IOCA 2021

Proceedings

A polynomial-time approximation to a 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: fmira8906@gmail.com; Tel.: +52 7442248438

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

Abstract: A *dominating set* of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that every vertex $v \in V \setminus S$ has at least one neighbor in set S. Finding a dominating set with the minimum cardinality in a graph $G = (V, E)$ is known to be NP-hard. A polynomial-time approximation algorithm for this problem, described here, works in two stages. At the first stage a dominant set is generated by a greedy algorithm, and at the second stage this dominating set is purified (reduced). The reduction is achieved by the analysis of the flowchart of the algorithm of the first stage and a special kind of clustering of the dominating set generated at the first stage. The clustering of the dominating set naturally leads to a special kind of a spanning forest of graph G, which serves as a basis for the second purification stage. The greedy algorithm of the first stage has essentially the same properties as the earlier known state-of-the-art algorithms for the problem. The second purification stage results in an essential improvement of the quality of the dominant set created at the first stage. We have measured the practical behavior of the algorithm of both stages on randomly generated problem instances. We have used two different random methods to generate our graphs, each of them yielding graphs with different structure.

Keywords: graph; dominating set; approximation ratio; approximation algorithm; time complexity

Citation: 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. *Algorithms* **2021**, *1*, 0. https://doi.org/

Published:

1. Introduction

Problem description. One of the most studied problems in combinatorial optimization and graph theory are covering and partitioning problems in graphs. A subset of vertices in a graph is a dominating set if every vertex of that graph which is not in that subset has at least one neighbor in that subset. More formally, given a simple connected undirected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, a set $S \subseteq V$ is called a *dominating set* of that graph if for all $v \in V$ either $v \in S$ or there exists a vertex u in $V \setminus S$ such that edge $(v, u) \in E$. A widely studied such problem is the dominating set problem. The Minimum Dominating Set problem consists in determining the minimum cardinality of a dominating set of $G = (V, E)$.

The *domination number* of graph G, denoted as $\gamma(G)$, is the minimum cardinality of a dominating set for that graph; we shall refer to a corresponding dominating set of cardinality $\gamma(G)$ as a $\gamma(G)$-set. A dominating set is *minimal* if by removing any of its elements the resultant reduced set becomes non-dominating. In fact, it is not so difficult to construct a polynomial-time algorithm that generates a minimal dominating set. However, as one can easily see, not necessarily a minimal dominating set approximates well a minimum dominating set. For instance, given a minimal dominating set, there may exist a non-minimal dominating set with a much smaller number of vertices.

The reader is referred to Haynes et al. [1] for further details on the basic graph terminology. The problem of domination in graphs was mathematically formalized by Berge and Ore in [2] and [3], respectively. Currently, this topic has been detailed in the two well-known books by Haynes, Hedetniemi, and Slater in [1] and [4]. The theory of domination in graphs is an area of increasing interest in discrete mathematics and combinatorial computing, in particular, because of a number of important real-life applications. Such applications arise, for instance, during the analysis of social networks (see Kelleher and Cozzens in [5]), in efficient identification of web communities (see Flake et al. in [6]), in bioinformatics (see Haynes et al. in [7]) and food webs (see Kim et al. in [8]), in the study of transmission of information in networks associated with defense systems (see Powel in [9]), also in distributed computing (see Kuhn et al. in [10], Kuhn and Wattenhofer in [11]). The variations of the domination problem and their applications have been widely studied, see for example, [1] and [4].

Our contributions. In this paper we propose a polynomial-time approximation algorithm, for the MINIMUM DOMINATING SET problem, which runs in two stages. The dominating set created at the first stage is reduced at the second purification stage. Although our greedy algorithm of stage 1 was developed independently from the earlier known algorithm, it turned out that we have used a similar heuristic approach as the above mentioned algorithms from [12] and [13] (a detailed comparison of our algorithm with these two algorithms and their descriptions are given later in Sections 2 and 4). The reduction of the dominating set of stage 1 is achieved at stage 2 by the analysis of the flowchart of the algorithm of the first stage and a special kind of clustering of the dominating set generated at the first stage. The clustering of the dominating set naturally leads to a special kind of a spanning forest of the original graph G, which serves as a basis for the purification stage.

