Global Journal of Computer Science and Technology: F
Graphics & Vision
Volume 25 Issue 1 Version 1.0 Year 2025
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Constructing Classic Graphs in Graph Theory using Python and Generative AI: A Case Study in Computational Visualization and Prompt Engineering

By Dr. Shanzhen Gao, Dr. Weizheng Gao, Dr. Julian D. Allagan, Dr. Jianning Su, Dr. Ephrem Eyob & Dr. Hank B. Strevel

Virginia State University

Abstract- This study explores the construction of several classic graphs in graph theory through Python programming, offering a hands-on computational approach to understanding their mathematical properties. The selected graphs-including the Wagner, Desargues, Herschel, Möbius-Kantor, Franklin, truncated icosahedral, and triangular grid graphs-are chosen for their historical significance and structural complexity. Using Python's turtle graphics module, each graph is visualized through trigonometric and geometric logic, illustrating core concepts such as regularity, symmetry, Hamiltonicity, and planarity. In addition to manual code development, the study integrates generative AI, specifically ChatGPT, to reproduce graph constructions via prompt engineering.

Keywords: *graph theory, python programming, graph visualization, classic graphs, generative AI, prompt engineering.*

GJCST-F Classification: For Code: 080199

Constructing Classic Graphs in Graph Theory using Python and Generative AI: A Case Study in Computational Visualization and Prompt Engineering

Dr. Shanzhen Gao [α], Dr. Weizheng Gao [σ], Dr. Julian D. Allagan [ρ], Dr. Jianning Su [ω], Dr. Ephrem Eyob [¥] & Dr. Hank B. Strevel [§]

Abstract- This study explores the construction of several classic graphs in graph theory through Python programming, offering a hands-on computational approach to understanding their mathematical properties. The selected graphs-including the Wagner, Desargues, Herschel, Möbius-Kantor, Franklin, truncated icosahedral, and triangular grid graphs-are chosen for their historical significance and structural complexity. Using Python's turtle graphics module, each graph is visualized through trigonometric and geometric logic, illustrating core concepts such as regularity, symmetry, Hamiltonicity, and planarity. In addition to manual code development, the study integrates generative AI, specifically ChatGPT, to reproduce graph constructions via prompt engineering. This dual approach showcases the educational potential of AI-assisted programming and reinforces algorithmic thinking. The work aims to bridge the gap between theoretical graph concepts and their algorithmic applications. It provides a replicable methodology that enhances student engagement, supports active learning, and promotes interdisciplinary exploration across mathematics, computer science, and education.

Keywords: graph theory, python programming, graph visualization, classic graphs, generative AI, prompt engineering.

I. Introduction

Graph theory, a foundational discipline within discrete mathematics, explores graphs comprising vertices (or nodes) connected by edges(or links). These abstract constructs are powerful tools for modeling pair wise relationships across various fields, including computer science, biology, physics, chemistry, transportation, and social network analysis. Graphs enable the representation of systems as diverse as internet connectivity, molecular structures, urban transportation networks, and social interactions. The construction and analysis of specific types of graphs provide deeper insights into the properties and behaviors of these complex systems.

This paper presents a computational approach to constructing classic graphs in graph theory using Python programming. The primary objective is to bridge the theoretical framework of graph theory with the practical application of computer programming to foster understanding, visualization, and manipulation of intricate graph structures. With its ease of use, extensive libraries, and built-in graphic capabilities like the turtle module, Python offers a suitable platform for modeling and animating these graphs in an educational and research context.

The study focuses on a collection of well-known graphs that hold historical, mathematical, and practical significance. These include the Wagner graph, Desargues graph, Herschel graph, Möbius–Kantor graph, Franklin graph, truncated icosahedral graph, and triangular grid graph. Each graph selected for this project embodies unique structural and topological properties that make it ideal for exploring advanced concepts such as regularity, symmetry, non-planarity, chromatic characteristics, and Hamiltonicity.

The Wagner graph, a 3-regular graph with 8 vertices and 12 edges, is a key structure in minor theory used in studying apex and toroidal graphs. Its girth of 4, radius and diameter of 2, and chromatic number of 3 exemplify important constraints in planar graph theory and are instrumental in Ramsey's theory.

The Desargues graph, with 20 vertices and 30 edges, is known for its high symmetry and serves as a model in stereochemistry. Its distance-transitive and Hamiltonian nature makes it a powerful object in mathematics and chemistry. This graph also belongs to

Author α: "Lifetime Fellow of the Institute for Combinatorics and its Applications". Department of Computer Information Systems. Reginald F Lewis College of Business. Virginia State University Petersburg VA 23806, USA.
Corresponding Author σ: Department of Mathematics, Computer Science and Engineering Technology. Elizabeth City State University. Elizabeth City, NC 27909, USA. e-mail: wegao@ecsu.edu
Author ρ: Department of Mathematics, Computer Science and Engineering Technology. Elizabeth City State University. Elizabeth City, NC 27909, USA.
Author ω: Department of Mathematics, Computer Science and Engineering. Perimeter College, Georgia State University. Atlanta, GA 30303, USA.
Author ¥ §: Reginald F Lewis College of Business. Virginia State University, Petersburg VA 23806, USA.

the family of cubic, distance-regular graphs and exhibits elegant structural harmony.

