THE ITALIAN DOMATIC NUMBER ON VARYING GRAPH FAMILIES

By

Keith G. Gallegos

Honors Scholarship Project

Submitted to the Faculty of

Olivet Nazarene University

for partial fulfillment of the requirements for

GRADUATION WITH UNIVERSITY HONORS

March, 2023

Bachelor of Science

in

Mathematics

Scholarship Project Advisor (printed)	Signature	Date
Honors Council Chair (printed)	Signature	Date
Honors Council Member (printed)	Signature	Date

ABSTRACT

A graph G is a mathematical object that consists of a set of vertices $V(G)$ and set of edges $E(G)$ such that an edge $e \in E(G)$ connects two distinct vertices in $V(G)$. An Italian dominating function $f : V(G) \to \{0, 1, 2\}$ assigns weights of 0, 1, or 2 to each vertex such that each vertex $v \in V(G)$ with a weight of 0 ($f(v) = 0$) has at least two neighbors $u_1, u_2 \in V(G)$ with weights of 1 ($f(u_1) = f(u_2) = 1$) or at least one neighbor $w \in V(G)$ with a weight of 2 ($f(w) = 2$). In short, if v is assigned a weight of 0, then $\sum_{w \in N(v)} f(w) \geq 2$ for $N(v)$ denoting all vertices adjacent to v. An Italian dominating family is a set of distinct Italian dominating functions $\{f_1,\ f_2,\ldots,\ f_n\}$ such that the sum of the weights of some vertex $v \in V(G)$ across all Italian dominating functions in this set does not exceed 2 for every vertex in the graph of G. In other words, an Italian dominating family must satisfy $\sum_{i=1}^{n} f_i(v) \leq 2$ for all $v \in V(G)$. The maximum cardinality of the Italian dominating family for G is called the Italian domatic number of G, denoted $d_I(G)$. In this report, we determine the Italian domatic number across varying families of Cartesian products, including $T_1 \square T_2$ for two trees that are not both stars, $C_{3r} \square C_{3s}$ for $r, s \geq 1$, $C_4 \square C_{3r}$ for $r \geq 1$, and $C_{5r} \square C_{5s}$ for $r, s \geq 1$. We also classify the Italian domatic number for all trees based upon the presence of a specific configuration in the tree.

Contents

1 Introduction

In the fields of computer science and mathematics, many problems can be measured or solved by examining connectivity between objects. One everyday example of this type of problem could be finding the shortest travel distance from one place to another on a map, while some less common examples include flight paths of planes and solving a maze. Graph theory is a field in mathematics that models and deals with these problems directly.

In graph theory, graphs are mathematical objects used to measure connectivity between arbitrary objects. Graphs in this context are not the same as algebraic graphs that model lines or functions. Graphs in graph theory consist of a collection of vertices along with edges that connect exactly two vertices. Everyday examples of graphs include family trees, tournament brackets, social networks, and maps with bordered territories. The map example may be hard to imagine, but if a territory represents a vertex of a graph while edges connect vertices of territories that are adjacent, then we can construct a graph with the vertices and edges described.

Graph theory has many useful applications in the real world, such as graph coloring or different graph covering problems. Graph coloring is used whenever it is desired that a graph be partitioned or grouped in a specific manner. Graph covering is where a specific set of vertices are chosen that "cover" or in some way account for all other vertices in the graph. One type of graph covering is known as domination, where a set of vertices from a graph is chosen such that all vertices not chosen to be part of this set are adjacent to at least one vertex that was chosen to be part of this set. Our research for this project focuses on a variation of domination known as Italian domination.

1.1 Terminology

Let $G = (V, E)$ be a simple undirected graph with vertex set $V = V(G)$ and edge set $E = E(G)$. The number of vertices in G (denoted $|V(G)|$) is called the order of G. For any two vertices $v_1, v_2 \in V(G)$, v_1 and v_2 are considered adjacent if there is an edge directly connecting the two vertices. The open neighborhood of a vertex $v \in V(G)$ is the set of all vertices in $V(G)$ adjacent to v. The open neighborhood of v is denoted $N(v)$. In the context of this paper, we will refer to this simply as the neighborhood of v. The degree of vertex v, denoted $deg(v)$ is equal to the number of vertices adjacent to v. In short, $deg(v) = |N(v)|$. The minimum degree of a graph G, denoted $\delta(G)$, is equal to the the smallest degree among all vertices in G, where $\delta(G) = min\{deg(v)|v \in V(G)\}$. The maximum degree of G, denoted $\Delta(G)$, is defined similarly to the minimum degree, where $\Delta(G) = max\{deg(v)|v \in V(G)\}$. Consider Figure 1 for some examples of graphs. For this paper, we work with specific graph structures and families. These specific groups include cycles, trees, paths, bipartite graphs, and Cartesian products.

Let a set of vertices $\{v_1, v_2, ..., v_m\}$ occur in a graph G where v_n is adjacent to v_{n+1} for all $n \in [1, m-1]$. If v_m is adjacent to v_1, then G contains a cycle of length m. If a graph G only consists of a cycle of length m, or in other words, there is no vertex outside of this cycle in G, then we denote this graph C_m. Figure 1(c) shows an example of a cycle on 8 vertices. Figure 1(b) is an example of a graph that contains cycles, but is not a cycle graph.

Trees are defined as any graph that has no cycle. An arbitrary tree will typically be denoted as T. Paths are a subset of trees where the maximum degree of the path graph is exactly 2. We denote paths on m vertices P_m. Figure 1(a) is an example of a tree.

Suppose a graph G can have its vertices partitioned into two distinct sets S_1 and S_2 such that $S_1 \cup S_2 = V(G)$. If any vertex in S_1 is only adjacent to vertices in S_2 and vice versa, G is said to be bipartite. In mathematical notation, for $v_1 \in S_1$ and $v_2 \in S_2$, then $N(v_1) \subseteq S_2$ and $N(v_2) \subseteq S_1$. If we were to color a bipartite graph red and blue for vertices in S_1 and S_2, respectively, then we can see that red vertices are only adjacent to blue vertices, and vice versa. This type of coloring is shown for C_8 in 1(c). It should also be noted that a graph G is bipartite if and only if G has no odd cycles. So if no odd cycles can be found in G, we can conclude that G is bipartite.

A Cartesian product of two graphs G and H, denoted $G\Box H$, is a graph defined on pairs of vertices $(g, h) \in V(G\Box H)$ where $g \in V(G)$ and $h \in V(H)$. The pair (g, h) is a single vertex in $G\Box H$. There is an edge between any two vertices (g_1, h_1) and (g_2, h_2) if and only if $g_1 = g_2$ and the edge between h_1 and h_2 is in $E(H)$ or $h_1 = h_2$ and the edge between g_1 and g_2 is in $E(G)$. There is also a straightforward, step-by-step

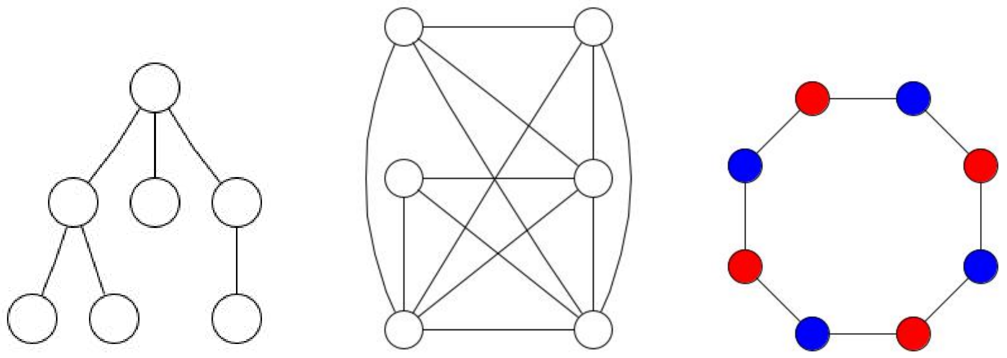
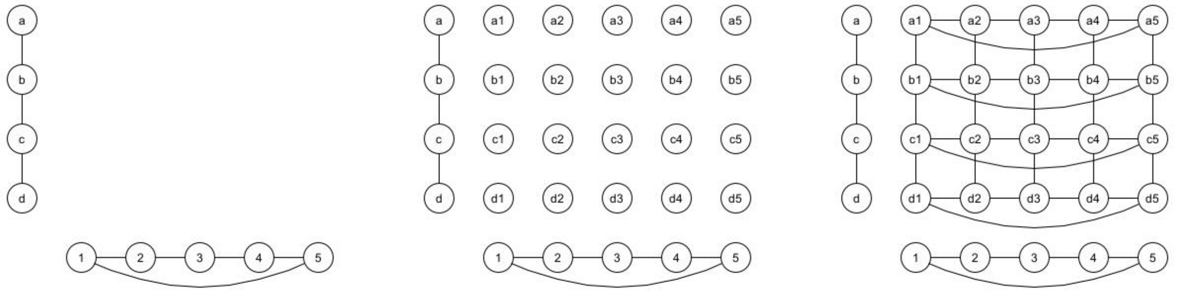

(a) (b) (c)

Figure 1: Examples of graphs



(a) (b) (c)

Figure 2: Creating a Cartesian product

way to create a Cartesian product of two graphs (say G_1 and G_2) using the following instructions:

1. Arrange G_1 and G_2 such that each appear linear. Orient one vertically and one horizontally illustrated in Figure 2(a)

2. Create vertices for the Cartesian product. The order of the Cartesian product should be the orders of G_1 and G_2 multiplied. Arrange them as shown in Figure 2(b)

3. Add edges. The edges for each column should be identical to the vertical graph, while the edges in each row should be identical to the horizontal graph. This is shown in Figure 2(c)

1.2 Research Specifics

A dominating set S on a graph G represents a subset of vertices in $V(G)$ such that for any $v \notin S$, v is adjacent to at least one vertex in S. In short, $N(v) \cap S \neq \emptyset$, for all $v \in V(G) \setminus S$ where $N(v)$ represents all vertices adjacent to v. Domination could also be defined using particular functions. Define a function $f : V(G) \to \{0, 1\}$ such that any vertex receives a weight of either 0 or 1. Then the support at f is a dominating set if for every $v \in V(G)$ such that $f(v) = 0$, then $\sum_{x \in N(v)} f(x) \geq 1$.

Domination on graphs have been studied extensively throughout series of papers such as [2], [9] and [13]. Through an article published by Ian Stewart [15] analyzing strategies for defending the Roman Empire, other concepts of domination were introduced. In Stewart's work, we see a map of the ancient Roman Empire demonstrating Constantine's plan for defending it given four groups of army legions. This can be seen in Figure 3. Constantine wanted to deploy these groups of army legions in a way such that they are deployed two groups at a time and such that every city in the ancient Roman Empire that did not have an army legion deployed in it was adjacent to a city that had exactly two army legions deployed in it. The idea behind

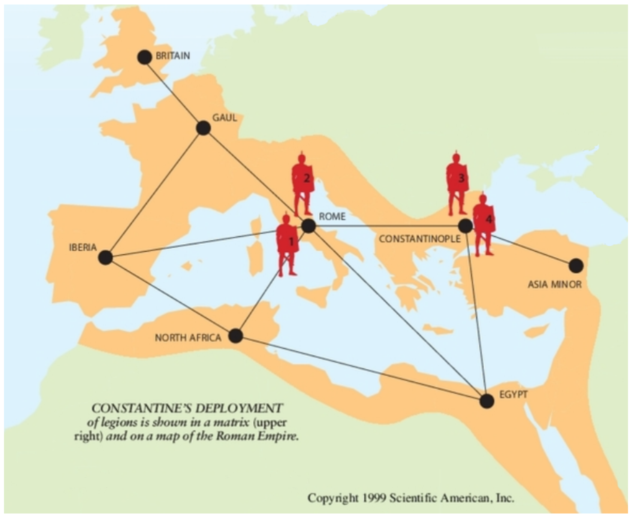

BRITAIN
GAUL
ROME
CONSTANTINOPLE
IBERIA
ASIA MINOR
NORTH AFRICA
EGYPT
1
2
3
4
CONSTANTINE'S DEPLOYMENT of legions is shown in a matrix (upper right) and on a map of the Roman Empire.
Copyright 1999 Scientific American, Inc.

Figure 3: Constantine's flawed attempt to defend the ancient Roman Empire given 4 groups of army legions

this is that for cities with no deployed legions, an adjacent city with two army legions deployed could send exactly one of its legions so that it could support the attacked city and still remain defended. The only problem was that given only four groups of these army legions, it would be impossible to defend the ancient Roman empire. The deployment demonstrated in Figure 3 has left Britain completely undefended.

Stewart's work ended up paving the way for two new variations of domination on graphs. These variations were later termed Roman domination and Roman {2}-domination. Roman domination was first introduced in a series of papers ([10], [6]) by Hedetniemi et al., while Roman {2}-domination first appeared in a paper by Chellali et al. [3]. Roman {2}-domination was termed Weak {2}-domination in [12] and later referred to as Italian domination in [11].

An Italian dominating function is a function $f : V(G) \rightarrow \{0, 1, 2\}$ (an assignment of weights to all vertices on G) such that for any vertex v where $f(v) = 0$, $\sum_{w \in N(v)} f(w) \geq 2$. For any Italian dominating function f, the sum of all weights throughout the function is known as the weight of the Italian domination function. The Italian domination number for G, $\gamma_I(G)$, is the minimum weight across all Italian dominating functions for G. Consider the graph shown in Figure 4(a). If all vertices except for vertex b were assigned a weight of 0 while vertex b was assigned a weight of 2, then we have constructed an Italian dominating function as shown in Figure 4(b) since all vertices with a weight of 0 are a neighbor to vertices whose weights sum to at least 2. This type of domination provides several alternative methods to successfully dominate the graph

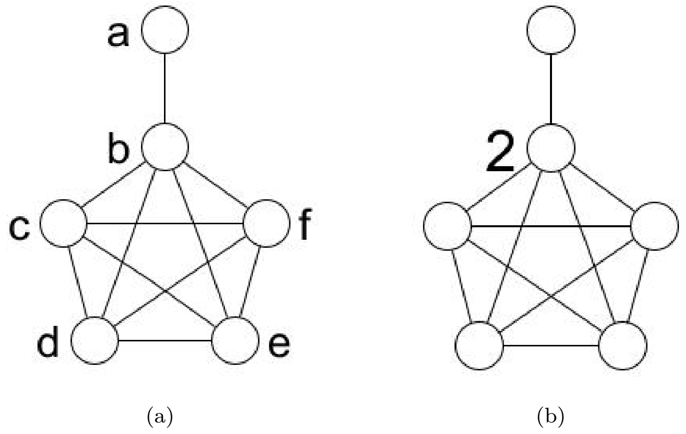


Figure 4: An example of an Italian dominating function

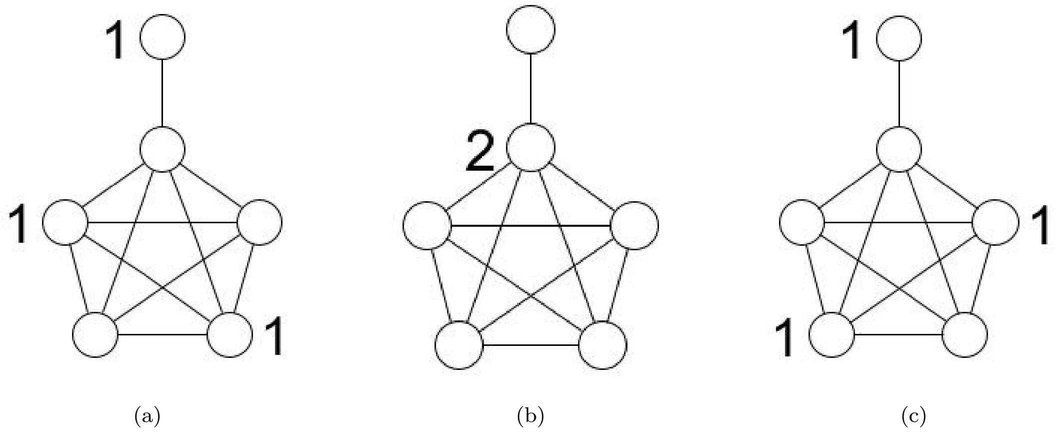


Figure 5: An example of an Italian dominating family that satisfies the Italian domatic number for a graph

pertaining to the ancient Roman Empire in Figure 3. Trial and error tells us that the Italian domination number for this graph is 4 since it cannot be dominated with a weight of 3. We can dominate this with a weight of 4 by giving Britain, Asia Minor, Iberia, and Egypt a weight of 1 to form one Italian dominating function. Another can be formed by giving Britain and Asia Minor a weight of 1 while giving Rome a weight of 2. From this perspective, Constantine could have defended the ancient Roman Empire if he would have deployed his 4 army legions in a similar manner as described above.

