Advanced Algorithms
1st Project - Exhaustive Search

José Pedro Marta Trigo
Departamento de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro

***Abstract*—This paper presents the work done for the first assignment of the Advanced Algorithms course. The paper will analyse and compare two distinct algorithmic approaches used: exhaustive search and greedy heuristic, in order to solve the minimum weight vertex cover problem.**

I. Introduction

In graph theory, a vertex cover of a graph is a set of vertices that includes at least one endpoint of every edge of the graph [1].
This paper aims to present, analyze and compare the results of two different algorithmic approaches (exhaustive search algorithms and greedy algorithms) to solve the minimum weight vertex cover problem:

> Find a minimum weight vertex cover for a given undirected graph G(V, E), whose vertices carry positive weights, with n vertices and m edges. A vertex cover of G is a set C of vertices, such that each edge of G is incident to, at least, one vertex in C. The weight of a vertex cover is the sum of its vertices' weights. A minimum weight vertex cover is a vertex cover whose total weight is as small as possible.

Exhaustive search is a very general problem-solving technique and algorithmic paradigm that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement. Brute-force search is simple to implement and will always find a solution if it exists, implementation costs are proportional to the number of candidate solutions – which in many practical problems tends to grow very quickly as the size of the problem increases [2].

A greedy algorithm is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage. In many problems, a greedy strategy does not produce an optimal solution, but a greedy heuristic can yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time [3].

II. Formal Computational Analysis

A. Exhaustive search

The exhaustive search algorithm will loop through all vertex combinations of size [2, n] and check if the combination includes at least one endpoint of every edge of the graph. If the previous condition is true, then it will calculate the total weight for that given vertex combination and compare it with the previously stored best result.
If the current result is better than the previously stored best result, the new result is stored as the best result for future comparisons. The iteration continues until all combinations have been checked. Pseudo code for the algorithm:

Algorithm 1 Exhaustive Search

$solutionWeight \leftarrow \infty$
$solutionCombination \leftarrow \{\}$
for $iteration = 2, \ldots, n$ **do**
    $combs \leftarrow \textsc{Combinations}(n)$ ▷ Generate vertex combinations of size n
    for each combination c in $combs$ **do**
        if INCLUDESALLEDGES(c) **then**
            $score \leftarrow \textsc{SumWeights}(c)$
            if $score < solutionWeight$ **then**
                $solutionWeight \leftarrow score$
                $solutionCombination \leftarrow c$
return solutionWeight, solutionCombination

The computational complexity of iterating all the vertex combinations is the number of k combinations for all k, where k is the size of the vertex cover: O(2^k). Iterating through each combination of size n takes linear time: O(n). Therefore, the computational complexity of this algorithm is O($2^k n$).

B. Greedy heuristic

A plausible heuristic for the the minimum weight vertex cover problem is to consider the number of edges a given vertex is connected to. Vertexes with a higher number of connected edges may be considered as better candidates for a possible solution by our algorithm. Therefore, the greedy heuristic algorithm will first iterate through every edge and count the number of edges connected to each vertex. Consequently, the algorithm chooses the vertexes with the highest number of connected edges first to form a combination that includes at least one endpoint of every edge of the graph. Pseudo code for the algorithm:

Algorithm 2 Greedy Heuristic

$vertexFreq \leftarrow$ COUNTVERTEXFREQ$(edges)$
$vertexFreq \leftarrow$ SORT$(vertexFreq)$
$candidates \leftarrow \{\}$
while True **do**
    $vertex \leftarrow vertexFreq.pop()$ ▷ pop's vertex with highest number of connected edges
    $candidates \leftarrow candidates.insert(vertex)$
    if INCLUDESALLEDGES(candidates) **then**
        $score \leftarrow$ SUMWEIGHTS$(candidates)$
return score, candidates

The greedy heuristic algorithm has a computational complexity of O(n) for computing the sorted vertex frequency table and log(n) complexity for choosing the locally optimum candidate (the vertex with the highest number of connected edges) to compute a locally optimum solution. Therefore, its computational complexity is O(n log(n)).

