Palestine Journal of Mathematics

Vol 14 (Special Issue IV), (2025) , 67–77

PYTHON-BASED ANALYSIS AND VERIFICATION OF PRIME LABELING IN ELECTRICAL NETWORKS AND LOGIC GATES

A. Anto Cathrin Aanisha and R. Manoharan

MSC 2020 Classifications: Primary 05C78; Secondary 68R10.

Keywords and phrases: Prime labeling, electrical network and logic circuits, Python coding.

The authors would like to thank the reviewers and editor for their constructive comments and valuable suggestions that improved the quality of our paper.

Anto Cathrin Aanisha would like to thank Sathyabama Institute of Science & Technology, Chennai, for providing essential resources, including library facilities, that contributed to the successful completion of this work.

Abstract This study investigates the concept of prime labeling in the graph representation of electrical networks and logic gates. Using Python coding, the research explores how the adjacency matrix of these graphs can be used to assign prime labels to the vertices, enhancing the analysis of network behavior and circuit logic.

1 Introduction

Graph theory, as a branch of mathematics in computer science, physics, and chemistry, deals with graphs, which are systems representing relationships between pairs of entities. It continues to be a fundamental area of mathematics due to its versatility in modeling intricate problems across diverse fields such as biochemistry, electrical engineering, computer science, and operations research. Its ability to model multifaceted relationships makes it an invaluable part of modern mathematics. From communication networks to algorithm design, graph theory significantly impacts critical areas of science and technology [1].

Furthermore, graph theory plays a vital role in network security and mathematics, especially in cryptography. The science of securing messages through disguised meanings employs graph-based methods to produce secure keys. The use of adjacency matrices in cryptographic algorithms strengthens security by ensuring that encryption and decryption keys are more robust than other methods [5].

Graphs consist of vertices (or nodes) and edges (connections between objects). They may be undirected, treating edges symmetrically, or directed, capturing specific directional relationships between vertices [11].

The use of graph theory in electrical networks began with Gustav Kirchhoff in 1845, who applied trees to analyze circuits. By modeling components as nodes and their connections as edges, he applied Kirchhoff's circuit laws-the foundation for modern electrical engineering analysis. This exemplifies how abstract graph concepts can provide practical insight into system design and optimization [13]. According to Kirchhoff's voltage law, the total voltage across components in a closed loop is zero, while Kirchhoff's current law states that the total current entering and exiting a node is conserved [15].

Graph theory is essential for representing information visually and solving complex problems efficiently. It helps resolve diverse real-world issues by analyzing structural relationships. A strong understanding of different graph types and techniques is required to effectively address these challenges. Applications range from network optimization to molecular modeling, demonstrating the broad utility of graph theory [8].

Electrical networks-systems composed of interconnected components-can be represented by graphs, where vertices correspond to terminals and edges to connections. Such representations

facilitate the application of Kirchhoff's laws to analyze and optimize systems, highlighting the essential relationship between graph theory and electrical engineering [3].

Graph theory also finds use in communication networks, such as multi-hop wireless networks, where node coloring problems arise [14]. In these contexts, graph theory supports efficient channel assignment and enhances system optimization. Its contributions are critical to the development of modern communication systems.

Graph theory is also widely used in the analysis of social, transportation, and biological networks. Its power lies in modeling complex systems using nodes and edges to represent elements and their relationships. This modeling capability allows researchers to better understand the structure and behavior of large-scale systems composed of many interacting components [2].

In addition, graph theory aids in the analysis of infrastructure networks such as roads, pipelines, and cables. Algorithms for solving shortest path problems are essential in these domains, where minimizing distance, cost, or other weights between nodes is a key objective [12].

In chemistry and pharmacology, graph theory facilitates the modeling of molecular structures for computational analysis. Atoms are represented by vertices, and chemical bonds by edges. These graph models help analyze molecular properties and compare molecular structures, which is important in chemical design and drug development [7].

Another active area in graph theory is graph labeling, with applications in coding theory, radar systems, and communication networks. Labeling schemes help solve practical problems by assigning values to graph elements. For example, [10] highlights how graph labeling improves wireless communication networks, especially in enhancing network security and structure.