The domatic number of a graph G was first introduced in [5] by Hedetniemi and Cockayne, which stems from the terms "domination" and "chromatic". The domatic number of a graph G is the maximum number of dominating sets in G such that no 2 sets share any common vertex. The domatic number of G is denoted $d(G)$. A variation of the domatic number known as the Italian domatic number of a graph has also been extensively studied ([16], [9]). In [16], Volkmann defined the Italian domatic number on simple undirected graphs as follows: Let $\mathcal{F} = \{f_1, f_2, ..., f_t\}$ be a set of Italian dominating functions. If $\sum_{i=1}^{t} f_i(v) \leq 2$ for all vertices $v \in V(G)$, then this is called an Italian dominating family. Then the Italian domatic number of G is the maximum cardinality of any such set of distinct Italian dominating functions. We will denote the Italian domatic number of G as $d_I(G)$. See Figure 5 for an example. We refer to the same example graph used in Figure 4(a). By examining the same vertex in each Italian dominating function, it is clear that the sum of the weights assigned to this vertex across all functions adds up to at most 2. This is enough to show that the Italian domatic number for this graph is at least 3. The following theorem allows us to conclude that it must be at most 3 and as a result, conclude that the Italian domatic number is 3 for this graph.

Theorem 1. [16] *For any graph G, $d_I(G) \leq \delta(G) + 2$ for $\delta(G)$ equal to the minimum degree of G.*

As far as lower bound is concerned, we know that $d(G) \leq d_I(G)$. We can define an Italian dominating function that corresponds to a dominating function by assigning a value of 2 to any vertex in the dominating set for G.

Returning to the graph of the ancient Roman Empire in Figure 3, we have already formed two distinct Italian dominating functions for this graph that satisfy the Italian domatic number. A third can be formed if we were to give Gaul and Constantinople a weight of 2 while giving Iberia, North Africa, and Egypt a weight of 1. These Italian dominating functions are shown in Figure 6, where a 3-tuple on vertex v represents ($f_1(v)$, $f_2(v)$, $f_3(v)$). In other words, the 3-tuples represent the weight distributions of a vertex in the first, second, and third Italian dominating function for the graph, respectively. These Italian dominating functions, along with Theorem 1, allow us to conclude that the Italian domatic number for the graph of the ancient Roman Empire is 3. However, one of the functions in this Italian dominating family has a weight of 7, which is greater than the number of army legions Constantine could work with. Although this is undesired, the Italian dominating family that corresponds to this graph's Italian domatic number certainly contains a number of Italian dominating functions of minimum weight, which in this case is 4.

The following theorems come from a series of papers and, along with Theorem 1, will also be used consistently throughout this research:

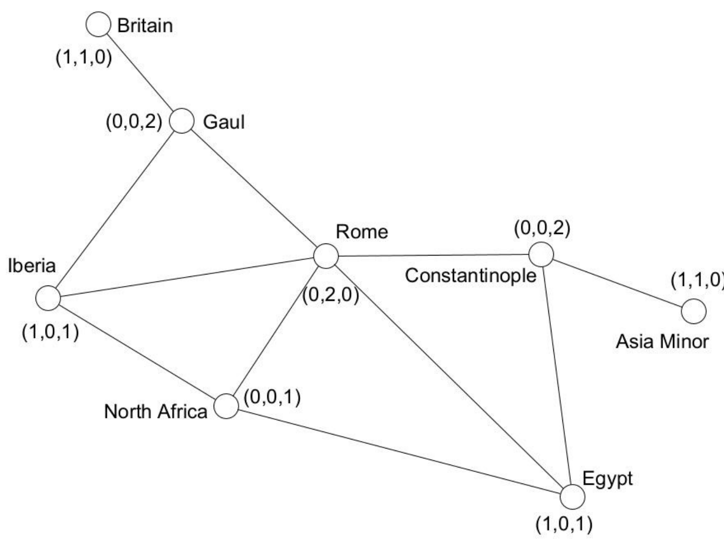


Figure 6: A version of the graph of the ancient Roman Empire from Figure 3 that includes weight distributions for an Italian dominating family

Theorem 2. [16] *For any graph G, $\gamma_I(G)d_I(G) \leq 2n$ for n equal to the order of G.*

Theorem 3. [3] *For a connected graph G with order n and maximum degree $\Delta(G)$, $\gamma_I(G) \geq \lceil \frac{2n}{\Delta(G)+2} \rceil$.*

2 Methodology

This work has two main focuses or areas of research. The first of these is to classify the Italian domatic number on some classes of Cartesian products, while the second is to classify the Italian domatic number of all trees. Both of these areas of research have varying motivations, and as a result have varying methodologies or methods of proof. For this reason, it is appropriate to discuss each methodology individually.

2.1 Cartesian Products

In research conducted on the Italian domatic number, there has been little work so far done on the Italian domatic number of Cartesian products. The goal for this section is to focus on Cartesian products that did not have a large Italian domatic number. By Theorem 1, we know that the Italian domatic number directly relates to how large the minimum degree of a graph is. Since the minimum degree of any Cartesian product is the sum of the minimum degrees of the two graphs used to produce it, we decided to prioritize Cartesian products of graphs with small minimum degrees. For this research, this includes trees (with minimum degree 1) and cycles (with minimum degree 2).

In the case for trees, it is more accurate to say that our motivation for this area of research stems from results about the Italian domatic number of paths. In [16], Volkmann shows that for a path on n vertices (denoted P_n), $d_I(P_n) = 3$ for $n \geq 6$. Because we had this result, we initially focused on paths. We know that $d_I(P_n \square P_m) \leq 4$, so our goal was to attempt to find an Italian dominating family of four distinct Italian dominating functions for the Cartesian product of two paths $P_n \square P_m$. By partitioning the Cartesian product of two paths (one path of at least length 4) into two bipartite graphs with minimum degree of at least 2, we were able to find the Italian domatic number for Cartesian products of paths. The same method worked for trees, where we could find the Cartesian product of two trees (such that one tree is not a star). The methods of proof for Cartesian products of trees (and subsequently, paths) consisted of partitioning the graph of the Cartesian product into two separate bipartite graphs. Partitioning these Cartesian products in such a way allowed us to generate a result for Cartesian products of any two trees that are not both stars. This result is given later in the paper.

After focusing on Cartesian products of graphs with a minimum degree of 1, we moved on to Cartesian products of graphs with minimum degree 2. The only connected graphs with a minimum degree of 2 are cycles, where C_n denotes a cycle of length n. This was our next logical step since the only difference between P_n and C_n is an edge connecting the two vertices of degree 1 in P_n to form C_n. However, this increases our minimum degree from 1 to 2, and the minimum degree of any Cartesian product of two cycles is now 4. Because of this, the methods of proof drastically change compared to Cartesian products of trees. In fact, we could only classify the Italian domatic number for some families of Cartesian products of cycles. These include $C_{3r}\square C_{3s}$, $C_4\square C_{3s}$, and $C_{5r}\square C_{5s}$ for positive integers r, s. For all three families, we initially found the Italian domatic number for each class when both r and s equal 1 and classified each of these findings as observations. We then used these along with Theorems 2 and 3 to improve the bound for d_I in Theorem 1 and to solve for the Italian domatic number of graphs in these families. However, to potentially improve the upper bound for the Italian domatic number using Theorem 2, we require some value for the Italian domination number γ_I. For the first two families, we use Lemma 4, which is introduced in this subsection to find γ_I.

Now that we have a way to classify the Italian domination number for each family of Cartesian products of cycles we work with, we can use Theorem 2 to find an upper bound that is potentially improved from that given in Theorem 1. As far as lower bounds are concerned, we typically use our observations or results found for each family where $r, s = 1$ and extend an Italian dominating family for the whole class that satisfies the Italian domatic number based on the Italian dominating family formed for these observations. If we are able to find an Italian dominating family with the same number of distinct functions as our upper bound for d_I from Theorem 2, then we have found the Italian domatic number for all graphs in the family. Our results for these classes of Cartesian products of cycles are shown later in this paper.

2.2 Trees

For the Italian domatic number on trees, our methodology is built around a somewhat experimental process, with a hypothesis, experimental procedure, results from the experimentation, and a conclusion. The only difference is that we must prove that the results yielded from the experimental procedure are true. In the case for trees, we know from [16] that for a graph G, $d_I(G) = 1$ if and only if G is an empty graph (a graph with 0 edges). We also know from Theorem 1 that for any tree T, $d_I(T) \leq 3$, meaning that any tree has an Italian domatic number of either 2 or 3. So our "hypothesis" or focus is to determine which class of trees have an Italian domatic number of 2, and subsequently determine which trees have an Italian domatic number of 3.

To start our "experimental procedure," we create an algorithm that outputs all trees on a specified number of vertices that do not have an Italian domatic number of 3. Since trees are graphs that contain no cycles, it is easier to use an iterator that loops through all possible trees on some number of vertices n. The specific details of the algorithm are outlined in another section, but our purpose for the algorithm is to determine all trees that do not have an Italian dominating family comprised of 3 Italian dominating functions. It should be noted that part of the requirements to form an Italian dominating family is that all functions in the family are distinct, but for this algorithm, we were unable to implement a way to determine if functions in a possible Italian dominating family for a tree were distinct. So our output for the algorithm provides us with trees that cannot be dominated with 3 Italian dominating functions, regardless of distinction. However, we used the concept of trees having Italian dominating families comprised of non-distinct Italian dominating functions to prove additional results about the Italian domatic number of trees. Because of the inability for the algorithm to identify distinct Italian dominating functions in trees, we decided to divide this section into subsections, one for classifying trees that can be dominated 3 (not necessarily distinct) Italian dominating functions that would otherwise satisfy the Italian domatic number, and another for classifying which trees can be dominated with 3 distinct Italian dominating functions.

For the first subsection, after obtaining our results from the algorithm, we focus on classifying trees that cannot be dominated with 3 Italian dominating functions (regardless of distinction). Since our output for the algorithm are trees that fall into this class, we analyze each tree in our output and try to find a pattern or form a conjecture for why these trees fail to contain 3 Italian dominating functions. We were able to find one structure that was common in all trees in the output for our algorithm. This structure is shown in Figure 10 later in the paper. The goals for this section are to show that the presence of this structure in any tree T implies that $d_I(T) = 2$ and to classify which trees do not contain this structure but still have an Italian domatic number of 2.

Once we classify trees that cannot be dominated with 3 (not necessarily distinct) Italian dominating functions, we move on to our next subsection about classifying those trees that can be dominated with 3 distinct Italian dominating functions. One tool that was essential for almost all proofs in this subsection is Lemma 24, or the Cut Edge Lemma. This essentially states that if a graph can be partitioned into two separate graphs that each have an Italian dominating family of three Italian dominating functions, then the original graph before partition must have an Italian domatic number of 3. We prove this lemma in this subsection, and onward from here most proofs that involve showing a tree T (in some class for trees) has an Italian domatic number of 3 involve demonstrating that an edge cut that satisfies Lemma 24 exists in T. There are some exceptions to this. The end goal for this subsection is to use the results from the previous subsection along with the results formed in this one to classify all trees that have an Italian domatic number of 3 and subsequently, those that have an Italian domatic number of 2.

3 Italian Domatic Number of Cartesian Products

A majority of this section will focus on the Italian domatic number of the Cartesian product of cycles. One way we do this is using Theorem 2, but this requires an Italian domination number γ_I. Listed below are two methods for finding the Italian domination number of different classes of Cartesian products of cycles:

Lemma 4. [8] *For $n \equiv 0 \pmod 3$,*

$$\gamma_I(C_n \Box C_m) = \begin{cases} \frac{mn}{3}, & m \equiv 0 \pmod 3 \\ \frac{mn+n}{3}, & m \equiv 1, 2 \pmod 3 \end{cases}$$

Theorem 5. [14] *For an integer n, $\gamma_I(C_5 \Box C_n) = 2n$ for $n \geq 5$.*

Along with these statements and Theorem 2, we have another method for finding lower bounds for the Italian domatic number of Cartesian products. However, we must first show a separate result:

Theorem 6. *Suppose G is a graph that can be partitioned into n connected components G_1, G_2, ... G_n through any number of edge cuts on G. Then $d_I(G) \geq \min\{d_I(G_1), d_I(G_2), ..., d_I(G_n)\}$.*

Proof. Let G be a graph that can be partitioned into n connected components G_1, G_2, ... G_n. Let $\min\{d_I(G_1), d_I(G_2), ..., d_I(G_n)\} = d$. Since $d_I(G_1)$, $d_I(G_2)$, ..., $d_I(G_n) \geq d$, there exists a corresponding family of at least d distinct Italian dominating functions for all G_1 through G_n. For each family, choose the first d Italian dominating functions from each. We will denote the first d Italian dominating functions as f_x^y, where $1 \leq x \leq d$ and $1 \leq y \leq n$ such that x represents xth function in the Italian dominating family for G_y. Now define the Italian dominating family for G as follows:

$$f_i(v) = \begin{cases} f_i^1 & v \in G_1 \\ f_i^2 & v \in G_2 \\ ... & \\ f_i^n & v \in G_n \end{cases}$$

For $1 \leq i \leq d$, we can form d distinct Italian dominating functions for the Italian dominating family for G, showing that $d_I(G) \geq d$, or in other words, $d_I(G) \geq \min\{d_I(G_1), d_I(G_2), ..., d_I(G_n)\}$. □

As a result from the above theorem, we can show a separate result specific to Cartesian products of any two graphs.

Corollary 7. *For any two graphs G and H, $d_I(G \Box H) \geq \max\{d_I(G), d_I(H)\}$.*

Proof. Without loss of generality (WLOG), suppose that $\max\{d_I(G), d_I(H)\} = d_I(G) = d$. In this case, we can partition $G \Box H$ into d partitions of the graph G by removing all edges from H prior to creating the Cartesian product. Since all partitions are identical, they all must have the same Italian domatic number. From Theorem 6, we can conclude that $d_I(G \Box H) \geq d$, or in other words, $d_I(G \Box H) \geq \max\{d_I(G), d_I(H)\}$. □

Although Corollary 7 provides a lower bound for the Italian domatic number of Cartesian products, this lower bound can often be improved upon. For graphs with large minimum degrees, it is often the case that Corollary 7 implies that the Italian domatic number is greater than some value less than the minimum degree of the Cartesian product. With all graphs that have been examined throughout this research, the Italian domatic number has never been less than the minimum degree of its corresponding graph. Although we have no way to prove this, it should be noted that the lower bound provided in this corollary can often be improved upon. However, before stating specific results of the Italian domatic number on Cartesian products, we must first define a specific class of trees.

Trees have many subclasses with specific properties. In particular, one subclass of trees are known as stars. Stars are trees on n vertices such that $n-1$ of these vertices are all leaves adjacent to the same vertex. In other words, if a tree has at least two vertices of degree 2 or more, then it is not a star. We will now continue on our research with Cartesian products, specifically starting with Cartesian products of trees.

