Randomized Minimum Edge Dominating Set

Ricardo Rodriguez (98388)
Universidade de Aveiro - DETI
AA - Algoritmos Avançados
ricardorodriguez@ua.pt

***Abstract*—This document describes the strategies, methodologies used for retrieving the minimum edge dominating set of an undirected connected graph through a randomized search, alongside with its results in comparison with the exhaustive and greedy methods.**

***Index Terms*—randomized, edge, adjacency**

I. Introduction

Graph theory problems have always gotten the interest from mathematicians, computer scientists and a variety of professionals who use it to solve problems (*e.g.* the travelling salesman problem [1]) and to help improve day-to-day tasks (*e.g.* find the shortest path between two locations).

In this report, we are going to look even further into the Minimum Edge Dominating Set problem, an NP-hard problem, and solve it using randomized search this time. The final chapter of this paper presents the final results of the randomized search and compare it according to the previous implemented search methods: exhaustive and greedy.

II. Problem

For a given undirected graph *G(V,E)*, with *n* vertices and *m* edges, a edge dominating set of *G* is a subset *D* of edges, such that every edge not in *D* is adjacent to, at least, one edge in *D*. A minimum edge dominating set is an edge dominating set of smallest possible size.

Additionally, this project requires that the undirected graph *G* is connected, but not the subset *D* of edges, and the number of edges sharing a vertex is randomly determined.

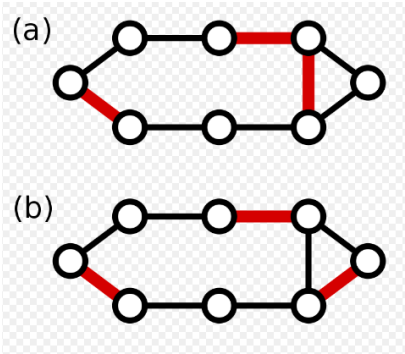

Fig. 1. Example of minimum edge dominating sets [2]

As we can see in *Fig. 1*, *(a)* and *(b)* are both minimum edge dominating sets. The edges with the color red constitute a minimum edge dominating set *D*, since the remaining edges in *G*, represented in black, are all connected to, at least, one of the edges belonging to *D*.

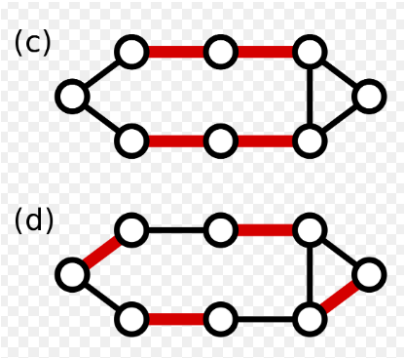

Fig. 2. Example of edge dominating sets [2]

In *Fig. 2*, *(c)* and *(d)* are also edge dominating sets. However, they are not the ones with the smallest possible size, since *(c)* and *(d)* subsets have both four edges while *(a)* and *(b)* subsets, represented in *Fig. 1*, have only three edges.

The minimum edge dominating set problem complexity augments when the number of edges in the undirected graph *G* increases. This being said, a graph *G* with 8 vertices and a 75% edge percentage has higher elapsed times to solve the problem than graph *H* with 9 vertices and a 50% edge percentage, since the number of edges in *G* is bigger than in *H*, although the number of vertices in *H* is higher.