Graph labeling includes a variety of schemes such as cordial, prime, graceful, and harmonious labeling. Specialized variants like Fibonacci labeling, sum divisor cordial (SDC) labeling [19], edge sum divisor cordial labeling [18], Fibonacci prime anti-magic labeling [20], odd Fibonacci mean labeling [21], k-th Fibonacci prime labeling [22], and radio mean labeling [23] are active areas of study.

Prime labeling assigns distinct positive integers to graph vertices such that adjacent vertices receive relatively prime labels. A graph that admits such labeling is known as a prime graph [6]. This type of labeling plays a key role in theoretical investigations and applications.

Python has recently been used to explore graph labelings computationally. For example, [25] generated over a thousand graceful and odd-even graceful labelings for the Grotzsch and Peterson graphs. In [26], Python code produced vertex labels for the union of subdivided star and bistar graphs. Similarly, [27] combined graph theory with Chaldean numerology and arithmetic number labeling to encrypt text.

In [17], Python was employed to analyze edge sum divisor cordial labelings of circular ladder graphs (when s is even), star subdivisions $S(K_{1,s})$, and bistar graphs $B_{s,s}$ (for odd s), all of which were proven to be ESDC graphs. All the graphs discussed here are simple, finite, connected, and directed, with no loops or multiple edges.

2 Methodology

The proposed recognition methodology can be divided into two important steps. Prime labeling is applied to two groups of graphs:

- The graph representation of logic circuits
- The graph representation of electrical circuit networks

3 Main Results

3.1 Application of Prime Labeling in Logic Circuit

In the simulation, a logic circuit is modeled as a directed graph in graph theory. Each logic gate within the circuit corresponds to a vertex (or node) in the graph, while the connections between the input and output ports of adjacent logic gates are represented as directed edges within the graph.

Each logic gate vertex has exactly one outgoing edge representing its output, and one or two incoming edges representing its input(s), depending on its type. Additionally, inputs and outputs of the circuit are also represented as vertices within the graph.

In this model, it is assumed that each input vertex of a logic gate connects to exactly one neighboring gate's output vertex, while a logic gate's output vertex may connect to the input vertices of one or more neighboring gates, creating a fan-out effect.

Input vertices within the graph only have outgoing edges, whereas output vertices only have incoming edges. Furthermore, the graph representation ensures that there are no loops present within the circuit.

3.1.1 Implementation of Prime Labeling in a Dependence Graph of a Logic Circuit for a Simple Markov Random Field

The below logic circuits, represented by variables $\{s_0, s_1, s_2, s_3, s_4, s_5\}$, translate into dependence graphs. These graphs, akin to Markov random fields, illustrate conditional dependencies among logic variables.

Absent directed logic flow, statistical dependence governs interactions. Nodes denote random logic variables, while edges signify their conditional relationships. Understanding such representations is pivotal for grasping circuit behavior and optimizing design strategies.

A logic circuit and its dependence graph for a simple Markov random field is shown below [9].

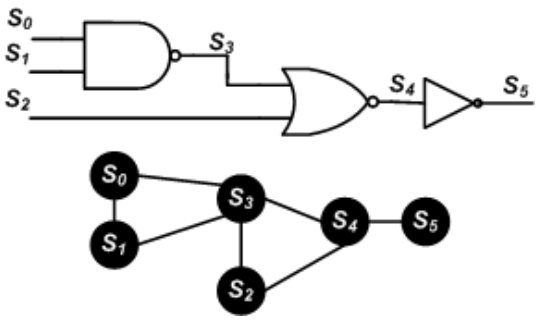

Figure 1. A logic circuit and its dependence graph for a simple Markov random field

Now, prime labeling applies in the following graph R which has been created by the above logic gate and its graph.

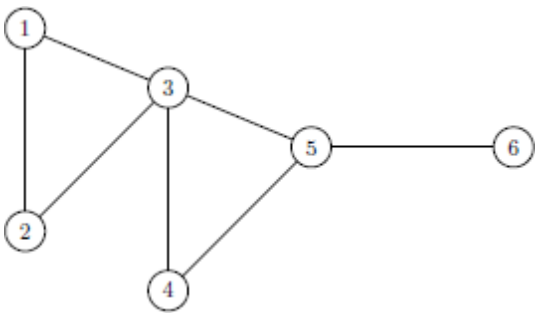

Figure 2. Implementation of prime labeling in the graph representation of above logic circuit.