Theorem 8. *For any graph $G = T_1 \Box T_2$ where T_1 and T_2 are trees that are not both stars, $d_I(G) = 4$.*

Proof. Given two tree graphs that not both stars, T_1 and T_2, let's say WLOG, T_1 is not a star. Since both T_1 and T_2 have leaves, $\delta(G) = 2$. By Theorem 1, the Cartesian product G has an Italian domatic number of at most 4. We want to show that it must be 4. Since trees are graphs with no cycles, any edge removal from a tree will result in two disjoint graphs as components. First, perform an edge removal operation, $T_1 - \{e\}$, such that the resulting graphs T_1' and T_1'' both have an order of n_1 and n_2, respectively, where $n_1, n_2 > 1$. Let H and J denote two new graphs such that $H = T_1' \Box T_2$ and $J = T_1'' \Box T_2$. H and J are both bipartite graphs, so Theorem 6 implies that $d_I(G) \geq 2$ since the Italian domatic number of any bipartite graph is at least 2. However, we can improve this bound with careful construction of Italian dominating functions using the 2 partitions of H and J. Let A, B represent the partitions of H and C, D represent the partitions of J. For all vertices $v \in H$, $f_A(v)$ and $f_B(v)$ can be represented as the following:

$$f_A(v) = \begin{cases} 1 & v \in A \\ 0 & v \in B \end{cases}$$

$$f_B(v) = \begin{cases} 1 & v \in B \\ 0 & v \in A \end{cases}$$

We can also represent the partitions of J, $f_C(v)$ and $f_D(v)$, similarly:

$$f_C(v) = \begin{cases} 1 & v \in C \\ 0 & v \in D \end{cases}$$

$$f_D(v) = \begin{cases} 1 & v \in D \\ 0 & v \in C \end{cases}$$

Since H is a bipartite Cartesian product with $\delta(H) = 2$, every vertex in partition A will be adjacent to at least 2 vertices in partition B, and vice versa. Similarly, we can say the same about partitions C and D for J. This means that f_A, f_B, f_C, and f_D are all Italian dominating functions. Let f_{AC}, f_{AD}, f_{BC}, and f_{BD} be defined for G as follows:

$$f_{AC}(v) = \begin{cases} f_A(v) & v \in H \\ f_C(v) & v \in J \end{cases}$$

$$f_{AD}(v) = \begin{cases} f_A(v) & v \in H \\ f_D(v) & v \in J \end{cases}$$

$$f_{BC}(v) = \begin{cases} f_B(v) & v \in H \\ f_C(v) & v \in J \end{cases}$$

$$f_{BD}(v) = \begin{cases} f_B(v) & v \in H \\ f_D(v) & v \in J \end{cases}$$

Because f_A, and f_B are distinct Italian dominating functions on H and f_C and f_D are distinct Italian dominating functions on J, f_{AC}, f_{AD}, f_{BC}, and f_{BD} are distinct Italian dominating functions on G. In the Italian dominating family for G formed by f_{AC}, f_{AD}, f_{BC}, and f_{BD}, the sum of the weights of every vertex across the family is exactly 2. Since there are four families, $d_I(G) = 4$. □

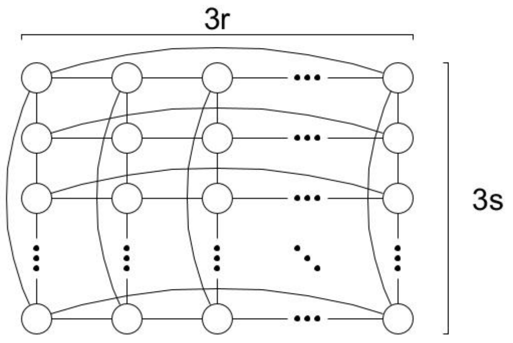


Figure 7: General form of G

Corollary 9. *For two paths, P_m and P_n, where $m > 3$ or $n > 3$, then $d_I(P_m \Box P_n) = 4$.*

Proof. Since paths are a subset of tree graphs, P_m and P_n are both trees. Theorem 8 applies since the only star graphs that are paths are paths with an order less than or equal to 3. It follows that $d_I(P_m \Box P_n) = 4$. □

The following definition from Gao et al. [4] provides a useful way to refer to different components of Cartesian products. This definition will be used throughout the proof of Theorem 11.

Definition 10 (Fiber)**.** *Let the fiber of any Cartesian product $G \Box H$ be the subgraph of $G \Box H$ induced by $V(G) \times \{h\}$ for some vertex $v \in H$.*

Theorem 11. *Suppose $G = C_{3r} \Box C_{3s}$ where C_{3r} and C_{3s} are cycles with orders $3r$ and $3s$, respectively, for $r, s > 1$. Then $d_I(G) = 6$.*

Proof. First, we note that the minimum degree of the Cartesian product of any two cycles is 4, so that $d_I(G) \leq \delta(G) + 2 = 6$. Since G is a Cartesian product of two cycles, G will have the form presented in Figure 7. Let V_t denote the vertices of a fiber of $C_{3r}^{C_t}$, where $C_{3r}^{C_t}$ is $V(C_{3r}) \times \{t\}$ for $t \in C_{3s}$ such that $v_0 \in V_t$ represents the first vertex, $v_1 \in V_t$ represents the second, and so on for $0 \leq t \leq 3r$. Let f_0, f_1, and f_2 represent different weight distributions for the set V_t, shown below:

$$f_i(v_t) = \begin{cases} 1, & t \equiv 0 \pmod 3 \\ 0, & t \not\equiv 0 \pmod 3 \end{cases}$$

$$f_j(v_t) = \begin{cases} 1, & t \equiv 1 \pmod 3 \\ 0, & t \not\equiv 1 \pmod 3 \end{cases}$$

$$f_k(v_t) = \begin{cases} 1, & t \equiv 2 \pmod 3 \\ 0, & t \not\equiv 2 \pmod 3 \end{cases}$$

Define a function f_{ijk}, where $i \neq j \neq k$, as the following:

$$f_{ijk} = \begin{cases} f_i(v) \text{ if } v \in C_{3r}^{C_t}, & t \equiv 0 \pmod 3 \\ f_j(v) \text{ if } v \in C_{3r}^{C_t}, & t \equiv 1 \pmod 3 \\ f_k(v) \text{ if } v \in C_{3r}^{C_t}, & t \equiv 2 \pmod 3 \end{cases}$$

For any vertex $t \in G$ that has a weight of 1, t has has a position equivalent to 0, 1, and 2 modulo 3 in every three consecutive fibers for $C_{3r}^{C_t}$. This means that every vertex with a weight of 0 is dominated by exactly two vertices (1 vertex in a similar row and similar column) with weight 1 since any position modulo 3 will be adjacent to both remaining positions in the fiber. This means f_{ijk} is an Italian dominating function. Six such functions exist that satisfy the Italian domatic number, f_{012}, f_{021}, f_{102}, f_{120}, f_{201}, and f_{210}. As a result, $d_I(G) \geq 6$. Because $d_I(G)$ is bounded below and above by 6, $d_I(G) = 6$. □

Observation 12. $d_I(C_3 \Box C_4) = 4$

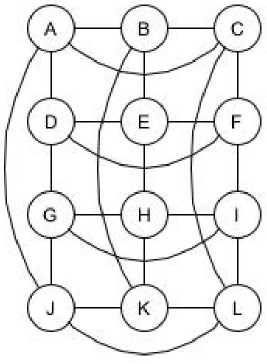


Figure 8: $C_3\square C_4$

	A	B	C	D	E	F	G	H	I	J	K	L
f_1	0	0	1	0	1	0	1	0	0	1	0	1
f_2	0	1	0	1	0	0	0	0	1	0	1	1
f_3	1	0	0	0	0	1	0	1	0	1	1	0
f_4	1	1	1	0	0	0	1	1	1	0	0	0

Table 1: Family of Italian dominating functions for $C_3\square C_4$

Proof. First note Figure 8 and the corresponding table for it. The figure is a graph $C_3\square C_4$, while the table represents an Italian dominating family that satisfies the Italian domatic number. Since there are four functions in the family, $d_I(G) \geq 4$. Now refer to Theorem 2, which states that $\gamma_I(G)d_I(G) \leq 2n$ for any graph G with order n. Since $3 \equiv 0 \pmod 3$, Lemma 4 applies for $C_3\square C_4$, meaning that $\gamma_I(C_3\square C_4) = \frac{4(3)+3}{3} = 5$. We can substitute this value and $3(4) = 12$ for $\gamma_I(G)$ and n, respectively, into our equation and obtain $5d_I(G) \leq 24$. Solving for $d_I(G)$ results with $d_I(G) \leq 4.8$. Since the Italian domatic number has to be an integer, we can conclude that $d_I(G) \leq 4$. Since $d_I(G)$ is bounded below and above by 4, $d_I(G) = 4$. □

Theorem 13. *For $n \equiv 0 \pmod 3$, $d_I(C_4\square C_n) = 4$*

Proof. From Theorem 2, we know that $\gamma_I(G)d_I(G) \leq 2s$ for some arbitrary graph G with order s. We can substitute $\frac{mn+n}{3}$ for $\gamma_I(G)$ using Lemma 4, along with substituting s for mn since the order of any Cartesian product of two graphs is the product of the order of the two individual graphs used to create it. We are interested in the case for $m = 4$, so we will make this substitution as well. After these substitutions, we obtain $\frac{4n+n}{3}d_I(G) \leq 8n$. By solving for $d_I(G)$, we obtain $d_I(G) \leq \frac{24}{5}$, meaning that the Italian domatic number of any $C_4\square C_n$ will be unaffected for any $n \equiv 0 \pmod 3$. Because of this, we know $d_I(C_4\square C_n) = d_I(C_4\square C_3) = 4$ from Observation 12 for any $n \equiv 0 \pmod 3$. □

Theorem 5 allows us to find the Italian domination number of graphs of the form $C_5\square C_n$ for any $n \geq 5$. Although the family of Cartesian products we will work with deviate from this family slightly, it still proves useful for the following observation:

Observation 14. $d_I(C_5\square C_5) = 5$

Proof. Consider Figure 9 and Table 2. The table represents a family of 5 distinct Italian dominating functions that satisfy the Italian domatic number for the graph of $C_5\square C_5$. This example shows that $d_I(C_5\square C_5) \geq 5$. According to Theorem 5, we know that $\gamma_I(C_5\square C_5) = 10$. From Theorem 2, we know that $\gamma_I(C_5\square C_5)d_I(C_5\square C_5) \leq 50$. By substituting, $10d_I(C_5\square C_5) \leq 50$, resulting in $d_I(C_5\square C_5) \leq 5$. Since $d_I(C_5\square C_5)$ is bounded above and below by 5, $d_I(C_5\square C_5) = 5$. □

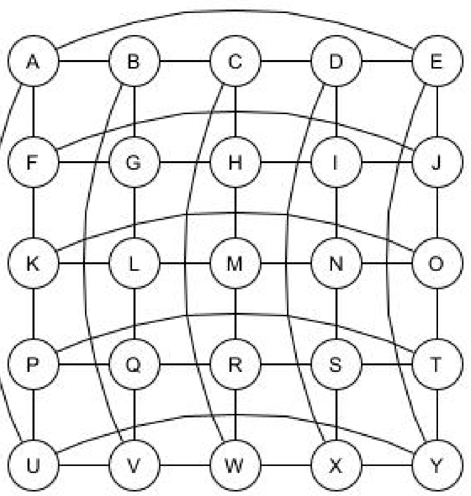


Figure 9: General form of $C_5 \Box C_5$

	A	B	C	D	E	F	G	H	I	J	K	L	M
f_1	0	1	0	1	0	1	0	0	0	1	0	0	2
f_2	0	0	1	0	1	1	1	0	0	0	0	0	0
f_3	1	0	0	1	0	0	1	1	0	0	0	0	0
f_4	0	1	0	0	1	0	0	1	1	0	2	0	0
f_5	1	0	1	0	0	0	0	0	1	1	0	2	0

	N	O	P	Q	R	S	T	U	V	W	X	Y
f_1	0	0	1	0	0	0	1	0	1	0	1	0
f_2	2	0	1	1	0	0	0	0	0	1	0	1
f_3	0	2	0	1	1	0	0	1	0	0	1	0
f_4	0	0	0	0	1	1	0	0	1	0	0	1
f_5	0	0	0	0	0	1	1	1	0	1	0	0

Table 2: Family of Italian dominating functions for $C_5 \Box C_5$

Theorem 15. *For constants $r,s \geq 1$, $d_I(C_{5r} \Box C_{5s}) \geq 5$*

Proof. We will consider induction to first show that $d_I(C_5 \Box C_{5s}) = 5$, and then show $d_I(C_{5r} \Box C_{5s}) = 5$. Refer to Observation 14 for a base case. Suppose that there exists some positive integer k such that $d_I(C_5 \Box C_{5k}) \geq 5$. For this example, we can form an Italian dominating family for $C_5 \Box C_{5k}$ similarly to $C_5 \Box C_5$ in Observation 14. By dividing the graph into 5x5 subsections and distributing weights to each subsection such that they are identical to exactly one Italian dominating function from the table corresponding to Figure 9. This process can be repeated five times, once for each function in the table for Figure 9. By appending another 5x5 subsection to the graph of $C_5 \Box C_{5k}$ so that we form $C_5 \Box C_{5(k+1)}$, we can distribute weights to this additional subsection in the same manner. Any vertices along the right-hand side of $C_5 \Box C_{5k}$ that were being dominated from another vertex on the left hand side of the graph will no longer be adjacent, but will remain dominated in $C_5 \Box C_{5(k+1)}$ since the weight distribution in the appended 5x5 subsection is identical to the weight distributions of all other 5x5 subsections. Five distinct Italian dominating functions can be formed since there are five distinct Italian dominating functions for the Italian dominating family of $C_5 \Box C_5$, showing that $d_I(C_5 \Box C_{5(k+1)}) \geq 5$. This implies that $d_I(C_5 \Box C_{5s}) \geq 5$ by induction. We will use induction again to show $d_I(C_{5r} \Box C_{5s}) \geq 5$. Consider our previous proof for $d_I(C_5 \Box C_{5s}) = 5$ as a base case for any positive integer s. Suppose there exists a positive integer l such that $d_I(C_{5l} \Box C_{5s}) \geq 5$. Similar to before, an Italian dominating family of 5 Italian dominating functions can be formed if every 5x5 subsection of $C_{5l} \Box C_{5s}$

have identical weight distributions that match one of the 5 functions in the table for Figure 9. We can form the graph of $C_{5(l+1)}\Box C_{5s}$ by appending a 5x5s subsection to the graph of $C_{5l}\Box C_{5s}$. If we distribute weights to the new subsection such that the weight distribution for this is identical to the weight distributions to all of the 5x5s subsections of $C_{5l}\Box C_{5s}$, we can form 5 distinct Italian dominating functions that satisfy the Italian domatic number for $C_{5(l+1)}\Box C_{5s}$, meaning $d_I(C_{5(l+1)}\Box C_{5s}) \geq 5$. This implies that $d_I(C_{5r}\Box C_{5s}) \geq 5$ by induction. □

Corollary 16. *For constants $r, s \geq 1$, we have the following:*

1. $d_I(C_{5r}\Box C_{5s}) = 6$*, if* $r, s \equiv 0 \pmod 3$
2. $d_I(C_{5r}\Box C_{5s}) = 5$ otherwise

Proof. Consider the following cases:

1. $r, s \equiv 0 \pmod 3$
2. $r \not\equiv 0 \pmod 3$ or $s \not\equiv 0 \pmod 3$
3. $r, s \not\equiv 0 \pmod 3$

For the first case, it is clear from Theorem 11 that $d_I(C_{5r}\Box C_{5s}) = 6$ since $5r$ and $5s$ are both multiples of 3.