In the next section we introduce the greedy algorithm of stage 1 and we describe our purification procedure of stage 2 moreover, study some useful properties of the dominating set delivered by that procedure. In Section 3 we report our experimental results.

2. Materials and Methods

2.1. Stage 1: Algorithm_Basic

In this subsection we describe our greedy algorithm of stage 1, Algorithm_Basic. It works on a number of iterations. Denote by S_h the dominant set formed by the algorithm by iteration h; $\overline{S_h}$, $\overline{S_h} = V \setminus S_h$, is the complement of the dominant set of iteration h. Initially, $S_0 := \emptyset$. At each iteration $h > 0$ the dominant set S_h of iteration h is obtained by adding vertex $v_h \in \overline{S_{h-1}}$ to the current dominant set S_{h-1} of the previous iteration.

At iteration $h > 0$, the selection of vertex v_h is based on the so-called *active degree* of the vertices in set $\overline{S_{h-1}}$. An active degree of vertex $v \in \overline{S_{h-1}}$ at iteration h is a derivation of the degree of that vertex in graph G that does not take into account the vertices already belonging to set S_{h-1} and the vertices adjacent to a vertex in set S_{h-1}. At iteration h, among all edges adjacent to vertex v, edge (v, x) is not counted in the modified degree of vertex v if either $x \in S_{h-1}$ or $x \in \overline{S_{h-1}}$ but x is a neighbor of a vertex in the set S_{h-1}. So the active degree of vertex v at iteration h is the number of neighbors of vertex v in set $\overline{S_{h-1}}$ not counting those vertices (neighbors of vertex v) in set $\overline{S_{h-1}}$ which are adjacent to a vertex in set S_{h-1}.

Initially at iteration 0, it sets $S_0 := \emptyset$ ($\overline{S_0} := V$, respectively). The initial settings are updated iteratively at each iteration $h > 0$ resulting in the current sets of vertices S_h and $\overline{S_h}$, correspondingly (the partially dominant set and the corresponding complement). At every iteration $h > 0$ vertex $v_h \in \overline{S_{h-1}}$ with the *maximum active degree* is selected and sets S_{h-1} and $\overline{S_{h-1}}$ are modified accordingly resulting in the updated sets S_h and $\overline{S_h}$ (vertex v_h is moved from set $\overline{S_{h-1}}$ to the set S_{h-1} resulting in the updated sets S_h and $\overline{S_h}$). So at every iteration $h > 0$, the updated set S_h contains one more vertex than set S_{h-1} of the previous iteration, whereas the updated set $\overline{S_h}$ contains one less vertex than the set $\overline{S_{h-1}}$ of the previous iteration. The algorithm halts at iteration $h_{\max}$ when $S = S_{h_{\max}}$ is already a dominating set. In Algorithm 1, a formal description is shown.

Denote by G_h the subgraph of graph G induced by set $\overline{S}_h$. We distinguish two kinds of vertices in graph G_h (in set $\overline{S}_h$). Consider iteration h when the set S_h is formed according to the made selection of vertex v_h. We call a vertex $p \in \overline{S}_{h-1}$ adjacent to vertex v_h in graph G_{h-1} a *pending vertex dependent* on vertex v_h by iteration h if there is no vertex in set S_{h-1} adjacent to vertex p (this notion will later be used at the purification stage).

Let p be a pending vertex dependent on vertex $v_{h'}$ by iteration h ($h' \leq h$). For the convenience, we shall consider an edge $(p, v) \in G_h$ incident to vertex p in graph G_h as a special kind of a *two-directional* edge. The direction of edge (v, p) incident to vertex p is marked as *dummy*, whereas the direction of the edge incident from vertex p, (p, v), is treated normally. Accordingly, we shall consider the graph G_h (set $\overline{S}_h$) *empty* if it contains only pending vertices (ones which are already covered by at least one vertex of set S_h).