The vertex set of the graph R is defined as:

$$W(R) = \{z_1, z_2, z_3, z_4, z_5, z_6\}. \tag{3.1}$$

The edge set of R is:

$$F(R) = \begin{cases} f_i = z_{2i-1}z_{2i}, & \text{for } 1 \le i \le 3; \\ f_{i+3} = z_i z_3, & \text{for } 1 \le i \le 2; \\ f_{i+5} = z_{i+2}z_5, & \text{for } 1 \le i \le 2. \end{cases} \tag{3.2}$$

A one-to-one correspondence function is defined by:

$$g : W(R) \to \{1, 2, \ldots, |W(R)|\}, \quad \text{given by } g(z_i) = i, \quad \text{for } 1 \le i \le 6. \tag{3.3}$$

Thus, $\gcd(g(z), g(w)) = 1$ for all adjacent vertices $z, w \in W(R)$.

Hence, a valid prime labeling is applied in Figure 2, which is derived from the logic gate circuit and its corresponding graph.

3.1.1(a) Python Implementation of Prime Labeling for the Graph Representation of the Above Logic Circuit

The following Python code simulates prime labeling for the graph described above using its adjacency matrix representation:

```
from math import gcd
import numpy as np

# Define the prime labeling function
def prime_labeling(vertices):
    return {vertex: i + 1 for i, vertex in enumerate(vertices)}

# Function to check if a graph satisfies the prime labeling condition
def is_prime_labeling(adjacency_matrix, labeling):
    n = len(adjacency_matrix)
    for i in range(n):
        for j in range(n):
            if adjacency_matrix[i][j] == 1:
                # Check if there is an edge
                if gcd(labeling[f"z{i+1}"], labeling[f"z{j+1}"]) != 1:
                    return False
    return True

# Input the adjacency matrix
def main():
    # Example adjacency matrix for the provided graph
    adjacency_matrix = np.array([
        [0, 1, 1, 0, 0, 0],  # z1
        [1, 0, 1, 0, 0, 0],  # z2
        [1, 1, 0, 1, 1, 0],  # z3
        [0, 0, 1, 0, 1, 0],  # z4
        [0, 0, 1, 1, 0, 1],  # z5
        [0, 0, 0, 0, 1, 0]   # z6
    ])

    # Define vertices
    vertices = [f"z{i+1}" for i in range(len(adjacency_matrix))]

    # Apply prime labeling
    labeling = prime_labeling(vertices)
```

```
    # Check if the graph satisfies the prime labeling condition
    prime_status = is_prime_labeling(adjacency_matrix, labeling)

    # Print the results
    print("Graph representation of above adjacency matrix is a prime graph:",
          prime_status)
    print("Vertex Labels:")
    for vertex, label in labeling.items():
        print(f"g({vertex}) = {label}")

# Run the main function
if __name__ == "__main__":
    main()
```

Output:

```
Graph representation of above adjacency matrix is a prime graph: True
Vertex Labels:
g(z1) = 1
g(z2) = 2
g(z3) = 3
g(z4) = 4
g(z5) = 5
g(z6) = 6
```

3.1.2 Implementation of Prime Labeling in a Graph Representation of a Logic Circuit

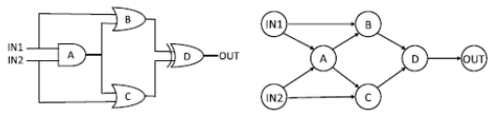

Figure 3. A Logic gate and its graph representation

Now, prime labeling is applied to the graph R, which has been constructed based on the above logic gate and its corresponding graph representation [16].