For the second case, WLOG, suppose that $r \not\equiv 0 \pmod 3$ and $s \equiv 0 \pmod 3$. From Lemma 4, we know that $\gamma_I(C_{5r}\Box C_{5s}) = \frac{25rs+5s}{3}$. We can substitute this value into the equation in Theorem 2 to get $\frac{25rs+5s}{3} d_I(C_{5r}\Box C_{5s}) \leq 50rs$. By solving for $d_I(C_{5r}\Box C_{5s})$, we obtain $d_I(C_{5r}\Box C_{5s}) \leq \lfloor \frac{30r}{5r+1} \rfloor$. For any value $r \not\equiv 0 \pmod 3$, $d_I(G) \leq 5$. We know from Theorem 15 that $d_I(C_{5r}\Box C_{5s}) \geq 5$, so $d_I(C_{5r}\Box C_{5s}) = 5$ since it is bounded above and below by 5

For the final case, we know that $\gamma_I(C_{5r}\Box C_{5s}) \geq 5$ from Theorem 15. For an upper bound, we may consider Theorems 2 and 3. From Theorem 3, we know that $\gamma_I(C_{5r}\Box C_{5s}) \geq \lceil \frac{2n}{\Delta(G)+2} \rceil = \lceil \frac{50rs}{6} \rceil$. Since 50, r, and s are not multiples of 3, $\lceil \frac{50rs}{6} \rceil > \frac{50rs}{6}$. By Theorem 2, we know $\gamma_I(C_{5r}\Box C_{5s}) d_I(C_{5r}\Box C_{5s}) \leq 50rs$. This implies $d_I(C_{5r}\Box C_{5s}) \leq \frac{50rs}{\gamma_I(C_{5r}\Box C_{5s})} \leq (50rs)/\lceil \frac{50rs}{6} \rceil < (50rs)/\frac{50rs}{6} = 6$. Since $d_I(C_{5r}\Box C_{5s}) < 6$, $d_I(C_{5r}\Box C_{5s}) \leq 5$. By Theorem 1, this implies that $d_I(C_{5r}\Box C_{5s}) = 5$.

From the cases above, $d_I(C_{5r}\Box C_{5s}) \geq 5$, with equality when $r \not\equiv 0 \pmod 3$ or $s \not\equiv 0 \pmod 3$ □

4 Italian Domatic Number Algorithm for Trees

For this section, the goal is to determine the Italian domatic number for all trees. In [16], it has been proven that if $d_I(G) = 1$, then G must be the empty graph (a graph with no edges). This paper also contains Theorem 1, which states that the maximum bound for the Italian domatic number of any graph G is $\delta(G) + 2$. Since all trees have a minimum degree of 1, this allows us to conclude that for any tree T, $d_I(T) = 2$ or $d_I(T) = 3$. This makes the goal for this section determining which trees have an Italian domatic number of 2. Any tree outside of this group will have an Italian domatic number of 3.

The first step to classifying trees by their Italian domatic number was to create an algorithm that can determine the number of Italian dominating families a given tree can contain. It should be noted that for this section, we modify our definition of an Italian dominating family such that distinct Italian dominating functions are no longer a requirement to form an Italian dominating family.

The algorithm in question is a dynamic programming algorithm developed on SageMath. In short, it starts with the leaves of the trees and works its way up to the root, determining the cardinality of the Italian dominating family for each subtree induced by varying vertices throughout the tree. The following definitions allow us to describe the algorithm in greater detail.

Definition 17 (Basic State)**.** *A state that a vertex v can take in an Italian dominating family for a tree T that satisfies the Italian domatic number for T.*

The basic states for the algorithm are defined below.

1. The state "A", which indicates v is being dominated by another vertex with weight 2 from above

2. The state “B”, which indicates v is being dominated from a total weight of 2 below across all children
3. The state “S”, which indicates v is being dominated from at least a weight of 1 above and a weight of 1 below
4. The state “O”, which indicates v contains a weight of 1
5. The state “T”, which indicates v contains a weight of 2

The basic states defined above assumes that the tree is constructed with the root above all other vertices in the graph. The children would be directly below the root, and so on until we reach the leaves. So when we say a vertex is dominated above, we assume it sees a weight of 2 from the its parent above, or from the neighbor closest to the root. Being dominated from below is defined similarly, while containing a certain weight implies an Italian dominating function assigns a specific vertex some nonzero weight.

This algorithm is designed to inform us whether a given tree has an Italian domatic number of 3 or not. In this case, the combination of any three basic states for a vertex v must be examined. Consider the following definition.

Definition 18 (Compound State). *The combination of three basics states for v.*

An example of a compound state for v is the compound state “AOO”, where v is being dominated above by a weight of 2 in the first Italian dominating function and contains a weight of 1 in the other two Italian dominating functions.

There are obviously compound states that would violate the Italian domatic number on G, such as compound states that contain more that one “T”. The total weight of a vertex throughout the Italian dominating family must be less than or equal to 2 in order to satisfy the Italian domatic number, so more than one basic state “T” for any single vertex cannot occur. Since a compound state is a combination of three basic states, and since there are five different basic states, there are 5^3 or 125 combinations of different compound states. It is clear that most of these 125 states will violate the Italian domatic number for G, so consider the following factors that would allow us to narrow down the 125 total combinations of compound states to result with compound states that do not violate the Italian domatic number.

1. We may assume that the sum of weights for v across all Italian dominating functions must be exactly 2, so exactly one “T” or two “O” basic states must be present in all compound states.
2. The states “A” and “S” cannot exist together in a compound state since the parent of v must have a sum weight of no more than 2. Similarly, “A” can only occur at most once in a compound state.

With the factors above, we can narrow the 125 total combinations of compound states to 27 valid compound states that v can take in an Italian dominating function for G. Since the order in which Italian dominating functions occur in the Italian dominating family for G does not matter, we can partition the 27 valid compound states into 7 different equivalence classes based on the basic states each compound state contains. The classes, along with the compound states, are listed below:

1. AOO, OAO, OOA
2. BOO, OBO, OOB
3. SOO, OSO, OOS
4. BBT, BTB, TBB
5. SST, STS, TSS
6. ABT, ATB, BTA, BAT, TBA, TAB
7. BST, BTS, STB, SBT, TBS, TSB

For a given vertex $v \in V(T)$, all compound states are boolean (true or false), meaning that v can either satisfy this compound state or that it cannot. If v can satisfy a certain compound state, then this compound state with v outputs the value 1. Otherwise, it outputs 0. Using basic and compound states, an algorithm for determining if a tree can contain a family of 3 (not necessarily distinct) Italian dominating functions that satisfy the vertex weight conditions for the Italian domatic number has been created. This algorithm allows us to determine which trees have an Italian domatic number of 2. Any tree that does not have a family of 3 Italian dominating functions is guaranteed to have an Italian domatic number of 2. The "possibility" for a partial solution to this algorithm is determined entirely by the compound states that the root (r) of the tree satisfies. The appendix in this report contains the full algorithm for determining the number of Italian dominating families a tree contains. Below is a description of each equivalence class and how the algorithm worked in each state.

4.1 The Compound State BST

To set $BST[v]$ to be true (for a parent vertex v), there are three conditions that must be met.

1. First, there must be a child (or children) with a combined weight of 2 in the first family
2. In the second family, at least 1 child has to have a weight of 1
3. The remaining children must have a true state.

We also note that we cannot extend any subtree on a child of v from a compound state in which the first or second basic state is A or S. This is because the first and second basic states of BST will result with v having a weight of 0 in both the first and second basic states. With this in mind, we can meet the previous requirements with two cases:

1. **A child u_1 with a weight of 2 in the first family, and a distinct child u_2 with a weight 1 in the second family.**

   Running over all possibilities of u_1 and u_2, we require that there is a "true" state for u_1 in which the first basic state is T. In particular,

   $$\text{childVal1} = TBA[u_1] \text{ or } TBB[u_1] \text{ or } TBS[u_1]$$

   We also require that there is a "true" state for u_2 in which the second basic state is either T or O. In particular,

   $$\text{childVal2} = BTA[u_2] \text{ or } OOA[u_2] \text{ or } BTB[u_2] \text{ or } BTS[u_2] \text{ or } OOB[u_2] \text{ or } OOS[u_2]$$

   Lastly, we require that the subtrees rooted at the remaining children are all valid. In particular,

   $$\text{childVal3} = \text{childVal3 and } (BTA[u_3] \text{ or } OOA[u_3] \text{ or } BTB[u_3] \text{ or } BTS[u_3] \text{ or } OOB[u_3] \text{ or } OOS[u_3])$$

   where childVal3 is set to true by default and u_3 represents all vertices that are not u_1 or u_2.

2. **A child u_1 with a weight of 1 in the first family, and a distinct child u_2 with a weight 1 in the first and second family.**

   Running over all the possibilities of u_1 and u_2, we require that there is a "true" state for u_1 in which the first basic state is O. We also require that the first two basic states for u_2 to both be O for u_2 to contain a "true" state. In particular,

   $$\text{childVal1} = OOA[u_1] \text{ or } OOB[u_1] \text{ or } OOS[u_1]$$
   $$\text{childVal2} = OOA[u_2] \text{ or } OOB[u_2] \text{ or } OOS[u_2]$$

   For all subtrees rooted at the remaining children, we consider the same case as before, where

   $$\text{childVal3} = \text{childVal3 and } (BTS[u_3] \text{ or } OOA[u_3] \text{ or } BTB[u_3] \text{ or } BTS[u_3] \text{ or } OOB[u_3] \text{ or } OOS[u_3])$$

   provided that childVal3 is set to "true" by default and that u_3 represents all children besides u_1 and u_2.

Then at the end, $BST[v]$ is "true" if either case 1 or case 2 is satisfied.

4.2 The Compound State BBT

To set $BBT[v]$ to be "true", there are three conditions that need to be met:

1. First, there must be a child (or children) with a combined weight of 2 in the first family
2. In the second family, there must also be a child (or children) with a combined weight of 2
3. The remaining children must have a true state.

Similar to the Compound State BST, the first and second basic states for the children of v must not be A since the first two basic states of $BBT[v]$ will give a weight of 0 to v. There are a number of ways to meet the requirements above. However, determining $BBT[v]$ can be simplified to considering the following two cases:

1. **A child u_1 with a weight of 2 in the first family, and a distinct child u_2 with a weight 2 in the second family.**

   By going through all possibilities of u_1 and u_2, we require that there is a "true" state for u_1 in which the first basic state is T. We require the same condition for u_2 in the second family. In particular,

   $$\text{childVal1} = TBA[u_1] \text{ or } TBS[u_1] \text{ or } TBB[u_1]$$
   $$\text{childVal2} = BTA[u_2] \text{ or } BTS[u_2] \text{ or } BTB[u_2]$$

   Notice that the first compound states in the statements for childVal1 and childVal2 are from the same equivalence class. The same is true for the second and third compound states in each of these statements. Although compound states from the same equivalence class are identical for any given vertex, the position of the T's in each of these statements was chosen so that it is clear that childVal1 satisfies the first basic state in BBT while childVal2 satisfies the second basic state. We also need the subtrees rooted at the remaining children to also be satisfied. The only compound state that the remaining children cannot take is the compound state SST. This is because the vertex v does not contain a weight of 1 in two separate families. So in particular,

   $$\text{childVal3} = \text{childVal3 and } (OOA[u_3] \text{ or } OOB[u_3] \text{ or } OOS[u_3] \text{ or } TBA[u_3] \text{ or } TBS[u_3] \text{ or } TBB[u_3])$$

   where childVal3 is "true" by default and u_3 represents all children besides u_1 and u_2.

2. **Two distinct children u_1 and u_2 that each contain a weight of 1 in both the first and second families.**

   We again go through all possibilities of compound states for both u_1 and u_2. We require that there is a "true" state for u_1 in which two basic states are O. Since the order in which the two O states occur does not affect the cardinality of the Italian dominating family, we can assume that the two O states occur as the first two states of the compound state u_1 haves. The same condition is required for u_2. In particular,

   $$\text{childVal1} = OOA[u_1] \text{ or } OOB[u_1] \text{ or } OOS[u_1]$$
   $$\text{childVal2} = OOA[u_2] \text{ or } OOB[u_2] \text{ or } OOS[u_2]$$

   For the subtrees rooted at the remaining children to be satisfied, we consider the same case as before, where

   $$\text{childVal3} = \text{childVal3 and } (OOA[u_3] \text{ or } OOB[u_3] \text{ or } OOS[u_3] \text{ or } TBA[u_3] \text{ or } TBS[u_3] \text{ or } TBB[u_3])$$

   where childVal3 is "true" by default and u_3 represents all children besides u_1 and u_2.

$BBT[v]$ is "true" if either of the cases above are satisfied.

4.3 The Compound State ABT

To set $ABT[v]$ to be true, there are two conditions for the children that must be met.

1. In the second family, there must be a child (or children) with a combined weight of 2
2. The remaining children must have a true state

We note that we cannot extend any subtree on a child of v in which the first and second basic states are either A or S, since v will have a weight of 0 in both the first and second basic state. Determining $ABT[v]$ can be simplified to considering the following two cases:

1. **A child u_1 with a weight of 2 in the second family**

   Running over all possibilities for u_1, we require that there is a "true" state for u_1 in which the second basic state is T. In particular,

   $$\text{childVal} = BTS[u_1] \text{ or } BTB[u_1] \text{ or } BTA[u_1]$$

   We also require that the subtrees rooted at the remaining children are all valid. In particular,

   $$\text{childVal} = \text{childVal and } (BTA[u_2] \text{ or } OOA[u_2] \text{ or } BTB[u_2] \text{ or } OOS[u_2] \text{ or } BTS[u_2] \text{ or } BOO[u_2])$$

   where childVal is set to true by default and u_2 represents all vertices that are not u_1.

2. **Distinct children u_1 and u_2 both with a weight of 1 in the second family.**

   Running over all the possibilities of u_1 and u_2, we require that there is a "true" state for u_1 in which the second basic state is O. We also require that the second basic state for u_2 to both be O for u_2 to contain a "true" state. In particular,

   $$\text{childVal1} = OOA[u_1] \text{ or } OOB[u_1] \text{ or } OOS[u_1]$$
   $$\text{childVal2} = OOA[u_2] \text{ or } OOB[u_2] \text{ or } OOS[u_2]$$

   For all subtrees rooted at the remaining children, we consider the same case as before, where

   $$\text{childVal3} = \text{childVal3 and } (BTA[u_3] \text{ or } OOA[u_3] \text{ or } BTB[u_3] \text{ or } OOS[u_3] \text{ or } BTS[u_3] \text{ or } BOO[u_3])$$

   provided that childVal3 is set to "true" by default and that u_3 represents all children besides u_1 and u_2.

$ABT[v]$ is "true" if either of the cases above are satisfied.

4.4 The Compound State AOO

For $AOO[v]$ to be set to true, there are actually no weight conditions that are required of the children. We simply require that the subtrees rooted at the children are valid. We note that in the compound state AOO, $f_i(v) \neq 2$ for any $i \in \{1, 2, 3\}$. Knowing this, we can meet the requirement with the following condition:

Any Compound State for the children of v that do not contain A or S

After going through all of the Compound States, we know that there are two classes of Compound States that do not contain the basic state A or S. In particular,

$$\text{childVal} = BBT[u] \text{ or } BOO[u]$$

where u represents all children of v. So long as this condition is met, $AOO[v]$ is "true".

4.5 The Compound State SOO

To set $SOO[v]$ to be true (for a parent vertex v), there are two conditions that must be met

1. First, there must be a child with a weight of at least 1 in the first family
2. The subtrees rooted at the remaining children must be valid

We also note that we cannot extend any subtree on a child of v from a compound state in which any basic state is A. This is because the all of the basic states of BST will result with v having a weight of 0 or 1. With this in mind, determining $SOO[v]$ can be simplified to considering the following case:
A child u_1 with a weight of at least 1 in the first family.
Running over the possibilities of states for u_1, we require that there is a "true" state for u_1 in which the first basic state is T or O. In particular,

$$\text{childVal} = OOS[u_1] \text{ or } OOB[u_1] \text{ or } TBB[u_1] \text{ or } TBS[u_1] \text{ or } TSS[u_1]$$

We also require that the subtrees rooted at the remaining children are all valid. In particular,

$$\text{childVal} = \text{childVal and } (BTS[u_2] \text{ or } BOO[u_2] \text{ or } SST[u_2] \text{ or } SOO[u_2] \text{ or } BBT[u_2])$$

where childVal is set to true by default and u_2 represents all vertices that are not u_1. Then at the end, $SOO[v]$ is "true" if the case above is satisfied.