The following remark gives a sufficient optimality condition for the algorithm Algorithm_Basic, and will also be useful in the estimation of the approximation ratio of our overall algorithm later on.

Remark 1. *If $|S_{h_{\max}}| \leq 2$ then $S_{h_{\max}}$ is a minimum dominating set.*

Proof. It will suffice to consider the two cases when $\gamma(G) = 1$ or $\gamma(G) = 2$. Indeed, suppose, first, that $\gamma(G) = 1$. Then there exists vertex $v \in V$ adjacent to every other vertex in V. Then at the initial iteration 0 of Algorithm_Basic the (active) degree of vertex v would be the maximum and this vertex will be selected as v_0, and, $S_{h_{max}} := \{v\}$ (among all such vertices, ties can clearly be broken arbitrarily). If now $S_{h_{\max}} = 2$, $\gamma(G)$ cannot be less than 2. $\square$

2.2. Stage 2: The purification procedure for Algorithm_Basic

In this subsection we complement Algorithm_Basic with the second *purification* stage, which eliminates specially determined redundant vertices from the dominating set $S_{h_{\max}}$ (the output of Algorithm_Basic). A vertex $x \in S_{h_{\max}}$ is said to be *purified* at stage 2 if it is eliminated from set $S = S_{h_{\max}}$. The purification procedure uses not only the output of Algorithm_Basic but also its flowchart. The following definitions are useful for the analysis of the flowchart of Algorithm_Basic.

Suppose at some iteration h of Algorithm_Basic a pending vertex p, dependent on vertex $v \in S_{h-1}$ gets included in set S_h. Then we call pair (v, p) *tied*. Notice that at any iteration h there may arise at most one new tied pair. Alternatively, we consider (v, p) as an edge. Suppose at iteration h of Algorithm_Basic, vertex z adjacent to a pending vertex p dependent on vertex v, gets included in set S_h. Then we call pair (v, z) *semi-tied*.

Proposition 1. *If there arise neither tied nor semi-tied pairs of vertices during the execution of Algorithm_Basic, then $S_{h_{\max}}$ is a minimum dominating set for graph G.*

Proof. By the way of contradiction, suppose there exists a dominating set S' such that $|S'| < |S_{h_{\max}}|$. If $S' \subset S_{h_{\max}}$ then there exist $v \in S_{h_{\max}}$ and $u \in S'$ such that u is adjacent to v. If (v, u) is not a tied pair, then there is a vertex $z \in S_{h_{\max}}$ such that edge (v, z) is a tied pair (since consider that u is inserted before v either v was dependent on u or it was dependent on z, in any case a tied pair is generated), which is a contradiction.

Otherwise, note that for every vertex $v \in S_{h_{\max}} \setminus S'$, if $N(v) \cap (S_{h_{\max}} \cap S') \neq \emptyset$, then there is a vertex $u \in S_{h_{\max}} \cap S'$ such that u is adjacent to v and a similar analysis as above leads to a contradiction. Now, by pigeonhole principle there exists a vertex u in S' having at least two neighbors, say v and w in set $S_{h_{\max}}$. Also, notice that $u \in \overline{S}_{h_{\max}}$. Similar to previous analysis, if (v, w) is not a semi-tied pair, then there is a vertex $z \in S_{h_{\max}}$ such that u is a dependent vertex on z. So, (v, z) or (w, z) is a semi-tied pair, which is a contradiction. It follows that $S_{h_{\max}}$ is a dominating set of minimum cardinality and Algorithm_Basic is optimal. $\square$

Algorithm 1 Algorithm_Basic

```
Input: A graph $G$.
Output: A dominating set $S = S_h$.
$h := 0$;
$v_0$ := any vertex with the maximum degree in graph $G$;
$S_1 := \{v_0\}$;
    { iterative step }
while $S_h$ is not a dominating set of graph $G$ do
    $h := h + 1$;
    $v_h$ := any vertex with the maximum active degree in set $\overline{S_{h-1}}$;
    $S_h := S_{h-1} \cup \{v_h\}$;
end while
```

Clusters and the induced spanning trees. A collection of tied pairs define independent structural components that reflect the flowchart of Algorithm_Basic. We define these components now.