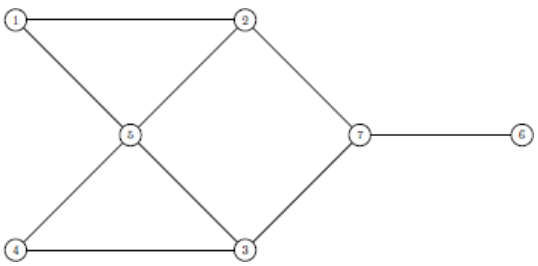

Figure 4. Implementation of prime labeling in the graph representation of above logic circuit

The vertex set of the graph R is defined as:

$$W(R) = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7\}. \tag{3.4}$$

The edge set of R is:

$$F(R) = \begin{cases} f_i = z_{2i-1}z_{2i}, & \text{for } 1 \le i \le 2; \\ f_{i+2} = z_i z_5, & \text{for } 1 \le i \le 4; \\ f_{i+6} = z_{i+1}z_7, & \text{for } 1 \le i \le 2; \\ f_9 = z_6 z_7. \end{cases} \tag{3.5}$$

The one-to-one correspondence function is defined as:

$$g : W(R) \to \{1, 2, \ldots, |W(R)|\}, \quad \text{given by } g(z_i) = i, \quad \text{for } 1 \le i \le 7. \tag{3.6}$$

Since $\gcd(g(z), g(w)) = 1$ for all adjacent vertices $z, w \in W(R)$, the graph R admits a prime labeling.

Hence, prime labeling is applied in Figure 4, which is created based on the above logic gate circuit and its corresponding graph.

3.1.2(a) Python Implementation of Prime Labeling for the Graph Representation of the Above Logic Circuit

The following Python program verifies whether a given logic-circuit-based graph admits a valid prime labeling using its adjacency matrix:

```
from math import gcd
import numpy as np

# Define the prime labeling function
def prime_labeling(vertices):
    return {vertex: i + 1 for i, vertex in enumerate(vertices)}

# Function to check if a graph satisfies the prime labeling condition
def is_prime_labeling(adjacency_matrix, labeling):
    n = len(adjacency_matrix)
    for i in range(n):
        for j in range(n):
            if adjacency_matrix[i][j] == 1:  # Check if there is an edge
                if gcd(labeling[f"z{i+1}"], labeling[f"z{j+1}"]) != 1:
                    return False
    return True

# Main function
def main():
    # Define the adjacency matrix
    adjacency_matrix = np.array([
        [0, 1, 0, 0, 1, 0, 0],  # z1
        [1, 0, 0, 0, 1, 0, 1],  # z2
        [0, 0, 0, 1, 1, 0, 1],  # z3
        [0, 0, 1, 0, 1, 0, 0],  # z4
        [1, 1, 1, 1, 0, 0, 0],  # z5
        [0, 0, 0, 0, 0, 0, 1],  # z6
        [0, 1, 1, 0, 0, 1, 0]   # z7
    ])

    # Define vertex names
```

```
    vertices = [f"z{i+1}" for i in range(len(adjacency_matrix))]

    # Apply prime labeling
    labeling = prime_labeling(vertices)

    # Check if the labeling satisfies the prime condition
    prime_status = is_prime_labeling(adjacency_matrix, labeling)

    # Print result
    print("Graph representation of above adjacency matrix is a prime graph:",
         prime_status)
    print("Vertex Labels:")
    for vertex, label in labeling.items():
        print(f"g({vertex}) = {label}")

# Run the main function
if __name__ == "__main__":
    main()
```

Output:

```
Graph representation of above adjacency matrix is a prime graph: True
Vertex Labels:
g(u1) = 1
g(u2) = 2
g(u3) = 3
g(u4) = 4
g(u5) = 5
g(u6) = 6
g(u7) = 7
```

3.2 Application of Prime Labeling in the Electric Circuit Networks

A circuit network is represented as a graph G with nodes corresponding to circuit junctions and edges representing the connections between them. Prime labeling assigns distinct positive integers (not exceeding the number of nodes) to each node such that adjacent nodes (connected by an edge) receive labels that are relatively prime, i.e., their greatest common divisor is one.

Electric circuits achieve completeness when current flows from the negative terminals of the power source. There are three fundamental types of electrical circuits:

- Series configuration
- Parallel configuration
- Hybrid (combination) configuration

Graph representations are instrumental in analyzing circuit networks, as they effectively capture current flow dynamics and the relationships between components such as resistors. Within these graph-based models, prime labeling techniques offer valuable insights into optimizing circuit layout and improving performance in terms of safety, efficiency, and design clarity.

