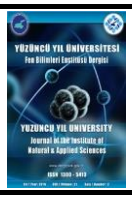
Yuzuncu Yil University
Journal of the Institute of Natural & Applied Sciences
https://dergipark.org.tr/en/pub/yyufbed



Research Article

Roman Domination in Mycielski Graphs: A Study of Some Graphs and a Heuristic Algorithm

Derya DOĞAN DURGUN[1], Emre Niyazi TOPRAKKAYA[*2]

[1]Manisa Celal Bayar University, Arts and Science Faculty, Mathematics, 45110, Manisa, Türkiye
[2]Manisa Celal Bayar University, Institute of Natural and Applied Sciences, Mathematics, 45110, Manisa, Türkiye
Derya DOĞAN DURGUN, ORCID No: 0000-0002-9099-5448,
Emre Niyazi TOPRAKKAYA, ORCID No: 0000-0003-1818-4330
*Corresponding author e-mail: toprakkayaemre@gmail.com

Article Info

Received: 25.02.2024
Accepted: 30.05.2024
Online August 2024

DOI:10.53433/yyufbed.1442759

Keywords
Algorithm,
Graph Theory,
Roman Domination

Abstract: Let $G = (V, E)$ be a graph. A function $f: V \to \{0,1,2\}$, if $\forall u$ for which $f(u) = 0$ is adjacent to $\exists v$ for which $f(v) = 2$, is called a Roman dominating function, and called in short terms RDF. The weight of an RDF f is $f(V) = \sum_{v \in V} f(v)$. The Roman domination number of a graph G, denoted by $\gamma_R(G)$, is the minimum weight of an RDF on G. This paper presents the results for Roman domination numbers of the Mycielski graphs obtained through Mycielski's construction of the comet, double comet, and comb graphs. An algorithm to determine the Roman domination number of any given graph is also provided.

Mycielski Graflarda Roma Baskınlığı: Bazı Grafların ve Bir Sezgisel Algoritmanın Çalışması

Makale Bilgileri

Geliş: 25.02.2024
Kabul: 30.05.2024
Online Ağustos 2024

DOI:10.53433/yyufbed.1442759

Anahtar Kelimeler
Algoritma,
Graf Teorisi,
Roma Baskınlığı

Öz: $G = (V, E)$ bir graf olsun. $f(u) = 0$ olan her u tepesinin, $f(v) = 2$ olan en az bir v tepesine bitişik olması koşulunu karşılayan bir Roma baskınlık fonksiyonu (RDF) $f: V \to \{0, 1, 2\}$. Bir RDF f'in ağırlığı $f(V) = \sum_{v \in V} f(v)$. Bir G grafının Roma baskınlık sayısı, $\gamma_R(G)$ ile gösterilir, G 'de bir RDF'nin minimum ağırlığıdır. Bu çalışmada, Mycielski'nin kuyruklu yıldız, çift kuyruklu yıldız ve tarak graflarını oluşturmasıyla elde edilen Mycielski graflarının Roma baskınlık sayılarına ilişkin sonuçlarını sunmaktadır. Ayrıca, herhangi bir grafın Roma baskınlık sayısını belirleyen bir algoritma sağlanmıştır.

1. Introduction

Let $G = (V, E)$ be a simple graph. $V(G)$ is the notation for the vertex set of a graph, and $E(G)$ is for the graph's edge set. The order of a graph means the number of the vertices of a graph, denoted by $|V| = n$. $N(v) = \{u \in V(G) | uv \in E(G)\}$ is the open neighborhood of a vertex v, and closed neighborhood of it is $N[v] = N(v) \cup \{v\}$. The degree of a vertex v is the number of edges incident to v, denoted by $deg(v)$. For a set $S \subseteq V$, $N(S) = \cup_{v \in S} N(v)$, and $N[S] = N(S) \cup S$. A set S of vertices is a dominating set if every vertex in the graph is either in S or adjacent to at least one vertex in S. The

domination number, denoted as $\gamma(G)$, is the smallest possible size of a dominating set that can dominate all the vertices of G. Such a set of G is called a $\gamma(G) - set$.