III. Graph Generator

Since the the minimum weight vertex cover problem is a graph related problem, a custom graph generator was built in order to randomly generate graphs with the predefined seed "98597". In total, 100 different graphs where generated with a vertex count ranging between 2 and 26 and an edge density of 12.5%, 25%, 50% and 75% for each vertex count. The vertexes's weights were derived from it's distance to the origin of the plane, the pair (0,0).

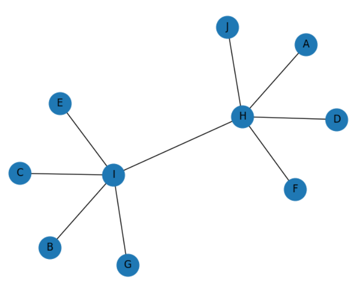

Fig. 1. Graph with 10 vertexes and 12.5% edge density

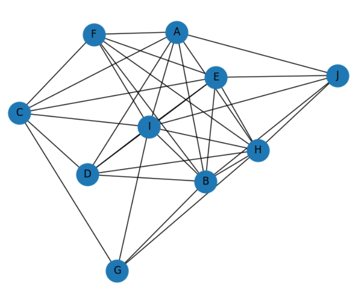

Fig. 2. Graph with 10 vertexes and 75% edge density

IV. Experiments

A. Results

Because the exhaustive search algorithm enumerates all possible candidates for the solution and checks whether each candidate satisfies the problem's statement or not, it is guaranteed to find the optimal solution if a valid one exists. Meanwhile, the greedy heuristic algorithm makes the locally optimal choice at each stage and does not guarantee an optimal solution, but can yield a locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time.

Table I displays the results of both the exhaustive search and greedy heuristic algorithms for each vertex count and edge densities:

TABLE I
Comparison of both algorithm's weight results

n	Exhaustive Search				Greedy Heuristic			
	12.5%	**25%**	**50%**	**75%**	**12.5%**	**25%**	**50%**	**75%**
2	15	15	15	15	15	15	15	15
3	15	18	15	15	15	18	15	15
4	33	30	33	30	34	34	33	34
5	15	14	33	47	15	14	33	47
6	30	30	49	66	41	34	70	74
7	41	33	47	84	41	33	55	95
8	29	29	69	92	29	33	69	106
9	44	46	78	103	44	46	78	141
10	32	42	89	121	32	42	89	142
11	37	52	106	145	37	52	116	160
12	43	72	113	136	43	83	113	151
13	35	82	139	168	35	82	158	196
14	45	94	152	184	45	129	200	236
15	29	105	177	204	29	161	244	245
16	29	107	168	221	29	164	172	242
17	39	124	212	226	39	130	234	237
18	57	145	203	245	59	223	256	270
19	67	160	213	259	82	202	236	268
20	52	151	235	261	56	187	266	305
21	65	141	215	272	81	217	260	320
22	81	186	242	276	102	283	302	306
23	118	189	278	285	199	308	336	341
24	110	224	270	316	186	311	342	343
25	128	209	289	307	196	321	375	356
26	118	219	299	323	196	319	335	372

Table II displays the vertex cover of both the exhaustive search and greedy heuristic algorithms at 75% edge density:

TABLE II
Comparison of both algorithm's combination results

n	Exhaustive Search	Greedy Heuristic
	75%	**75%**
2	{B}	{B}
3	{B}	{B}
4	{A, D}	{C, D}
5	{C, D, E}	{A, C, D}
6	{B, C, E, F}	{C, D, E, F}
7	{A, B, D, F, G}	{B, C, D, F, G}
8	{A, B, C, D, H}	{A, B, D, F, G, H}
9	{A, B, C, D, E, H}	{A, B, C, D, F, G, H, I}
10	{A, B, C, D, F, H, J}	{A, B, C, D, F, H, I, J}

B. Basic Operations

Regarding Basic Operations, both algorithms's basic operations count increases as the number of vertexes also increases. However, the increase is much heavier in the exhaustive search approach.