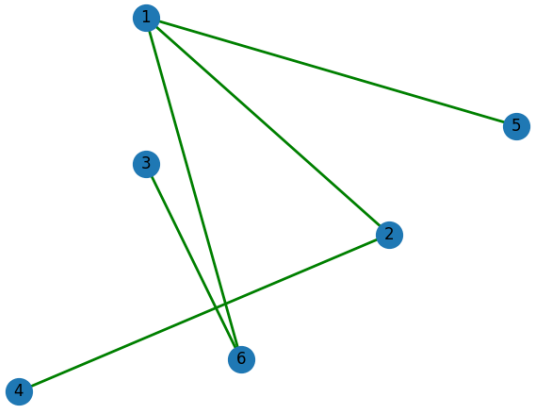

Fig. 3. Undirected graph with 6 vertices and a 25 edge percentage

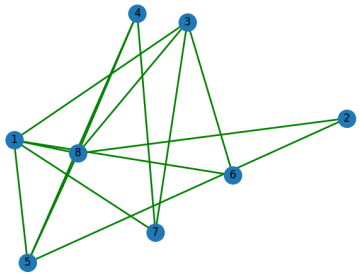

Fig. 4. Undirected graph with 8 vertices and a 50 edge percentage

III. Solution

The programming language used in this project was *Python3*. There are faster programming languages for this kind of computational problems like *C* or *C++* but, since the major goal of this project is to comprehend the difference between different algorithms and how to improve them, the programming language is indifferent.

The graph vertices have integer valued 2D coordinates between 1 and 100, a big range to facilitate the its visualization. All of them should neither be coincident nor to close to each other, having more than one unit of euclidean distance separating them from each other.

Besides that, for each number of vertices the program calculates the minimum edge dominating set with 12.5%, 25%, 50% and 75% of the maximum number of edges.

The minimum (*min*) and maximum (*max*) number of edges of an undirected connected graph with *n* vertices are described in the equations below:

$$min = n - 1 \tag{1}$$

$$max = (n * (n - 1))/2 \tag{2}$$

This being said, a graph with 8 vertices and with a 50% edge percentage is going to have 14 edges, while a graph with 4 vertices and a 25% edge percentage is going to have 3, since the percentage of edges in the graph must be bigger or equal than the minimum number of edges of an undirected connected graph (*Fig. 5)*.

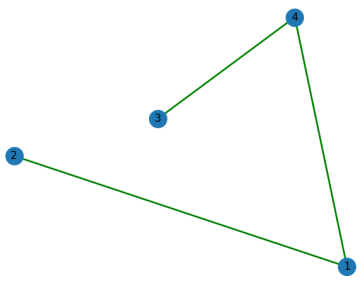

Fig. 5. An undirected graph with 4 vertices and with a 25% edge percentage (*max = ((4*3)/2)*25% = 6*0.25 = 1.5* does not satisfy the minimum number of edges requirement (*min = n-1 = 4-1 = 3*). Thus, the program must create a graph with 3 edges.

A. Adjacency List

In graph theory, an adjacency list is a collection of unordered lists used to represent a finite graph. Each unordered list within an adjacency list describes the set of adjacent neighbours of a particular vertex in the graph. [3]

On the other side, an adjacency matrix is a square matrix also used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent (1) or not (0) in the graph. [4]

Fig. 3 depicts an undirected graph, alongside with its adjacency matrix and adjacency list.

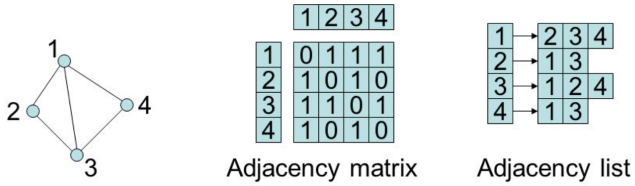

Fig. 6. Adjacency matrix and adjacency list for an undirected graph

For this problem, it's better to use an adjacency list in favor of an adjacency matrix, since it's faster to retrieve and iterate through all the edges connected to a vertex.

Using an adjacency matrix would be better in problems where we need to find if two vertices are connected with each other, which is not the case. Additionally, the space complexity is of an adjacency matrix is quadratic $O(n^2)$ whereas the space used in an adjacency list depends on the number of vertices and edges, usually saving more memory when the graph's edges are sparse.

B. Randomized Search

In computer science, a randomized algorithm uses a source of randomness to find a solution to a given problem. Although it's not as used as brute-force and greedy ones, randomized algorithms can speed up the elapsed times of a search and reduce its space complexity because it doesn't compute all possible solutions or follow a set of heuristics.