In this paper, we consider an algorithm for the Roman domination number, defined by Ian Stewart (Stewart, 1999). Roman dominating function (RDF) on a graph $G = (V, E)$ is a function $f: V \rightarrow \{0, 1, 2\}$ satisfying the condition that every vertex u for which $f(u) = 0$ is adjacent to at least one vertex v for which $f(v) = 2$ (Dreyer, 2000). The weight of an RDF is the value $w(f) = \Sigma_{v \in V}\ f(v)$. The Roman domination number of a graph is the minimum weight of an RDF on G, denoted by $\gamma_R(G)$. $\gamma_R(G) - function$ is a Roman dominating function of G with weight $\gamma_R(G)$ (Henning & Hedetniemi, 2003). A Roman dominating function f on a vertex set V can be represented by the ordered partition (V_0, V_1, V_2) of V, where $V_i = \{v \in V \,|f(v) = i\}$ and $f(v) \in \{0, 1, 2\}$ (Chambers et al., 2009). Its weight is $w(f) = |V_1| + 2|V_2|$. If someone needs additional details about the parameters of domination, as well as the associated terminologies, can refer to the information provided in the literature (Haynes et al., 1998; West et al., 2001).

Cockayne et al. who proposed the Roman dominating function on graphs ask the following questions: "*What are the algorithmic, complexity, and approximation properties of Roman domination?"* and, *"Can you construct a polynomial algorithm for computing the value $\gamma_R(G)$ for any interval graph G?"* (Cockayne et al., 2004). Liedloff et al. showed that there are linear-time algorithms for obtaining the Roman domination number in interval graphs and cographs. They also introduced a dynamic programming algorithm for solving the same problem specifically in interval graphs (Liedloff et al., 2008).

In the context of corporate structures, different departments, whether large or small, have their distinct chain of command. For smooth and efficient operations, employees and managers must communicate effectively. To simplify the chain of command, each employee should fall under the supervision of at least one manager. For instance, consider the organizational chart of a company, where each person is represented by a vertex. If an employee is supervised by a manager, connect the corresponding vertices with an edge. In graphs with this property, a selected group of employees (S) or the complement set of managers $(V(G) - S)$ can serve as dominating sets, reflecting the organizational hierarchy. We explore a variant of domination, known as Roman domination.

This manuscript extends our previous research, which we initially shared as a preprint (Durgun & Toprakkaya, 2021). In this paper, the Roman domination number of some Mycielski graphs is given. Additionally, an algorithm is presented to obtain the Roman domination number.

2. Roman Domination Numbers of Mycielski Graphs of Some Graphs

In this section, theorems for Roman domination numbers of the Mycielski graphs obtained by Mycielski's construction of the comet, double comet, and comb graphs are given. Mycielski graph of a graph G is the graph $\mu(G) = (V', E')$ with vertex set $V' = V \cup \{v' : v \in V\} \cup \{w\}$ and edge set $E' = E \cup \{vu' : vu \in E\} \cup \{v'w : v', w \in V'\}$.

Theorem 2.1. Let $G = C_{t,r}$ be a comet graph where $t \geq 2$ and $r \geq 1$. Then the Roman domination number of $\mu(G)$ is equal to

$$\gamma_R\left(\mu\left(C_{t,r}\right)\right) = \begin{cases} \left(2\frac{t}{3} + 1\right) + 2 & t \equiv 0\ (mod\ 3) \\ 2\left\lfloor\frac{t}{3}\right\rfloor + 2 & otherwise \end{cases} \quad (1)$$

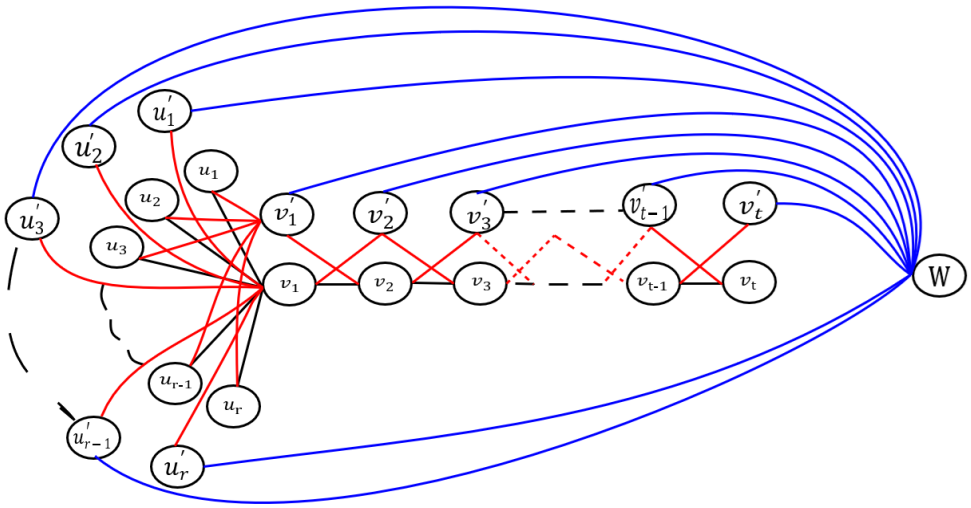


Figure 1. Mycielski of a comet graph.