Table III displays the number of basic operations performed by the exhaustive search and greedy heuristic algorithms for each vertex count and edge densities:

TABLE III
COMPARISON OF BOTH ALGORITHM'S BASIC OPERATIONS

n	Exhaustive Search				Greedy Heuristic			
	12.5%	25%	50%	75%	12.5%	25%	50%	75%
2	37	37	37	37	22	22	22	22
3	112	112	112	112	36	36	36	36
4	297	297	297	334	69	69	69	78
5	780	780	815	920	64	64	102	166
6	1777	1834	1990	2347	97	105	242	228
7	4222	4222	5045	5558	125	121	191	485
8	9994	9891	11550	13123	129	129	283	558
9	22218	22406	25949	28452	149	194	557	1077
10	48730	51138	57182	60081	161	241	560	1265
11	112250	110209	122989	128172	181	276	1015	1728
12	243562	235458	256245	276251	217	372	1127	2038
13	561214	565854	569219	528061	209	441	1550	2892
14	1.2e6	1.1e6	1.1e6	1.1e6	237	667	1875	2386
15	2.7e6	2.2e6	2.4e6	2.3e6	241	1137	4221	3665
16	5.6e6	5.4e6	5e6	4.7e6	257	1590	1829	4840
17	1.3e7	1.1e7	1e7	9.6e6	290	1187	2038	4355
18	2.4e7	1.99e7	2.2e7	2e7	370	2028	4336	4465
19	5.1e7	4.4e7	4.1e7	4.1e7	554	2167	4288	5514
20	9.9e7	9.3e7	8.6e7	8.2e7	581	2292	3484	10044
21	2.4e8	1.8e8	1.7e8	1.7e8	653	4028	4692	10588
22	4.1e8	3.5e8	3.4e8	3.4e8	819	5259	8942	11698
23	9.2e8	7.5e8	7.2e8	7e8	1915	2653	10507	7150
24	1.6e9	1.45e9	1.4e9	1.4e9	2513	6590	9702	12359
25	3.8e9	3.2e9	2.9e9	2.9e9	2040	11398	10138	16835
26	8.9e9	6.1e9	5.9e9	6e9	2893	10060	8278	16113

C. Tested solutions

In the greedy heuristic approach, the number of tested solutions is always one since the solution is progressively built by always choosing the local optimum result during each iteration.

The Exhaustive Search method computes all possible vertex combinations of size n, hence why we see such a sharp increase in the number of tested solutions.

Table IV displays the number of tested solutions performed by the exhaustive search and greedy heuristic algorithms for each vertex count and edge densities:

TABLE IV
COMPARISON OF BOTH ALGORITHM'S TESTED SOLUTIONS

n	Exhaustive Search				Greedy Heuristic			
	12.5%	25%	50%	75%	12.5%	25%	50%	75%
2	3	3	3	3	1	1	1	1
3	7	7	7	7	1	1	1	1
4	15	15	15	15	1	1	1	1
5	31	31	31	31	1	1	1	1
6	63	63	63	63	1	1	1	1
7	127	127	127	127	1	1	1	1
8	255	255	255	255	1	1	1	1
9	511	511	511	511	1	1	1	1
10	1023	1023	1023	1023	1	1	1	1
11	2047	2047	2047	2047	1	1	1	1
12	4095	4095	4095	4095	1	1	1	1
13	8191	8191	8191	8191	1	1	1	1
14	16383	16383	16383	16383	1	1	1	1
15	32767	32767	32767	32767	1	1	1	1
16	65535	65535	65535	65535	1	1	1	1
17	131071	131071	131071	131071	1	1	1	1
18	262143	262143	262143	262143	1	1	1	1
19	524287	524287	524287	524287	1	1	1	1
20	1e6	1e6	1e6	1e6	1	1	1	1
21	2.1e6	2.1e6	2.1e6	2.1e6	1	1	1	1
22	4.2e6	4.2e6	4.2e6	4.2e6	1	1	1	1
23	8.4e6	8.4e6	8.4e6	8.4e6	1	1	1	1
24	1.7e7	1.7e7	1.7e7	1.7e7	1	1	1	1
25	3.4e7	3.4e7	3.4e7	3.4e7	1	1	1	1
26	6.7e7	6.7e7	6.7e7	6.7e7	1	1	1	1