The Herschel graph, though relatively small with 11 vertices and 18 edges, holds significance as the smallest non-Hamiltonian polyhedral graph. It provides a clear example of the limits of Hamiltonian cycles in three-dimensional graph structures and plays a crucial role in understanding planar graph exceptions.

The Möbius–Kantor graph, a cubic bipartite symmetric graph with 16 vertices and 24 edges, offers insights into the behavior of generalized Petersen graphs. It is a helpful substructure within higher-dimensional graphs like the hypercube and plays a key role in studying symmetry and Cayley graphs.

The Franklin graph, famous for its application to topological coloring problems, was used by Philip Franklin to disprove the Heawood conjecture on the Klein bottle. With 12 vertices and 18 edges, it remains an essential case study in topological graph theory and graph coloring.

The truncated icosahedral graph, or the Buckminster Fullerene graph, is derived from an Archimedean solid and features 60 vertices and 90 edges. Its structure models the carbon molecule C_{60} and is widely recognized in chemistry and architecture. As a 3-regular graph, it demonstrates the complexity and symmetry of polyhedral graphs and the feasibility of rendering them computationally.

Finally, the triangular grid graph is a visual model for lattice structures. It represents the layout of triangular tilings and has applications in physics, chemistry, and game theory. These graphs are important in modeling networks with hexagonal or triangular symmetry.

In this research, each graph is constructed algorithmically using Python's turtle module. The process involves deriving polar coordinates for vertex placement, calculating edge connections using trigonometry, and applying stylized rendering for clear visualization. This computational method enables students and researchers to dynamically visualize and interact with graph properties, fostering deeper engagement with abstract concepts.

Additionally, this study incorporates generative artificial intelligence tools, such as ChatGPT, to reproduce and verify Python programs for each graph. Through prompt engineering, the researchers crafted queries that guided the AI in generating accurate code representations of each graph structure. This dual approach-combining manual programming with AI-assisted verification-demonstrates the synergy between human logic and machine learning in computational mathematics.

The integration of Python into graph theory education offers numerous pedagogical advantages. It provides a platform for visual experimentation, encourages algorithmic thinking, and bridges the gap between abstract mathematical definitions and concrete visual outputs. Furthermore, using AI tools introduces learners to emerging technologies in computational science, enhancing their digital literacy and programming fluency.

This paper aims to serve as both a research contribution and a teaching resource. Illustrating how classic graphs can be constructed through code provides a replicable methodology for instructors and students to explore graph theory in an interactive, project-based environment. Including code, figures, and AI-generated examples supports active learning and promotes inquiry-based exploration.

In the following sections, the mathematical characteristics of each selected graph are discussed in detail, followed by their respective Python implementations. Through this work, the authors aim to deepen understanding, inspire further research, and promote computational literacy in graph theory.

II. Methodologies

This study adopts a computational and algorithmic methodology to construct and analyze classic graphs in graph theory using Python programming. The primary goal is to bridge mathematical abstraction with visual comprehension by implementing graph structures through code. Python's flexibility, particularly its turtle graphics module, is the foundation for this approach, enabling accurate and dynamic graph rendering based on geometric and trigonometric calculations.

The methodology follows a systematic, replicable process applied across all selected graphs. Each begins with a thorough mathematical analysis-defining vertex counts, edge arrangements, degrees, symmetries, and other graph invariants. Next, the design transitions to coordinate planning, typically utilizing polar geometry to determine optimal vertex placement around circular or polygonal paths. Edges are drawn between these vertices by calculating precise movements and rotations within the turtle environment.

Each Python program is customized to the individual graph's structure. This includes 3-regular graphs like the Wagner and Franklin graphs, symmetric bipartite graphs like the Möbius–Kantor graph, and complex polyhedral graphs like the truncated icosahedral graph. The visualizations often employ color-coded vertices, labeled nodes, and edge stylizations to reflect graph attributes like chromatic number, connectivity, and planarity for enhanced educational value and clarity.

In addition to constructing the graphs manually through code, this study integrates generative AI tools, specifically ChatGPT, by using prompt engineering to generate alternative Python implementations. This dual strategy-manual coding followed by AI-assisted reproduction-enables cross-verification of graphical outputs and exposes learners to collaborative AI programming practices.

Overall, this methodology emphasizes reproducibility, educational accessibility, and computational precision. It supports the theoretical exploration of graph properties and enables hands-on learning and experimentation. By merging algorithmic thinking with visual modeling, this framework equips students and researchers with a powerful toolkit for advancing their understanding of graph theory through Python.

III. Wagner Graph

In the mathematical field of graph theory, the Wagner graph is a 3-regular graph with 8 vertices and 12 edges (Bondy & Murty, 2007). It is the 8-vertex Möbius ladder graph. It is nonplanar but has a crossing number of one, making it an apex graph. It can be embedded without crossings on a torus or projective plane and is also a toroidal graph. It has girth 4, diameter 2, radius 2, chromatic number 3, chromatic index 3, and is both 3-vertex-connected and 3-edge-connected. It is a vertex-transitive graph but is not edge-transitive. Its whole automorphism group is isomorphic to the dihedral group D_8 of order 16, the group of an octagon symmetries, including rotations and reflections ("Wagner Graph," 2024).

