Minimum Weight Dominating Set

Manuel Gomes, 88939

Abstract **–This report present four algorithms to solve a minimum weight dominating set graph problem. The first one is an exhaustive search algorithm, while the rest are based on greedy heuristics. A formal analysis is presented for each algorithm. After their implementation in Python 3, an experimental analysis of each algorithm is described.**

Resumo **–Este relatório apresenta quatro algoritmos com o objetivo resolver o problema de grafos "minimum weight dominating set". O primeiro é um algoritmo de procura extensiva, enquanto que o resto são baseados em heurísticas vorazes. Uma análise formal é apresentada para cada algoritmo. Após a sua implementação em Python 3, uma análise experimental de cada algoritmo é descrita.**

I Introduction

Consider a finite, undirected graph $G(V, E)$, where V is the set of graph vertices and E is the set of graph edges. Two vertices u, v of G connected by edge (u, v) are called adjacent nodes. Two edges which share a vertex are also called adjacent. One edge *dominates* its adjacent. A set of edges M of G is called an *edge dominating set* if all edges of set $E - M$ are adjacent, and thus dominated, by the edges of M [1]. The weight of an edge dominating set is the sum of its edges' weight. A *minimum weight edge dominating set* is an edge dominating set whose total weight is as small as possible.

The objective behind this report is to apply exhaustive search and greedy algorithms to retrieve the minimum weight edge dominating set for a general graph G. The report is divided in five section: the first (section I), where the problem is introduced; the second (section II), where the algorithms used are described; the third (section III), where a formal analysis for the algorithms is presented; the fourth (section IV), where results for experiments conducted with the algorithms are detailed; and the fifth (section V), where conclusions are extracted.

II Algorithms Used

To retrieve the minimum weight dominating set of a graph G, two different types of algorithm were used. The first algorithm was an exhaustive search algorithm, while the rest followed a greedy heuristics.

II-A Exhaustive Search

Exhaustive search is a brute-force approach to combinatorial problems that consists of generating every element of the problem domain, verify if it fulfils a specific condition and then finding a desired element [2].

The exhaustive search used for the problem at hand is based on generating every possible combination of edges of the graph. Then, for every single combination, verify if it is a dominating set. If so, compute the sum of weights of every edge that belongs to the combination. Then, compare that sum with the current minimum weight. If the sum is smaller than the minimum, the combination is a better solution than the current minimum set. Therefore, the combination at hand is the now the current best solution and the sum of the weights is the current minimum weight. This algorithm is better illustrated in Algorithm 1.

Algorithm 1 Exhaustive search algorithm

```
Inputs:
    $G(V, E) \leftarrow$ graph with set of vertices $V$
    and set of edges $E$
Initialize:
    $l_c \leftarrow$ list of every combination of edges in
    $E$
    $w_{min} \leftarrow \infty$
    $set_{min} \leftarrow [\cdot]$
for $c$ in $l_c$ do
    if $c$ is dominating set of $G(V, E)$ then
      $w_c = \sum$ weight of edges in $c$
        if $w_c < w_{min}$ then
            $w_{min} \leftarrow w_c$
            $set_{min} \leftarrow c$
        end if
    end if
end for
```

II-B Greedy Heuristics

A greedy approach is a general design technique applicable to optimization problems. It consists a solution through a set of steps, expanding a partially constructed solution along them. On each step, a choice needs to take place. That choice needs to be: feasible, so it satisfies the problem's constraints; locally optimal, i.e., it has to be the best possible choice among all feasible choices available; and irreversible, i.e., the choice, once made, cannot be changed [2].

For the current problem, three different greedy heuristics were developed: minimum weight, maximum connection and one based on the work of Chaurasia and Singh [3].

In the first one, edges of the graph are sorted in ascending order by weight. Then, the edge with the least weight is added to the solution set. The solution set is checked to verify if it is a dominating set. If so, the solution is found and the algorithm stops. If not, add the next edge to the solution edge. The algorithm is described in Algorithm 2.

