Advanced Algorithms

2nd Project - Randomized Algorithms for Combinatorial Problems

Simão Teles Arrais – 85132
Departamento de Eletronica, Telecomunicações e Informática
Universidade de Aveiro

***Abstract* - The main objective of this article is to analyse the Maximum Independent Set Problem with an randomized algorithm approach. The analysis is done for graphs of different sizes and it includes a formal analysis of the computational complexity for the created algorithm. All this is complemented with an experimental analysis based on their execution times, number of basic operations and solutions tested/found.**

I. Introduction – Maximum Independent Set

In graph theory, an independent set is a subset of vertices where no two of which are adjacent.

This paper aims to analyse the solution devised and the results it produces. It also compares some results with the ones obtained previously in the 1st Assignment.

For a given undirected graph G(V, E) with V vertices and E edges, what is the maximum independent set?

The Maximum Independent Set is a graph optimization problem and many graph optimization problems like this one have significance for Computer Science and Software Engineers since it's a problem that has correspondence in real life.

II. Description of the algorithm

The randomized algorithm devised is a simple solution for the problem, it's mainly composed by two while loops.

One to decide when to stop testing candidate solutions of a certain size and start testing larger solutions and another one to decide when to stop testing altogether.

The algorithm accepts as input the Graph and the time (in seconds) the program has to iterate through as many candidate solutions as it possibly can.

```
Algorithm 1: Randomized Maximum Independent Set
  Input  : Graph Object - graph
           Time until algorithm stops - outer_loop_time
  Output: Set containing the vertices that makes the biggest
           independent set found
tested_candidates ← []
bestsolution ← {}
while vertices available and elapsed_time < outer_loop_time do
    if graph_vertices is independent_set then
        bestsolution ← graph_vertices
        break
    comb ← combinations(candidate_size)
    inner_limit ← comb/log10(comb)
    while i < comb and i < inner_limit do
        candidate_set ← generate_candidate(candidate_size)
        while candidate_set in tested_candidates do
            candidate_set ← generate_candidate(candidate_size)
        end
        tested_candidates.append(candidate_set)
        if candidate_set is independent_set then
            bestsolution ← candidate_set
            break
    end
end
return bestsolution
```

Figure 1: Pseudo Code of Randomized Algorithm

It generates candidate solutions while it's possible to calculate a solution of a higher size or the time to run the algorithm hasn't passed.

It starts off by checking if the candidate solution to be generated is the last one, therefore, it's the biggest one which is equal to the number of existing vertices. If it's not, it will calculate all the combinations of candidate solutions for that size and a limit to know when to stop testing solutions of a certain size and start testing solutions with higher size.

To calculate that limit, the following formula was applied:

$$limit = combinations * \frac{1}{log10(combinations)}$$

Figure 2: Formula to calculate the inner loop limit

This formula was chosen because it makes use of the number of combinations for solutions of a certain size and the log function to attenuate the factorial growth. For a lower number of combinations, the algorithm will test all/majority of solutions and for a higher number of combinations it will test less, and less since it becomes more improbable to find an independent set.

The generation of candidate solutions is done while there are combinations to be tested or the limit for candidate sets tested as been met. To generate candidate sets it makes use of the *random* module *sample function.*

III. Graph Generator

For this Graph Independent Set Problem, it made use of the custom graph generator that was built in Assignment 1. The generator takes into account the chosen number of vertices, maximum number of edges that graph can have, an edge density percentage between 4 values (12,5%; 25%; 50% and 75%) and a seed which in this case is the same as the Student ID value "85132".

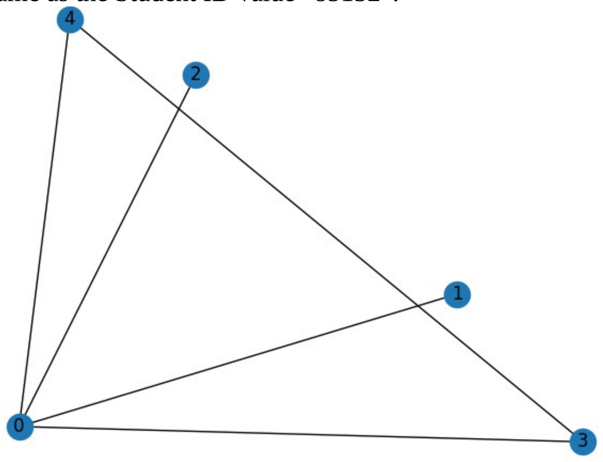


Figure 3: Graph with 5 vertices and 50% of maximum edgedensity

All vertices have coordinates randomly calculated between 1 and 100, there are no vertices that coincide nor are too close to each other (Euclidean Distance needs to be above a certain threshold) and the number of edges that share a vertex is completely randomly determined. Everything that needs to be calculated for a given choosing is completely random down to the core.