3.2.1 Implementation of Prime Labeling in an Electric Circuit Network

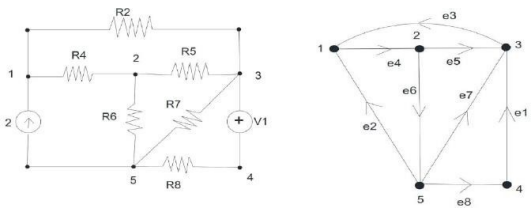

Figure 5. An Electric Network and its Graph Representation

Now, prime labeling is applied to the following graph R, which has been created based on the above electric circuit network 1 and its corresponding graph representation [4].

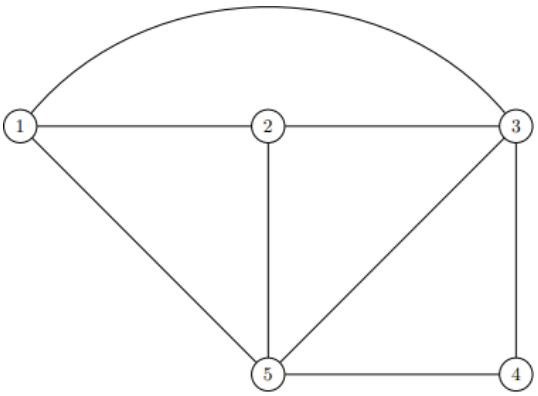

Figure 6. Implementation of prime labeling in the graph representation of above Electric Network

The vertex set of the graph R is:

$$W(R) = \{z_1, z_2, z_3, z_4, z_5\}. \tag{3.7}$$

The edge set of R is:

$$F(R) = \{f_1 = z_3z_4,\ f_2 = z_1z_5,\ f_3 = z_1z_3,\ f_{i+5} = z_{i+1}z_5 : 1 \le i \le 3,\ f_{i+3} = z_iz_{i+1} : 1 \le i \le 2\}. \tag{3.8}$$

The one-to-one correspondence function is defined by:

$$g : W(R) \to \{1, 2, \ldots, |W(R)|\}, \quad \text{given by } g(z_i) = i; \quad 1 \le i \le 5. \tag{3.9}$$

So, $\gcd(g(z), g(w)) = 1$ where z and w are arbitrary adjacent vertices.

Hence, prime labeling is applied in Figure 6, which has been created by the above electric circuit network 1 and its graph.

3.2.1(a) Python Implementation of Prime Labeling for the Graph Representation of the Above Electric Network

```
from math import gcd
import numpy as np

# Define the prime labeling function
def prime_labeling(vertices):
    return {vertex: i + 1 for i, vertex in enumerate(vertices)}

# Function to check if a graph satisfies the prime labeling condition
```

```
def is_prime_labeling(adjacency_matrix, labeling):
    n = len(adjacency_matrix)
    for i in range(n):
        for j in range(n):
            if adjacency_matrix[i][j] == 1:  # Check if there is an edge
                if gcd(labeling[f"z{i+1}"], labeling[f"z{j+1}"]) != 1:
                    return False
    return True

# Input the adjacency matrix
def main():
    # Updated adjacency matrix based on the edge set:
    # F(R) = {f1 = z3 z4, f2 = z1 z5, f3 = z1 z3,
    #         fi+5 = z_{i+1} z5 (1 <= i <= 3), fi+3 = z_i z_{i+1} (1 < <= i <= 2)}
    adjacency_matrix = np.array([
        [0, 1, 1, 0, 1],  # z1 connected to z2, z3, z5
        [1, 0, 1, 0, 1],  # z2 connected to z1, z3, z5
        [1, 1, 0, 1, 1],  # z3 connected to z1, z2, z4, z5
        [0, 0, 1, 0, 1],  # z4 connected to z3, z5
        [1, 1, 1, 1, 0]   # z5 connected to z1, z2, z3, z4
    ])

    # Define vertices
    vertices = [f"z{i+1}" for i in range(len(adjacency_matrix))]

    # Apply prime labeling
    labeling = prime_labeling(vertices)

    # Check if the graph satisfies the prime labeling condition
    prime_status = is_prime_labeling(adjacency_matrix, labeling)
2
    # Print the results
    print("Graph representation of above adjacency matrix is a prime graph:", prime_sta
    print("Vertex Labels:")
    for vertex, label in labeling.items():
        print(f"g({vertex}) = {label}")

# Run the main function
if __name__ == "__main__":
    main()
```

