IOCA 2021

Proceedings

Algorithms for finding minimum dominating set in a graph[†]

Frank Angel Hernández Mira [1], Ernesto Parra Inza [2], José María Sigarreta Almira [3] and Nodari Vakhania [2*]

[1] Centro de Ciencias de Desarrollo Regional, Universidad Autónoma de Guerrero; fmira8906@gmail.com
[2] Centro de Investigación en Ciencias, Universidad Autónoma del Estado de Morelos; eparrainza@gmail.com (E.P.I), nodari@uaem.mx (N.V)
[3] Facultad de Matemáticas, Universidad Autónoma de Guerrero; josemariasigarretaalmira@hotmail.com
[*] Correspondence: nodari@uaem.mx

[†] 1st International Online Conference on Algorithms (IOCA 2021), https://ioca2021.sciforum.net/, September 27 - October 10/2021.

Abstract: In a simple connected graph $G = (V, E)$, a subset of vertices $S \subseteq V$ is a dominating set in graph G if any vertex $v \in V \setminus S$ is adjacent to some vertex x from this subset. It is known that this problem is NP-hard, and hence there exists no exact polynomial-time algorithm that finds an optimal solution to the problem. This work aims to present an exact enumeration and heuristic algorithm that can be used for large-scale real-life instances. Our exact enumeration algorithm begins with specially derived lower and upper bounds on the number of vertices in an optimal solution and carries out a binary search within the successively derived time intervals. The proposed heuristic accomplishes a kind of depth-first search combined with breadth-first search in a solution tree. The performance of the proposed algorithms is far better than that of the state-of-the-art ones. For example, our exact algorithm has solved optimally problem instances with order 300 in 165 seconds. This is a drastic breakthrough compared to the earlier known exact method that took 11036 seconds for the same problem instance. On average, over all the 100 tested problem instances, our enumeration algorithm is 168 times faster.

Keywords: graph; dominating set; enumeration algorithm; heuristic; time complexity

Citation: Hernández Mira, F.A.; Parra Inza, E.; Sigarreta Almira, J.M.; Vakhania N. Algorithms for finding minimum dominating set in a graph. *IOCA2021* **2021**, *1*, 0. https://doi.org/

Published:

1. Introduction

Finding a minimum dominating set in a graph is a traditional discrete optimization problem. In a simple connected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, a subset of vertices $S \subseteq V$ is a *dominating set* in graph G if any vertex $v \in V$ is *adjacent* to some vertex x from this subset (i.e., there is an edge $(v, x) \in E$) unless vertex v itself belongs to set S. Any subset S with this property will be referred to as a *feasible solution*, whereas any subset of vertices from set V will be referred as a *solution*. The number of vertices in a solution will be referred to as its *size (order)*. The objective is to find an *optimal solution*, a feasible solution with the minimum possible size $\gamma(G)$.

Since the number of subsets of set V is an exponent of n, a complete enumeration of all solutions would imply an exponential cost 2^n. For example, for a graph with only 20 nodes, such an enumeration would take centuries on modern computers. Since the problem is known to be *NP-hard*, there exists no exact algorithm that finds an optimal solution in polynomial time. An exact exponential-time enumeration algorithm was suggested in [9–11]. Although the authors in [10,11] do not provide any experimental study of their algorithm, they give an exact expression for its running time in terms of the number of vertices n, $O(1,4969^n)$. On one hand, this is better than the cost 2^n for a complete enumeration of all feasible solutions. On the other hand, the algorithm remains impractical. For instance, for graphs with already 100 nodes, it would run for more than 5 years. Non-exact heuristic algorithms that run in a reasonable time but which may deliver solutions of unacceptable quality were proposed in [1–8].

In this work, we aim to develop an exact implicit enumeration algorithm that can be used in real-life scenarios with graphs with a considerable number of vertices. Our algorithm applies the lower and upper limits, L and U, respectively, on the number of vertices in an optimal solution. A lower bound L is derived from an earlier known results (see [1,12,13]), and an upper bound U is the size of a feasible solution obtained by the approximation algorithm from [1]. The proposed search method allows discarding a considerable number of solutions in the enumeration process. We combined it with a binary search that made the overall enumeration process more efficient. Instead of enumerating the solutions of all possible sizes from the range $[L, U]$, the sizes are derived from this interval by the binary division process (reducing $U - L$ possible solution sizes to $\log(U - L)$).