Algorithm 2 Minimum weight greedy heuristics

```
Inputs:
    $G(V,E) \leftarrow$ graph with set of vertices $V$
    and set of edges $E$
Initialize:
    $l_{E,W} \leftarrow$ list of $E$ with corresponding set
    of weights $W$
    $w_{min} \leftarrow 0$
    $set_{min} \leftarrow [\cdot]$
$l_{E,W-sorted} = l_{E,W}$ sorted in ascending order by weight
for $edge,\ weight$ in $l_{E,W-sorted}$ do
    $set_{min} \leftarrow$ add $edge$
    $w_{min} += weight$
    if $set_{min}$ is dominating set of $G(V,E)$ then
        break
    end if
end for
```

The second one is similar to the first, with the difference lying in the sorting step. Instead of sorting in ascending order by weight, this heuristics sorts the edges in descending order by number of adjacent edges, as seen in Algorithm 3.

Algorithm 3 Maximum connection greedy heuristics

```
Inputs:
    $G(V,E) \leftarrow$ graph with set of vertices $V$
    and set of edges $E$
Initialize:
    $l_{E,W,NA} \leftarrow$ list of $E$ with corresponding
    sets of weights $W$ and of the number of
    adjacent edges $NA$
    $w_{min} \leftarrow 0$
    $set_{min} \leftarrow [\cdot]$
$l_{E,W,NA-sorted} = l_{E,W,NA}$ sorted in descending order
by number of adjacent edges
for $edge,\ weight,\ n_{adjacent}$ in $l_{E,W,NA-sorted}$ do
    $set_{min} \leftarrow$ add $edge$
    $w_{min} += weight$
    if $set_{min}$ is dominating set of $G(V,E)$ then
        break
    end if
end for
```

The third greedy algorithm is based on the work by Chaurasia-Singh [3]. In this algorithm, a weight ratio for a certain edge is calculated by dividing the sum of the weights of its adjacent edges by the weight of the edge itself. The edges are then sorted in descending order by their weight ratio. The algorithm is exemplified in Algorithm 4.

Algorithm 4 Chaurasia-Singh greedy heuristics

```
Inputs:
    $G(V,E) \leftarrow$ graph with set of vertices $V$
    and set of edges $E$
Initialize:
    $l_{E,W,W_A} \leftarrow$ list of $E$ with corresponding
    sets of weights $W$, and of weight of
    adjacent edges $W_A$
     $w_{min} \leftarrow 0$
     $set_{min} \leftarrow [\cdot]$
     $l_{W_r \leftarrow [\cdot]}$
for $edge,\ weight,\ weight_{adjacents}$ in $l_{E,W,W_A}$ do
    $W_{ratio} = weight_{adjacents}/weight$
  $l_{W_r} \leftarrow$ add $W_{ratio}$
end for
$l_{E,W,W_A,W_r} \leftarrow [l_{E,W,W_A},\ l_{W_r}]$
$l_{E,W,W_A,W_r-sorted} = l_{E,W,W_A,W_r}$ sorted in descending order by weight ratio $w_r$
for $edge,\ weight,\ weight_{adjacent}, weight_{ratio}$ in $l_{E,W,W_A,W_r-sortedE-W-Asorted}$ do
    $set_{min} \leftarrow$ add $edge$
    $w_{min} += weight$
    if $set_{min}$ is dominating set of $G(V,E)$ then
        break
    end if
end for
```

III Formal Analysis

This section will be focused on performing a formal analysis for the algorithm described in section II. This formal analysis will consist of presenting their basic operations, defining a closed formula for the number of basic operations, their order of growth, and the worst and best cases.

For the exhaustive search algorithm, the basic operation defined is verifying if a set is a dominating set, due to its repetitiveness and computational demand. This operation takes place for every combination of edges in a graph. For a generic graph $G(V,E)$ with m vertices and n edges, the combination of edges can be given by a set of binary number with n elements. Every binary number corresponds to a certain edge. When it is zero, the edge is not present in the combination. When it is one, the edge is present in the combination. This concept is better illustrated by Equation 1 and Equation 2.

$$edges = [e_1,\ e_2,\ ...,\ e_(n-1),\ e_n] \tag{1}$$

$$combination = [0,\ 1,\ ...,\ 1,\ 0],\ \text{with length } n \tag{2}$$

So, if every combination can be defined by a set of binary numbers with length n, it can be concluded that the closed formula for the number of basic operations is given by Equation 3.