The Wagner graph is triangle-free and has independence number three, providing one half of the proof that the Ramsey number R(3,4) (the least number n such that any n-vertex graph contains either a triangle or a four-vertex independent set) is 9 (Soifer, 2008).

Wagner graph has 392 spanning trees; it and the complete bipartite graph $K_{3,3}$ have the most spanning trees among all cubic graphs with the same number of vertices (Jakobson & Rivin, 1999).

The Wagner graph is also one of four minimal forbidden minors for the graphs of tree width at most three (the other three being the complete graph K_5, the graph of the regular octahedron, and the graph of the pentagonal prism) and one of four minimal forbidden minors for the graphs of branch width at most three (the other three being K_5, the graph of the octahedron, and the cube graph) ("Wagner Graph," 2024; Bodlaender, 1998; Bodlaender & Thilikos, 1999).

Wagner's famous conjecture asserts that for any infinite set of graphs, one of its members is isomorphic to a minor of another (all graphs in this paper are finite) (Wagner, 1970). The project proving Wagner's conjecture was started by Robertson and Seymour, later joined by Thomas, and completed in 2004, which led to entirely new concepts and a new way of looking at graph theory (Lovász, 2006).

Python Program for Wagner Graph

```
import turtle
import math
t = turtle.Turtle()
t.speed("fastest")
for k in range(8):
   t.penup()
   x1 = 300*math.cos(math.pi*((45*k)%360)/180)
   y1 = 300*math.sin(math.pi*((45*k)%360)/180)
   t.goto(x1,y1)
   t.pendown()
   x2 = 300*math.cos\
(math.pi*((45*k+45)%360)/180)
   y2 = 300*math.sin(math.pi*((45*k+45)%360)/180)
   t.goto(x2,y2)
radius = 300
for k in range(8):
   t.penup()
   t.goto(0,0)
   t.setheading(45*k)
   t.forward(radius)
   t.pendown()
   t.fillcolor("blue")
   t.begin_fill()
   t.circle(4)
   t.end_fill()
radius = 300
for k in range(4):
   t.penup()
   x1 = radius*math.cos(math.pi*((45*k)%360)/180)
   y1 = radius*math.sin(math.pi*((45*k)%360)/180)
   t.goto(x1,y1)
   t.pendown()
   x2 = radius*math.cos\
(math.pi*((45*k+180)%360)/180)
   y2 = radius*math.sin\
(math.pi*((45*k+180)%360)/180)
   t.goto(x2, y2)
t.hideturtle()
```

Using the previous algorithm, we generate the Wagner graph as shown in Figure 1.

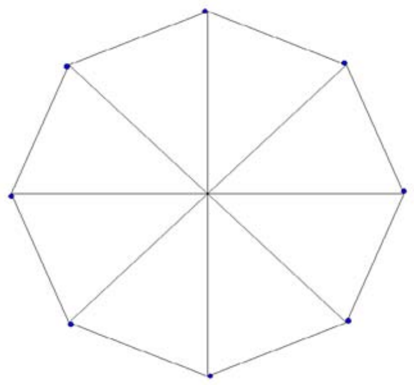

Figure 1: Wagner Graph

Using ChatGPT to Reproduce the Above Graph

After uploading the above graph to ChatGPT, we asked, "Based on the attached image/graph, could you develop a Python program to reproduce it?"

The Python program generated by ChatGPT produced the following image in Figure 1(a).

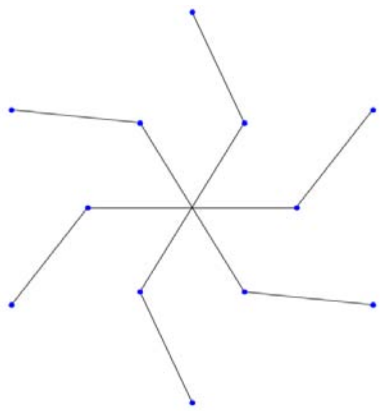

Figure 1(a)

After uploading the above graph to ChatGPT, we asked, "Based on the attached image/graph, could you develop a Python program to reproduce it?"

The Python program generated by ChatGPT produced the following image in Figure 1(b).

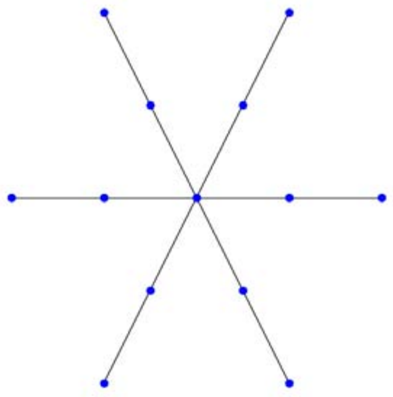

Figure 1(b)

IV. Desargues Graph

The Desargues graph is a distance-transitive, cubic graph with 20 vertices and 30 edges. It is named after Girard Desargues, arises from several different combinatorial constructions, has a high level of symmetry, is the only known non-planar cubic partial cube, and has been applied in chemical databases. It is a symmetric graph with symmetries that take any vertex to any other vertex and any edge to any other edge. Its symmetry group has order 240, and is isomorphic to the product of a symmetric group on 5 points with a group of order 2 ("Desargues Graph," 2024).