Remarkably, the overall enumeration algorithm has solved optimally problem instances with up to 300 vertices in 165 seconds. This is a drastic breakthrough compared to the state-of-the-art exact method from Van Rooij and Bodlaender [10], which required 11036 seconds for the same instance. On average, for random instances for graphs with a density approximately equal to 0.5 ($density(G) = \frac{2|E|}{|V|(|V|+1)}$), our algorithm found optimal solutions 168 times faster. The proposed heuristic algorithm carries out a kind of depth-first search combined with breadth-first search in a solution tree. Among 150 randomly generated problem instances with up to 770 vertices with the density between 0.2 and 0.5, our heuristic has improved the earlier known state-of-the-art solutions (Hernandez et al. [1]) in 98.62% of the instances.

2. Materials and Methods

2.1. Implicit enumeration

In this section, we describe our algorithm. For that, we define some basic properties on a graph. The *diameter* $d(G)$ of graph G is the maximum number of edges on the shortest path between any pair of vertices in that graph, and the *radius* $r(G)$ in graph G is the minimum number of edges on the shortest path (one with the minimum number of edges) between any pair of vertices in that graph. A *support vertex* is a vertex adjacent to a degree one vertex. We denote the set of support vertices in graph G by $Supp(G)$, the set of vertices with decree one by $l(G)$, and the degree of a vertex with the maximum number of neighbors in graph G by $\Delta(G)$.

Initially, we create a feasible solution using the approximation algorithm from [1]. This solution defines the initial upper bound U on the size of a feasible solution, whereas the initial lower bound L is obtained based on the following known results.

Theorem 1. *[12]* $\gamma(G) \geq \frac{n}{\Delta(G)+1}$.

Theorem 2. *[12]* $\gamma(G) \geq \frac{2r(G)}{3}$ *and* $\gamma(G) \geq \frac{d(G)+1}{3}$.

Theorem 3. *[12]* $|Supp(G)| \leq \gamma(G) \leq n - |Leaf(G)|$.

The next corollary is an immediate consequence of the Theorems 1, 2 and 3.

Corollary 1. $L = \max\{\frac{n}{\Delta(G)+1}, \frac{2r(G)}{3}, \frac{d(G)+1}{3}, s\}$ *is a lower bound on the number of vertices in a minimum dominating set.*

The solutions of the size $\nu \in [L, U]$ are generated and tested for the feasibility based on the specially formed priority list of solutions. The sizes are derived by the standard binary division search accomplished in the interval $[L, U]$ as described below. For each created solution σ of size ν feasibility condition is verified, i.e., it is verified if the solution forms a dominating set. Below we describe the general framework of the algorithm, and then we describe how the priority lists are created.

- If solution σ is feasible, then the current upper bound U is updated to ν and the algorithm proceeds with the reduced in this way time interval (continuing with the next to ν (smaller) trial value from the interval $[L, \nu)$ derived by the binary division); if all trial νs were already tested, then the algorithm returns the created feasible solution with the minimum size and halts.
- If solution σ is not feasible, then the algorithm calls *Procedure_Next*(ν) which selects the next to σ solution of size ν from the corresponding priority list (see below), and the algorithm is repeatedly invoked for that solution.
- If *Procedure_Next*(ν) returns NIL, *that is*, all the (potentially optimal) solutions of size ν were already tested for the feasibility (hence none of them being feasible), the current lower bound L is updated to ν and the algorithm proceeds with the reduced in this way time interval (continuing with the next to ν (larger) trial value from the interval $[\nu, U)$ derived by the binary division); if all trial νs were already derived, then the algorithm returns the created feasible solution with the minimum size halts.