$$T(n) = 2^n \tag{3}$$

Therefore, we conclude the order of magnitude of the algorithm is exponential, as seen in Equation 4.

$$O(n) = 2^n \tag{4}$$

As the algorithm always needs to verify every single combination, the best and worst case scenario are given by Equation 5 and Equation 6.

$$W(n) = 2^n \tag{5}$$

$$B(n) = 2^n \tag{6}$$

For the greedy heuristics, the basic operation defined is the sorting of the lists of edges, due to behind the most time consuming operation. The sorting algorithm used was the `sort()` function in Python 3, which according to documentation [4], has a order of magnitude given by Equation 7.

$$O(n) = nlog_2(n) \tag{7}$$

Regarding the best and worst case scenario, the algorithm can found an adequate solution after one iteration or after iterating for every edge. Therefore, the best and worst cases can be defined by Equation 8 and Equation 9.

$$W(n) = n \tag{8}$$

$$B(n) = 1 \tag{9}$$

IV Results

After implementing the algorithms described in section II, experimental results were retrieved. These results range from number of operation, elapsed time and relative error. The results were taken in a machine with the AMD Ryzen 5 5600X processor and implemented in Python 3.

For the exhaustive search algorithm, the results obtained are presented in Table I, Figure 1, and Figure 2. In Table I, V and E are the number of vertices and edges of the graph; p is the fraction between E and the maximum number of edges; execution time is the time elapsed during the computation of the algorithm; number of basic operations is how many basic operations (defined in section III) took place; number of dominating sets is the number of sets generated that were dominating sets; and the time per operation is the estimated elapsed time for a basic operation. From this data we can verify that the number of basic operations presents an exponential order of growth ($1^{0.693x} = 2^x$, $R^2 = 1$, Figure 1), following the behavior expected by Equation 3 and Equation 4. The execution time also presents an exponential growth ($2.74e\text{-}5^{0.712x}$, $R^2 = 1$, Figure 2) with the number of edges. The time per basic operation in seconds is $3,71e\text{-}5 \pm 8,36e\text{-}6$. From these experiences, the largest graph able to be computed was one with 9 vertices and 23 edges (Figure 3). Equation 10 is used to calculate how long a computation takes using the same hardware.

TABLE I
Results from exhaustive search algorithm

V	p	E	Execution Time (s)	# of Basic Operations	# of Dominating Sets	Time per Operation (s)
4	0,125	4	7,89E-04	16	13	5,26E-05
4	0,25	6	2,68E-03	64	57	4,26E-05
4	0,5	6	2,57E-03	64	57	4,07E-05
4	0,75	6	2,76E-03	64	57	4,38E-05
5	0,125	4	6,00E-04	16	11	4,00E-05
5	0,25	6	3,22E-03	64	49	5,11E-05
5	0,5	7	5,12E-03	128	107	4,03E-05
5	0,75	8	1,06E-02	256	227	4,15E-05
6	0,125	5	1,22E-03	32	21	3,93E-05
6	0,25	8	6,64E-03	256	203	2,61E-05
6	0,5	12	1,74E-01	4096	3835	4,24E-05
6	0,75	13	2,33E-01	8192	7823	2,85E-05
7	0,125	6	1,59E-03	64	33	2,53E-05
7	0,25	10	2,76E-02	1024	871	2,70E-05
7	0,5	15	1,21E+00	32768	31017	3,71E-05
7	0,75	18	1,14E+01	262144	256281	4,34E-05
8	0,125	8	6,32E-03	256	159	2,48E-05
8	0,25	12	1,15E-01	4096	3407	2,80E-05
8	0,5	18	9,86E+00	262144	247957	3,76E-05
8	0,75	23	3,51E+02	8388608	8274471	4,19E-05
9	0,125	10	2,63E-02	1024	731	2,57E-05
9	0,25	17	4,03E+00	131072	118053	3,07E-05
9	0,5	23	3,61E+02	8388608	8064383	4,30E-05

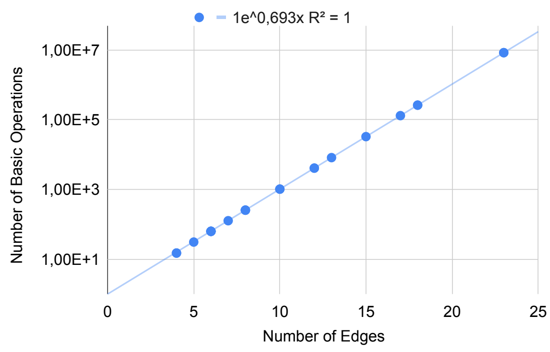

Fig. 1
Correlation between number of edges and number of basic operation for exhaustive search. Logarithmic scale.