Proposition 2.2. $\boldsymbol{\gamma_R(\mu(C_{t,r})) = \gamma_R(C_{t,r}) + 2}$

Theorem 2.3. For $p = n - a - b$ and $p \neq 2$, let $G = DC(n,a,b)$ be a double comet graph. The Roman domination number of $\mu(G)$ is equal to

$$\gamma_R\left(\mu\left(DC(n,a,b)\right)\right) = \begin{cases} 2\left(\frac{p}{3}+1\right)+2 & p \equiv 0\ (mod\ 3) \\ 2\left\lfloor\frac{p}{3}\right\rfloor+2 & p \equiv 1\ (mod\ 3) \\ \left(2\left\lfloor\frac{p}{3}\right\rfloor+1\right)+2 & p \equiv 2\ (mod\ 3) \end{cases} \quad (2)$$

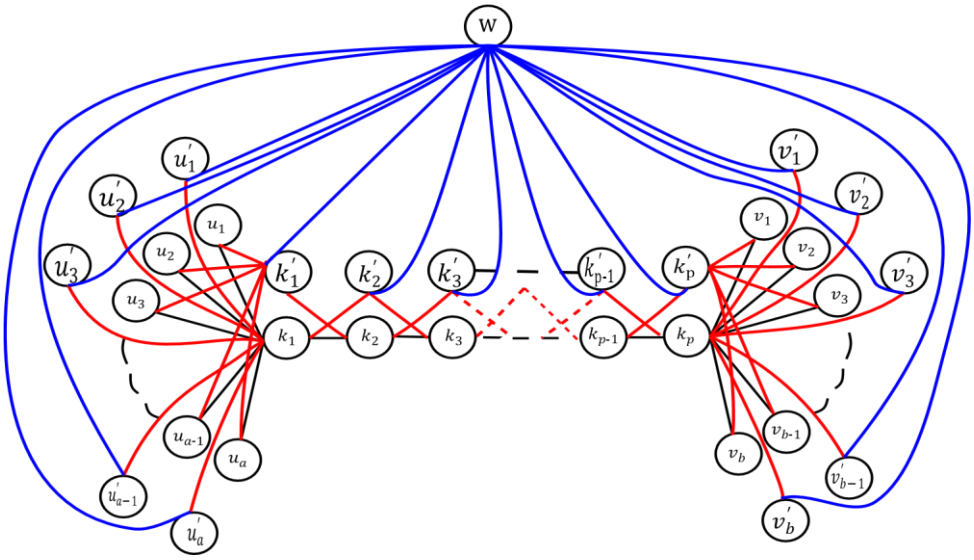


Figure 2. Mycielski of a double comet graph.

Proposition 2.4. $\gamma_R(\mu(DC(n,a,b))) = \gamma_R(DC(n,a,b)) + 2$

Observation 1. We take $p \neq 2$, unlike the definition of the double comet graph, because in the case of $p = 2$, the graph is a Special Roman graph (Kazemi, 2012). Accordingly, when $p = 2$, the value of $\gamma_R(\mu(DC(n,a,b)))$ is equal to $\gamma_R(DC(n,a,b)) + 1$.