2.2. *Procedure_Next*(ν)

For each trial $\nu \in [L, U]$, the solutions of size at most ν are generated in a special priority order that is intended to help in a faster convergence to a feasible solution. Heuristic considerations are used to determine that order. As briefly noted, *Procedure_Next*(ν) determines the (next) solution $\sigma_h(\nu)$ of size ν at iteration h. An auxiliary subroutine *Procedure_Priority_LIST*$()$ generates a priority list of vertices which is used for the creation of solution $\sigma_h(\nu)$. While creating this list, the support and leaf vertices are ignored: by Theorem 3, for every iteration h, all vertices from set $Supp(G)$ can be included in solution $\sigma_h(\nu)$ and no vertex from set $l(G)$ is to be part of it.

Iteratively, if solution $\sigma_h(\nu)$ is feasible, *Procedure_Next*(ν) returns that solution and the algorithm switches to the next trial ν; it halts if all trial sizes were already considered. Otherwise, *Procedure_Next*(ν) creates next solution $\sigma_{h+1}(\nu)$ which is obtained from solution $\sigma_h(\nu)$ by a vertex interchange as follows. Let $v \notin \sigma_h(\nu)$ be the next vertex from the priority list, and let $v' \in \sigma_h(\nu)$ be the lowest active degree vertex in solution $\sigma_h(\nu)$.

If the size of solution $\sigma_h(\nu)$ is less than ν, then

$$\sigma_{h+1}(\nu) := \sigma_h(\nu) \cup \{v\};$$

If the size of solution $\sigma_h(\nu)$ is already ν (it cannot be more than ν), then

$$\sigma_{h+1}(\nu) := (\sigma_h(\nu) \setminus \{v'\}) \cup \{v\}.$$

Procedure_Next(ν) verifies the feasibility of each solution $\sigma_h(\nu)$ generated.

Remark 1. *The feasibility of every generated solution of a given size is verified in time $O(n)$.*

Now we can give a formal description of our enumeration algorithm.

2.3. *Algorithm proposed*

Let $s = |Supp(G)|$ and $l = |l(G)|$. Now, with the above mentioned and Remark 1, we obtain the following lemma.

Lemma 1. *The time complexity of $Algorithm\_BDS$ is*

$$O\left(n \log(\frac{n}{2} - 1) \binom{n}{n/4}\right).$$

Proof of Lemma 1. Since binary search in the interval $[L, U]$ is carried out, the total number of external iterations in Algorithm 1 (i.e., the number of different sizes ν) is at most $\log(U - L)$. For a given size ν, the number of the generated solutions of that size is

Algorithm 1 Algorithm_BDS

```
Input: A graph $G$.
Output: A $\gamma(G)$-set $S$.
$Supp(G)$ := Set of support vertex of graph $G$;
$l(G)$ := Set of leaf vertex of graph $G$;
$L := \max\{\frac{n}{\Delta(G)+1}, \frac{2r}{3}, \frac{d+1}{3}, |Supp|\}$;
$S$ := Feasible solution proposed in [1];
$U := |S|$;
$\nu := \lfloor (L+U)/2 \rfloor$;
$Procedure\_Priority\_LIST()$;
    { iterative step }
while $U - L > 1$ do
    if $Procedure\_Next(\nu)$ returns $NIL$ then
        $L := \nu$;
        $\nu := \lfloor (L+U)/2 \rfloor$;
    else    { A feasible solution was found ($\sigma_h(\nu)$)}
        $U := \nu$;
        $\nu := \lfloor (L+U)/2 \rfloor$;
        $Procedure\_Priority\_LIST()$;
    end if
end while
```

bounded by $\binom{n-s-l}{\nu-s}$. Indeed, in the worst case all solutions of cardinality ν are considered. By Theorem 3, the set $Supp(G)$ forms part of all generated solutions, whereas no leaf vertex belongs to any created solution and hence $\binom{n-s-l}{\nu-s}$ is an upper bound on the number of solutions of size ν that the algorithm creates. To establish the feasibility of solution $\sigma_h(\nu)$, $Procedure\_Next(\nu)$ verifies if every vertex $x \in V(G)$ is in $\sigma_h(\nu)$ or if it is adjacent to a vertex in $\sigma_h(\nu)$, which clearly tales time $O(n)$. For the purpose of this estimation, let us assume that $\nu = \lfloor (U+L)/2 \rfloor$ (as the maximum number of combinations is reached for this particular ν). We may also express ν in terms of n as $\nu = \lfloor (\frac{n}{2}+1)/2 \rfloor = \frac{n}{4}$ using $U \le n/2$, $L \ge 1$, $s \ge 0$ and $l \ge 0$. Summing up the above, we have an overall bound $O\Big(n \log(\frac{n}{2}-1)\binom{n}{n/4}\Big)$ on the cost of the algorithm. □