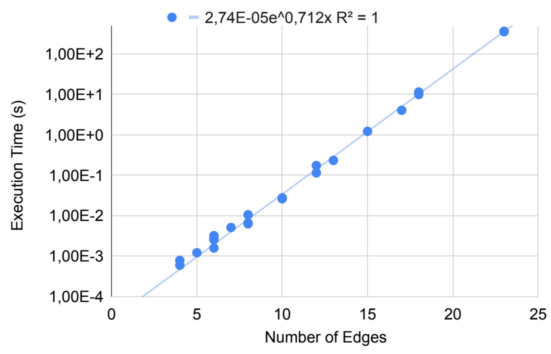

Fig. 2
Correlation between number of edges and elapsed time for exhaustive search. Logarithmic scale.

$$time = T(n) * 3,71e\text{-}5 = 2^n * 3,71e\text{-}5 \tag{10}$$

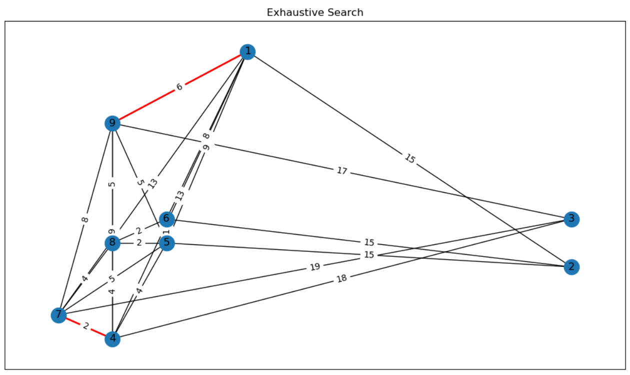

Fig. 3

Largest graph able to be processed. 9 nodes and 23 edges.

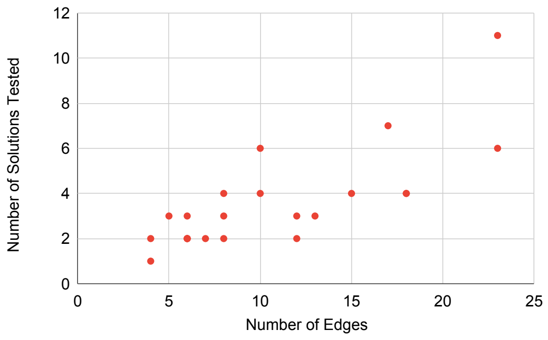

Fig. 4

Correlation between number of edges and number of solutions tested for minimum weight greedy algorithm.

For the minimum weight greedy heuristics, the results obtained are presented in Table II, Figure 4, Figure 5, and Figure 6. The columns for Table II are equal to the ones in Table I, with the exception of: number of solutions tested, which is the number of sets tested before a dominating one was formed; and accuracy ratio, which is a measure used to compare the weight of the solution obtained with the weight of the optimal solution. This data verifies that the number of solutions tested grows with the number of edges. The elapsed time increases with the number of edges. From the accuracy ratio, it can be inferred that the algorithm is relatively accurate in smaller graph, becoming less accurate in larger graphs.