If a randomized algorithm has only one iteration it's likely to return a non-optimal solution and if it has a huge amount of iterations, it's very likely to return an optimal or even the best solution possible. However, the elapsed time of the search function increases proportionally to the number of iterations. Thus, a good randomized algorithm should have a number of iterations sufficient enough to find optimal solutions without spending too much computing time.

The best approach to the minimum edge dominating set problem using the randomized methodology is to pick a random edge from all the existing ones and remove its neighbours from the list, since they're adjacent to the chosen edge. After this, if the list of edges is empty, the program has found an edge dominating set and returns the iteration's result.

If the iteration edge dominating set has a smaller size than the current best one, it's stored as the new best feasible solution computed. After all the iterations, the randomized search algorithm returns the best computed optimal solution.

Algorithm 1 Randomized Search Iteration

```
iteration ←0
best_result ←[]
randomized_iterations ←calculate number of iterations
while iteration ≠ randomized_iterations do
  result ←[]
  cont ←true
  while cont do
    v1 ←get random vertex from adjacency_list
    v2 ←get random neighbour from v2
    edge ←(v1,v2)
    neighbours ←set(popped v1 from adjacency_list) +
    popped v2 from adjacency_list) - set(edge)
    for each item neighbour in copy of neighbours do
      if neighbour in adjacency_list and v1 ∈
      adjacency_list[k] then
        remove v1 from adjacency_list[k]
      end if
      if neighbour in adjacency_list and v2 ∈
      adjacency_list[k] then
        remove v2 from adjacency_list[k]
      end if
      if not adjacency_list[k] then
        delete k from adjacency_list
      end if
    end for
    cont ←true if any value of adjacency_list != []
    append edge to result list
  end while
  if (lenght(result) less than lenght(best_result)) or
  best_result == [] then
    best_result ← result
  end if
  iteration ← iteration + 1
end while
```

The function used to calculate the number of randomized iterations is the presented below:

Algorithm 2 Compute number of random iterations

```
edges ←list of all graph's edges
α ←length(edges)
num_edge_combinations ← 2^α − 1
iterations ← maximum value of [5, round(0.1 ×
log10(num_edge_combinations))]
```

It's important to have a minimum number of random iterations (*e.g* 5) to find an optimal solution when the number of edges is very low. However, that number is not sufficient to find optimal solutions for bigger graphs and it needs to be computed according to the number of edges in the graph. Because of this, using 10% of the base 10 logarithm of the number of edge combinations (*num_edge_combinations*) produces a number good enough to test different dominating set combinations and find an optimal solution to the problem without spending too much time.

By performing a formal analysis on the randomized algorithm, we can verify that the computational complexity of the search is equal to the number of randomized iterations needed to perform the algorithm, which is $O(0.1 \times \log(2^n))$ = $O(\log(2^n))$ = $O(n \times \log(2))$ = $O(n)$. Each iteration is being executed while the list of untouched vertices is not empty, meaning a edge dominating set solution was not found yet. This operation has a $O(n)$ complexity and it has a step which copies a list of the adjacency list keys that also has $O(n)$ complexity. Thus, the computational complexity of all the algorithm is $O(n) \times O(n \times n)$ = $O(n^3)$, being *n* the number of edges in the graph.

IV. Results

Fig. 7 presents an example of a mininum edge dominating set computed by the randomized search method.

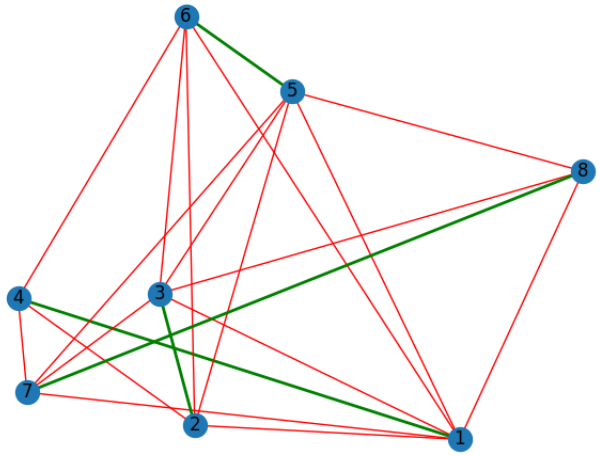

Fig. 7. Minimum edge dominating set (*in green*) of an undirected graph with 8 vertices and a 75% edge percentage computed by the randomized algorithm

The following chapters will present the results between the basic operations, solution length, elapsed time and candidate solutions of the exhaustive, greedy and randomized algorithm, alongside with a discussion of their performance and how they compare with each other.