4.6 The Compound State SST

To set $SST[v]$ to be "true", there are 2 conditions that need to be met:

1. First, there must be two children that have a weight of at least one in the first and second family
2. The remaining children must have a true state

Similar to the compound state BST, the first and second basic states for the children of v must not be A or S since the first two basic states of $SST[v]$ will give a weight of 0 to v. There are a number of ways to meet the requirements above. However, determining $SST[v]$ can be simplified to considering the following two cases:

1. **A child u_1 that contains a weight of 1 in both the first and second family**

   For this case, we only consider one child since it is possible for any vertex in a tree to have two O basic states. Since the order of these states do not matter, we may consider one child with a weight of 1 in the first and second basic states. By going through all possibilities for u_1 we require that there is a "true" state for u_1 in which the first and second basic states are O. In particular,

   $$\text{childVal1} = OOA[u_1] \text{ or } OOB[u_1] \text{ or } OOS[u_1]$$

   We also require that the subtrees rooted at the remaining children are valid. In particular,

   $$\text{childVal2} = \text{childVal2 and } (OOA[u_2] \text{ or } OOB[u_2] \text{ or } OOS[u_2] \text{ or } TBA[u_2] \text{ or } TBS[u_2] \text{ or } TBB[u_2])$$

   where childVal2 is "true" by default and u_2 represents all children besides u_1.

2. **Two distinct children u_1 and u_2 that each contain a weight of 2 in both the first and second families.**

   We must now go through all possibilities of compound states for both u_1 and u_2. We require that there is a "true" state for u_1 in which the first basic state is T. The same condition is required for u_2. In particular,

   $$\text{childVal1} = TBA[u_1] \text{ or } TBS[u_2] \text{ or } TBB[u_3]$$
   $$\text{childVal2} = BTA[u_2] \text{ or } BTS[u_2] \text{ or } BTB[u_2]$$

For the subtrees rooted at the remaining children to be satisfied, we consider the same case as before, where

$$\text{childVal3} = \text{childVal3 and } (OOA[u_3] \text{ or } OOB[u_3] \text{ or } OOS[u_3] \text{ or } TBA[u_3] \text{ or } TBS[u_3] \text{ or } TBB[u_3])$$

where childVal3 is "true" by default and u_3 represents all children besides u_1 and u_2.

$SST[v]$ is "true" if either of the cases above are satisfied.

4.7 The Compound State BOO

To set $BOO[v]$ to be "true", there are two conditions that need to be met:

1. First, there must be a child (or children) with a combined weight of 2 in the first family
2. The remaining children must have a true state

None of the basic states for the children of v can be A since all of the basic states of $BOO[v]$ will give a weight of 0 or 1 to v. There are a number of ways to meet the requirements above. However, $BOO[v]$ can be satisfied with the following two cases:

1. **A child u_1 with a weight of 2 in the first family.**

   By going through all possibilities for u_1, we require that there is a "true" state for u_1 in which the first basic state is T. In particular,

   $$\text{childVal} = TBB[u_1] \text{ or } TBS[u_1] \text{ or } TSS[u_1]$$

   We also require that the subtrees rooted at the remaining children of v are also valid. In particular,

   $$\text{childVal} = \text{childVal and } (SBT[u_2] \text{ or } BOO[u_2] \text{ or } SST[u_2] \text{ or } SOO[u_2] \text{ or } BBT[u_2])$$

   where u_2 represents all children besides u_1.

2. **Two distinct children u_1 and u_2 that each contain a weight of 1 in the first family.**

   We now go through all possibilities of compound states for both u_1 and u_2. We require that there is a "true" state for u_1 in which the first basic state is O. The same condition is required for u_2. In particular,

   $$\text{childVal1} = OOS[u_1] \text{ or } OOB[u_1] \text{ or } TBB[u_1] \text{ or } TBS[u_1] \text{ or } TSS[u_1]$$
   $$\text{childVal2} = OOS[u_2] \text{ or } OOB[u_2] \text{ or } TBB[u_2] \text{ or } TBS[u_2] \text{ or } TSS[u_2]$$

   For the subtrees rooted at the remaining children to be satisfied, we consider the same case as before, where

   $$\text{childVal3} = \text{childVal3 and } (SBT[u_3] \text{ or } BOO[u_3] \text{ or } SST[u_3] \text{ or } SOO[u_3] \text{ or } BBT[u_3])$$

   where childVal3 is "true" by default and u_3 represents all children besides u_1 and u_2.

$BOO[v]$ is "true" if either of the cases above are satisfied.

4.8 Algorithm Output

Once given an input for the number of vertices n for a tree, the algorithm would run and print each tree on n vertices that did not have 3 Italian dominating functions in their respective Italian dominating families. To distinguish which trees did not have 3 Italian dominating functions, we examined the values for the two valid compound states at the root of the tree, BBT and BOO. In every other compound state, there is at least 1 basic state that looks to a parent vertex in order to be dominated. This clearly cannot apply for the root of a tree since the root has no parents. If there was no way to dominate the root r of a tree T with 3 Italian dominating functions that satisfies the Italain domatic number, then $d_I(T) = 2$. In other words, if $BBT[r] = 0$ and $BOO[r] = 0$, then $d_I(T) = 2$. Refer to Figure 11 for a couple examples of outputs we received from the algorithm for trees where $d_I(T) = 2$.

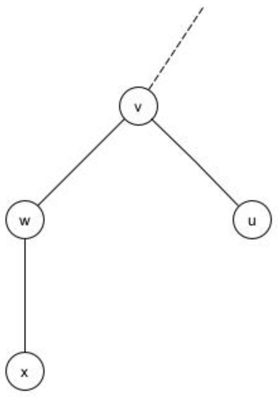


Figure 10: A forbidden substructure in a tree if $d_I(T) = 3$

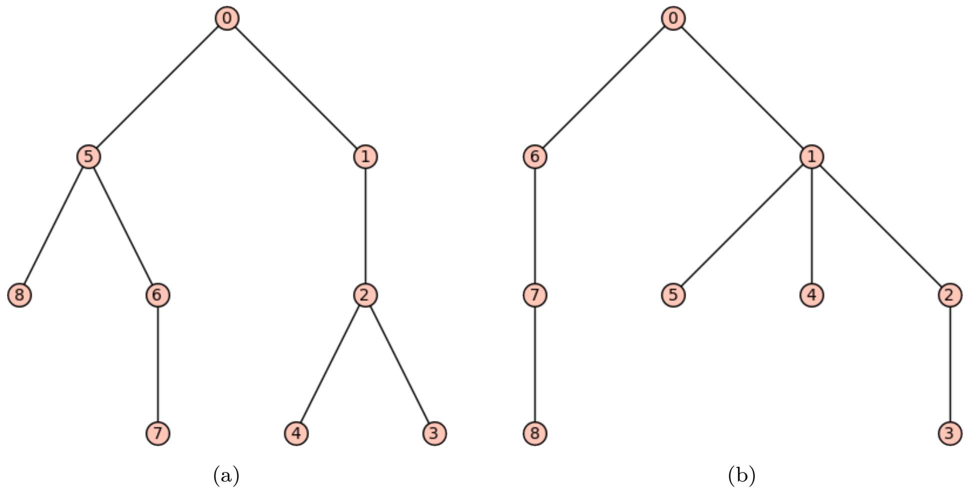


Figure 11: Algorithm Outputs

5 Results for Trees

In this section, our goal is to characterize the trees that have an Italian domatic number of 3 (and consequently, those trees with an Italian domatic number of 2). There are only two ways in which a tree can fail to have an Italian domatic number of 3. The first case is where it would be impossible to define three distinct Italian dominating functions on the tree such that the combined weight of each vertex in the tree is at most 2 throughout the Italian dominating family. The second way that a tree can fail to have an Italian domatic number of 3 is if we can define three Italian dominating functions for the Italian dominating family of the tree, but not in such a way that all three of these functions are distinct. First, we will consider trees with three Italian dominating functions that satisfy $\sum_{i=1}^{3} f_i(v) \leq 2$ for all vertices v in the tree, regardless of whether these functions are distinct. From this result, we will consider the second case and classify those trees that have an Italian domatic number of 3.

5.1 Trees with a Family of 3 Italian Dominating Functions

In this subsection, we would like to show that for trees that have a family of three Italian dominating functions, the tree does not contain the substructure in Figure 10. We will start by showing that this is a necessary condition.

Lemma 19. *If the removal of a vertex $v \in V(T)$ results in a path on 2 vertices and an isolated vertex, then $d_I(T) = 2$.*

Proof. Let G be a graph that contains the substructure in Figure 10. We will denote the vertices of the substructure the same way that is shown in its figure. Clearly, the removal of vertex v leaves a path on two vertices (the path formed between vertices w and x) and an isolate (vertex u) as components. Suppose, for contradiction, that $d_I(G) = 3$.

Suppose for contradiction that G has an Italian dominating family $\mathcal{F} = \{f_1, f_2, f_3\}$. Then, by definition, $\sum_{i=1}^{3} f_i(v) \leq 2$ for all vertices $v \in V(G)$. Consider the vertices w and x from Figure 10. We know that $f_i(x) = 0$ for some $i \in \{1, 2, 3\}$. WLOG, suppose that $f_1(x) = 0$. Consequently, $f_1(w) = 2$. This means for the remaining families, $f_2(w) = f_3(w) = 0$. As a result, $f_2(x), f_3(x) > 0$, implying that $f_2(x) = f_3(x) = 1$. This shows that for any leaf a and parent b, $f_i(a) = 0$ and $f_i(b) = 2$ for some $i \in \{1, 2, 3\}$ and that $f_k(a) = 1$ and $f_k(b) = 0$ for $k \neq i$. If we were to apply this to vertices x (leaf) and v (parent of x) from Figure 10, we would find that v has a nonzero weight in only one of the three functions in the Italian dominating family. Since vertex w also has a nonzero weight in only one of the three functions in the Italian dominating family, it is required to be dominated by v in both of its remaining Italian dominating functions where w has a weight of 0 since u only has a weight of 1 in these two functions for w. Since it is not possible for v to have a nonzero weight in two Italian dominating functions, we cannot dominate w in at least one Italian dominating function in the Italian dominating family for G. By contradiction, if the deletion of a vertex in a graph G leaves a path on 2 vertices and an isolated vertex as components, $d_I(G) = 2$ □

The method of proof above where we show the weights of leaves and their parents in a graph that has an Italian domatic number of 3 will prove to be useful for several other proofs later in this paper. It will be convenient to list this as its own result.

Observation 20. *Let G be a graph with a leaf v (and parent u). Suppose $d_I(G) = 3$ and that $\mathcal{F} = \{f_1, f_2, f_3\}$ is the corresponding Italian dominating family for this graph. Then there exists some $i \in \{1, 2, 3\}$ such that*

1. *$f_i(v) = 0$ and $f_i(u) = 2$, and*
2. *$f_k(v) = 1$ and $f_k(u) = 0$ for all $k \neq i$*

To show that it is sufficient just to exclude the configuration in Figure 10, we will use induction on the number vertices and the maximum degree of the tree. We want to assume that the maximum degree of the tree is at least 3, so it is useful to consider paths separately. In [16], Volkmann handles the cases for paths with the following theorem.

Theorem 21. *If P_n is a path with $n \geq 6$, the $d_I(P_n) = 3$.*

To conclude our discussion of paths, we should consider the cases when $n < 6$. For $n = 3$ and $n = 5$, we have a star and 2-subdivided star, respectively. We can dominate each of these with 3 Italian dominating functions; however, these functions will not be distinct. In particular, we can alternate labels between (1, 1, 0) and (0, 0, 2) between each adjacent vertex, starting at a leaf, and define f_1, f_2, and f_3 so that each label on a vertex v is of the form $(f_1(v),\ f_2(v),\ f_3(v))$. The path P_4, rooted at any interior vertex is the same configuration as Figure 10, so by Lemma 19, P_4 does not have a family of 3 Italian dominating functions.

For convenience, we introduce the following definition:

Definition 22. *A tree T is said to be "bad" if the removal of some vertex $v \in V(T)$ results in components that are a path on two vertices and an isolated vertex. Any other tree that is not "bad" is considered "good."*

The definition above defines the type of tree referred to in the hypothesis of Lemma 19. It makes it easier to state most of the rest of the theorems in this paper.

Theorem 23. *If T is a "good" tree on three or more vertices, then T has an Italian dominating family comprised of three Italian dominating functions.*

Proof. We will proceed by induction. First, on the number of vertices, and then by the maximum degree. For the base case, we can use the previous discussion of paths, specifically, the only path without a family of three Italian dominating functions is P_4, a "bad" tree.

Now consider a tree T. If T is a "bad" tree, then we may apply Lemma 19. Therefore, we may assume T is a "good" tree, not a path so that it has a vertex of at least degree 3, and that any "good" subtree of T on 3 or more vertices has a family of three Italian dominating functions.

Let the root of T be r such that r is a vertex of at least degree 3. We can label the subtrees formed by removing r as T_1, T_2,...,T_m for $m \geq 3$. Now we define three new trees T', T'', and T''' as follows:

- T' is the subtree induced by $\{r\} \cup (V(T) - V(T_1))$
- T'' is the subtree induced by $\{r\} \cup (V(T) - V(T_2))$
- T''' is the subtree induced by $\{r\} \cup (V(T) - V(T_3))$

We can now consider two cases based on the subtrees defined above. The case where all of the subtrees are "good" trees, or the case where at least one of them is a "bad" tree.

We will first consider the case where all T', T'', and T''' are "good" trees. Since T', T'', and T''' are all "good", each of them must have Italian dominating families comprised of three Italian dominating functions. In each of these families there are two cases for what may occur at the root r. Either r has a weight of 2 in one function and 0 in the other two, or r has a weight of 1 in two functions and a weight of 0 in the remaining function. By the pigeonhole principle, two of the Italian dominating families for T', T'', and T''' must fall into the same case. WLOG, assume that the families for T' and T'' fall into the same case. Now let $\{f'_1, f'_2, f'_3\}$ and $\{f''_1, f''_2, f''_3\}$ be the families of three Italian dominating functions for T' and T'', respectively, and index the Italian dominating families so that they agree on the vertex r, i.e. $f'_1(r) = f''_1(r)$, $f'_2(r) = f''_2(r)$, and $f'_3(r) = f''_3(r)$. Now, we can define the following:

$$f_i(v) = \begin{cases} f'_i(v) & v \in V(T') \\ f''_i(v) & v \in V(T_1) \end{cases}$$

For every vertex $v \in V(T')$ (particularly r), f_i satisfies all the requirements of an Italian dominating function since f_i is an Italian dominating function on T'. Similarly, for every vertex $v \in V(T_1)$, f_i satisfies the requirements of an Italian dominating function. Since this can be done for $i = 1$, $i = 2$, or $i = 3$, T has an Italian dominating family comprised of three Italian dominating functions.

We will now consider the case where at least one of the subtrees T', T'', or T''' are "bad". This case is more complicated, and the goal will be to sufficiently restrict the trees in this case so that a simple algorithm can be used to determine the Italian dominating functions f_1, f_2, and f_3.

If removing T_1, T_2, or T_3 creates a "bad" tree, it must be the case that r is adjacent to a leaf and has a neighbor adjacent to a leaf. In fact, it must serve as vertex w from Figure 10 and must consequently have a degree of exactly 3. If we cannot choose another vertex in $V(T)$ to root T, then we may assume T has no vertices of degree 4 or more and that every vertex of degree 3 in $V(T)$ is adjacent to a leaf.