TABLE II

Results from minimum weight greedy algorithm

V	p	E	Execution Time (s)	# of Solutions Tested	Accuracy Ratio
4	0.125	4	1,05E-04	2	1,80
4	0.25	6	1,03E-04	2	1,00
4	0.5	6	9,78E-05	2	1,00
4	0.75	6	1,15E-04	2	1,00
5	0.125	4	7,20E-05	1	1,00
5	0.25	6	1,05E-04	2	1,00
5	0.5	7	9,78E-05	2	1,00
5	0.75	8	1,08E-04	2	1,00
6	0.125	5	9,58E-05	3	1,50
6	0.25	8	1,03E-04	4	1,61
6	0.5	12	1,05E-04	3	1,00
6	0.75	13	9,87E-05	3	1,00
7	0.125	6	8,51E-05	3	1,00
7	0.25	10	1,11E-04	4	1,60
7	0.5	15	1,78E-04	4	1,56
7	0.75	18	2,03E-04	4	1,27
8	0.125	8	8,85E-05	3	1,00
8	0.25	12	8,03E-05	2	1,00
8	0.5	18	1,34E-04	4	1,31
8	0.75	23	3,10E-04	6	1,75
9	0.125	10	1,48E-04	6	2,13
9	0.25	17	1,96E-04	7	2,56
9	0.5	23	5,75E-04	11	4,44

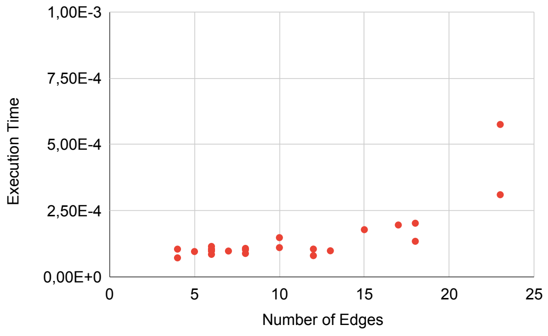

Fig. 5

Correlation between number of edges and elapsed time for minimum weight greedy algorithm.

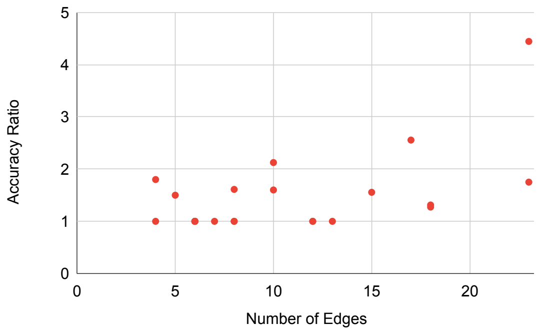

Fig. 6

Correlation between number of edges and accuracy ratio for minimum weight greedy algorithm.

For the maximum connection greedy heuristics, the results obtained are presented in Table III, Figure 7, Figure 8, and Figure 9. The columns for Table III are equal to the ones in Table II. The data shows that the number of solutions tested grows with the number of edges. However, this growth is smaller than the previous heuristics. The elapsed time increases with the number of edges, very similarly to the previous heuristics. With the accuracy ratio, it can be stated that the algorithm, although not completely inaccurate, is not particularly accurate in smaller graphs. However, the inaccuracy does not seem to grow, at least not in a noticeable rate, for larger graphs.

TABLE III
Results from maximum connection greedy algorithm

V	p	E	Execution Time (s)	# of Solutions Tested	Accuracy Ratio
4	0.125	4	1,54E-04	1	1,00
4	0.25	6	2,10E-04	2	1,13
4	0.5	6	2,15E-04	2	1,13
4	0.75	6	2,14E-04	2	1,13
5	0.125	4	1,48E-04	1	1,00
5	0.25	6	2,12E-04	2	2,00
5	0.5	7	2,13E-04	2	1,00
5	0.75	8	2,28E-04	2	1,08
6	0.125	5	1,73E-04	2	1,69
6	0.25	8	1,80E-04	3	1,89
6	0.5	12	2,17E-04	3	1,36
6	0.75	13	2,31E-04	3	2,14
7	0.125	6	1,69E-04	3	1,58
7	0.25	10	2,03E-04	2	1,80
7	0.5	15	3,58E-04	3	2,11
7	0.75	18	2,93E-04	3	1,27
8	0.125	8	1,61E-04	2	1,35
8	0.25	12	2,19E-04	3	3,27
8	0.5	18	2,86E-04	4	2,06
8	0.75	23	4,77E-04	3	1,19
9	0.125	10	2,01E-04	3	1,63
9	0.25	17	2,55E-04	3	1,44
9	0.5	23	6,18E-04	4	3,22