A. Basic Operation Comparison

Table I shows the number of basic operations done by the three different algorithms for graphs with 75% edge percentage.

As we can see, the number of basic operations on exhaustive and randomized search algorithms grow exponentially while greedy search grows in a linear way (half of the number of vertices in the graph).

B. Solution Length

Table II shows the length of the minimum edge dominating sets computed by the three different algorithms, for graphs with 75% edge percentage.

As expected, exhaustive search guarantees that the computed solution is the best one (with the smaller size). However,

TABLE I
NUMBER OF BASIC OPERATIONS

Vertices	Exhaustive Search	Greedy Search	Randomized Search
5	18	2	45
10	35376	5	89
14	58553054	7	133
30		15	602
50		25	2804
75		37	9383
100		50	22404
125		62	43567
150		75	75572

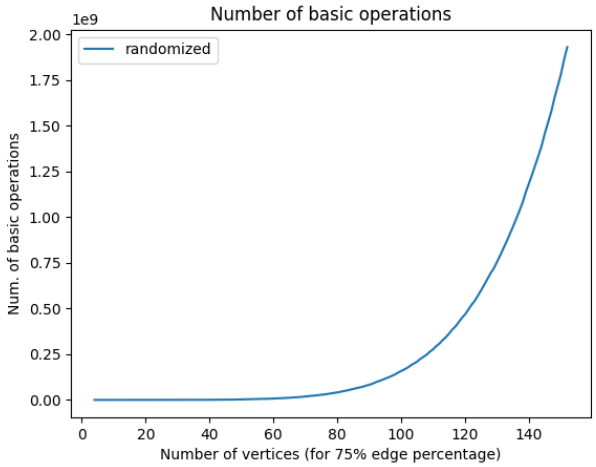

Fig. 8. Basic operations

we can see that the randomized algorithm typically has a better solution than the greedy algorithm and its solutions have the same size as the exhaustive algorithm's solutions. Thus, the randomized search provides an optimal, or even the best, solutions for a given graph.

C. Tested Solutions/Configurations

Table III shows the number of candidate solutions processed by the three different algorithms, for graphs with 75% edge percentage.

The number of computed solution candidates grows exponentially for the exhaustive search as the number of possible edge combinations also increases. The same happens for the randomized search, although in a very slow pace, because the algorithm has to eliminate the neighbours of each random chosen edge and this iterations take more time as the graph expands. Finally, the greedy search has only one computed solution, as expected.

TABLE II
MIN. EDGE DOMINATING SET LENGTH (THE LESS THE BETTER)

Vertices	Exhaustive Search	Greedy Search	Randomized Search
5	2	2	2
10	4	5	4
14	6	7	6
30		15	14
50		25	24
75		37	36
100		50	49
125		62	61
150		75	74

TABLE III
TESTED SOLUTIONS/CONFIGURATIONS

Vertices	Exhaustive Search	Greedy Search	Randomized Search
5	14	1	5
10	10647	1	5
14	11794293	1	5
30		1	10
50		1	28
75		1	63
100		1	112
125		1	175
150		1	252