To further restrict the class of trees we need to consider, suppose it is possible to choose r so that one of the components of $T - \{r\}$ is a path P_k for $k \geq 3$. We can then label the vertices of this path u_1, u_2,...,u_k such that u_1 is the leaf of T.

If $k \neq 4$, remove u_1 and u_2 from T and use induction to find the Italian dominating functions f_1, f_2, and f_3 for the resulting tree. We can extend these functions to the full tree (which includes u_1 and u_2) by setting $f_k(u_1) = f_k(u_3)$ and $f_k(u_2) = 2$ if $f_k(u_1) = 0$ and $f_k(u_2) = 0$ otherwise for $k = 1, 2, 3$.

This process can be repeated for $k = 4$, but we need to take caution in removing the vertices so we do not create a "bad" tree. If $k = 4$, remove u_1, u_2, and u_3 from T and again use induction to find the Italian dominating functions for the resulting tree. These functions can again be extended to the full tree by setting $f_k(u_1) = f_k(u_3) = f_k(u_4)$ and $f_k(u_2) = 2$ if $f_k(u_1) = 0$ and $f_k(u_2) = 0$ otherwise for $k = 1, 2, 3$.

We are now ready to handle trees that do not fall into the first case and do not have any pendant paths (or paths rooted at a child of r such that the child would be a leaf in the path) P_m for $m \geq 2$. In other words, the removal of any vertex of degree 3 from T should not leave a path P_m as a component. First, choose any vertex in T, then find the farthest vertex of degree 3 and call this vertex r. Root T at r. We note that r should be adjacent to two leaves as there are no pendant path components on this tree.

	Leaves			Non-leaves		
Distance level k	f_1	f_1	f_1	f_1	f_1	f_1
$k \equiv 0 \pmod 3$	1	1	0	0	2	0
$k \equiv 1 \pmod 3$	1	0	1	2	0	0
$k \equiv 2 \pmod 3$	0	1	1	0	0	2

Table 3: Labels for vertices in the second case of Theorem 23

Now we can partition all $V(T)$ based on distance from r. We distribute weights based on the definition for f_1, f_2, and f_3 given in Table 3. We can verify that f_1, f_2, and f_3 are Italian dominating functions of T by considering all types of vertices at specific distance levels. For example, consider some $v \in V(G)$ that is distance k from r, where $k \equiv 0 \pmod 3$.

- If v is a vertex of degree 3, then $f_1(v) = f_3(v) = 0$. If v has two children that are leaves, then both leaves would have a weight of 1 for f_1 and f_3. Vertex r (the root) would fall into this category. If this is not the case, v is not the root and has both a parent and child that are not leaves. The parent would be assigned a weight of 2 in f_3 and the child is assigned a weight of 2 in f_1.
- If v is a vertex of degree 2, again $f_1(v) = f_3(v) = 0$. If v was adjacent to a leaf, then consider the parent of v. The parent cannot be degree 3 as this would imply the parent was adjacent to a leaf and has a neighbor adjacent to a leaf, implying T is a "bad" tree. The parent of v cannot be a vertex of degree 2 as this would imply T contains a pendant path. Therefore, v can't be adjacent to a leaf and has one parent and one child that are non-leaves. Then the parent is assigned a weight of 2 in f_3 and the child is assigned a weight of 2 in f_1.
- If v is a leaf, only $f_3(v) = 0$, and the parent of v is a non-leaf such that $f_3(v) = 2$.

Showing the cases for $k \equiv 1 \pmod 3$ and $k \equiv 2 \pmod 3$ would result similarly. This demonstrates that f_1, f_2, and f_3 form an Italian dominating family for T, thus completing the inductive proof.

□

Theorem 23 along with Lemma 19 imply that a tree T is "good" if and only if it has an Italian dominating family comprised with three (not necessarily distinct) Italian dominating functions.

5.2 Trees with a Family of 3 Distinct Italian Dominating Functions

The second situation where a tree fails to have an Italian domatic number of 3 is when it is possible to define 3 Italian dominating functions on a tree T such that $\sum_{i=1}^{3} f_i(v) \leq 2$ for all vertices $v \in V(T)$, but impossible to do this in such a way where f_1, f_2, and f_3 are all distinct. Figure 12 shows examples of trees that fall into this category.

In this subsection, we will show that most trees that have families of 3 Italian dominating families actually have families of 3 distinct Italian dominating families. The following lemma serves to be a very useful tool for this goal.

Lemma 24. *Suppose that a graph G has a cut edge e such that G' and G'' are the components after the operation of $G - e$. If G' and G'' each have families of at least three (not necessarily distinct) Italian dominating functions, then $d_I(G) \geq 3$.*

Proof. Let G be a graph with a cut edge e and components G' and G''. Let the Italian dominating families for G' and G'' be $\mathcal{F}' = \{f'_1, f'_2, f'_3\}$ and $\mathcal{F}'' = \{f''_1, f''_2, f''_3\}$, respectively. If either of the families $\mathcal{F}'$ or $\mathcal{F}''$ contain three distinct functions, then we may define an Italian dominating family for G as follows:

$$f_i(v) = \begin{cases} f'_i(v) & v \in G' \\ f''_i(v) & v \in G'' \end{cases}$$

However, if both families $\mathcal{F}'$ and $\mathcal{F}''$ contain nondistinct functions, then one function in each family must repeat itself. WLOG, we can rewrite the families for G' and G'' as $\mathcal{F}' = \{f'_1, f'_1, f'_2\}$ and $\mathcal{F}'' = \{f''_1, f''_1, f''_2\}$,

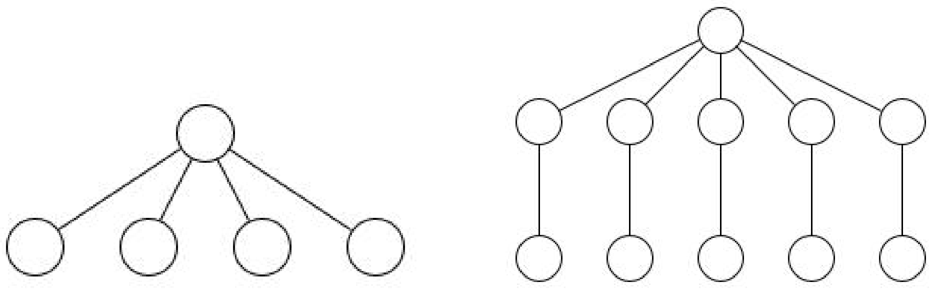


Figure 12: Examples of Trees that are good where $d_I(T) = 2$

respectively. We can still define a family of three Italian dominating functions for G using the following functions below:

$$f_1(v) = \begin{cases} f_1'(v) & v \in G' \\ f_1''(v) & v \in G'' \end{cases}$$

$$f_2(v) = \begin{cases} f_1'(v) & v \in G' \\ f_2''(v) & v \in G'' \end{cases}$$

$$f_i(v) = \begin{cases} f_2'(v) & v \in G' \\ f_1''(v) & v \in G'' \end{cases}$$

Each function above is an Italian dominating function, and each function is distinct. □

Two important classes of trees are the star and 2-subdivided star. A star is the complete bipartite graph $K_{1,n}$, where n represents the number of children for the root of the tree. The 2-subdivided star, denoted $K_{1,n}^2$, is the star $K_{1,n}$ where each edge to a leaf has been subdivided once. Figure 12 shows $K_{1,4}$ and $K_{1,5}^2$ in parts (a) and (b) of the figure, respectively.

We can apply Lemma 24 to trees that have exactly one vertex of degree 3 or more. Trees with no vertices of degree 3 or more (i.e., paths) are handled by Theorem 21 in [16].

Lemma 25. *Let T be any "good" tree with exactly one vertex of degree 3. Then T is either a star, a 2-subdivided star, or $d_I(T) = 3$.*

Proof. Let r be the root of T. We will approach this proof by examining the structure of the trees rooted at the children of r. Suppose r is removed from T, resulting in three or more subtrees rooted at the children of r. We require that T is "good", so it is impossible for both an isolated vertex and some graph of P_2 to be among these subtrees at the same time. Otherwise, the forbidden configuration from Figure 10 is formed in T. Suppose the removal of r results in only isolates or only graphs of P_2 as subtrees. In these scenarios, T would either be a star or 2-subdivided star, respectively. These structures only have an Italian domatic number of 2. For this reason, suppose that at least one of the subtrees rooted at the children of r is a path of three vertices or more. Suppose there is a path P_s of at least length 3 rooted at a child of r such that $s \neq 4$. In this case, we can cut the edge between r and the child that is the root of P_s. The two components remaining after this operation each have Italian dominating families of three functions, meaning that by Lemma 24, $d_I(T) = 3$.

In the case where P_4 is the only type of path present among the trees rooted at the children of r (excluding possible isolates or graphs of P_2), then we cannot perform the same cut as before. P_4 does not have an Italian dominating family of three functions, so consider the cases where the subtrees rooted at the children of r contain isolates and paths of length four, just paths of length four, or paths of length two and paths of length four. In the first case and second case, an edge cut can still be made. Suppose that a child c is the vertex that roots some P_4 subtree. Instead of cutting the edge between r and c, we can simply cut the other edge connected to c. The components formed by this include a star P_3 and another tree component with either isolates or P_4 paths rooted at the children of r. Repeating the same initial edge cut we made for all P_4 paths rooted at the children of r will result with a number of P_3 components and a star component. Using Lemma 24, we can inductively conclude that $d_I(T) = 3$.

For the last case, no cut edge is present, but there is a way to dominate this. Suppose that we remove all P_4 subtrees such that a 2-subdivided star component and a number of P_4 components remains. We will now partition all vertices of T as follows:

1. r: the root of T
2. S_1: set of all children of r among the 2-subdivided star component from above
3. S_2: set of all leaves among the 2-subdivided star component
4. M_1: set of all children of r that root a P_4 component
5. M_2: set of all children of all vertices in M_1
6. M_3: set of all children of all vertices in M_2
7. M_4: set of all children of all vertices in M_3

Now consider Table 4 for the Italian dominating family for the tree described above.

	r	S_1	S_2	M_1	M_2	M_3	M_4
f_1	1	0	1	1	1	0	1
f_2	1	0	1	0	1	0	1
f_3	0	2	0	1	0	2	0

Table 4: The Italian dominating family of the tree for the final case of the proof for Lemma 25.

Since the Italian dominating family above is composed of three distinct Italian dominating functions, $d_I(T) = 3$.

From all the cases above, if T is a "good" tree with exactly one vertex of degree 3, then either T is a star, T is a 2-subdivided star, or $d_I(T) = 3$. ☐

Our method of proof for proving that the hypothesis from Lemma 25 holds for all "good" trees is to examine the distance between degree 3 vertices. However, it is useful to consider the following case separately:

Lemma 26. *Any "good" tree T with two adjacent vertices, both adjacent to leaves, must satisfy $d_I(T) = 3$*

Proof. We know T is a tree with two adjacent vertices that are both adjacent to leaves. Let the two adjacent vertices be u and v, adjacent to w and x, respectively. Since T is good, Theorem 23 implies that T has a family of three Italian dominating functions. We can denote this family as $\mathcal{F} = \{f_1, f_2, f_3\}$. By Observation 20, each leaf must have a weight of 1 in two functions. So in at least one Italian dominating function, x and w must agree. WLOG, say $f_1(x) = f_1(w)$. So the remaining functions f_2 and f_3 either agree at w and x, or they do not.

If they do not agree, then we may assume that $f_2(w) = 1$ and $f_2(x) = 0$, as well as $f_3(w) = 0$ and $f_3(x) = 1$. This implies f_1, f_2, and f_3 are all distinct, meaning $d_I(T) = 3$.

If they do agree, we may assume $f_2(x) = f_2(w) = 1$ and $f_3(x) = f_3(w) = 0$. We note that this implies $f_3(v) = f_3(x) = 2$ and $f_k(v) = f_k(x) = 0$ for $k = 1, 2$. Now consider the subtrees T_1 and T_2 induced by cutting the edge (u, v) from T. Since f_1, f_2, and f_3 agree at vertices u and v, restricting them to either T_1 or T_2 would form families of three Italian dominating functions for each subtree. By Lemma 24, $d_I(T) = 3$. ☐

We are now ready to show the main result, that any "good" tree T that is not a star or 2-subdivided star satisfies $d_I(T) = 3$.

Theorem 27. *Let T be any "good" tree on three or more vertices that is not a star or 2-subdivided star. Then $d_I(T) = 3$*

Proof. To begin, we can use Theorem 21, which states $d_I(P_n) = 3$ for $n \geq 6$. We note that P_5 is a 2-subdivided star, P_4 is "bad", and that P_3 is a star. Furthermore, we can use Lemma 25 for vertices with exactly one vertex of degree three.

Therefore, consider a "good" tree T with at least two vertices of degree three, r and s, respectively. Assume r and s are the "closest" degree 3 or more vertices, or that the path between these two degree 3 or more vertices is the shortest path that can be formed between any two degree 3 or more vertices on T. We will first consider exactly two vertices of degree 3, and then consider cases based on the distance between the vertices of degree three or greater. In this context, distance refers to edges, and we will refer to the vertices of degree 3 or greater simply as degree 3 vertices. The first two cases, where r and s are either close or far apart, are the easiest to show.

1. Case 1: The vertices r and s are adjacent (distance of 1 apart).

   Consider the components formed by removing the edge (r, s). Let the two components be T_1, which contains r, and T_2, which contains s. Both r and s had degree 3 or more before this cut was made. If a copy of the configuration from Figure 10 occurs in either of the components, then either r or s would serve as vertex w from the figure and must be adjacent to a leaf. Furthermore, they would need to be adjacent to vertex v from the figure. Vertex v would have to be degree 3 and be adjacent to a leaf. Then T was a "good" tree with two adjacent vertices, each adjacent to leaves. By Lemma 26, $d_I(T) = 3$. Otherwise, T_1 and T_2 are "good" trees on three or more vertices. By Theorem 23 and Lemma 24, $d_I(T) = 3$.

2. Case 2: Vertices r and s are a distance of more than three apart.

   In this case, let $\{t_1, t_2, ..., t_k\}$ be the path on k vertices between r and s for $k \geq 3$. We can cut the edge (r, t_1) to form components T_1 (containing r) and T_2 (containing t_1). Since r had degree 3 or more, it would have to serve as the vertex w in Figure 10, which would imply that there are two closer degree 3 or more vertices in T (since v from the figure is at least degree 3). The vertex t_1 would have degree 1 in T_2, but is to far away from another degree 3 vertex to form the forbidden configuration in Figure 10. As a result, we may assume T_1 and T_2 are "good" components. By Theorem 23 and Lemma 24, $d_I(T) = 3$.

3. Case 3: Vertices r and s are a distance of three apart.

   In this case, let the vertices between r and s be t_1 (adjacent to r) and t_2 (adjacent to s). We must divide this case into subcases. First, let us assume that the removal of r or s does not leave a P_2 component in T. This implies neither r nor s has a neighbor adjacent to a leaf. In this case, we can cut the edge (t_1, t_2) to obtain subtrees T_1 and T_2 containing t_1 and t_2, respectively. Since r does not have a neighbor adjacent to a leaf, this edge cut does not form a "bad" subtree for T_1. The same argument can be made for T_2. The only way that either T_1 or T_2 would be "bad" is if the configuration in Figure 10 was present in T, which cannot be true since we require T to be "good". By Theorem 23 and Lemma 24, $d_I(T) = 3$.

   For the second subcase, let us assume that either the removal of r or the removal of s from T leaves a P_2 component. In this case, either r has a neighbor adjacent to a leaf or s has a neighbor adjacent to a leaf. WLOG, assume s is adjacent to a copy of P_2 and cut the edge (r, t_1) to obtain two subrees T_1 (containing neither t_1 nor t_2) and T_2 (containing both t_1 and t_2). In T_1, r has a degree of at least 2, meaning that if the forbidden configuration from Figure 10 were to appear from this edge cut, r would have to be adjacent to both an isolate and a copy of P_2, implying that this configuration was present in T. Since T is "good", this cannot be the case. So T_1 is "good". For T_2, we know s was already adjacent to a copy of P_2 before the edge cut. The edge cut now makes s adjacent to one additional copy of P_2 in T_2. Similar to before, if this cut were to cause the configuration from Figure 10 to occur, s would have to be adjacent to an isolate, implying that the forbidden configuration would have been present in T prior to the edge cut. Since T is "good", this cannot be the case. This implies T_2 is "good". By Theorem 23 and Lemma 24, $d_I(T) = 3$.