In such a sense, we shall refer to a sequence of the tied pairs of the form $(v, p_1), (p_1, p_2), (p_2, p_3), \ldots, (p_{l-1}, p_l)$ as a *chain*. We may look at a chain as a sequence of the corresponding edges, a path from vertex v to vertex p_l. Two different chains of tied pairs have either (i) no vertex in common or (ii) solely one vertex in common or (iii) one or more tied pairs in common (a sub-chain of tied pairs of vertices). We shall refer to a maximum set (one with the maximum cardinality) of two or more different chains such that, for any two different chains from the set either (ii) or (iii) is satisfied, as a *cluster of chains*, or a cluster, for short.

As it is easily seen, a cluster C possesses a single vertex such that all chains of that cluster have that vertex in common. We call this vertex the *root* of cluster C and denote it by $r(C)$. We note that the structure of a cluster does not generate cycles among its vertices. The following remark is straightforward.

Description of the purification procedure. We are ready to describe our purification procedure that purifies (reduces) dominant set S, the output of Algorithm_Basic. First we describe the basic version of the procedure which we extend a bit later. The procedure purifies trees $T(C_1), \ldots, T(C_k)$ in this order. Iteratively, it purifies the vertices of the next tree $T(C_i)$ and outputs the purified tree, that we denote by $T'(C_i)$; $T^h(C_i)$ is the partially purified tree of iteration h, and T^h is the corresponding forest. The output of all the k calls of the procedure is a purified forest T'.

At iteration h of the procedure for cluster C_i, let $v \in T^h(C_i)$. We call vertex $x \in V(G) \setminus V(T(C_1) \cup \cdots \cup T(C_k))$ a *semi-private* neighbor of vertex v if vertex v is the only (remained) neighbor of vertex x in the current purified forest $T'(C_1), \ldots, T'(C_{i-1}), T^{h-1}(C_i), T'(C_{i+1}), \ldots, T'(C_k)$. Note that, not necessarily, x is a private neighbor of vertex v in graph G (a private neighbor is also a semi-private one for vertex v, but not necessarily vice-versa).

At iteration h, we distinguish two types of the vertices in tree $T^h(C_i)$: the currently purified ones and ones which were set as non-purified at some previous iteration, the so-called *firm* ones. The procedure carries out a four-degree bottom-up look-ahead checking. Given a yet unprocessed firm vertex, its parent and grandparent are purified and the grand-grandparent is set to be firm (this rule has an exception which is explicitly stated a bit later). Thus an up-going chain of four vertices (a, b, c, d), that we call a *quadruple*, is considered at once so that the endpoint a of that quadruple is already firm, the endpoint d, is set firm and the intermediate vertices are set purified (in general, a processed chain $(a, \ldots, d)$ may contain from two to four vertices; for the sake of simplicity we shall not distinguish them and shall refer to them as quadruples).

The purification procedure starts by purifying all the leaves with no (private or) semi-private neighbor in tree $T(C_i)$. If leaf v is so purified, then its parent is set to be firm. Otherwise (leaf v has a semi-private neighbor), leaf v is set firm. Once all the leaves are so processed, the corresponding firm vertices form the initial set $PF^0(C_i)$ of the firm vertices in tree $T^0(C_i)$, the *pending firm* ones.

Iteratively, at an iteration $h > 0$, the highest level leftmost pending firm vertex with a non-firm parent is looked for. We denote the chosen in this way vertex at iteration h by a_h. If there exists no vertex a_h, i.e., the parent of any pending firm vertex is already firm or there is no pending firm vertex having a parent (the highest level pending firm vertex is the root), then the purification procedure halts. Once vertex a is selected, the quadruple (a, b, c, d) with $a = a_h$ is determined and is processed as follows: The parent b of vertex a_h is purified and its grandparent c (if it exists) is also purified if vertex c has not earlier been set firm (from another up-going branch) or c is the root. In the latter case, the root is set firm. If vertex c is not the root, its parent d is set to be firm (unless it was already set firm at an earlier iteration). In this way, at most two vertices b and c, predecessors of vertex a_h are purified from vertex a_h at iteration h.