In chemistry, the Desargues graph is known as the Desargues–Levi graph; it is used to organize systems of stereoisomers of 5-ligand compounds. In this application, the thirty edges of the graph correspond to pseudorotations of the ligands ("Desargues Graph," 2024; Balaban, Fărcaşiu, & Bănică, 1966; Mislow, 1970).

It has chromatic number 2, chromatic index 3, radius 5, diameter 5, and girth 6. It is also a 3-vertex-connected and a 3-edge-connected Hamiltonian graph. It has a book thickness of 3 and a queue number of 2 ("Desargues Graph," 2024).

All the cubic distance-regular graphs are known (Brouwer, Cohen, & Neumaier, 1989). The Desargues graph is one of the 13 such graphs ("Desargues Graph," 2024).

Python Program for Creating Desargues Graph

```
import turtle
import math
t = turtle.Turtle()
t.speed("fastest")
t.penup()
t.goto(0,-300)
t.pendown()
t.circle(300)
radius = 300
for k in range(10):
    t.penup()
    t.goto(0,0)
    t.setheading(36*k)
    t.forward(radius)
    t.pendown()
    t.fillcolor("blue")
    t.begin_fill()
    t.circle(4)
    t.end_fill()
radius = 150
for k in range(10):
    t.penup()
    t.goto(0,0)
    t.setheading(36*k)
    t.forward(radius)
    t.pendown()
    t.fillcolor("red")
    t.begin_fill()
    t.circle(4)
    t.end_fill()
for k in range(10):
    t.penup()
    x1 = 300*math.cos(math.pi*((36*k)%360)/180)
    y1 = 300*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 150*math.cos(math.pi*((36*k)%360)/180)
    y2 = 150*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x2,y2)
for k in range(10):
    t.penup()
    x1 = 150*math.cos(math.pi*((36*k)%360)/180)
    y1 = 150*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 150*math.cos\
(math.pi*((36*k+108)%360)/180)
    y2 = 150*math.sin\
(math.pi*((36*k+108)%360)/180)
    t.goto(x2,y2)
t.hideturtle()
```

Using the previous algorithm, we generate the Desargues graph shown in Figure 2.

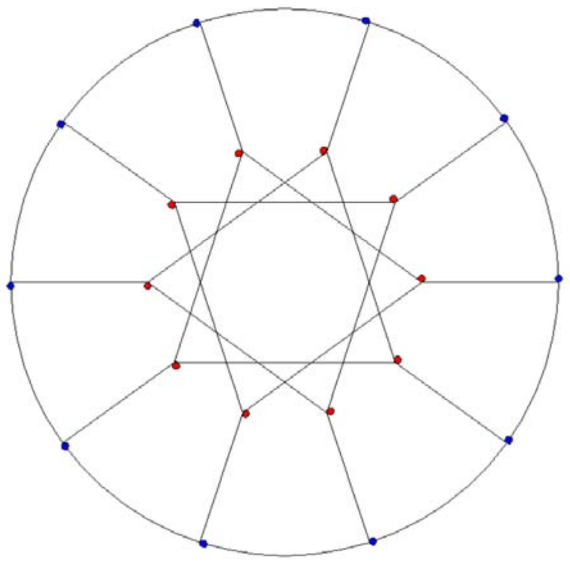
Figure 2: Desargues Graph

Using ChatGPT to Reproduce the Above Graph

After uploading the above graph to ChatGPT, we asked, "Based on the attached image/graph, could you develop a Python program to reproduce it?"

The Python program generated by ChatGPT produced the following image in Figure 2 (a).

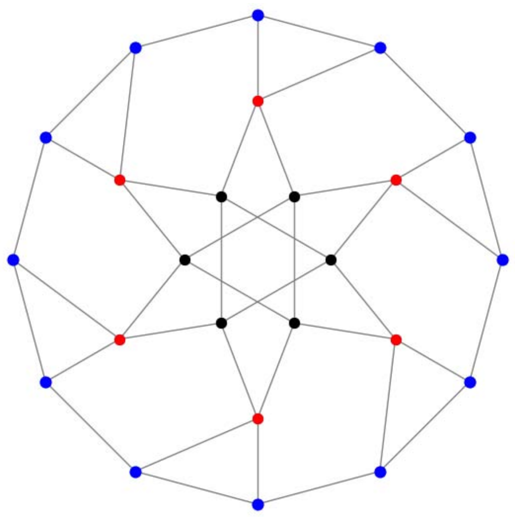
Figure 2 (a)

After uploading the above graph to ChatGPT, we asked, "Based on the attached image/graph, could you develop a Python program to reproduce it?"

The Python program generated by ChatGPT produced the following image in Figure 2 (b).