2.4. A combined DFS and BFS search

Our second algorithm DBS (Depth Breadth Search), combines depth-first search with a breadth-first search in solution tree T, a binary tree of depth n, in which vertex v^i is associated with level i. The path from the root to a leaf uniquely defines a solution in that tree. With each solution, a binary number with n digits is naturally associated, with 0 entry in position i if vertex v^i does not pertain to that solution, and with the entry 1 otherwise. Every path in tree T from the root to a leaf represents a binary number of n digits and the corresponding solution. If the edge of this path at level i of the tree is marked as 0 then vertex v^i does not belong to that solution, and if that edge is marked as 1 then it belongs to the solution.

Given solution $\sigma = (v^1, v^2, \ldots, v^{U_0})$ obtained by the greedy algorithm from [1], we define an auxiliary parameter $\beta = \lfloor \alpha(U-s) \rfloor + s$, for $0 < \alpha < 1$, as the size of a base solution $\sigma(\beta)$ (U is the current upper bound, initially it is U_0, $s = |Supp(G)|$). A base solution is constructed by the procedure and serves as a basis for the construction of the following larger sized solutions sharing the β vertices with solution $\sigma(\beta)$. In case none of these extensions of solution $\sigma(\beta)$ turn out to be feasible, the current base solution is replaced by another base solution of size β and the search similarly continues.

The set of vertices in a base solution is determined according to one of the following alternative rules:

1. The first β vertices $(v^1, v^2, \ldots, v^{\beta})$ from solution σ.
2. Randomly selected β vertices from solution σ.
3. Randomly selected β vertices from set $V \setminus l(G)$.

With each of these options, the vertices in a newly determined base solution are selected in such a way that it does not coincide with any of the earlier formed base solutions. Typically, the procedure creates the first base solution by rule *(1)* or rule *(2)*. The following base solutions are obtained by rule *(3)* until the last such generated base solution coincides with an earlier created one. If this occurs, then the remaining base solutions are created just in the lexicographic order (according to their binary representations).

Each base solution is iteratively extended by one vertex per iteration and each of these extensions are checked for feasibility until either *(i)* one of them turns out to be feasible or *(ii)* an extension of size $U-1$ is created. In the latter case *(ii)* if the corresponding extension of size $U-1$ is not feasible, the next base solution with size β is constructed and the procedure is repeated for the newly created base solution. In the former case *(i)*, the current lower bound is updated; correspondingly, the parameter β is also updated, the first base solution of the new size β is created and the procedure is again repeated for this newly created base solution. Procedure DBS halts if the extensions of all the base solutions of the current size β were tested and none of them turned out to be feasible. Then $\gamma(G)=U$ and the procedure return the corresponding feasible solution of size U, which is minimal.

The definition of the upper bound U, lower bound L, and the fact that none of the solutions of size lower to $U-1$ is feasible, immediately follows from the following remark.

Remark 2. *The Procedure DBC returns a minimal dominating set.*

Remark 3. *If none extension of all base solutions of current size β is feasible and $\beta > L$, then $\beta < \gamma(G) \le U$.*

We complete the description of Procedure DBS by specifying how the extensions of each base solution are generated. A base solution $\sigma(\beta)$ is iteratively extended in at most $U-1-\beta-s$ iterations, by one vertex per iteration. Let $\sigma_i(\beta)$ be the extension of solution $\sigma(\beta)$ by iteration i, $\beta < i < U$, where we let $\sigma_0(\beta) = \sigma(\beta)$. Then $\sigma_i(\beta) = \sigma_{i-1}(\beta) \cup x_i$, where $x_i \in V \setminus l(G) \setminus \sigma_{i-1}(\beta)$ is determined by one of the following selection rules. In case solution $\sigma(\beta)$ was formed by rule *(1)*, x_i is selected randomly, whereas if solution $\sigma(\beta)$ was formed by either of the rules *(2)* and *(3)*,x_i is set to be a vertex with the higher active degree[1] from set $V \setminus l(G) \setminus \sigma_{i-1}(\beta)$.