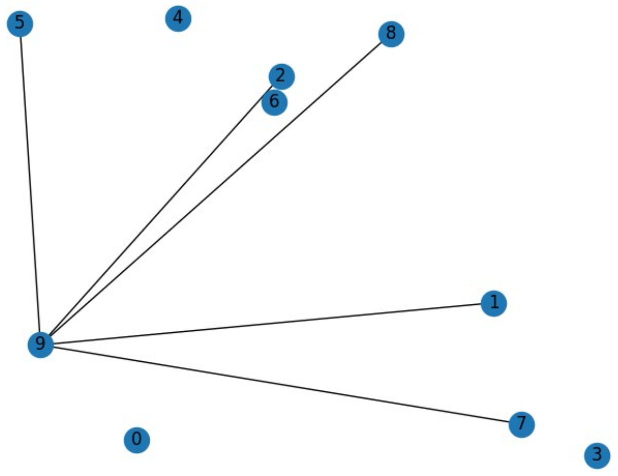


Figure 4: Graph with 10 vertices and 12.5% ofmaximum edge density

IV. Formal Computational Analysis

The randomized solution mainly performs two nested while loops. In the outer loop, the solution considers sets of vertices of increasing size and in the inner loop, the method generates a certain number of random sets of vertices of the size considered in the outer loop.

The number of sets that are tested in the inner loop is determined by the combinations for sets of that size and to calculate that, it makes use of *comb* function from *math* module which gives an *O(1)* solution, só the time complexity of the inner loop is *O(k)* where *k* is the number of vertices that the candidate sets will have.

Since the inner loop is nested within the outer loop, which has a time complexity of *O(n)*, where *n* is the number of vertices Graph. The overall time complexity of the solution is *O(n*k)*. Being the best case scenario *O(n)* when all *k* solutions were found at the first try.

This means the running time of the method increases as the number of vertices in the Graph increases and as the size of the sets being considered increases. This, of course, with the exeption of when the condition to stop running the program is met.

V. *Experiments*

To compare the results with the ones in Assignment 1, the same number of experiments were run. The solution was run with 20 Graphs across all the edge densities.

Also, the experiments were run with a maximum time of execution of 5 minutes. Any value that is not represented in the table, has surpassed the 5 minute limit.

A. Execution Time

Randomized Algorithm - Execution Time

N	**12,5%**	**25%**	**50%**	**75%**
2	9.44e-05	6.13e-05	6.56e-05	8.70e-05
3	4.01e-05	2.17e-05	5.60e-05	3.81e-05
4	5.48e-05	3.24e-05	2.43e-05	7.37e-05
5	6.34e-05	3.81e-05	5.65e-05	1.52e-04
6	3.81e-05	1.51e-04	1.70e-04	3.41e-04
7	1.10e-04	1.80e-04	3.75e-04	5.06e-04
8	9.56e-05	6.76e-04	7.06e-04	8.59e-04
9	3.18e-04	1.33e-03	1.49e-03	2.87e-03
10	1.98e-04	6.44e-04	5.09e-03	4.63e-03
11	7.28e-04	3.42e-04	6.32e-03	1.04e-02
12	3.45e-04	2.74e-03	1.04e-02	2.24e-02
13	9.78e-04	2.45e-03	3.56e-02	6.24e-02
14	1.33e-02	3.02e-02	1.23e-01	1.86e-01
15	1.61e-03	3.21e-02	2.24e-01	6.07e-01
16	2.18e-03	4.70e-02	6.13e-01	2.05e+00
17	3.32e-02	5.81e-01	1.70e+00	9.79e+00
18	3.70e-03	8.86e-01	1.35e+01	4.11e+01
19	4.28e-01	5.38e-01	1.01e+02	2.02e+02
20	1.51e-01	3.67e+00	8.37e+01	----------

Table 1: Randomized Algorithm execution time in seconds

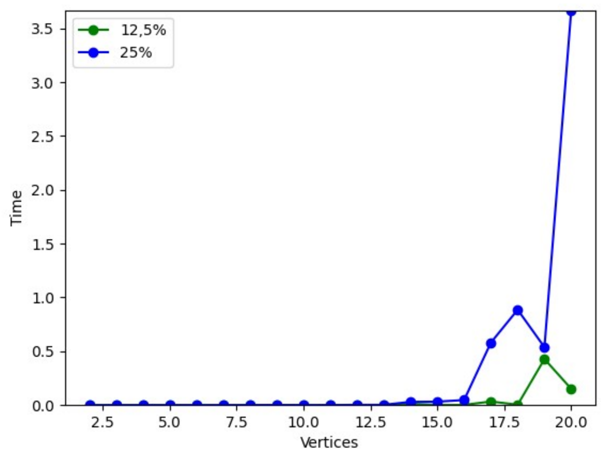


Figure 5: Time comparison plot between 12,5% and 25%

By looking at Table 1 and Figure 5 we can draw interesting conclusions.