Output:

```
Graph representation of above adjacency matrix is a prime graph: True
Vertex Labels:
g(z1) = 1
g(z2) = 2
g(z3) = 3
g(z4) = 4
g(z5) = 5
```

3.2.2 Implementation of Prime Labeling in an Electric Circuit Network

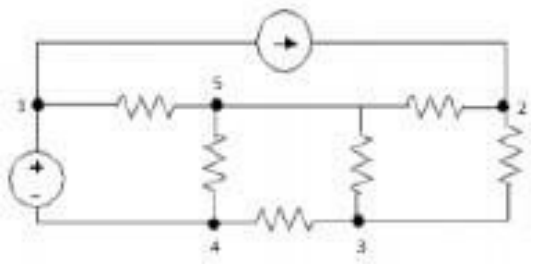

Figure 7. Graph Representation of an Electric Network

Now, prime labeling is applied to the following graph R, which has been created based on the above electric circuit network [15].

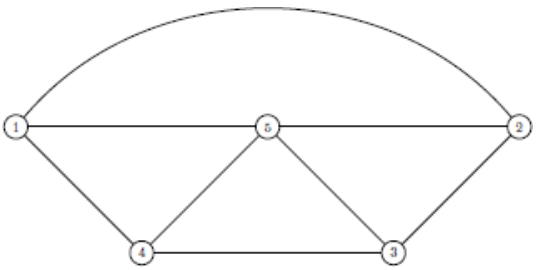

Figure 8. Implementation of Prime Labeling in the Graph Representation of above Electric Network

The vertex set of the graph R is:

$$W(R) = \{z_1, z_2, z_3, z_4, z_5\}. \tag{3.10}$$

The edge set of R is:

$$F(R) = \{f_i = z_i z_{i+1} : 1 \leq i \leq 4; \quad f_5 = z_1 z_4; \quad f_{i+5} = z_i z_5 : 1 \leq i \leq 3\}. \tag{3.11}$$

A one-to-one correspondence function is defined as:

$$g : W(R) \to \{1, 2, \ldots, |W(R)|\}, \quad \text{given by } g(z_i) = i, \quad 1 \leq i \leq 5. \tag{3.12}$$

So, $\gcd(g(z), g(w)) = 1$ for all adjacent vertices $z, w \in W(R)$.

Hence, prime labeling is applied in Figure 8, which has been created by the above electric circuit network.

4 Conclusion

This study demonstrates how prime labeling can be effectively applied to graphs representing electrical networks and logic gates. By utilizing Python and adjacency matrices, the methodology facilitates deeper understanding of network behavior and logic circuit structures. The implementation supports the analysis and verification of such systems, offering a useful mathematical tool for circuit optimization and computational exploration.

References

[1] Applications of Graph Methods, pp. 295-299, (2019). https://doi.org/10.1142/9789813278479_0012.

[2] S. M. Arul et al., Graph theory and algorithms for network analysis, E3S Web Conf., 399, (2023). https://doi.org/10.1051/e3sconf/202339908002.

[3] O. K. Berdewad and S. D. Deo, Application of graph theory in electrical network, Int. J. Sci. Res., 5(3), 981-982, (2016). https://doi.org/10.21275/v5i3.nov162021.

[4] G. Dhanalakshmi and D. A. Emiltet, A view point on applications of graph labeling in communication networks, 11, 682-690, (2022).

[5] G. Kaur et al., Applications of graph theory in science and computer science, Int. J. Adv. Eng. Manag., 2(6), 736-739, (2008). https://doi.org/10.35629/5252-0206736739.

[6] B. Kavitha and C. Vimala, Some prime labeling of graph, (2022), 1-9. https://doi.org/10.4172/JSMS.8.5.006.