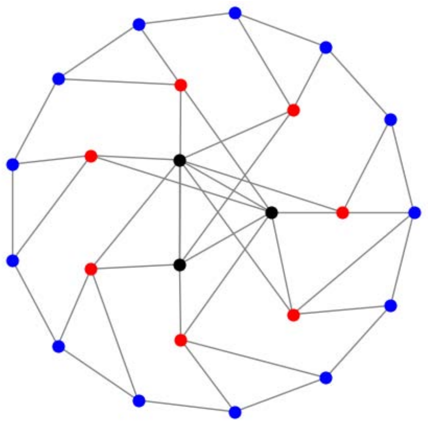
Figure 2 (b)

V. Herschel Graph

The Herschel graph is a bipartite undirected graph with 11 vertices and 18 edges. It is a polyhedral graph (the graph of a convex polyhedron). It is the smallest polyhedral graph that does not have a Hamiltonian cycle, a cycle passing through all its vertices. It is named after British astronomer Alexander Stewart Herschel, because Herschel studied Hamiltonian cycles in polyhedral graphs (but not of this graph) (Wikipedia contributors, 2023).

Herschel graph has three vertices of degree four and eight vertices of degree three. Each two distinct degree-four vertices share two degree-three neighbors, forming a four-vertex cycle with these shared neighbors. Three of these cycles pass through six of the eight degree-three vertices. Two more degree-three vertices do not participate in these four-vertex cycles; instead, each is adjacent to three of the six vertices (Wikipedia contributors, 2023; Lawson-Perfect, 2013).

Herschel's graph also provides an example of a polyhedral graph for which the medial graph has no Hamiltonian decomposition into two edge-disjoint Hamiltonian cycles. The medial graph of the Herschel graph is a 4-regular graph with 18 vertices, one for each edge of the Herschel graph; two vertices are adjacent in the medial graph whenever the corresponding edges of the Herschel graph are consecutive on one of its faces (Wikipedia contributors, 2023; (Bondy & Häggkvist, 1981).

Python Program for Creating Herschel Graph

```
import turtle
import math
t = turtle.Turtle()
t.speed("fastest")
t.penup()
t.goto(0,0)
t.pendown()
t.fillcolor("blue")
t.begin_fill()
t.circle(4)
t.end_fill()
t.penup()
t.goto(0,-300)
t.pendown()
t.circle(300)
radius = 300
for k in range(4):
   t.penup()
   t.goto(0,0)
   t.setheading(90*k)
   t.forward(radius)
   t.pendown()
   t.fillcolor("blue")
   t.begin_fill()
   t.circle(4)
   t.end_fill()
radius = 150
for k in range(6):
   t.penup()
   t.goto(0,0)
   t.setheading(60*k)
   t.forward(radius)
   t.pendown()
   t.fillcolor("red")
   t.begin_fill()
   t.circle(4)
   t.end_fill()
for k in range(1,3,1):
   t.penup()
   x1 = 150*math.cos(math.pi*((60*k)%360)/180)
   y1 = 150*math.sin(math.pi*((60*k)%360)/180)
   t.goto(x1,y1)
   t.pendown()
   t.goto(0,0)
for k in range(4,6,1):
   t.penup()
   x1 = 150*math.cos(math.pi*((60*k)%360)/180)
   y1 = 150*math.sin(math.pi*((60*k)%360)/180)
   t.goto(x1,y1)
   t.pendown()
   t.goto(0,0)
t.penup()
x1 = 300*math.cos(math.pi*((90*0)%360)/180)
y1 = 300*math.sin(math.pi*((90*0)%360)/180)
t.goto(x1,y1)
t.pendown()
x2 = 150*math.cos(math.pi*((60*0)%360)/180)
y2 = 150*math.sin(math.pi*((60*0)%360)/180)
t.goto(x2,y2)
t.penup()
x1 = 300*math.cos(math.pi*((90*1)%360)/180)
y1 = 300*math.sin(math.pi*((90*1)%360)/180)
t.goto(x1,y1)
t.pendown()
x2 = 150*math.cos(math.pi*((60*1)%360)/180)
y2 = 150*math.sin(math.pi*((60*1)%360)/180)
t.goto(x2,y2)
t.penup()
x1 = 300*math.cos(math.pi*((90*1)%360)/180)
y1 = 300*math.sin(math.pi*((90*1)%360)/180)
t.goto(x1,y1)
t.pendown()
x2 = 150*math.cos(math.pi*((60*2)%360)/180)
y2 = 150*math.sin(math.pi*((60*2)%360)/180)
t.goto(x2,y2)
t.penup()
x1 = 300*math.cos(math.pi*((90*2)%360)/180)
y1 = 300*math.sin(math.pi*((90*2)%360)/180)
t.goto(x1,y1)
t.pendown()
```

```
x2 = 150*math.cos(math.pi*((60*3)%360)/180)
y2 = 150*math.sin(math.pi*((60*3)%360)/180)
t.goto(x2,y2)
t.penup()
x1 = 300*math.cos(math.pi*((90*3)%360)/180)
y1 = 300*math.sin(math.pi*((90*3)%360)/180)
t.goto(x1,y1)
t.pendown()
x2 = 150*math.cos(math.pi*((60*4)%360)/180)
y2 = 150*math.sin(math.pi*((60*4)%360)/180)
t.goto(x2,y2)
t.penup()
x1 = 300*math.cos(math.pi*((90*3)%360)/180)
y1 = 300*math.sin(math.pi*((90*3)%360)/180)
t.goto(x1,y1)
t.pendown()
x2 = 150*math.cos(math.pi*((60*5)%360)/180)
y2 = 150*math.sin(math.pi*((60*5)%360)/180)
t.goto(x2,y2)
t.penup()
x1 = 150*math.cos(math.pi*((60*0)%360)/180)
y1 = 150*math.sin(math.pi*((60*0)%360)/180)
t.goto(x1,y1)
t.pendown()
x2 = 150*math.cos(math.pi*((60*1)%360)/180)
y2 = 150*math.sin(math.pi*((60*1)%360)/180)
t.goto(x2,y2)
t.penup()
x1 = 150*math.cos(math.pi*((60*0)%360)/180)
y1 = 150*math.sin(math.pi*((60*0)%360)/180)
t.goto(x1,y1)
t.pendown()
x2 = 150*math.cos(math.pi*((60*5)%360)/180)
y2 = 150*math.sin(math.pi*((60*5)%360)/180)
t.goto(x2,y2)
t.penup()
x1 = 150*math.cos(math.pi*((60*3)%360)/180)
y1 = 150*math.sin(math.pi*((60*3)%360)/180)
t.goto(x1,y1)
t.pendown()
x2 = 150*math.cos(math.pi*((60*2)%360)/180)
y2 = 150*math.sin(math.pi*((60*2)%360)/180)
t.goto(x2,y2)
t.penup()
x1 = 150*math.cos(math.pi*((60*3)%360)/180)
y1 = 150*math.sin(math.pi*((60*3)%360)/180)
t.goto(x1,y1)
t.pendown()
x2 = 150*math.cos(math.pi*((60*4)%360)/180)
y2 = 150*math.sin(math.pi*((60*4)%360)/180)
t.goto(x2,y2)
t.hideturtle()
```