The Procedure DBS does not guarantee the optimal solution, but it allows to improve the solutions of [1]. Some computational experiments are discussed in Table 3 of the Results section.

3. Results

In this section, we describe our computational experiments. We have implemented the algorithms in C++ using Visual Studio for Windows 10 (64 bits) on a personal computer with Intel Core i7-9750H (2.6 GHz) and 12 GB of RAM DDR4. The order and the size of an instance were generated randomly utilization function *random()*. To complete the set E(G), each new edge was added in between two yet non-adjacent vertices randomly until the corresponding size was attained. The results for the instances are shown in Table 1. We can observe a significant difference in the time of the algorithms tested. We have obtained that for 100% of the analyzed instances, with a density greater or equal to 0.5, $Time(BDS) \approx \frac{1}{168} Time(MSC)$. The $Time(A)$ function returns the time in seconds that it takes for algorithm A to give a response.

Table 1. Graphs with density ≈ 0.5.

No.	$\lvert V(G)\rvert$	$\lvert E(G)\rvert$	Time BDS (s)	Time MSC (s)	Lower Bounds				$\gamma(G)$	Upper Bounds	
					$\frac{n}{\Delta(G)+1}$	$\frac{d+1}{3}$	$\frac{2r}{3}$	$\lvert Supp(G)\rvert$		$\lvert S\rvert$	$n-\Delta(G)$
1	121	2979	4.14543	48.6736	1	1	1	1	5	6	56
2	121	2979	0.12797	48.4455	1	1	1	2	4	4	59
3	121	3015	0.114021	49.8324	1	1	1	1	4	4	60
4	127	3267	4.99421	61.1054	1	1	1	1	5	6	58
5	130	3449	0.941822	71.8736	2	1	1	1	5	5	56
6	131	3533	0.161308	75.7072	1	1	1	1	4	5	61
7	134	3632	5.90682	85.7137	1	1	1	1	5	5	59
8	136	3687	6.43478	92.3635	1	1	1	1	5	5	68
9	138	3912	6.74142	104.484	1	1	2	1	5	6	64
10	141	4085	0.189412	115.303	1	1	1	1	4	4	61
11	142	4074	7.31829	113.367	1	1	1	2	5	6	71
12	145	4085	8.31298	110.847	1	1	1	2	5	5	70
13	147	4328	8.35853	144.31	1	1	1	1	5	5	74
14	151	4575	10.1057	171.704	1	1	1	1	5	6	70
15	157	4996	11.652	220.932	1	1	1	3	5	5	75
16	163	5329	0.275169	284.979	1	2	1	1	4	4	78
17	167	5229	6.8164	288.135	1	1	1	4	5	5	80
18	171	5984	18.9668	421.062	1	1	1	1	5	6	86
19	176	6201	19.9994	480.029	1	2	1	1	5	5	87
20	182	6629	23.0523	611.24	1	1	1	2	5	6	87
21	185	6849	24.667	666.839	1	1	1	1	5	5	93
22	189	7147	23.8574	770.36	1	1	1	1	5	5	91
23	194	7529	28.294	915.361	1	1	1	1	5	5	96
24	198	7842	30.5348	1041.98	2	1	1	1	5	5	101
25	201	8081	31.0313	1136.67	2	1	1	2	5	5	103
26	207	8569	35.9424	1361.77	2	1	1	1	5	6	105
27	209	8735	37.3502	1454.44	1	2	1	1	5	5	104
28	215	9243	42.9108	1810.09	1	1	1	1	5	5	103
29	217	9415	42.881	1892.35	1	1	1	3	5	5	107
30	221	9765	49.91	2011.15	2	1	1	1	6	7	115
31	226	10211	51.2046	2278.89	2	1	1	1	5	5	114
32	230	10575	4.6399	2537.6	1	2	1	4	5	5	111
33	233	10852	57.8398	2793.59	2	1	1	1	5	6	118
34	238	11322	65.053	3093.64	2	1	1	1	5	6	123
35	242	11705	67.0996	3463.78	1	1	1	2	5	5	119
36	250	12491	83.1936	4065.44	1	2	1	1	5	5	125
37	257	13199	93.983	4919.85	2	1	1	1	5	6	131
38	264	13927	112.518	5925.73	2	1	1	2	5	5	135
39	269	14459	112.609	6102.11	1	1	1	1	5	6	134
40	275	15111	114.446	6887.98	2	1	1	1	5	6	144
41	283	16002	418.72	8040.4	1	1	1	1	5	6	141
42	290	16803	148.712	9215.59	2	1	1	1	5	5	152
43	296	17505	155.861	10269.4	2	1	1	1	5	6	152
44	300	17981	165.301	11035.8	2	1	1	1	5	5	151

