Randomized algorithm for maximum matching problem

Pedro Figueiredo 97487

Abstract - **This report presents the work done on the second project of the Advanced Algorithms course. The paper will analyse and compare two different algorithmic approaches developed, exhaustive search and randomized search in order to solve a maximum matching problem.**

I. INTRODUCTION

Randomized algorithms are a type of non-deterministic algorithms, which can exhibit different behaviours, due to changes in external states (like altering random values) on different runs, and is often used to obtain results that are approximate solutions, when obtaining the actual solution proves to be costly computationally with a deterministic algorithm, like **exhaustive search**.

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."

II. FORMAL COMPUTATIONAL ANALYSIS

A. Exhaustive Search

The randomized algorithm starts with a similar flow to the one of the previously used algorithms.

Taking in an incidence matrix, it gathers the edges (with the corresponding connected vertices), as well as the number of vertices. This information is then passed to a function named monte carlo(edges and vertices, num total vertices), which contains the main algorithmic logic of the solution.

First, we create *list_of_vertices*, which will provide a list of all vertices.

Afterwards, we limit the number of matchings analyzed to 60% of the existing matchings (therefore making a smaller execution time possible).

In every execution of the loop, we select a random vertice from list_of vertices. Afterwards, we compare the vertice to the vertices of every edge. If the vertice does not match either of the other, it means that edge is part of the vertice's matrix, and we then increment a temporary variable which will contain the size of the matching.

Should the matching size stored in the temporary variable be bigger than the current maximum matching (stored in variable *max_matching*), it will override the current *max_matching* value, taking it's place.

$$E * 2V\text{-}1 + (.6*V)*E$$

Image 1 – Computational Complexity

Above we can see an equation of the computational complexity of the algorithm. Taking into account both functions, we can define the algorithm as having O(E) complexity.

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 adn the number of solutions testes were all recorded on a file named *res_random.txt.*

Comparisons with the performance of a corrected version of the exhausted search algorithm were also stored in a file named *compare.txt.*

A. Solutions Found

N	Exhaustive Search			Randomized Search		
	25	**50**	**75**	**25**	**50**	**75**
10	12	22	30	11	22	30
20	46	89	133	46	89	132
30	106	210	313	106	210	313
40	191	379	565	191	376	565
50	302	599	897	299	599	897
60	435	870	1300	435	870	1292
80	781	1552	2328	778	1552	2327
100	1227	2450	3661	1222	2450	3655

Table 1 – Solutions for different graph instances

Table 1 shows the maximum matchings found by each search method, for different numbers of vectors and edge densities.

From the 24 differents graphs whose results are on the table, we achieved the same result for 17 examples, only missing 7, with the biggest difference between being 8, for a matching of 1300 elements (60 vertices, 75% edge density).

We see a good performance by the randomization algorithm, producing results with a very small error margin.

B. Number of Basic Operations

N	Exhaustive Search			Randomized Search		
	25	50	75	25	50	75
10	130	240	350	78	144	210
20	980	1920	2880	588	1152	1728
30	3300	6570	9840	1980	3842	5904
40	7840	15640	23440	4704	9384	14064
50	15400	30700	46000	9240	18420	27600
60	26640	53160	79740	15984	31896	47844
80	63280	126480	189680	37968	75888	113808
100	123900	247600	371400	74340	148560	222840

Table 2 – Number of basic operations for different graph instances

Table 2 compares the number of basic operations required by each of the algorithms for each specific graph. As we can, the number of operations is significantly lower for the randomized search (by an average of around 40%).

<OTHER GRAPHS>

C. Execution Time

N	Exhaustive Search			Randomized Search		
	25	50	75	25	50	75
10	$1.1*10^{-4}$	$1.1*10^{-4}$	$1.4*10^{-4}$	$7.6*10^{-5}$	$1.0*10^{-4}$	$1.3*10^{-4}$
20	$3.0*10^{-4}$	$5.7*10^{-4}$	$9.8*10^{-4}$	$2.6*10^{-4}$	$4.9*10^{-4}$	$8.6*10^{-4}$
30	$8.3*10^{-4}$	$1.6*10^{-3}$	$2.7*10^{-3}$	$7.4*10^{-4}$	$1.5*10^{-3}$	$2.3*10^{-3}$
40	$1.9*10^{-3}$	$5.1*10^{-3}$	$6.8*10^{-3}$	$1.6*10^{-3}$	$3.7*10^{-3}$	$6.1*10^{-3}$
50	$3.7*10^{-3}$	$1.0*10^{-2}$	$1.4*10^{-2}$	$3.2*10^{-3}$	$8.7*10^{-3}$	$1.2*10^{-2}$
60	$6.2*10^{-3}$	$1.5*10^{-2}$	$1.9*10^{-2}$	$5.9*10^{-3}$	$1.3*10^{-2}$	$1.7*10^{-2}$
80	$1.9*10^{-2}$	$3.0*10^{-2}$	$4.7*10^{-2}$	$1.6*10^{-2}$	$2.6*10^{-2}$	$4.2*10^{-2}$
100	0,034	0,063	0,109	0,031	0,054	0,095

Table 3 – Execution time for different graph instances

Table 3 compares the execution times of both algorithms. As we see, the randomized search algorithm is around 40% faster when compared to exhaustive search.