Theorem 2.5. Let $G = P_n^+$ be a comb graph. The Roman domination number of $\mu(G)$ is equal to

$$\gamma_R\left(\mu(P_n^+)\right) = \begin{cases} 4\dfrac{n}{3} + 2 & n \equiv 0\ (mod\ 3) \\ \left(4\left\lfloor\dfrac{n}{3}\right\rfloor + 2\right) + 2 & n \equiv 1\ (mod\ 3) \\ \left(4\left\lceil\dfrac{n}{3}\right\rceil - 1\right) + 2 & n \equiv 2\ (mod\ 3) \end{cases} \tag{3}$$

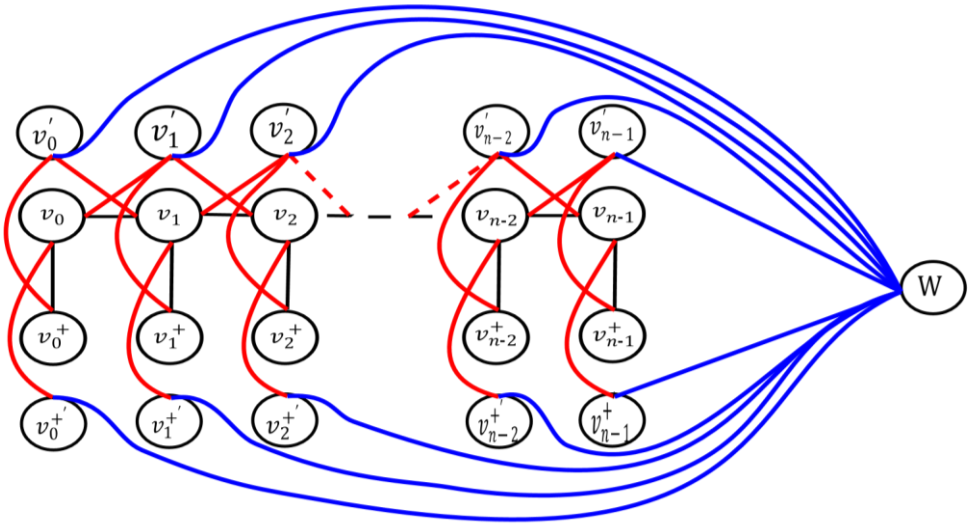


Figure 3. Mycielski of a comb graph.

Proposition 2.6. $\gamma_R(\mu(P_n^+)) = \gamma_R(P_n^+) + 2$

3. An Algorithm for Computing the Roman Domination Number

In this section, we present Algorithm 1 for computing $\gamma_R(G)$ of a graph G.

Our method for sorting the vertices of a graph involves analyzing their degrees, the degree sum of adjacent vertices, and the sum of all vertex degrees. This algorithm repeatedly partitions the graph into smaller sections until it's fully sorted. Once completed, it returns the Roman domination number.

To implement this algorithm for a graph G, its adjacency matrix is entered as data and, the number of vertices is stored in a variable. The while loop runs until the counter reaches the total number of vertices in G. Initially, each vertex dominates itself, so every element in the *'roman domination'* matrix variable is assigned the value of 1.

Within the while loop, the matrix *'degree'* stores the degrees of each vertex in the graph. The algorithm calculates the sum of the degrees of all vertices, assigning this value to the variable *'deg sum'*. Additionally, it calculates the sum of the degrees of adjacent vertices for each vertex and stores this in the *'neighbor deg'* matrix. On lines 13-14 of the algorithm, it checks whether the graph's vertices are isolated. If they are, the value $'\deg sum + 1'$ is assigned to the sum of their adjacent vertices' degrees to keep them separate from others. The indices of the graph's vertices are stored in the *'index'* matrix, ordered in descending order according to their vertex degrees. In rows 16-24, the algorithm sorts vertices according to their neighbor deg values if their degrees are equal, replacing the *'index'* values accordingly. If the value in the first column of the *'index'* variable is not zero, the value $'2'$ is written in the corresponding place in the *'roman domination'* matrix, indicating that the vertex held in the *'index'* variable is included in V_2. The algorithm then deletes the vertex and its adjacent vertices from the graph's adjacency matrix, incrementing the counter by $'1'$. As the while loop runs again, it repeats these operations on smaller partitions until the counter reaches the total number of vertices.

Upon the termination of the while loop, the summation of all values within the *'roman domination'* matrix is calculated and subsequently assigned to the *'gamma roman'* variable. This

resultant value is then returned as the output. An alternative approach would be to extract the values contained within the *'roman domination'* matrix to identify the vertices belonging to V_0, V_1, and V_2.