Once quadruple (a, b, c, d) is processed, the current set of the pending firm vertices $PF^{h-1}(C_i)$ is updated by deleting from it vertex a (which becomes a non-pending firm vertex), and by including vertex d into the updated set (a new pending firm vertex).

Lemma 1. *If the basic version of the purification procedure purifies vertex $b \in T^h(C_i)$ at iteration h then that vertex cannot be set firm at any later iteration.*

Proof. Let vertex b be purified from vertex a_h at iteration h. By way of contradiction, suppose vertex b is set firm from vertex a_g at iteration $g > h$. By the construction of the procedure, there must exist two intermediate vertices between vertices a_g and b , as otherwise vertex b would not have been set firm. But then the level of vertex a_g is greater than that of vertex a_h and hence the procedure could not select vertex a_h at iteration h as the highest level pending firm vertex. $\square$

From here on, we will refer to the augmented version of the purification procedure as *the purification procedure*, and to the overall two-stage algorithm as *Algorithm_Extended*. It is a known fact that the number of vertices in a minimum dominating set is bounded above by $\frac{n}{2}$, i.e., $\gamma(G) \leq \frac{n}{2}$, for every simple graph G of order n, and this bound is tight. We prove that the same bound is valid for the dominating set delivered by Algorithm_Extended.

Lemma 2. $|S^*| \leq \frac{n}{2}$, *where S^* is the dominating set returned by Algorithm_Extended.*

Proof. Let us consider any (non-purified) vertex $v \in T^*(C)$. Only the following two cases are possible. Either (1) vertex v was left non-purified as a son of a purified vertex $z \in T(C)$ or as the parent of such a vertex, or (2) vertex v was left non-purified as the semi-private neighbor of some vertex x. In case (1) there is a neighbor of vertex v in the complement $\bar{S^*}$ (the corresponding purified vertex z), and in case (2) there exist vertex x from $\bar{S^*}$. Thus with any vertex $v \in S^*$ a unique vertex from $\bar{S^*}$ is associated. Hence, $|S^*| \leq \frac{n}{2}$. $\square$

3. Results and Discussion

In this final section we describe our computation experiments. We have implemented our algorithms in C++ using Windows 10 operative system for 64 bits on a personal computer with Intel Core i7-9750H (2.6 GHz) and 16 GB of RAM DDR4. We have generated two sets of the problem instances obtained by using two different pseudo-random methods to generate the graphs. For the first class, each new edge was added in between two yet non-adjacent vertices randomly until the corresponding size was attained. For the second class of instances, we have followed a more particular rules while adding each new edge according to the chosen structure (e.g., see the example of a graph in Figure 1).

The results for the first class of instances are shown in Table 1. Over all the tested problem instances, we have obtained in average 0.89% of the reduction of the size of the dominating sets at stage 2. We may also observe that the reduction of the size of the dominating sets at the purification stage becomes more notable as the order of the graphs increases.