When analyzing graphs with a density of approximately 0.2, the execution times of the analyzed algorithms behave differently from the cases analyzed previously. In low-density instances, the MSC algorithm is faster than the algorithm proposed in this paper. The results of the experiments, with this type instance, can be seen in Table 2.

Table 2. Graphs with density ≈ 0.2.

No.	$\lvert V(G)\rvert$	$\lvert E(G)\rvert$	Time BDS (s)	Time MSC (s)	Lower Bounds				$\gamma(G)$	Upper Bounds	
					$\frac{n}{\Delta(G)+1}$	$\frac{d+1}{3}$	$\frac{2r}{3}$	$\lvert Supp(G)\rvert$		$\lvert S\rvert$	$n-\Delta(G)$
1	50	286	1.89587	0.616285	2	1	1	2	6	6	33
2	60	357	3.13747	0.715912	2	1	1	1	6	7	40
3	70	524	7.04679	1.46234	2	1	1	1	6	6	45
4	80	678	12.5997	2.81906	2	1	1	1	6	7	53
5	90	842	390.903	5.18101	3	1	1	1	7	8	63
6	100	1031	803.741	8.31738	2	1	1	2	7	7	67
7	101	1051	804.208	8.69229	3	1	1	1	7	7	70
8	106	1154	1143.92	10.8275	2	1	1	1	7	8	71
9	113	1306	1594.92	16.2542	3	1	1	1	7	7	77
10	117	1398	3364.4	19.955	3	1	1	2	8	9	84
11	121	1493	2240.49	23.1156	3	1	1	1	7	7	87

The computational experiments with the Procedure DBS showed that in 98.62% of the analyzed instances the solution given by [1] was improved. Table 3 shows some of these results.

Table 3. Results Procedure DBS.

No.	$\lvert V(G)\rvert$	$\lvert E(G)\rvert$	$\lvert S\rvert$	Solution 1				Solution 2			
				β	$\sigma_i(\beta)$ generates	Time(s)	$\lvert DS\rvert$	β	$\sigma_i(\beta)$ generates	Time(s)	$\lvert DS\rvert$
1	600	84557	12	4	43	37.815	11				
2	610	87490	12	4	3225	2876.67	11				
3	620	90472	13	3	21	20.953	12	3	6	25.745	11
4	630	93505	13	4	107	90.532	12	3	35266	29750	11
5	640	96587	13	4	22	23.207	11				
6	650	102571	9	2			No solution found				
7	660	105798	10	3	4080	4635.46	8				
8	670	109076	9	2	18	24.966	8				
9	680	109417	12	4	65	86.754	11				
10	690	117488	10	3	812	1055.51	9				
11	700	116132	13	4	39	55.176	12				
12	710	120941	10	3	12	21.208	9				
13	720	127996	9	2	11299	15411.1	8				
14	730	131598	9	2	25142	37801.4	8				
15	740	130162	13	4	52	72.597	12	3	31056	40681.6	11
16	750	137096	10	3	4498	6453.14	9				
17	760	138953	10	3	9	18.662	9				
18	770	141210	13	4	13	24.707	12	3	9561	16496.5	11