Algorithm 1: Roman Domination Algorithm

Input: An undirected graph $G = (V, E)$
Output: Roman domination number of the graph G

```
1  begin
2      i = |V|;
3      a = 1;
4      gamma roman = 0;
5      roman domination =A matrix of dimension 1xi consisting of 1's;
6      while a < i do
7          deg sum = 0;
8          for m = 1 to i do
9              degree[m] = |N(m)| ;
10             deg sum = ∑ degree[m];
11         for m = 1 to i do
12             neighbor deg[m] = ∑_{u∈N(m)} deg(u);
13             if neighbor_deg[m] = 0 then
14                 neighbor deg[m] = deg sum + 1;
15         index =the variable holding the indices of the vertices in V sorted in descending
            order according to the degree[m] values;
16         for m = 1 to i − 1 do
17             if index[m] ≠ 0 then
18                 for n = m + 1 to i do
19                     if index[n] ≠ 0 then
20                         if degree[index[m]]=degree[index[n]] then
21                             if neighbor deg[index[m]] > neighbor deg[index[n]] then
22                                 memo=index[m];
23                                 index[m]=index[n];
24                                 index[n]=memo;
25         if index[1] ≠ 0 then
26             roman domination[index[1]] = 2;
27         for each vertex v ∈ Adj(index[1]) in V do
28             roman domination[v] = 0;
29         Delete index[1] vertex in V and each vertex v ∈ Adj(index[1]) in V ;
30         a = a + 1;
31     for m = 1to i do
32         gamma_roman = gamma roman + roman domination[m];
33     Return gamma roman;
```

Figure 4. Roman Domination Algorithm.

Example 3.1. The figures below illustrate a step-by-step demonstration of our algorithm. Our algorithm uses a simple example of G graph as input.

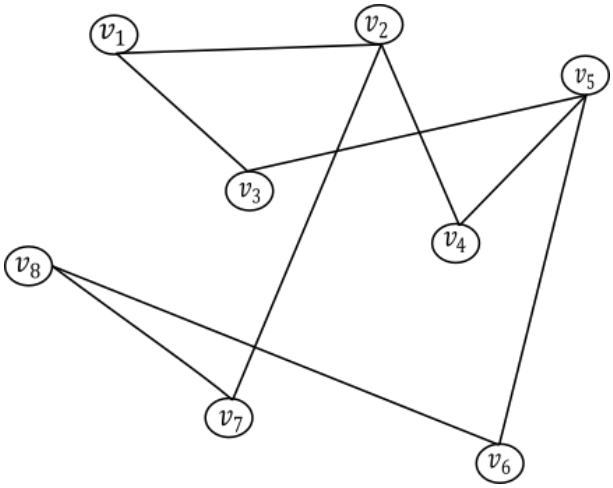


Figure 5. Input graph G and initial conditions of V_1, V_2. $V_2 = \emptyset$ and $V_1 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$.

Table 1. Initial situation of the graph

a	***i***	***index***	***roman domination***	***gamma roman***
1	8	[0 0 0 0 0 0 0 0]	[1 1 1 1 1 1 1 1]	0

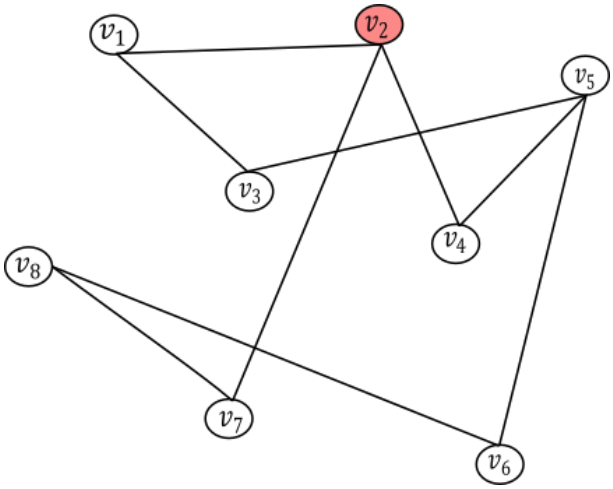


Figure 6. First cycle of the algorithm $V_2 = \{v_2\},\ V_1 = \{v_1, v_3, v_4, v_5, v_6, v_7, v_8\}$.