4. Case 4: Vertices r and s are a distance of two apart.

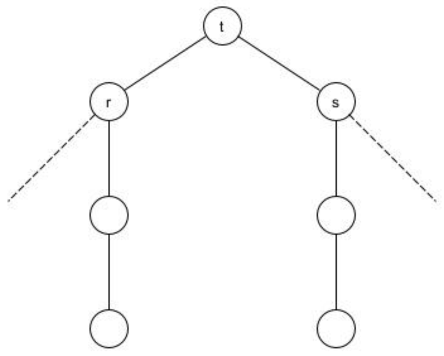


Figure 13: A tree used for Case 4 in Theorem 27. Deleting any edge on the path between r and s creates a copy of the configuration in Figure 10 rooted at r or at s.

In this case, let t be the vertex between them. It is helpful to divide this case into the subcases where either r or s is not adjacent to a copy of P_2 and the case where both r and s are adjacent to a copy of P_2.

For the first case, WLOG, suppose r is not adjacent to a copy of P_2. In this case, cut the edge (t, s) to form two subtrees T_1 (containing r) and T_2 (containing s). For T_1 to contain a "bad" component from this cut, r would have to be adjacent to a copy of P_2 (since it is now adjacent to an isolate in T_1 from the edge cut), which cannot be true since it was stated at the start of this subcase that r is not adjacent to a copy of P_2. Any other copy of the "bad" configuration in T_1 would imply that it is present in T, which cannot be true as T is "good". This implies T_1 is "good". The same holds true for s in T_2. If a "bad" configuration were to occur in T_2 from the edge cut, then s would have to be adjacent to an isolate and a copy of P_2. Since s is adjacent to a copy of P_2 and since T is "good", it is impossible for s to be adjacent to an isolate, as this would imply T has a "bad" configuration. So T_2 is also "good". By Theorem 23 and Lemma 24, $d_I(T) = 3$.

For the final subcase, assume both r and s are adjacent to a copy of P_2. This case is more difficult to handle, as there is no edge cut that can be made on the path between r and s without leaving a "bad" component in one of the two subtrees induced by the cut. This is shown in Figure 13. However, removing the vertex t from T results in two subtrees T_1 (containing r) and T_2 (containing s). For T_1 or T_2 to be "bad", the removal of t must result in its neighbors becoming leaves for the forbidden configuration in Figure 10. However, since r and s each have degree of at least 3 in T, neither will become leaves in T_1 or T_2. Beyond this, if a "bad" component were to appear in T_1 or T_2, then they would have to be present in T, which is impossible since T is "good", so T_1 and T_2 are "good". By Theorem 23, we know that both T_1 and T_2 have Italian dominating families of 3 functions. Now to extend these families for T_1 an T_2 back for T, we have to determine how to add t such that it is dominated. Since r is adjacent to a copy of P_2, it must be the case that in two of the three Italian dominating functions for T_1, r has a weight of 1. This is true as r has a neighbor of degree 2 that is adjacent to a leaf. Call this neighbor u. By Observation 20, it must be the case that u has a weight of 0 in two Italian dominating functions and a weight of 2 in the remaining function. This would imply that r needs to match the weight distribution of the leaf adjacent to u, as this leaf must have a weight of 1 in two of its Italian dominating families by Observation 20. This leaf alone cannot dominate u with weights of 1, so r must have a weight of 1 in two Italian dominating families. The same is true for s in T_2. Now let $\mathcal{F}' = \{f_1', f_2', f_3'\}$ and $\mathcal{F}'' = \{f_1'', f_2'', f_3''\}$ be the Italian dominating families for T_1 and T_2, respectively. With a convenient choice of indexing, say that $f_1'(r) = 1$, $f_2'(r) = 1$, $f_3'(r) = 0$, $f_1''(s) = 1$, $f_2''(s) = 0$, and $f_3''(s) = 1$. Now we can extend the Italian dominating functions for T_1 and T_2 to the entirety of T as follows: For each $v \in V(T)$,

$$f_1(v) = \begin{cases} f_1'(v) & v \in T_1 \\ f_1''(v) & v \in T_2 \\ 0 & v = t \end{cases}$$

$$f_2(v) = \begin{cases} f_2'(v) & v \in T_1 \\ f_2''(v) & v \in T_2 \\ 1 & v = t \end{cases}$$

$$f_3(v) = \begin{cases} f_3'(v) & v \in T_1 \\ f_3''(v) & v \in T_2 \\ 1 & v = t \end{cases}$$

It is clear that the functions above are Italian dominating functions, and since no two tuples $(f_1(r), f_2(r), f_3(r))$, $(f_1(s), f_2(s), f_3(s))$, and $(f_1(t), f_2(t), f_3(t))$ remain the same, f_1, f_2, and f_3 are all distinct. This means that $d_I(T) = 3$

□

Theorem 27 classifies the Italian domatic number for all trees, which is our main result for this section. As a result of this theorem, we can show a separate result for the Italian domination number of trees that follows directly from what we have shown above.

5.3 An Upper Bound for the Italian Domination Number for Trees

From our main result in Theorem 27, we can clearly see that an upper bound for the Italian domination number γ_I of any "good" tree T can be no greater than the maximum average of weight distributions of all vertices throughout all Italian dominating functions in the Italian dominating family for T, implying that $\gamma_I(T) \leq \lfloor \frac{2n}{3} \rfloor$. This also follows directly from Theorem 2. However, this tells us nothing about the upper bound for the Italian domination number of a "bad" tree. We would like to show that there does indeed exist a reasonable upper bound for the Italian domination number of "bad" trees.

Theorem 28. *Let T be a tree, then $\gamma_I(T) \leq \lfloor \frac{2n}{3} + \frac{n_1}{6} \rfloor$ for n_1 equal to the number of leaves in T.*

Proof. If T is "good," then it has an Italian dominating family comprised of three Italian dominating functions. By the discussion above, $\gamma_I(T) \leq \lfloor \frac{2n}{3} \rfloor \leq \lfloor \frac{2n}{3} + \frac{n_1}{6} \rfloor$, so the bound holds for "good" trees. If T is "bad" then there must exist at least one vertex $v \in T$ such that the deletion of this vertex will leave at least one isolate and at least one P_2 graph as components. Let K be the set of all such vertices in T. For each vertex $v \in K$, let m be the number of isolate components that would result from the removal of v and n be the number of P_2 components that would result from the removal of v. If $m > n$, remove all isolate components that would result from the removal of v, but do not remove v. Unless the resulting tree from this operation is P_5 (which is a "good" tree), then v will not be a leaf nor will it be adjacent to a leaf, so this operation does not introduce another "bad" configuration from Figure 10. If $m \leq n$, remove all leaves from the P_2 components that would result from the removal of v, but do not remove v. It is easier to see that since the degree of v does not change with this operation, we do not introduce another "bad" configuration. Repeat this process for all $v \in K$. After doing this, we should have a "good" tree, say T'. The maximum number of leaves that could be removed from T to form T' is $\frac{n_1}{2}$, and this is only possible if $m = n$ for all $v \in K$. So we have that $\gamma_I(T') \leq \lfloor \frac{2}{3}(n - \frac{n_1}{2}) \rfloor = \lfloor \frac{2n}{3} - \frac{n_1}{3} \rfloor$. One way to reintroduce all of the leaves we removed to form T' back to form an Italian dominating function for T is to give each removed leaf a weight of 1. This means that $\gamma_I(T) \leq \gamma_I(T') + \frac{n_1}{2} = \lfloor \frac{2n}{3} - \frac{n_1}{3} \rfloor + \frac{n_1}{2} \leq \frac{2n}{3} - \frac{n_1}{3} + \frac{n_1}{2} = \frac{2n}{3} + \frac{n_1}{6}$. Since $\gamma_I(T)$ must be an integer, we can conclude that $\gamma_I(T) \leq \lfloor \frac{2n}{3} + \frac{n_1}{6} \rfloor$. □

6 Discussion

For this research, one of the main results has been finding an upper bound for the Italian domination number for any tree given Figure 10's presence in the tree. This result is only one of many that find an upper bound for the Italian domination number. For instance, in [3], Chellali et al. prove that $\gamma_I(G) \leq \gamma_R(G)$ for any graph G, where $\gamma_R(G)$ represents the Roman domination number of a graph. We will not go into specifics about Roman domination here, but keep this bound in mind, as we know from [10] that $\gamma_R(G) \leq 2\gamma(G)$. In this statement, $\gamma(G)$ represents the domination number of a graph. This allows us to conclude that $\gamma_I(G) \leq 2\gamma(G)$ for any graph G. We know from [7] that $\gamma(T) \leq n - n_1$ for any tree T with n_1 leaves and order $n \geq 3$. With all of this in mind, we can conclude that $\gamma_I(T) \leq 2(n - n_1)$ for any tree T. However, we have a separate bound for $\gamma_I(T)$ from Theorem 28 that states $\gamma_I(T) \leq \lfloor \frac{2n}{3} + \frac{n_1}{6} \rfloor$ for all trees T. We may ease the restrictions of this bound by saying that $\gamma_I(T) \leq \frac{2n}{3} + \frac{n_1}{6}$.

Now we would like to compare each of the two bounds for the Italian domination number of trees. It is clear that both bounds rely on the number of vertices n and the number of leaves n_1 present in a given tree. It is also clear that $n_1 < n$ for all trees with $n \geq 3$. As n_1 approaches n, $\frac{2n}{3} + \frac{n_1}{6}$ clearly increases while $2(n - n_1)$ clearly decreases, implying that $2(n - n_1)$ serves as a better upper bound in this case. However, for trees with relatively small ratios of leaves to order, it appears that $\frac{2n}{3} + \frac{n_1}{6}$ will be smaller that $2(n - n_1)$, implying that $\frac{2n}{3} + \frac{n_1}{6}$ will be a better upper bound. This further implies that there must exist some ratio of leaves to order of a tree where one bound is strictly better than the other. In other words, there must exist some n_1 in terms of n such that $\frac{2n}{3} + \frac{n_1}{6} < 2(n - n_1)$. Algebraically solving for n_1 in terms of n yields that $n_1 < \frac{8n}{13}$. With this, we can conclude that if the proportion of leaves to order of a tree is less than $\frac{8}{13}$, then our bound $\frac{2n}{3} + \frac{n_1}{6}$ from Theorem 28 will be more accurate than the bound $2(n - n_1)$. Otherwise, if the proportion of leaves to order is greater than $\frac{8}{13}$, the bound $2(n - n_1)$ is more accurate. It is clear that the bounds will be equal for $n_1 = \frac{8n}{13}$.

Although the results in this research are plentiful, there are still conjectures, questions, and other statements that we did not have the ability to answer or prove during this research. The conjecture and questions may pave the way for future research in this field of mathematics. Some of the most prominent of these conjectures and questions are listed below.

Conjecture 29. *For any graph G, $d_I(G) \geq \delta(G)$*

In our research and previous research conducted on the Italian domatic number of a graph, no counterexample has been found to the conjecture above. However, there has been no way to prove this statement either. If this conjecture were true, it would imply that the Italian domatic number of any graph could only be 3 possible values, $\delta(G)$, $\delta(G) + 1$, and $\delta(G) + 2$. The proof of this conjecture would also make the bound presented in Corollary 7 obsolete, which is $d_I(G \square H) \geq \max\{d_I(G), d_I(H)\}$. For G and H with large minimum degrees, $d_I(G)$ and $d_I(H)$ will not exceed $\delta(G \square H)$.

Question 30. *Can the algorithm for determining the number of Italian dominating functions present in a tree be improved such that the algorithm can determine distinction or create the Italian dominating functions for a tree?*

The algorithm we developed to find the number of Italian dominating families a tree could hold was pivotal for us classifying the Italian domatic number for all trees. However, the algorithm can clearly be improved in a number of ways. One way is that it also generates the Italian dominating functions for a given tree. With this, it will certainly be easier to find the Italian dominating functions that correspond to the Italian dominating family of a tree, but it may also allow others to determine the Italian domination number for some tree. This may lead to improvements on our upper bound for the Italian domination number of a tree. With this in mind, it may also be convenient if the algorithm could determine distinction among the Italian dominating functions in the Italian dominating family of a tree.

Question 31. *Do our findings on the Italian domatic number for trees and Cartesian products of cycles imply anything about the 2-rainbow domatic number of these same graph families?*

It should be noted that 2-rainbow domination on a graph is very similar to Italian domination. Where Italian domination assigns weights to each vertex of a graph, 2-rainbow domination assigns sets (empty set or any set involving 1 and 2) to each vertex. To further emphasize this, it has even been shown in [1] that the 2-rainbow domination number of a tree ($\gamma_{r2}(T)$) is equal to the Italian domination number of the same tree $\gamma_I(T)$. We were able to bound the Italian domination number for trees based on whether or not it had a "bad" configuration from Figure 10 and subsequently, the tree's Italian domatic number. For research involving either the Italian domatic number or the 2-rainbow domatic number, it would be useful to know if any of our results on the Italian domatic number for trees and Cartesian products of cycles would be useful in finding the 2-rainbow domatic number for these same graph families.

Outside of the field of mathematics, graph theory can be used to model many different objects, structures, or relations, including city blocks or districts, faculty office hour overlap at a university, and many other examples. The Italian domatic number of a graph represents a set of Italian dominating functions with relatively minimal weights in relation to one another. This is implied with Theorem 2, which states that $\gamma_I(G)d_I(G) \leq 2n$ for all graphs G with order n. So as the Italian domatic number increases, the Italian domination number tends to decrease. In a case where minimally optimizing weight distributions on a graph is necessary, such as the ancient Roman Empire example from the start of this research, a larger Italian domatic number implies a set of these "minimally optimized" functions exist. For example, refer to the faculty office hour example stated above and say a university wanted to create a number of faculty committees with committee members and committee leaders such that everyone not on a specific committee could meet with either one leader or two members who are on it. Let us also say that no faculty member had the time to be leader of multiple committees or the time to participate as a member in more than two committees. This would imply faculty members on and off the committee that could meet have an overlap in office hours and that any committee formed would represent an Italian dominating function. In this case, the Italian domatic number represents the number of committees that could be formed.

As seen above, minimally optimizing weight distributions across a graph could equate to minimizing structures or objects over an area. This could be useful for trying to determine the minimum number of emergency service buildings (fire station, police department, or hospital) to distribute over a populated city such that specific criteria are met. Stationing legions of troops or soldiers over some country, as seen in the ancient Roman Empire example, represent another such example related to optimizing weight distributions over a graph. Based on the criteria required for the aforementioned weight distributions, the Italian domatic number of a graph can be used to find a set of Italian dominating functions with relatively minimal weight.