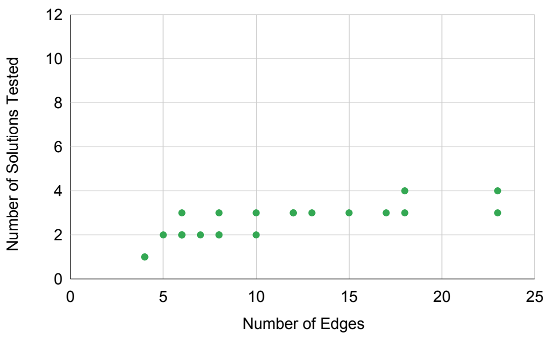

Fig. 7
Correlation between number of edges and number of solutions tested for maximum connection greedy algorithm.

For the Chaurasia-Singh greedy heuristics, the results obtained are presented in Table IV, Figure 10, Figure 11, and Figure 12. The columns for Table IV are equal to the ones in Table II. The data shows that the number of solutions tested grows with the number of edges, similar to both previous heuristics. The elapsed time increases with the number of edges, very similarly to both previous heuristics. The accuracy ratio for this algorithm is smaller then the two previous ones.

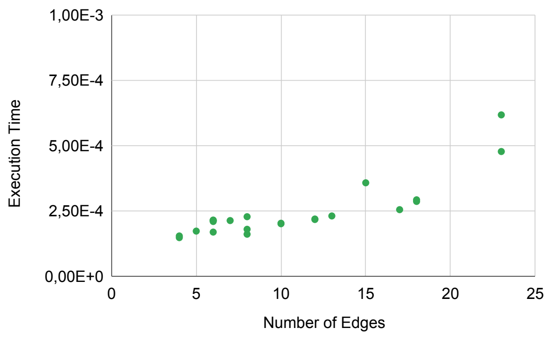

Fig. 8
Correlation between number of edges and elapsed time for maximum connection greedy algorithm.

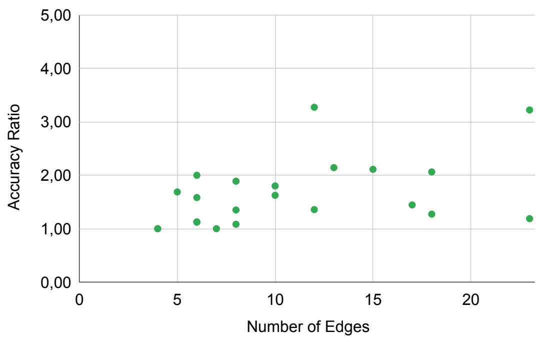

Fig. 9
Correlation between number of edges and accuracy ratio for maximum connection greedy algorithm.

TABLE IV
Results from Chaurasia-Singh greedy algorithm

V	p	E	Execution Time (s)	# of Solutions Tested	Accuracy Ratio
4	0.125	4	1,61E-04	1	1,80
4	0.25	6	2,28E-04	2	1,00
4	0.5	6	2,04E-04	2	1,00
4	0.75	6	2,13E-04	2	1,00
5	0.125	4	1,51E-04	1	1,00
5	0.25	6	2,38E-04	2	1,00
5	0.5	7	2,28E-04	2	1,00
5	0.75	8	2,48E-04	2	1,00
6	0.125	5	2,19E-04	2	1,50
6	0.25	8	2,14E-04	3	1,61
6	0.5	12	2,31E-04	3	1,00
6	0.75	13	2,38E-04	3	1,00
7	0.125	6	1,87E-04	3	1,00
7	0.25	10	2,10E-04	3	1,60
7	0.5	15	3,74E-04	3	1,56
7	0.75	18	3,15E-04	3	1,27
8	0.125	8	1,90E-04	3	1,00
8	0.25	12	2,28E-04	3	1,00
8	0.5	18	2,84E-04	3	1,31
8	0.75	23	5,49E-04	4	1,75
9	0.125	10	2,21E-04	3	2,13
9	0.25	17	2,81E-04	3	2,56
9	0.5	23	6,62E-04	3	4,44