Table 2. First Iteration

a	***i***	***index***	***roman domination***	***gamma roman***
1	8	[2 5 4 1 3 6 7 8]	[1 2 1 1 1 1 1 1]	0

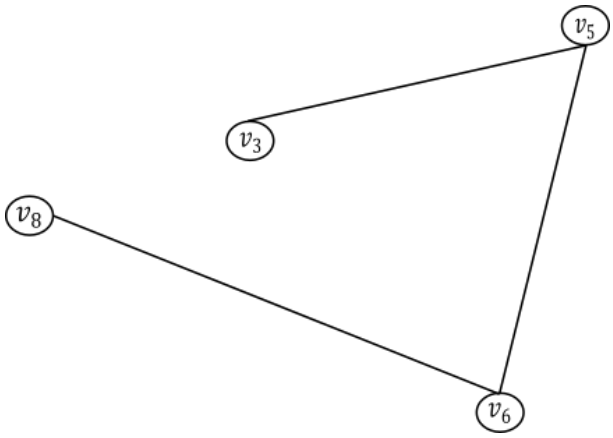


Figure 7. The reduced graph at the beginning of the second cycle $V_2 = \{v_2\},\ V_1 = \{v_3, v_5, v_6, v_8\}$.

Table 3. Situation at the beginning of the second cycle

a	*i*	*index*	*roman domination*	*gamma roman*
2	8	[2 5 4 1 3 6 7 8]	[0 2 1 0 1 1 0 1]	0

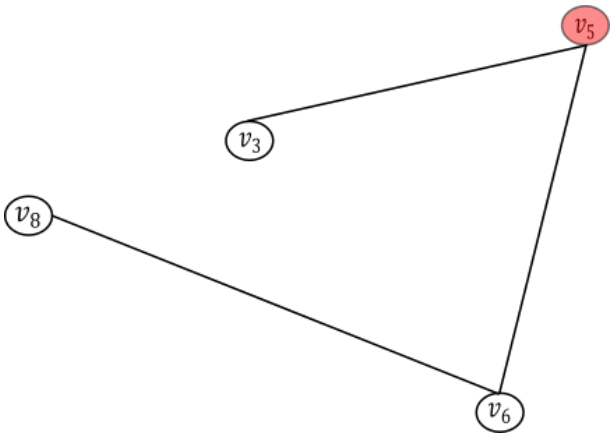


Figure 8. Second cycle of the algorithm $V_2 = \{v_2,\ v_5\},\ V_1 = \{v_3,\ v_6, v_8\}$.

Table 4. Second Iteration

a	*i*	*index*	*roman domination*	*gamma roman*
2	8	[5 6 3 8 0 0 0 0]	[0 2 1 0 2 1 0 1]	0

Figure 9. The reduced graph at the beginning of the third cycle $V_2 = \{v_2, v_5\},\ V_1 = \{v_8\}$.

Table 5. Situation at beginning of the third cycle

a	*i*	*index*	*roman domination*	*gamma roman*
3	8	[5 6 3 8 0 0 0 0]	[0 2 0 0 2 0 0 1]	0

In the end, our algorithm exits the loops by doing no changes in the *roman domination* matrix when only isolated vertices left because the $index[1]$ variable will be equal to 0 in every cycle. Then it calculates the result of the equation $\gamma_{R(G)} = \min\{|V_1| + 2|V_2|\}$ and returns the value, $\gamma_{R(G)} = 5$.

Table 6. Situation at the end

a	*i*	*index*	*roman domination*	*gamma roman*
8	8	[0 0 0 0 0 0 0 0]	[0 2 0 0 2 0 0 1]	5

3.1. Correctness of the Roman domination algorithm

The algorithm's correctness will be proven by demonstrating the dominance and minimality properties. Loop invariants and mathematical reasoning will be used to prove these properties.

Proof. Consider the input graph $G = (V, E)$ and set of vertices D selected by the algorithm. At the outset, all vertices in the Roman domination matrix of G are initially dominated by 1. During each iteration of the while loop, the algorithm chooses the remaining vertex with the highest degree, except in the case of equality when it also looks for the neighbor degree. The algorithm then updates the Roman domination values of the selected vertex and its adjacent vertices, setting the former to 2 and the latter to 0. This process continues until all vertices are removed. At each iteration, the algorithm sets the Roman domination value of the selected vertex to 2, which is greater than the value of any previously unselected vertex (which is 1). The algorithm terminates when all vertices have been removed from G. At this point, the set of vertices D selected by the algorithm dominates all other vertices. Every vertex has either been selected and given a Roman domination value of 2, or has been adjacent to a selected vertex, or has a Roman domination value of 1, and therefore dominates itself. Thus, we have demonstrated that the Dominance Property holds.

