cdot 10^{-4}$

Table 2 – Execution time for different instances

In a similar fashion to the number of basic operations, we also see a growth in execution time across the board.

The reasons behind the increase are also the same as previously stated: Higher number of edges justifies the increase in execution time across different densities and equal numbers of vertices, the higher number of operations required to perform an exhaustive search justify that, although having the same number of vertices and same density, it performs slower than a comparative greedy search.

B.3 CONFIGURATIONS TESTED

Table 3 shows the number of configurations tested for the pair of algorithms and differently sized graphs (N means number of vertices).

N	Exhaustive search				Greedy Search			
	12,5	25	50	75	12,5	25	50	75
4	1	2	3	5	4	4	4	4
5	2	3	5	8	5	5	5	5
6	2	4	8	12	6	6	6	6
8	4	7	14	21	8	8	8	8
10	6	12	23	34	10	10	10	10
15	14	27	53	79	15	15	15	15
20	24	48	95	143	20	20	20	20

Table 3 – Configurations tested for different instances

For the exhaustive search, the number of configurations tested correspond to the number of existing edges (because we generate the list, for every edge, of all edges without common vertices).

For the greedy search, the number is unaffected by edge density, because, for every vertice, it generates the number of edges touching it. As such, it creates a configuration for every vertice, meaning that if the

number of vertices remains unchanged, so will the number of configurations.

IV. EXPERIMENTAL AND FORMAL ANALYSIS COMPARISON

Figures 1, 2 and 3 show the number of basic operations, the execution time and the number of solutions tested (respectively) in a logarithmic scale for the greedy and exhaustive search approach compared with the formal analysis complexity.

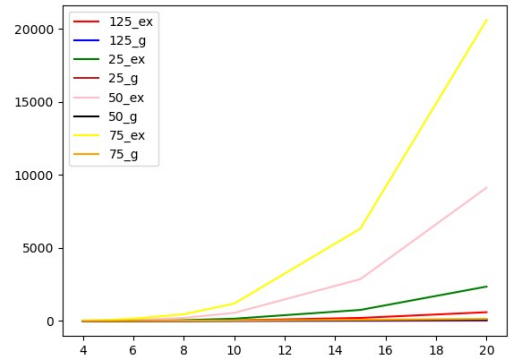


Figure 1 – Comparison of experiments' number of basic operations with formal analysis

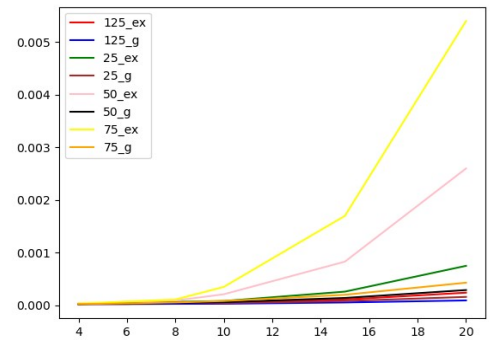


Figure 2 – Comparison of execution time with formal analysis

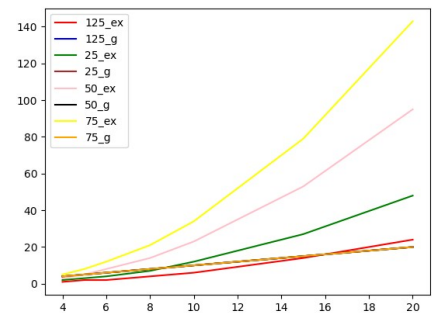


Figure 3 – Comparison of number of tested configurations with formal analysis

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 do not have a relevant impact in the number of basic operations done or the execution time.

V. LARGER PROBLEM INSTANCES

With 20 vertices the exhaustive search algorithms takes around 0.006 seconds to find the solution while the greedy approach takes only 0.0005 seconds, which is about 10 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.

Table 4 shows the approximated values of execution times for large number of vertices (time will be in seconds).

N	Exhaustive search	Greedy Search
20	$5.4 \cdot 10^{-3}$	$4.3 \cdot 10^{-4}$
50	0.25	0.006
100	4.9	0.05

As we can see, for as high as 100 vertices the values are still fairly small, standing at 5 seconds for the exhaustive search, and 0.05 for the greedy search.

We can also notice an exponential growth in search time, with a doubling in number of vertices being corresponded by an increase of search time, in the case of exhaustive search, of 20 times.

We were unable to determine an expression that corresponded to the values we received, as such the estimations for larger instances are little more than guess work.

Even so, my estimation of time spent by the algorithm, for a graph with **500** vertices, is of **4 minutes and 35 seconds**, and of **43 minutes and 20 seconds** for an instance with **1000** vertices. For the same instances using the greedy search, the prediction is of 37 seconds for greedy search, and 8 minutes for 1000, making it significantly faster and more viable for large instance problems.

VI. 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"
- [2] J.Madeira, "AA 02 Algorithm Design Strategies II"
- [3] J.Madeira, "AA 05 Algorithm Design Strategies V"