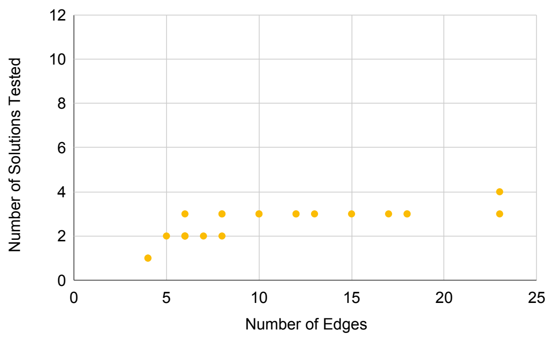

Fig. 10
Correlation between number of edges and number of solutions tested for Chaurasia-Singh greedy algorithm.

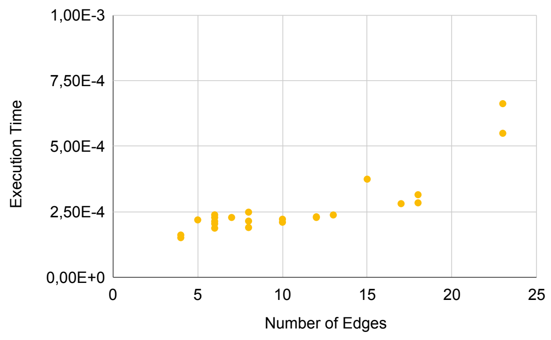

Fig. 11
Correlation between number of edges and elapsed time for Chaurasia-Singh greedy algorithm.

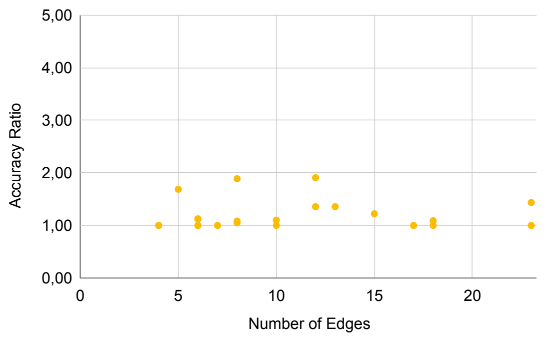

Fig. 12
Correlation between number of edges and accuracy ratio for Chaurasia-Singh greedy algorithm.

V Conclusions

In this report, four different algorithm to solve the minimum weight dominating set problem were proposed. The first one is an exhaustive search algorithm, which based on its formal analysis it can be concluded that it is time-consuming but it guarantees an optimal solution. The next three are greedy algorithms, which based on their formal analysis, they take less time than the exhaustive one, however, the solution is not guaranteed to be optimal. The greedy algorithm are based on three different choices: choosing the edges with minimum weigh, choosing the edges with maximum number of connections, and choosing the edges with the largest ratio between the weight of adjacent edges and the weight of the edge itself.

Results prove that the exhaustive search and the greedy algorithms follow the behavior defined by their formal analysis. It is also verified that the most accurate greedy algorithm is the one based on the work of Chaurasia and Singh.

Possible future work on this subject is the testing of exhaustive search in more complex graphs using better hardware, to further verify if it follows its formal analysis. Another possible problem to tackle is the sorting in the greedy algorithms. Currently, the sorting used was already created, giving the author no control over it, including not being able to verify how many basic operations it performed. Still regarding the greedy algorithms, a n-log-n regression could be done on the data retrieved from these algorithms, to verify if it follows their formal analysis

References

[1] Mihalis Yannakakis and Fanica Gavril, "Edge dominating sets in graphs", *SIAM J. Appl. Math.*, vol. 38, pp. 364–372, 06 1980.

[2] A. Levitin, *Introduction to the Design & Analysis of Algorithms*, Always learning. Pearson, 2012.
URL: `https://books.google.pt/books?id=dQeqKQEACAAJ`

[3] S. N. Chaurasia and A. Singh, "A hybrid evolutionary algorithm with guided mutation for minimum weight dominating set", *Applied Intelligence*, vol. 43, no. 3, pp. 512–529, 2015, Cited By :15.
URL: `www.scopus.com`

[4] "sort() in Python", `https://www.geeksforgeeks.org/sort-in-python/`, 2022, [Online; accessed 19-November-2022].