Let's assume there's a vertex, $v \in D$, that, if removed from D, would result in a new set D' that doesn't dominate all other vertices. Since v is in D, it must've been picked by the algorithm at some point during iteration k. If we were to remove v from D, no vertex adjacent to v would have its Roman domination value set to 0 in subsequent iterations. This means there's a vertex, u, adjacent to v where $roman\ domination[u] = 2$. However, this contradicts the algorithm's procedure, which ensures that the Roman domination value of any vertex adjacent to the selected vertex is set to 0. Therefore, our assumption was incorrect, and the set of vertices selected by the algorithm is minimal.

Applying the Dominance Property and Minimality Property, it can be inferred that the Roman Domination Algorithm accurately calculates the Roman domination number for any input graph. This algorithm guarantees that the chosen vertex set dominates all other vertices and is the smallest possible set.

□

3.2. Complexity of the Roman domination algorithm

The Roman Domination Algorithm's time complexity increases quadratically with the number of vertices in the input graph, which can cause performance issues when dealing with large graphs. However, for simple, undirected, and planar graphs, the overall time complexity can be estimated as $O(|V|) + O(|V|\log|V|)$. Planar graphs have a limited degree, which enables quick degree calculation in linear time, resulting in a lower time complexity compared to the general case where degree calculation has a complexity of $O(|V|^2)$.

In practice, the time complexity of the Roman Domination Algorithm for planar graphs is dominated by the sorting term $O(|V|log|V|)$ due to the necessity of sorting the vertices based on their degrees.

4. Conclusion

Our algorithm has undergone thorough testing on various established graph categories, including paths, cycles, and trees. Moreover, we have conducted tests on graph classes that have been

explored in the context of Roman domination numbers, as well as on their Mycielski graph counterparts. In addition, we have tested our algorithm on any graph generated by the Watts-Strogatz model using the MATLAB program, and we have validated the accuracy of our results through manual verification. In a recent publication, we conducted extensive calculations for different classes of graphs, with different numbers of vertices (Durgun & Toprakkaya, 2023). Our findings provided conclusive evidence regarding the algorithm's effectiveness.

It has been crafted to handle intricate data structures with ease, delivering top-notch results every time. By following our intuitive, user-friendly instructions and leveraging our helpful visual aids, our algorithm empowers users to interpret and analyze complex data structures with greater ease, ultimately resulting in more informed decisions and increased productivity.

References

Chambers, E. W., Kinnersley, B., Prince, N., & West, D. B. (2009). Extremal problems for Roman domination. *SIAM Journal on Discrete Mathematics*, *23*(3), 1575-1586. https://doi.org/10.1137/070699688

Cockayne, E. J., Dreyer Jr, P. A., Hedetniemi, S. M., & Hedetniemi, S. T. (2004). Roman domination in graphs. *Discrete Mathematics*, *278*(1-3), 11-22. https://doi.org/10.1016/j.disc.2003.06.004

Dreyer, P. A. (2000). *Applications and variations of domination in graphs*. (PhD), Rutgers University, New Jersey.

Durgun, D. D., & Toprakkaya, E. N. (2021). Roman domination of the Comet, Double Comet, and Comb Graphs. *arXiv preprint arXiv:2102.07902*. https://doi.org/10.48550/arXiv.2102.07902

Durgun, D. D., & Toprakkaya, E. N. (2023). Roman domination on some graphs. *Journal of Modern Technology and Engineering*, *8*(2), 96-104.

Haynes, T. W., Hedetniemi, S., & Slater, P. (1998). *Fundamentals of domination in graphs*. CRC Press. https://doi.org/10.1201/9781482246582

Henning, M. A., & Hedetniemi, S. T. (2003). Defending the Roman Empire - A new strategy. *Discrete Mathematics*, *266*(1-3), 239-251. https://doi.org/10.1016/S0012-365X(02)00811-7

Kazemi, A. P. (2012). Roman domination and Mycielski's structure in graphs. *Ars Combinatoria*, *106*, 277-287.

Liedloff, M., Kloks, T., Liu, J., & Peng, S. (2008). Efficient algorithms for Roman domination on some classes of graphs. *Discrete Applied Mathematics*, *156*(18), 3400-3415. https://doi.org/10.1016/j.dam.2008.01.011

Stewart, I. (1999). Defend the Roman Empire!, *Scientific American*, *281*(6), 136-138. https://doi.org/10.1038/scientificamerican1299-136

West, D. B. (2001). *Introduction to graph theory*. Prentice Hall Upper Saddle River.