D. Configurations / Solutions tested

N	Exhaustive Search			Randomized Search		
	25	50	75	25	50	75
10	10	10	10	6	6	6
20	20	20	20	12	12	12
30	30	30	30	18	18	18
40	40	40	40	24	24	24
50	50	50	50	30	30	30
60	60	60	60	36	36	36
80	80	80	80	48	48	48
100	100	100	100	60	60	60

Table 4 – Configurations/Solutions for different graph instances

Here we can see the different amount solutions tested in different contexts.

As we can see, the randomized search checks only 60% of the solutions of exhaustive search.

IV. Results Comparison

The figures below shows a comparison betwwen the exhaustive search and randomized search for all the previously described metrics.

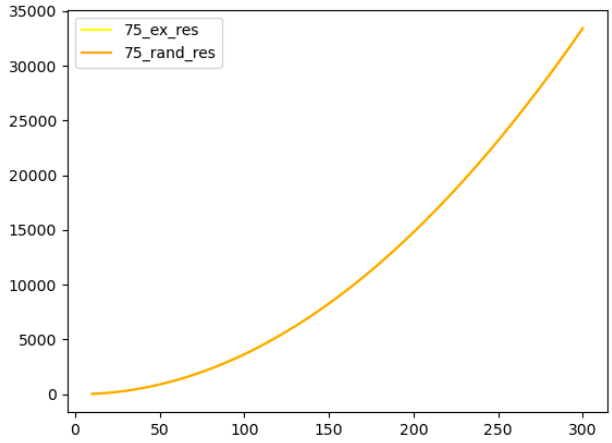
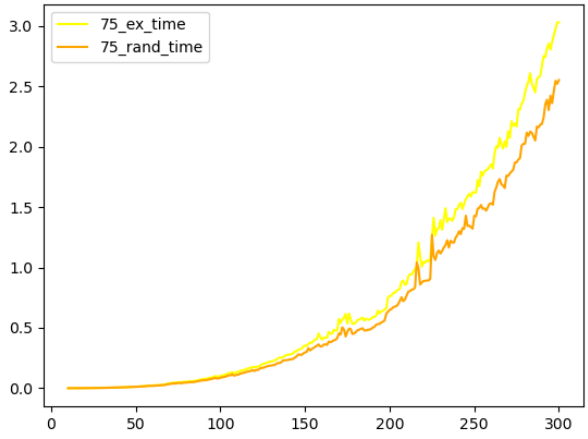
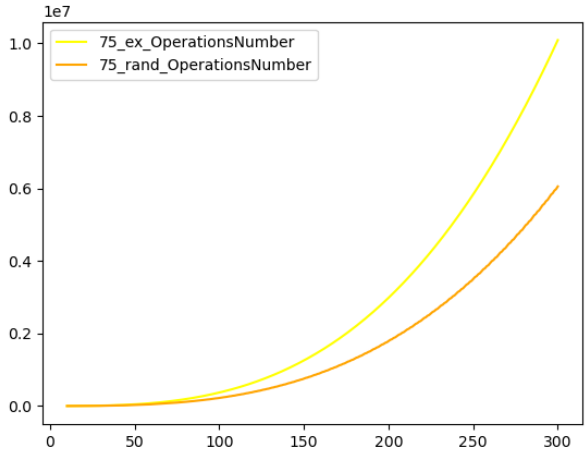


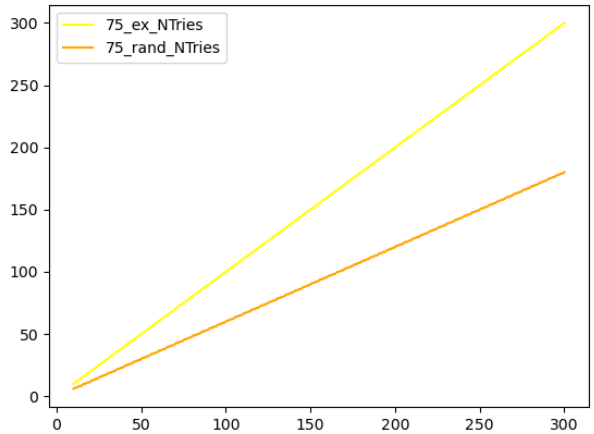
Figure 1 – Comparison of results between algorithms



times



operations



ed

The 4 figures paint a clear picture.

The randomized search algorithm is faster, testing less solutions and also performing less operations.

However, we present the same level of results, having a very high level of accuracy, as we can see from the fact the lines in figure 1 overlap.

V. Larger Instances

The biggest case we tested for was a graph with **300 vertices** and **33638 edges**. Exhaustive search got the result in about 3 seconds, while randomized search took about 2.2 seconds.

The program shows exponential growth (slightly less aggressive in it's progress for the randomized search).

Table 5 shows results for singular tests in especially larger graphs (SWm means Swmedium, a graph from the teacher suggestions) .

N	Randomized Search			
	Max_matching	**Time (s)**	**N_Operations**	**N_Solutions**
400	59605	7,27	14364240	240
500	93253	13,5	28069200	300
600	134390	24,8	48519360	360
700	183025	40,1	77065380	420
800	239172	46,9	115056480	480
SWt	12	$4.5*10^{-5}$	112	8
SWm	1271	0,02	191100	150

Table 5 – Information for different graph instances

Looking at the values, I estimate that the calculation for a graph with 1000 vertices would take a little over a minute.

Should the graph have a million vertices, I estimate that it would take around 25 hours.

REFERENCES

[1] J.Madeira, "Intro Randomized Algorithms"
[2] J.Madeira, "Intro Randomized Algorithms II"
[3] J.Madeira, "Intro Randomized Algorithms III"
[4] R.R. de Paula, "Método de Monte Carlo e Aplicações"