We generate the Herschel graph shown in Figure 3 using the previous algorithm.

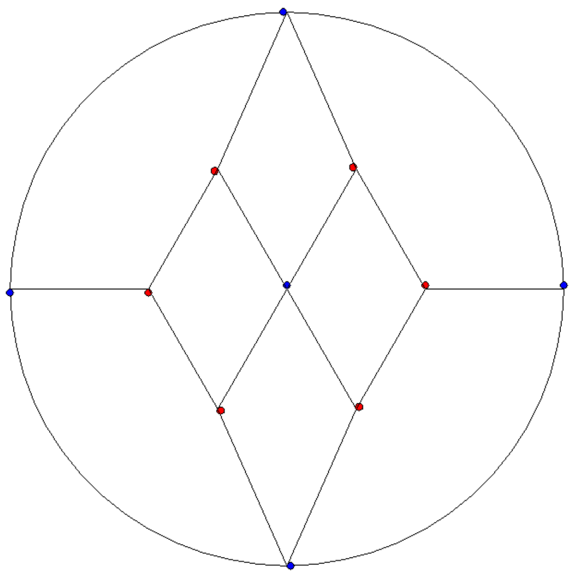

Figure 3: Herschel Graph

After uploading the above graph to ChatGPT, we asked, “Based on the attached image/graph, could you develop a Python program to reproduce it?”

The Python program generated by ChatGPT produced the following image in Figure 3 (a).

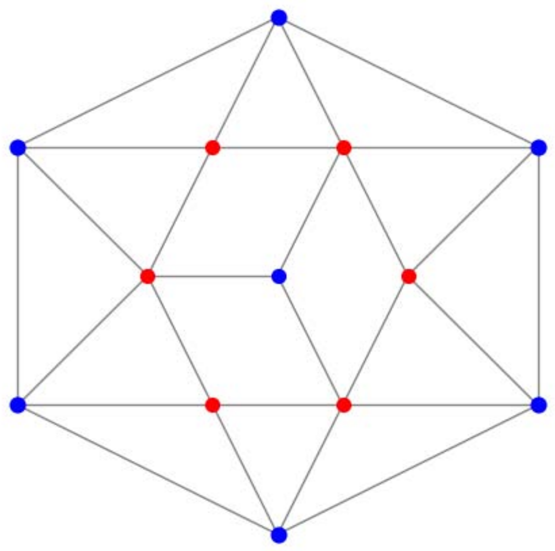

Figure 3 (a)

After uploading the above graph to ChatGPT, we asked, “Based on the attached image/graph, could you develop a Python program to reproduce it?”

The Python program generated by ChatGPT produced the following image in Figure 3 (b).