7 Appendix

```
from sage.graphs.trees import TreeIterator
def setAOO(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,BOO,
    ↪ OBO,OOB,SST,STS,TSS,SOO,OSO,OOS):
    value = 1
    childVal = 0
    for w in children[v]:
        childVal = (BBT[w] or BOO[w])
        value = childVal and value
    AOO[v] = value
    OAO[v] = value
    OOA[v] = value

def setABT(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,BOO,
    ↪ OBO,OOB,SST,STS,TSS,SOO,OSO,OOS):
    value = 0
    value1 = 0
    value2 = 0
    childVal = 0
    childVal1 = 0
    childVal2 = 0
    childVal3 = 1
    for w in children[v]: #Handles when v is being dominated below by 1 child with weight 2
        childVal1 = (BTS[w] or BTB[w] or BTA[w])
        childVal3 = 1
        for u in children[v]: #Vertex w has weight 2, this handles all other children of v
            if w != u:
                childVal3 = childVal3 and (BTA[u] or OOA[u] or BTB[u] or OOS[u] or BTS[u] or BOO[u])
        value1 = value1 or (childVal1 and childVal3)
    for x in children[v]: #Handles when v is being dominated below by 2 children with weight 1
        for y in children[v]:
            if x != y: #Vertices x and y are children with weight 1
                childVal1 = OOA[x] or OOB[x] or OOS[x]
                childVal2 = OOA[y] or OOB[y] or OOS[y]
                childVal3 = 1
                for z in children[v]: #All other vertices in children of v
                    if z != x and z != y:
                        childVal3 = childVal3 and (BTA[z] or OOA[z] or BTB[z] or OOS[z] or BTS[z] or
                            ↪ BOO[z])
                value2 = value2 or (childVal1 and childVal2 and childVal3)
    value = value1 or value2 #!!!Not sure how far to indent this
    ABT[v] = value
    ATB[v] = value
    BAT[v] = value
    BTA[v] = value
    TBA[v] = value
    TAB[v] = value
def setSOO(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,BOO,
    ↪ OBO,OOB,SST,STS,TSS,SOO,OSO,OOS):
    value = 0
    childVal = 0
    for w in children[v]: #Handles when v is being dominated below by 1 child with weight 1
        childVal = (OOS[w] or OOB[w] or TBB[w] or TBS[w] or TSS[w])
        for u in children[v]: #Vertex w has weight 1, this handles all other children of v
            if w != u:
                childVal = childVal and (BTS[u] or BOO[u] or SST[u] or SOO[u] or BBT[u])
        value = value or childVal
    SOO[v] = value
    OSO[v] = value
    OOS[v] = value

def setBOO(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,BOO,
    ↪ OBO,OOB,SST,STS,TSS,SOO,OSO,OOS):
    value = 0
    value1 = 0
    value2 = 0
    childVal = 0
    for w in children[v]: #Handles when v is being dominated below by 1 child with weight 2
        childVal = (TBB[w] or TBS[w] or TSS[w])
        for u in children[v]: #Vertex w has weight 1, this handles all other children of v
            if w != u:
                childVal = childVal and (SBT[u] or BOO[u] or SST[u] or SOO[u] or BBT[u])
        value1 = value1 or childVal
    for u in children[v]:
        for w in children[v]:
            if u != w:
                childVal1 = OOS[w] or OOB[w] or TBB[w] or TBS[w] or TSS[w]
                childVal2 = OOS[u] or OOB[u] or TBB[u] or TBS[u] or TSS[u]
                childVal3 = 1
                for z in children[v]: #All other vertices in children of v
                    if z != w and z != u:
                        childVal3 = childVal3 and (SBT[z] or BOO[z] or SST[z] or SOO[z] or BBT[z])
                value2 = value2 or (childVal1 and childVal2 and childVal3)
    value = value1 or value2
    BOO[v] = value
    OBO[v] = value
    OOB[v] = value

def setBST(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,BOO,
    ↪ OBO,OOB,SST,STS,TSS,SOO,OSO,OOS):
    value = 0
    value1 = 0
    value2 = 0
    childVal = 0
    for u in children[v]: #Vertex u has weight 2 first
        childVal1 = TBA[u] or TBB[u] or TBS[u]
        for w in children[v]: #Vertex w has weight 1 in middle
            if w != u:
                childVal2 = BTA[w] or OOA[w] or BTB[w] or BTS[w] or OOB[w] or OOS[w]
                childVal3 = 1
                for x in children[v]: #Every other vertex
                    if x != w and x != u:
                        childVal3 = childVal3 and (BTA[x] or OOA[x] or BTB[x] or BTS[x] or OOB[x] or
                            ↪ OOS[x])
                value1 = value1 or (childVal1 and childVal2 and childVal3)
    for u in children[v]: #Handles when v is being dominated below by 2 children with weight 1
        for w in children[v]:
            if u != w: #Vertices x and y are children with weight 1
                childVal1 = OOA[u] or OOB[u] or OOS[u]
                childVal2 = OOA[w] or OOB[w] or OOS[w]
                childVal3 = 1
                for y in children[v]: #Every other vertex in children of v
                    childVal3 = childVal3 and (BTS[y] or OOA[y] or BTB[y] or BTS[y] or OOB[y] or OOS[
                        ↪ y])
                value2 = value2 or (childVal1 and childVal2 and childVal3)
    value = value1 or value2
    BST[v] = value
    BTS[v] = value
    STB[v] = value
    SBT[v] = value
    TSB[v] = value
    TBS[v] = value
def setBBT(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,BOO,
    ↪ OBO,OOB,SST,STS,TSS,SOO,OSO,OOS):
    value = 0
    value1 = 0
    value2 = 0
    childVal = 0
    #Start with 2 children with weight 2
    for u in children[v]:
        childVal1 = TBA[u] or TBS[u] or TBB[u]
        for w in children[v]:
            if u != w:
                childVal2 = BTA[w] or BTS[w] or BTB[w]
                childVal3 = 1
                for z in children[v]: #All other vertices in children of v
                    if z != w and z != u:
                        childVal3 = childVal3 and (OOA[z] or OOB[z] or OOS[z] or TBA[z] or TBS[z] or
                            ↪ TBB[z])
                value1 = value1 or (childVal1 and childVal2 and childVal3)
    #2 children with weight 1
    #Only consider OOA, OOB, and OOS compound states since the other states the other combinations
        ↪ of compounds states with 2's and 1's end up falling under this category
    for u in children[v]:
        childVal1 = OOA[u] or OOB[u] or OOS[u]
        for w in children[v]:
            if u != w:
                childVal2 = OOA[w] or OOB[w] or OOS[w]
                childVal3 = 1
                for z in children[v]: #All other vertices in children of v
                    if z != w and z != u:
                        childVal3 = childVal3 and (OOA[z] or OOB[z] or OOS[z] or TBA[z] or TBS[z] or
                            ↪ TBB[z])
                value2 = value2 or (childVal1 and childVal2 and childVal3)
    value = value1 or value2
    BBT[v] = value
    BTB[v] = value
    TBB[v] = value
def setSST(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,BOO,
    ↪ OBO,OOB,SST,STS,TSS,SOO,OSO,OOS):
    value = 0
    value1 = 0
    value2 = 0
    childVal = 0
    #1 child with weight 1
    for u in children[v]:
        childVal1 = OOA[u] or OOB[u] or OOS[u]
        childVal2 = 1
        for z in children[v]: #All other vertices in children of v
            if z != u:
                childVal2 = childVal2 and (OOA[z] or OOB[z] or OOS[z] or TBA[z] or TBS[z] or TBB[z])
        value1 = value1 or (childVal1 and childVal2)
    #2 children with weight 2
    for u in children[v]:
        childVal1 = TBA[u] or TBS[u] or TBB[u]
        for w in children[v]:
            if u != w:
                childVal2 = BTA[w] or BTS[w] or BTB[w]
                childVal3 = 1
                for z in children[v]: #All other vertices in children of v
                    if z != w and z != u:
                        childVal3 = childVal3 and (OOA[z] or OOB[z] or OOS[z] or TBA[z] or TBS[z] or
                            ↪ TBB[z])
                value2 = value2 or (childVal1 and childVal2 and childVal3)
    value = value1 or value2
    SST[v] = value
    STS[v] = value
    TSS[v] = value

#following sets up tree iterator for trees with 9 vertices
#can be altered by changing input value
for G in TreeIterator(9):
    #G = graphs.RandomTree(num_vertices)
    G.is_tree()
    num_vertices = G.order()
    #G.show(layout='tree',tree_root=0,tree_orientation='down')
    L = G.lex_BFS(reverse=True)

    #Compound States sorted by Equivalence class
    ABT = zero_vector(SR, num_vertices)
    ATB = zero_vector(SR, num_vertices)
    BAT = zero_vector(SR, num_vertices)
    TAB = zero_vector(SR, num_vertices)
    TBA = zero_vector(SR, num_vertices)
    BTA = zero_vector(SR, num_vertices)

    AOO = zero_vector(SR, num_vertices)
    OAO = zero_vector(SR, num_vertices)
    OOA = zero_vector(SR, num_vertices)

    BBT = zero_vector(SR, num_vertices)
    BTB = zero_vector(SR, num_vertices)
    TBB = zero_vector(SR, num_vertices)

    BST = zero_vector(SR, num_vertices)
    BTS = zero_vector(SR, num_vertices)
    SBT = zero_vector(SR, num_vertices)
    STB = zero_vector(SR, num_vertices)
    TBS = zero_vector(SR, num_vertices)
    TSB = zero_vector(SR, num_vertices)

    BOO = zero_vector(SR, num_vertices)
    OBO = zero_vector(SR, num_vertices)
    OOB = zero_vector(SR, num_vertices)

    SST = zero_vector(SR, num_vertices)
    STS = zero_vector(SR, num_vertices)
    TSS = zero_vector(SR, num_vertices)

    SOO = zero_vector(SR, num_vertices)
    OSO = zero_vector(SR, num_vertices)
    OOS = zero_vector(SR, num_vertices)
```

```
order=zero_vector(SR, num_vertices)
children = [[] for i in range(num_vertices)]
count=0
for v in G.lex_BFS(reverse=True):
    #Records ordering in reverse breadth first search
    count += 1
    order[v] = count
#Sets children arrays
for v in G.lex_BFS(reverse=True):
    for w in G.neighbors(v):
        if order[w]<order[v]:
            children[v].append(w)
for v in G.lex_BFS(reverse=True):
    setABT(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,
        ↪ BOO,OBO,OOB,SST,STS,TSS,SOO,OSO,OOS)
    setAOO(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,
        ↪ BOO,OBO,OOB,SST,STS,TSS,SOO,OSO,OOS)
    setBBT(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,
        ↪ BOO,OBO,OOB,SST,STS,TSS,SOO,OSO,OOS)
    setBST(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,
        ↪ BOO,OBO,OOB,SST,STS,TSS,SOO,OSO,OOS)
    setBOO(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,
        ↪ BOO,OBO,OOB,SST,STS,TSS,SOO,OSO,OOS)
    setSST(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,
        ↪ BOO,OBO,OOB,SST,STS,TSS,SOO,OSO,OOS)
    setSOO(v,children,ABT,ATB,BAT,TAB,TBA,BTA,AOO,OAO,OOA,BBT,BTB,TBB,BST,BTS,SBT,STB,TBS,TSB,
        ↪ BOO,OBO,OOB,SST,STS,TSS,SOO,OSO,OOS)

#first if statement shows trees with 3 (not necessarily distinct) Italian dominating families
    ↪ while second shows trees with d_I = 2
#if (BBT[0] == 0 or BOO[0] == 1) and (BBT[0] == 1 or BOO[0] == 0):
if BBT[0] == 0 and BOO[0] == 0:
    G.show(layout='tree',tree_root=0,tree_orientation='down')
    print(0,":","AOO=", AOO[0], "ABT=", ABT[0], "SOO=", SOO[0], "BST=", BST[0], "BOO=", BOO[0],
        ↪ "BBT=", BBT[0], "SST=", SST[0])
```

References

[1] Boštjan Brešar, Michael A. Henning, and Douglas F. Rall. "Rainbow domination in graphs". In: *Taiwanese J. Math.* 12.1 (2008), pp. 213–225. ISSN: 1027-5487. DOI: 10.11650/twjm/1500602498. URL: https://doi.org/10.11650/twjm/1500602498.

[2] Gerard J. Chang. "The domatic number problem". In: vol. 125. 1-3. 13th British Combinatorial Conference (Guildford, 1991). 1994, pp. 115–122. DOI: 10.1016/0012-365X(94)90151-1. URL: https://doi.org/10.1016/0012-365X(94)90151-1.

[3] Mustapha Chellali et al. "Roman {2}-domination". In: *Discrete Appl. Math.* 204 (2016), pp. 22–28. ISSN: 0166-218X. DOI: 10.1016/j.dam.2015.11.013. URL: https://doi.org/10.1016/j.dam.2015.11.013.

[4] K. Choudhary, S. Margulies, and I.V. Hicks. "Integer domination of Cartesian product graphs". In: *Discrete Mathematics* 338.7 (2015), pp. 1239–1242. ISSN: 0012-365X. DOI: https://doi.org/10.1016/j.disc.2015.01.032. URL: https://www.sciencedirect.com/science/article/pii/S0012365X15000527.

[5] E. J. Cockayne and S. T. Hedetniemi. "Towards a theory of domination in graphs". In: *Networks* 7.3 (1977), pp. 247–261. ISSN: 0028-3045. DOI: 10.1002/net.3230070305. URL: https://doi.org/10.1002/net.3230070305.

[6] Ernie J. Cockayne et al. "Roman domination in graphs". In: *Discrete Math.* 278.1-3 (2004), pp. 11–22. ISSN: 0012-365X. DOI: 10.1016/j.disc.2003.06.004. URL: https://doi.org/10.1016/j.disc.2003.06.004.

[7] Wyatt J. Desormeaux, Teresa W. Haynes, and Michael A. Henning. "Improved bounds on the domination number of a tree". In: *Discrete Appl. Math.* 177 (2014), pp. 88–94. ISSN: 0166-218X. DOI: 10.1016/j.dam.2014.05.037. URL: https://doi.org/10.1016/j.dam.2014.05.037.

[8] Hong Gao et al. "More Results on Italian Domination in $C_n \square C_m$". In: *Mathematics* 8.4 (2020). ISSN: 2227-7390. DOI: 10.3390/math8040465. URL: https://www.mdpi.com/2227-7390/8/4/465.

[9] A. Giahtazeh, H. R. Maimani, and A. Iranmanesh. "On the Roman 2-domatic number of graphs". In: *Discrete Mathematics, Algorithms and Applications* 13.05 (2021), p. 2150052. DOI: 10.1142/S179383092150052X. eprint: https://doi.org/10.1142/S179383092150052X. URL: https://doi.org/10.1142/S179383092150052X.

[10] Michael A. Henning and Stephen T. Hedetniemi. "Defending the Roman Empire—a new strategy". In: vol. 266. 1-3. The 18th British Combinatorial Conference (Brighton, 2001). 2003, pp. 239–251. DOI: 10.1016/S0012-365X(02)00811-7. URL: https://doi.org/10.1016/S0012-365X(02)00811-7.

[11] Michael A. Henning and William F. Klostermeyer. "Italian domination in trees". In: *Discrete Appl. Math.* 217.part 3 (2017), pp. 557–564. ISSN: 0166-218X. DOI: 10.1016/j.dam.2016.09.035. URL: https://doi.org/10.1016/j.dam.2016.09.035.

[12] Zepeng Li, Zehui Shao, and Jin Xu. "Weak {2}-domination number of Cartesian products of cycles". In: *J. Comb. Optim.* 35.1 (2018), pp. 75–85. ISSN: 1382-6905. DOI: 10.1007/s10878-017-0157-6. URL: https://doi.org/10.1007/s10878-017-0157-6.

[13] J. Nieminen. "Two bounds for the domination number of a graph". In: *J. Inst. Math. Appl.* 14 (1974), pp. 183–187. ISSN: 0020-2932.

[14] Zofia Stepień et al. "2-rainbow domination number of $C_n \square C_5$". In: *Discrete Appl. Math.* 170 (2014), pp. 113–116. ISSN: 0166-218X. DOI: 10.1016/j.dam.2014.01.027. URL: https://doi.org/10.1016/j.dam.2014.01.027.

[15] Ian Stewart. "Defend the Roman Empire!" In: *Scientific American* 281.6 (Dec. 1999), pp. 136–138. DOI: 10.1038/scientificamerican1299-136.

[16] Lutz Volkmann. "The Roman {2}-domatic number of graphs". In: *Discrete Appl. Math.* 258 (2019), pp. 235–241. ISSN: 0166-218X. DOI: 10.1016/j.dam.2018.11.027. URL: https://doi.org/10.1016/j.dam.2018.11.027.