4. Conclusions

We proposed an exact branch and bound and an approximation heuristic algorithms for the domination problem in general graphs which outperform the state-of-the-art exact and approximation, respectively, algorithms from Van Rooij et al. [10] and Hernández et al. [1], respectively. The first proposed exact Binary Domination Search algorithm combines upper and lower bounds with binary search. The initial lower bound is obtained directly from the earlier known properties and the initial upper bound is obtained by the earlier known best heuristic algorithm for the problem. The practical behavior of the algorithm was tested on a considerable number of the randomly generated problem instances with a size up to 300. On random instances with graphs with an average density of 0.5 algorithms from Van Rooij et al. [10] delayed 168 times more than the Binary Domination Search algorithm. The approximate Depth Breadth Search heuristic combine depth-first search with breadth-first search and was able to improve solutions delivered by the earlier known state-of-the-art algorithm (Hernández et al. [1]) in 98.62% of the tested instances.

Author Contributions: The authors contributed equally to this research. Investigation, F.A.H.M., E.P.I., J.M.S.A. and N.V.; writing—review and editing, F.A.H.M., E.P.I., J.M.S.A. and N.V. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by SEP PRODEP publication grant. The fourth author was supported by SEP PRODEP 511/6 grant and CONACyT 2020-000019-01NACV-00008 grant.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hernández Mira, F.A.; Parra Inza, E.; Sigarreta Almira, J.M.; Vakhania, N. A polynomial-time approximation to a minimum dominating set in a graph. *Theoretical Computer Science. A submitted manuscript*.
2. Campan, A.; Truta, T.M.; Beckerich, M. Fast Dominating Set Algorithms for Social Networks. MAICS, 2015, pp. 55–62.
3. Eubank, S.; Kumar, V.A.; Marathe, M.V.; Srinivasan, A.; Wang, N. Structural and algorithmic aspects of massive social networks. Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, 2004, pp. 718–727.
4. Foerster, K.T. Approximating fault-tolerant domination in general graphs. 2013 Proceedings of the Tenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO). SIAM, 2013, pp. 25–32. doi:https://doi.org/10.1137/1.9781611973037.4.
5. Gibson, M.; Pirwani, I.A. Approximation algorithms for dominating set in disk graphs. *arXiv preprint arXiv:1004.3320* **2010**.

6. Hunt III, H.B.; Marathe, M.V.; Radhakrishnan, V.; Ravi, S.S.; Rosenkrantz, D.J.; Stearns, R.E. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *Journal of algorithms* **1998**, *26*, 238–274. doi:https://doi.org/10.1006/jagm.1997.0903.
7. Jha, A.; Pradhan, D.; Banerjee, S. The secure domination problem in cographs. *Information Processing Letters* **2019**, *145*, 30–38. doi:https://doi.org/10.1016/j.ipl.2019.01.005.
8. Chvatal, V. A greedy heuristic for the set-covering problem. *Mathematics of operations research* **1979**, *4*, 233–235. doi:https://doi.org/10.1287/moor.4.3.233.
9. Gaspers, S.; Kratsch, D.; Liedloff, M.; Todinca, I. Exponential time algorithms for the minimum dominating set problem on some graph classes. *ACM Transactions on Algorithms (TALG)* **2009**, *6*, 1–21. doi:https://doi.org/10.1145/1644015.1644024.
10. Van Rooij, J.M.; Bodlaender, H.L. Exact algorithms for dominating set. *Discrete Applied Mathematics* **2011**, *159*, 2147–2164. doi:https://doi.org/10.1016/j.dam.2011.07.001.
11. Fomin, F.V.; Grandoni, F.; Kratsch, D. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM (JACM)* **2009**, *56*, 1–32. doi:https://doi.org/10.1145/1552285.1552286.
12. Haynes, T. *Domination in Graphs: Volume 2: Advanced Topics*; Routledge, 2017.
13. Cabrera Martínez, A.; Hernández-Gómez, J.C.; Inza, E.P.; Sigarreta, J.M. On the Total Outer k-Independent Domination Number of Graphs. *Mathematics* **2020**, *8*, 194. doi:https://doi.org/10.3390/math8020194.