For obvious reasons, as we start to have higher edge density, the execution time will be longer since it's less likely to generate a candidate solution that's an independent set.

It's possible to see the randomeness of the algorithm in action on Figure 5, since the execution time doesn't grow at a specific rate but rather at different rates or sometimes, for Graphs of different sizes, the execution time is even lower with bigger Graphs.

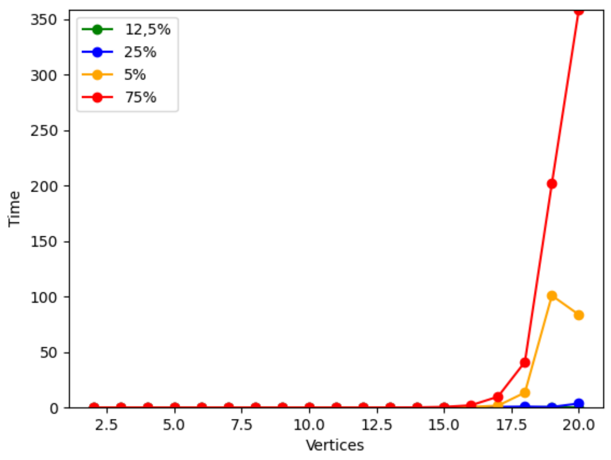
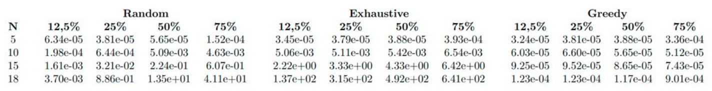


Figure 6: Time comparison plot between all edge densities

Also, it's important to point that as the edge density got higher, it seems to be less likely the execution time of the solution decreases from one Graph to the next one. That's due to the fact that when Graphs get bigger they will have a higher number of possible candidate sets and theyselves are more likely to not be independent since the edge density is high.



	Random				Exhaustive				Greedy			
N	12,5%	25%	50%	75%	12,5%	25%	50%	75%	12,5%	25%	50%	75%
5	6.34e-05	3.81e-05	5.65e-05	1.52e-04	3.45e-05	3.79e-05	3.88e-05	3.93e-04	3.24e-05	3.81e-05	3.88e-05	3.36e-04
10	1.98e-04	6.44e-04	5.09e-03	4.63e-03	5.06e-03	5.11e-03	5.42e-03	6.54e-03	6.03e-05	6.60e-05	5.65e-05	5.12e-05
15	1.61e-03	3.21e-02	2.24e-01	6.07e-01	2.22e+00	3.33e+00	4.33e+00	6.42e+00	9.25e-05	9.52e-05	8.65e-05	7.43e-05
18	3.70e-03	8.86e-01	1.35e+01	4.11e+01	1.37e+02	3.15e+02	4.92e+02	6.41e+02	1.23e-04	1.23e-04	1.17e-04	9.01e-04

Figure 7: Time comparison between algorthms

Even though the algorithms created in Assignment 1 don't have the exact same objective, they are very similar nontheless and it's valid to make a comparison between them.

The randomized algorithm has a natural advantage towards the exhaustive one but the same can't be said about the greedy algorithm. The difference between execution times on the randomized and greedy algorithm are already huge with Graphs with 18 vertices. Though, t an edge density of 12,5% the algorithms are very similiar.