D. Execution Time

Regarding execution times, the greedy heuristic has a massive advantage over the exhaustive search algorithm because it doesn't need to generate and iterate over all vertex combinations and test all possible solutions to check their result. Another important observation is that edge density does not influence execution times.

Table V displays the execution time in seconds of both the exhaustive search and greedy heuristic algorithms for each vertex count and edge densities:

TABLE V
COMPARISON OF BOTH ALGORITHM'S EXECUTION TIME IN SECONDS

n	Exhaustive Search				Greedy Heuristic			
	12.5%	25%	50%	75%	12.5%	25%	50%	75%
2	5.9e-5	5.9e-5	5.6e-5	5.6e-5	1e-4	5.6e-5	5.7e-5	5.8e-5
3	5.2e-5	4.8e-5	6.6e-5	6.6e-5	7.1e-5	4.5e-5	4.1e-5	4.1e-5
4	6.1e-5	5.9e-5	8.2e-5	9.5e-5	1e-4	4.4e-5	6.9e-5	4.2e-5
5	9.7e-5	9.6e-5	1.6e-4	1.4e-4	5.6e-5	4.1e-5	5.5e-5	4.3e-5
6	1.7e-4	1.7e-4	2.1e-4	2.4e-4	5.7e-5	4.5e-5	4.9e-5	5.2e-5
7	3.3e-4	3.3e-4	4.2e-4	4.2e-4	8.3e-5	4.7e-5	5.1e-5	6.5e-5
8	7.4e-4	6.7e-4	8.0e-4	9.1e-4	8.9e-5	7.2e-5	5.6e-5	6.5e-5
9	1.4e-3	1.4e-3	1.6e-3	1.7e-3	6e-5	8.7e-5	6.2e-5	9.4e-5
10	2.8e-3	2.9e-3	3.2e-3	3.2e-3	8e-5	9.3e-5	7.0e-5	9.4e-5
11	6.0e-3	6.0e-3	6.7e-3	6.3e-3	4.5e-5	9.2e-5	9.6e-5	1.6e-4
12	1.2e-2	1.2e-2	1.3e-2	1.3e-2	4.6e-5	6.6e-5	1.7e-4	1.2e-4
13	3.3e-2	2.5e-2	2.7e-2	2.5e-2	7e-5	6.9e-5	1.6e-4	1.6e-4
14	6.0e-2	5.2e-2	5.5e-2	4.8e-2	4.8e-5	1.5e-4	1.4e-4	1.2e-4
15	1.2e-1	1.1e-1	1.1e-1	9.5e-2	5.2e-5	1.8e-4	1.6e-4	1.4e-4
16	2.4e-1	2.0e-1	2.2e-1	2.0e-1	5.4e-5	9.6e-5	1.4e-4	1.7e-4
17	5.7e-1	4.4e-1	4.4e-1	4.0e-1	5.5e-5	9.7e-5	1.8e-4	2.1e-4
18	9.6e-1	7.5e-1	7.9e-1	7.8e-1	1.2e-4	2.0e-4	1.5e-4	2.7e-4
19	2	1.7	1.5	1.5	8-1e-5	2.1e-4	1.9e-4	3.3e-4
20	3.8	3.4	3	3.1	7.5e-5	1.3e-4	1.7e-4	4.6e-4
21	10	6.4	6.1	7.9	8e-5	1.9e-4	2.4e-4	4.1e-4
22	15.4	12.3	11.9	13.4	1.2e-4	1.8e-4	2.9e-4	4.2e-4
23	35	26.9	24.4	28	9.6e-5	6.7e-4	3.5e-4	3.1e-4
24	61.6	55.6	47	57	1.2e-4	2.2e-4	3.9e-4	4.2e-4
25	139.3	102.6	102.3	102	2.1e-4	2.1e-4	2.9e-4	4.1e-4
26	313.5	191.7	201	204	2.6e-4	2.4e-4	4.2e-4	6.3e-4