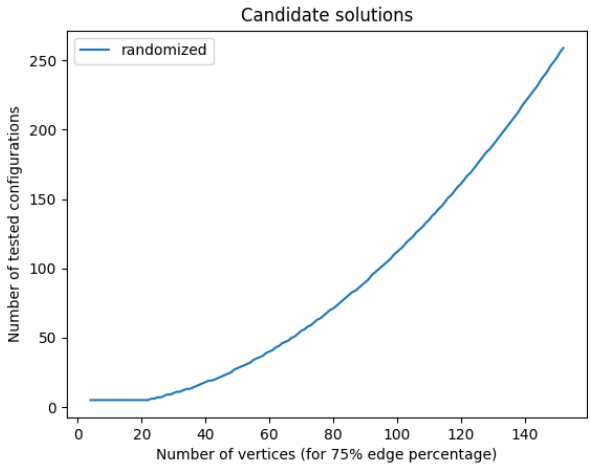

Fig. 9. Basic operations

D. Elapsed Times

Table IV presents the computational time, in seconds, to find a minimum edge dominating set for each one of the three different algorithms, for graphs with 75% edge percentage.

TABLE IV
ELAPSED TIMES (IN SECONDS)

Vertices	Exhaustive Search	Greedy Search	Randomized Search
5	3.69×10^{-5}	5.51×10^{-5}	0.0001
10	0.1216	9.46×10^{-5}	0.0003
14	209.56	0.0007	0.0005
30		0.0076	0.0052
50		0.0495	0.0276
75		0.2807	0.1591
100		1.124	0.5692
125		3.034	1.3996
150		7.3556	3.3380

Analysing Table IV, we can verify that all algorithms grow in a exponential manner but with very different paces. Exhaustive search has the worst elapsed time, since it's computing all possible combinations for bigger graphs. Greedy search is

the second slowest algorithm, it's better than the exhaustive search but takes longer time than the randomized algorithm, since the greedy algorithm has to follow a set of heuristics that consume more time than the randomness nature of the randomized algorithm.

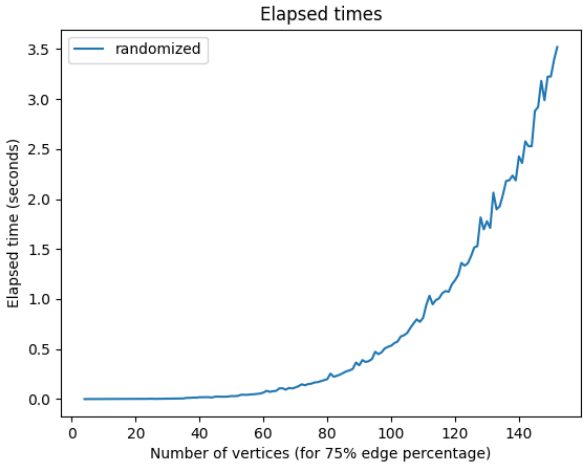

Fig. 10. Basic operations

E. Benchmarking Results

The results of the randomized search algorithm for the pointer instances provided by the professor for benchmarking purposes are displayed in Table V.

TABLE V
SEARCH METHODS RESULTS ON WEB GRAPH INSTANCES

File	Vertices	Operations	Solution Size	Elapsed Time (s)
SWtinyG.txt	13	101	4	0.00036
SWmediumG.txt	250	17878	115	0.13076

V. CONCLUSION

According to the obtained results, we can conclude that a randomized algorithm should always be taken for consideration when implementing a search algorithm for a given problem, since it's faster than other common algorithms, such as the exhaustive and greedy ones, and typically computes an optimal solution (or even the best one) when the number of random iterations is decent.

Randomized algorithms' biggest advantage is that it can calculate optimal solutions for bigger problems in a small fraction of time, since its computational complexity is lower than other search methodologies.

REFERENCES

[1] "Discrete mathematics:" Traveling Salesman Problems. [Online]. Available: https://www.jiblm.org/mahavier/discrete/html/chapter-6.html. [Accessed: 31-Oct-2022].
[2] Edge dominating set (2021) Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Edge_dominating_set (Accessed: October 31, 2022).
[3] Adjacency list (2022) Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Adjacency_list (Accessed: October 31, 2022).
[4] Adjacency matrix (2022) Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Adjacency_matrix (Accessed: October 31, 2022).
[5] Fundamentals of data structures- graph. Available at: https://bournetocode.com/projects/AQA_A_Theory/pages/graph.html (Accessed: October 31, 2022).