B. Basic Operations

Randomized Algorithm – Basic Operations				
N	12,5%	25%	50%	75%
2	23	23	23	23
3	41	41	35	53
4	62	63	53	89
5	91	88	103	212
6	98	247	275	387
7	209	278	502	593
8	209	717	753	937
9	401	1085	1351	1853
10	399	106	3249	3335
11	766	451	4231	6038
12	504	1581	5626	9304
13	985	2152	12367	17526
14	5451	10436	26694	33567
15	1790	12745	34488	63940
16	1745	13638	59531	118936
17	11227	59488	96220	221511
18	2031	66929	209862	381838
19	46577	49046	547737	798963
20	27331	124422	416974	1014359

Table 2: Randomized Algorithm execution basic operations

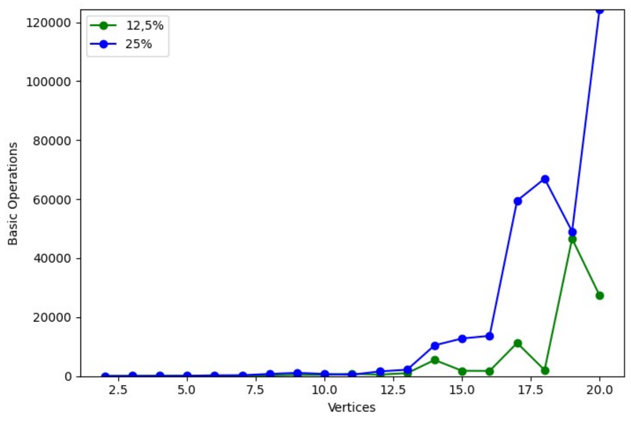


Figure 8: Basic operations comparison plot between 12,5% and 25%

As the execution time increases, naturally, the basic operations needed also increases. In fact, by analysing Figure 5 and Figure 8 or even Figure 6 and Figure 9, it's possible to see the similarity in both graphs on how they are growing/drawn. We can draw the conclusion that there is a direct correlation between them.

Just like in the Execution Times, we can see the randomeness of the algorithm in action on Figure 8, since the basic operations don't grow at a specific rate but rather at different rates or sometimes, even for Graphs of bigger sizes, the number of basic operations is lower than the Graphs of smaller sizes.

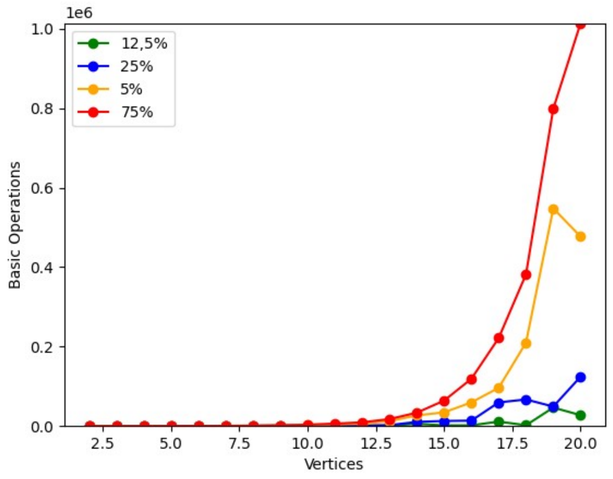
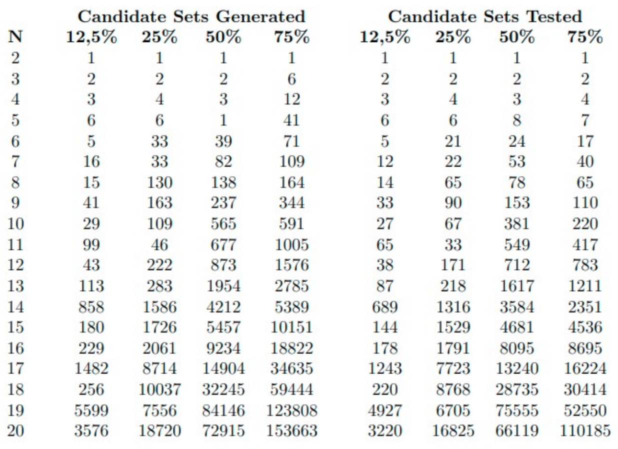


Figure 9: Basic operations comparison plot between all edge densities

C. Number of Solutions



	Candidate Sets Generated				Candidate Sets Tested			
N	**12,5%**	**25%**	**50%**	**75%**	**12,5%**	**25%**	**50%**	**75%**
2	1	1	1	1	1	1	1	1
3	2	2	2	6	2	2	2	2
4	3	4	3	12	3	4	3	4
5	6	6	1	41	6	6	8	7
6	5	33	39	71	5	21	24	17
7	16	33	82	109	12	22	53	40
8	15	130	138	164	14	65	78	65
9	41	163	237	344	33	90	153	110
10	29	109	565	591	27	67	381	220
11	99	46	677	1005	65	33	549	417
12	43	222	873	1576	38	171	712	783
13	113	283	1954	2785	87	218	1617	1211
14	858	1586	4212	5389	689	1316	3584	2351
15	180	1726	5457	10151	144	1529	4681	4536
16	229	2061	9234	18822	178	1791	8095	8695
17	1482	8714	14904	34635	1243	7723	13240	16224
18	256	10037	32245	59444	220	8768	28735	30414
19	5599	7556	84146	123808	4927	6705	75555	52550
20	3576	18720	72915	153663	3220	16825	66119	110185