No.	Order	Size	Clusters	Algorithm_Basic	Algorithm_Extended	Purification
1	841	1647	32	155	154	1
2	908	978	76	235	231	4
3	809	1106	46	178	178	0
4	992	1365	46	226	223	3
5	984	1465	58	208	208	0
6	1011	1811	36	205	205	0
7	1196	2189	42	228	227	1
8	1533	1897	93	363	357	6
9	1538	1722	80	377	373	4
10	1673	3116	67	322	319	3
11	2041	2139	141	519	509	10
12	2393	2812	142	573	564	9
13	2031	3157	114	428	424	4
14	2562	3475	148	578	575	3
15	2064	3438	83	418	415	3
16	3048	3262	222	791	779	12
17	3286	6146	138	629	624	5
18	3089	4966	128	633	629	4
19	3127	4376	177	686	678	8
20	3904	7687	151	725	724	1
21	4041	6344	170	837	829	8
22	4466	6895	200	930	921	9
23	4578	7988	185	901	894	7
24	4389	8482	163	818	817	1
25	4493	8911	171	818	816	2
26	5071	8385	231	1003	997	6
27	5964	8147	320	1326	1308	18
28	5342	8173	265	1112	1105	7
29	5427	7470	304	1203	1192	11
30	5346	8036	282	1133	1129	4
31	6041	6385	345	1536	1502	34
32	6786	7552	378	1692	1649	43
33	6052	7097	295	1452	1430	22
34	6956	11942	252	1378	1372	6
35	6820	12432	282	1305	1299	6
36	7042	11426	324	1419	1411	8
37	7061	11426	313	1439	1422	17
38	7922	15072	269	1465	1464	1
39	7052	10747	343	1480	1469	11
40	7841	8791	442	1927	1881	46
41	8051	10426	449	1831	1813	18
42	8179	13192	388	1664	1654	10
43	8684	17256	303	1592	1588	4
44	8547	12357	461	1848	1837	11
45	8195	11525	465	1794	1783	11
46	9589	10256	496	2401	2359	42
47	9041	15010	415	1818	1805	13
48	9917	15763	408	2011	1997	14
49	9293	14002	439	1952	1937	15
50	9590	13364	519	2086	2067	19

Table 1: The results for the randomly generated graphs

The results for the second class of instances are shown in Table 2. These instances were created with the intention to verify the efficiency of the purification stage for the graphs for which the first stage would deliver a poor solution. Taking as an example the graph depicted in Figure 1, we observe that, if we run Algorithm_Basic for that graph, it will first include the central (bold) vertex into the formed dominant set, then it will add all the (gray) vertices adjacent to the former vertex to the formed dominant set, and then it will also include all the (bold) vertices adjacent to the latter vertices in that set. We may observe such a behavior of the algorithms of the first and the second stages with a significant reduction of the number of vertices in the purified dominating sets for the problem instances of the second class in Table 2.

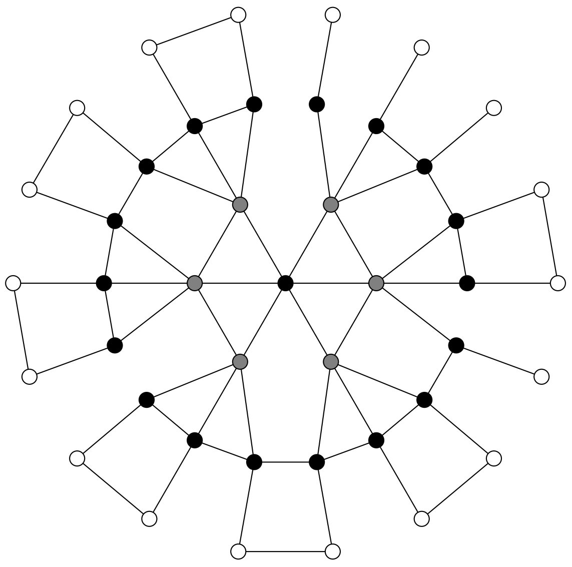

Figure 1. A graph from the second class

No.	Order	Size	Clusters	Algorithm_Basic	Algorithm_Extended	Purification
1	3619	4588	1	1975	1646	329
2	3651	4468	1	1027	833	194
3	3773	4775	1	2059	1716	343
4	3795	4824	1	2071	1726	345
5	4016	4876	1	1129	916	213
6	4180	5286	1	2281	1901	380
7	4290	5491	1	1717	1431	286
8	4485	5643	1	1795	1496	299
9	4620	5885	1	1849	1541	308
10	4673	5714	1	1314	1066	248
11	4740	6020	1	1897	1581	316
12	4965	6039	1	1396	1132	264
13	5200	6597	1	2801	2401	400
14	5525	7056	1	2976	2551	425
15	5720	7279	1	3081	2641	440
16	5759	7309	1	3102	2659	443
17	5768	7084	1	1621	1315	306
18	5785	7403	1	3116	2671	445
19	5841	7096	1	1642	1332	310
20	6019	7728	1	3242	2779	463
21	6058	7664	1	3263	2797	466
22	6383	8126	1	3438	2947	491
23	6409	8115	1	3452	2959	493
24	6571	8055	1	1847	1498	349
25	6837	8580	1	3144	2695	449
26	7350	9291	1	3431	2941	490
27	7447	9015	1	2093	1698	395
28	7545	9464	1	3522	3019	503
29	7560	9565	1	3529	3025	504
30	7593	9242	1	2134	1731	403