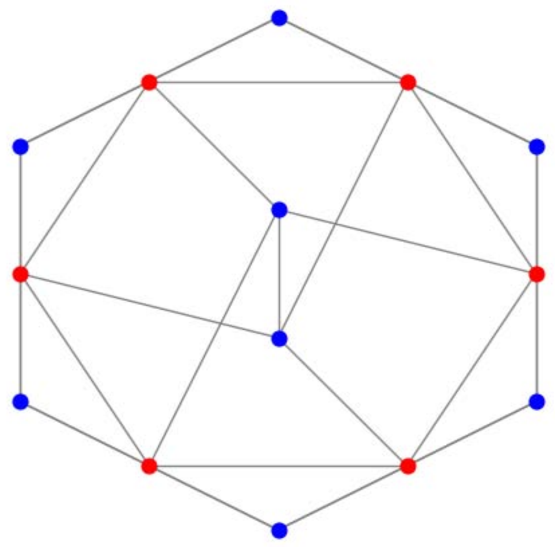

Figure 3 (b)

VI. Möbius–Kantor Graph

The Möbius–Kantor graph is a symmetric bipartite cubic graph with 16 vertices and 24 edges named after August Ferdinand Möbius and Seligmann Kantor. It can be defined as the generalized Petersen graph G(8,3): that is, it is formed by the vertices of an octagon, connected to the vertices of an eight-point star in which each point of the star is connected to the points three steps away from it (Wikipedia contributors, 2024).

Möbius–Kantor graph is a subgraph of the four-dimensional hypercube graph, formed by removing eight edges from the hypercube. Since the hypercube is a unit distance graph, the Möbius–Kantor graph can also be drawn in the plane with all edges unit length, although such a drawing will necessarily have some pairs of crossing edges. The Möbius–Kantor graph also often occurs as an induced subgraph of the Hoffman–Singleton graph. Each of these instances is, in fact, an eigenvector of the Hoffman-Singleton graph, with an associated eigenvalue of -3. Each vertex not in the induced Möbius–Kantor graph is adjacent to exactly four vertices in the Möbius–Kantor graph, two each in half of a bipartition of the Möbius–Kantor graph. The Möbius–Kantor graph cannot be embedded without crossings in the plane; it has crossing number 4, and is the smallest cubic graph with that crossing number (Wikipedia contributors, 2024; Coxeter, 1950).

The automorphism group of the Möbius–Kantor graph is a group of order 96. It acts transitively on the graph's vertices, edges, and arcs. Therefore, the Möbius–Kantor graph is a symmetric graph. It has automorphisms that take any vertex to any other vertex and any edge to any other edge. According to the Foster census, the Möbius–Kantor graph is the unique cubic symmetric graph with 16 vertices, and the smallest cubic symmetric graph is not also distance-transitive. The Möbius–Kantor graph is also a Cayley graph (Wikipedia contributors, 2024).

Python Program for Creating Möbius–Kantor Graph

```
import turtle
import math
t = turtle.Turtle()
t.speed("fastest")
t.penup()
t.goto(0,-300)
t.pendown()
t.circle(300)
radius = 300
for k in range(8):
    t.penup()
    t.goto(0,0)
    t.setheading(45*k)
    t.forward(radius)
    t.pendown()
    t.fillcolor("blue")
    t.begin_fill()
    t.circle(4)
    t.end_fill()
radius = 150
for k in range(8):
    t.penup()
    t.goto(0,0)
    t.setheading(45*k)
    t.forward(radius)
    t.pendown()
    t.fillcolor("red")
    t.begin_fill()
    t.circle(4)
    t.end_fill()
for k in range(8):
    t.penup()
    x1 = 150*math.cos(math.pi*((45*k)%360)/180)
    y1 = 150*math.sin(math.pi*((45*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 300*math.cos\
(math.pi*((45*k+45)%360)/180)
    y2 = 300*math.sin\
(math.pi*((45*k+45)%360)/180)
    t.goto(x2,y2)
for k in range(4):
    t.penup()
    x1 = 150*math.cos(math.pi*((45*k)%360)/180)
    y1 = 150*math.sin(math.pi*((45*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 150*math.cos\
(math.pi*((45*k+135)%360)/180)
    y2 = 150*math.sin\
(math.pi*((45*k+135)%360)/180)
    t.goto(x2,y2)
for k in range(4):
    t.penup()
    x1 = 150*math.cos(math.pi*((45*k)%360)/180)
    y1 = 150*math.sin(math.pi*((45*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 150*math.cos\
(math.pi*((45*k+225)%360)/180)
    y2 = 150*math.sin\
(math.pi*((45*k+225)%360)/180)
    t.goto(x2,y2)
t.penup()
x1 = 150*math.cos(math.pi*((45*4)%360)/180)
y1 = 150*math.sin(math.pi*((45*4)%360)/180)
t.goto(x1,y1)
t.pendown()
x2 = 150*math.cos(math.pi*((45*7)%360)/180)
y2 = 150*math.sin(math.pi*((45*7)%360)/180)
t.goto(x2,y2)
t.hideturtle()
```

Using the previous algorithm, we generate Möbius the Möbius-Kantor graph as shown in Figure 4.