Figure 10: Randomized Algorithm candidate sets generated/tested

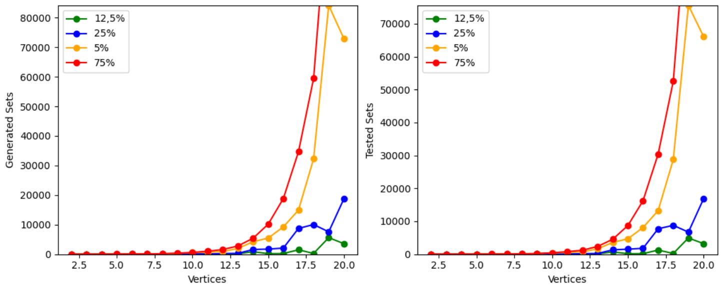


Figure 11: Candidate sets generated/tested plot between all edge densities

By analysing Figure 10 and Figure 11 it's valid to say that, once again, the correlation here in the growth rate with time excution and basic operations is massive. Being this, of course, because most of the resources and time is spent on generating candidate sets.

It's also interesting to note that the values for candidate sets generated and candidate sets tested is not that different accross the same N and edge density. This means that the solution instead of spending most of the time generating candidate sets that already have been tested, it's spending time on trying to find a set that is independent. This is specially true for Graphs with a higher number ot vertices and edge density.

D. Best Solutions

Randomized Algorithm – Size of Best Solution				
N	**12,5**	**0,25**	**0,5**	**0,75**
2	2	2	2	2
3	3	3	2	2
4	4	3	3	2
5	4	4	3	2
6	5	4	3	2
7	5	5	4	3
8	7	5	5	3
9	7	5	4	3
10	9	8	4	3
11	8	9	5	3
12	11	8	6	4
13	11	10	6	5
14	9	8	6	4
15	12	10	7	4
16	13	11	8	4
17	12	10	9	5
18	15	11	9	6
19	13	13	9	3
20	15	13	11	6

Table 3: Randomized Algorithm best solution sizes

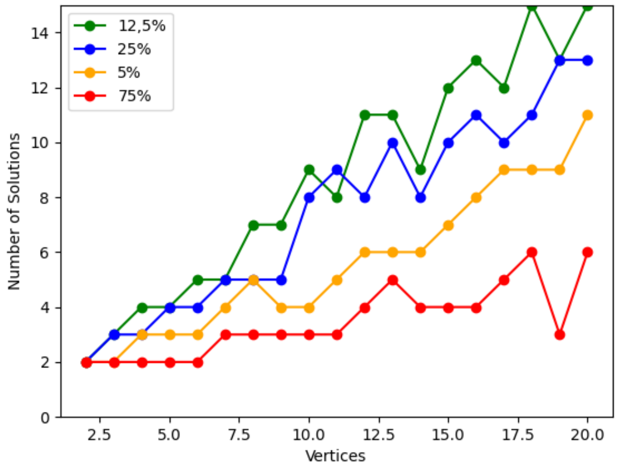


Figure 12: Best solution size plot between all edge densities

A quick at Table 3 and Figure 12 we can see that for the first time, the order of the values have changed. The bigger values are in the Graphs with smaller edge densities. This goes according to what it was expected, since, there's a much greater opportunity to find bigger independet sets in Graphs with lower edge density.

The bigger the Graph and the lower edge density is, the bigger will be the size of the Maximum Independent Set.

E. Max Graph Processed

Randomized Algorithm- Max Graph				
	12,5%	**25%**	**50%**	**75%**
N	31	25	22	20

Table 4: Randomized Algorithm biggest Graph

The Table 4 represents the biggest Graph that it was possible to process at different edge density.
As said previously, the tests were run with a maximum limit of 5 minutes. These values represent the size of the Graph that it took for the solution to surpass those 5 minutes.

VI. Conclusion

With this analysis, it can be concluded that the randomized implementation it's a middle term between the exhaustive and greedy search. All have their pros and cons. The exhaustive search is a simple approach to find an optimal solution but only feasible for smaller computational problems because of it's exponential increase in execution time. The greedy and randomized solution are more suitable for larger problems or when an approximation for the optimal solution has more value than the optimal solution itself. For smaller computational problems, the difference is negligible.

Links

https://github.com/simaoarrais/Advanced-Algorithms