No.	Order	Size	Clusters	Algorithm_Basic	Algorithm_Extended	Purification
31	7665	9741	1	3578	3067	511
32	7680	9759	1	3585	3073	512
33	7812	9545	1	2195	1781	414
34	8396	10190	1	2359	1913	446
35	8550	10842	1	4561	3991	570
36	8688	10584	1	2441	1980	461
37	8715	10996	1	4649	4068	581
38	8820	11168	1	4705	4117	588
39	8835	11173	1	4713	4124	589
40	8865	11241	1	4729	4138	591
41	9076	11181	1	4951	4126	825
42	9142	11246	1	4987	4156	831
43	9406	11563	1	5131	4276	855
44	9626	11827	1	5251	4376	875
45	9648	11847	1	5263	4386	877
46	14033	17077	1	3946	3196	750
47	14910	18162	1	4192	3394	798
48	15787	19238	1	4439	3595	844
49	16664	20304	1	4685	3795	890
50	18418	22437	1	5178	4194	984

Table 2: The results of the generated second class graphs.

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. Haynes, T. W., Hedetniemi, S. T., and Slater, P. J. *Domination in Graphs (Advanced Topics)*, **1998**, Marcel Dekker Publications, New York, 9780824700348.
2. Berge, C. (1962). The theory of graphs and its applications, *Methuen and Co.* Ltd., London.
3. O. Ore, Theory of Graphs, *A. M. S. Colloquium Publications 38* (1962), 270 pages.
4. Haynes, T. W., Hedetniemi, S. T., and Slater, P. J. Fundamentals of domination in graphs, **1998**, volume 208 of *Monographs and Textbooks in Pure and Applied Mathematics.*, 9780824700331.
5. Kelleher, L. L., and Cozzens, M. B. Dominating sets in social network graphs. *Mathematical Social Sciences* **1988**, 16(3), 267-279, https://doi.org/10.1016/0165-4896(88)90041-8.
6. G. W. Flake, S. Lawrence, and C. L. Giles, Efficient Identification of Web Communities, *Conference on Knowledge Discovery and Data Mining* **2000**, 150-160, 10.1145/347090.347121.
7. Haynes, T., Knisley, D., Seier, E., and Zou, Y. A quantitative analysis of secondary RNA structure using domination based parameters on trees. *BMC bioinformatics* **2006**, 7(1), 108. https://doi.org/10.1186/1471-2105-7-108
8. Kim, B. J., Liu, J., Um, J., and Lee, S. I. Instability of defensive alliances in the predator-prey model on complex networks. *Physical Review E*, **2005**, 72(4). 10.1103/PhysRevE.72.041906
9. M. Powel, Alliance in graph, *Proc. on th 255 of the USA Military Academy*, **2004**, 1350-1415.
10. Kuhn, F., Moscibroda, T., and Wattenhofer, R. What cannot be computed locally!. *In Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, **2004** (pp. 300-309). ACM. 10.1145/1011767.1011811
11. Kuhn, F., and Wattenhofer, R. Constant-time distributed dominating set approximation. *Distributed Computing* **2005**, 17(4), 303-310. https://doi.org/10.1007/s00446-004-0112-5
12. V. Chvátal, A Greedy Heuristic for the Set Covering problem. *Mathematics of Operations Research* 4, (1979) 233-235.
13. Parekh, A. K. Analysis of a greedy heuristic for finding small dominating sets in graphs. *Information processing letters*, **1991**, 39(5), 237-240. https://doi.org/10.1016/0020-0190(91)90021-9