[7] R. D. More et al., A literature review on applications of graph theory in various fields, 6(1), 1-8, (2021).

[8] B. L. Natarajan and M. K. Balaji, Application of graph theory in online network services to determine the shortest journey, Int. J. Adv. Netw. Appl., 10(5), 4030-4034, (2019). https://doi.org/10.35444/ijana.2019.10058.

[9] K. Nepal et al., Designing nanoscale logic circuits based on Markov random fields, J. Electron. Test., 23(2-3), 255-266, (2007). https://doi.org/10.1007/s10836-006-0553-9.

[10] N. L. Prasanna et al., Applications of graph labeling in communication networks, Orient. J. Comput. Sci. Technol., 7(1), 139-145, (2014).

[11] A. Prathik et al., An overview of application of graph theory, Int. J. ChemTech Res., 9(2), 242-248, (2016).

[12] A. B. Sadavare and R. V. Kulkarni, A review of application of graph theory for network, Int. J. Comput. Sci. Inf. Technol., 3(6), 5296-5300, (2012).

[13] S. V. M. Sarma, Applications of graph theory in human life, Int. J. Comput. Appl., 1(2), 21-30, (2012).

[14] H. Tamura et al., On applications of graph/network theory to problems in communication systems, ECTI Trans. Comput. Inf. Technol., 5(1), 15-21, (1970). https://doi.org/10.37936/ecti-cit.201151.54227.

[15] A. Verma et al., Using graph theory for automated electric network solving and analysis, (2021). https://doi.org/10.13140/RG.2.2.16919.50087.

[16] W. C. Xiao et al., Parallelizing a discrete event simulation application using the Habanero-Java multicore library, Proc. 6th Int. Workshop Program. Models Appl. Multicores Manycores, 86-95, (2015). https://doi.org/10.1145/2712386.2712402.

[17] A. A. C. Aanisha and R. Manoharan, Edge sum divisor cordial labeling of some graphs with Python implementation, 45(2), (2024).

[18] A. A. C. Aanisha et al., Edge sum divisor cordial labeling of crown & comb graphs, 15(83), 1-5, (2024).

[19] A. Aanisha and R. Manoharan, Sum divisor cordial labelling of sunflower graphs, AI4IoT 2023, 305-308, (2024). https://doi.org/10.5220/0012614800003739.

[20] G. Chitra and V. Mohanapriya, Fibonacci antimagic labeling of some special graphs, 6(9), 11-13, (2018).

[21] C. G., Odd Fibonacci mean labeling of some special graphs, Int. J. Math. Trends Technol., 66(1), 115-126, (2020). https://doi.org/10.14445/22315373/ijmtt-v66i1p515.

[22] P. K. et al., Kth Fibonacci prime labeling of graphs, Int. J. Math. Trends Technol., 68(5), 61-67, (2022). https://doi.org/10.14445/22315373/ijmtt-v68i5p510.

[23] K. Sunitha et al., Radio mean labeling of path and cycle related graphs, Glob. J. Math. Sci. Theory Pract., 9(3), 337-345, (2017).

[24] P. K. et al., Kth Fibonacci prime labeling of graphs (duplicate entry), (2022). https://doi.org/10.14445/22315373/ijmtt-v68i5p510.

[25] J. A. Gadhiya and R. Solanki, Labeling of some graphs using Python programming, 20(17), 2288-2294, (2022). https://doi.org/10.48047/Nq.2022.20.17.Nq880294.

[26] W. N. Taware et al., J. Northeast. Univ., 25(4), 2588-2612, (2022). http://dbdxxb.cn/wp-content/uploads/2022/12/Dr-Kunal-Sharma.pdf

[27] A. S. Purnalakshimi, Graph labelling and Chaldean numerology in cryptography, May, (2023).

Author information

A. Anto Cathrin Aanisha, Research Scholar, Department of Mathematics, Sathyabama Institute of Science and Technology, Chennai, India.
E-mail: antocathrinaanisha@gmail.com

R. Manoharan, Department of Mathematics, Sathyabama Institute of Science and Technology, Chennai, India.
E-mail: mano_rl@yahoo.co.in