V. Experimental and Formal Analysis Comparison

Figures 3 and 4 show the execution time growth for the exhaustive search and greedy heuristic algorithms. As observed, the results validate the formal analysis.

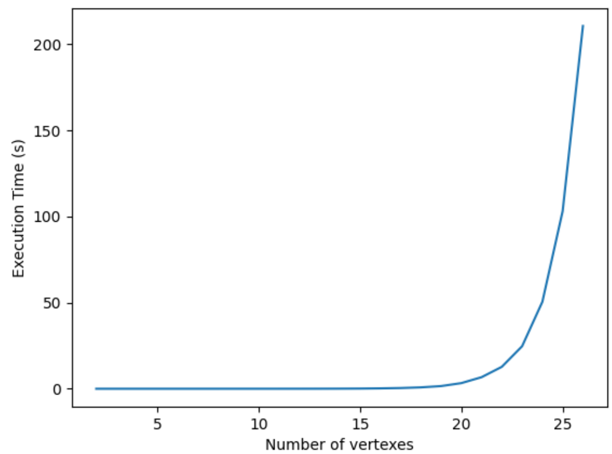

Fig. 3. Comparison of exhaustive search execution time with formal analysis

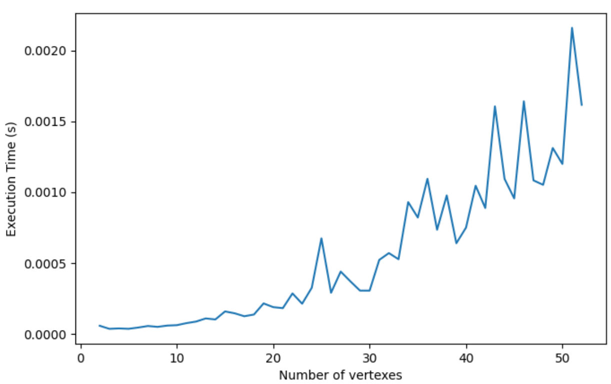

Fig. 4. Comparison of greedy heuristic execution time with formal analysis

VI. Larger Problem Instances

For the exhaustive search algorithm, the execution time for 20 vertexes is approximately 4.35457s. By knowing this we can approximate the execution time of larger instances with the following formula:

$$T(n) = \frac{2^n n}{2^{20} * 20} * 4.35457$$

The execution time of 26 vertexes using exhaustive search was 313 seconds. We can now compare this value to our approximation, using the formula:

$$T(26) = \frac{2^{26} * 26}{2^{20} * 20} * 4.35457 = 362s$$

The approximation error was 15% which already gives us a good enough estimate for larger problem instances.

For the greedy heuristic algorithm this formula can be used:

$$T(n) = \frac{n * log(n)}{20 * log(20)} * 7.73e - 05$$

VII. Conclusion

The exhaustive search algorithm is a simple implementation approach to find the optimal solution for smaller computational problems. However, the exponential increase in execution time means that they are unsuitable for large problems. On the other hand, greedy heuristic algorithms are preferred when a faster approximation is more valued than an optimal result due to their reduced computational complexity.

References

[1] ”Vertex cover” https://en.wikipedia.org/wiki/Vertex_cover
[2] ”Brute-force search” https://en.wikipedia.org/wiki/Brute-force_search
[3] ”Greedy algorithm” https://en.wikipedia.org/wiki/Greedy_algorithm