

Minimum Weight Dominating Set

Manuel Gomes, 88939

Abstract –...

Resumo –...

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 sections: 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.

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 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

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 need 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 with its weight. 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:

$l_{E,W,A,W_A} \leftarrow$ list of E with corresponding
sets of weights W , of adjacent edges A
and of weight of adjacent edges W_A
 $w_{min} \leftarrow 0$
 $set_{min} \leftarrow [\cdot]$
 $l_{W_r} \leftarrow [\cdot]$

for $edge, weight, edges_{adjacents}, weight_{adjacents}$ in
 l_{E,W,A,W_A} **do**

$W_{ratio} = weight_{adjacents}/weight$
 $l_{W_r} \leftarrow$ add W_{ratio}

end for

$l_{E,W,A,W_A,W_r} \leftarrow [l_{E,W,A,W_A}, l_{W_r}]$

$l_{E,W,A,W_A,W_r-sorted} = l_{E,W,A,W_A,W_r}$ sorted in de-
scending order by weight ratio w_r

for $edge, weight, edges_{adjacent}, weight_{adjacent}, weight_{ratio}$
in $l_{E-W-Asorted}$ **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 their repetitiveness. 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.

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

$$combination = [0, 1, \dots, 1, 0], \text{ with length } n \quad (2)$$

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.

$$T(n) = 2^n \quad (3)$$

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

$$O(n) = 2^n \quad (4)$$

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

$$W(n) = 2^n \quad (5)$$

$$B(n) = 2^n \quad (6)$$

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.

$$O(n) = n \log_2(n) \quad (7)$$

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.

$$W(n) = n \quad (8)$$

$$B(n) = 1 \quad (9)$$

IV. RESULTS

V. CONCLUSIONS

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].