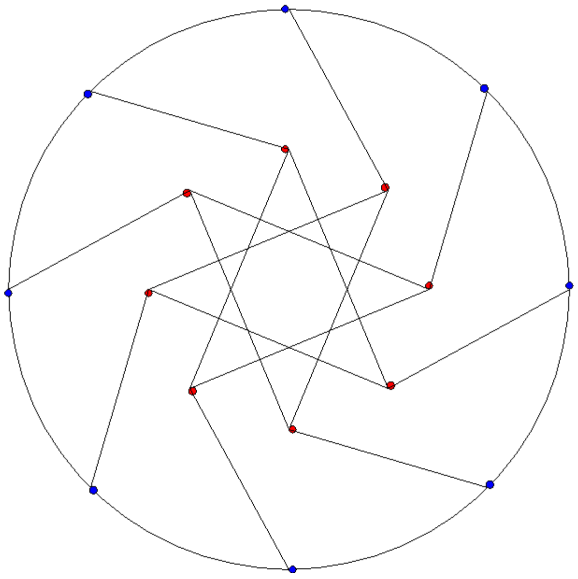

Figure 4: Möbius–Kantor graph

After uploading the above graph to ChatGPT, we asked, "Based on the attached image/graph, could you develop a Python program to reproduce it?"

The Python program generated by ChatGPT produced the following image in Figure 4 (a).

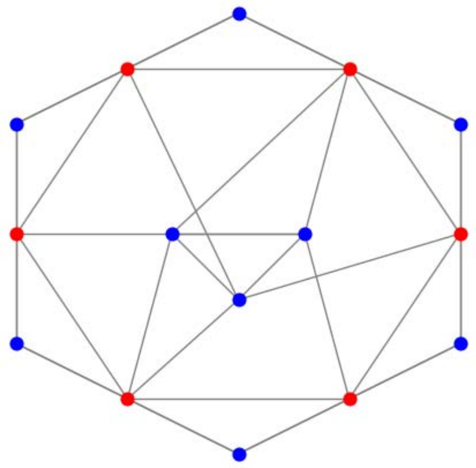

Figure 4 (a)

After uploading the above graph to ChatGPT, we asked, "Based on the attached image/graph, could you develop a Python program to reproduce it?"

The Python program generated by ChatGPT produced the following image in Figure 4 (b).

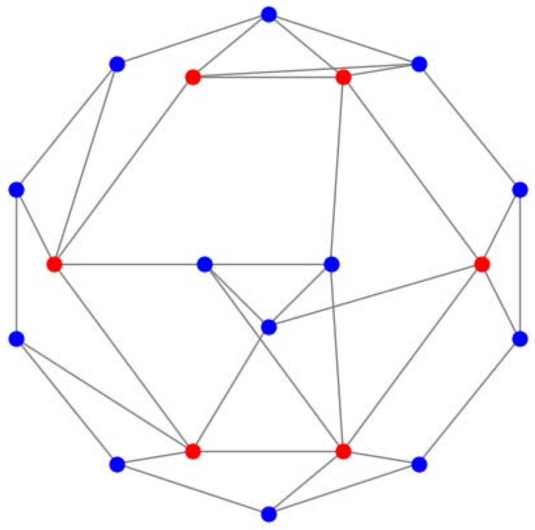

Figure 4 (b)

VII. Franklin Graph

The Franklin graph is a 3-regular graph with 12 vertices and 18 edges. Franklin graph is named after Philip Franklin, who disproved the Heawood conjecture on the number of colors needed when a two-dimensional surface is partitioned into cells by a graph embedding (Wikipedia contributors, 2022; Franklin, 1934).

The Heawood conjecture implied that the maximum chromatic number of a map on the Klein bottle should be seven, but Franklin proved that in this case six colors always suffice. (The Klein bottle is the only surface for which the Heawood conjecture fails.)

The Franklin graph can be embedded in the Klein bottle so that it forms a map requiring six colors, showing that six colors are sometimes necessary in this case. Franklin graph is Hamiltonian and has chromatic number 2, chromatic index 3, radius 3, diameter 3 and girth 4. It is also a 3-vertex-connected and 3-edge-connected perfect graph (Wikipedia contributors, 2022).

Python Program for Creating Franklin Graph

```
import turtle
import math
t = turtle.Turtle()
t.speed("fastest")
t.pensize(1)
for k in range(12):
   t.penup()
   x1 = 300*math.cos(math.pi*((30*k)%360)/180)
   y1 = 300*math.sin(math.pi*((30*k)%360)/180)
   t.goto(x1,y1)
   t.pendown()
   x2 = 300*math.cos\
(math.pi*((30*k+30)%360)/180)
   y2 = 300*math.sin\
(math.pi*((30*k+30)%360)/180)
   t.goto(x2,y2)
radius = 300
for k in range(12):
   t.penup()
   t.goto(0,0)
   t.setheading(30*k)
   t.forward(radius)
   t.pendown()
   t.fillcolor("blue")
   t.begin_fill()
   t.circle(4)
   t.end_fill()
radius = 300
for k in range(0,12,2):
   t.penup()
   x1 = radius*math.cos(math.pi*((30*k)%360)/180)
   y1 = radius*math.sin(math.pi*((30*k)%360)/180)
   t.goto(x1,y1)
   t.pendown()
   x2 = radius*math.cos\
(math.pi*((30*k+150)%360)/180)
   y2 = radius*math.sin\
(math.pi*((30*k+150)%360)/180)
   t.goto(x2,y2)
t.hideturtle()
```

Using the previous algorithm we generate the Franklin graph as shown in Figure 5.

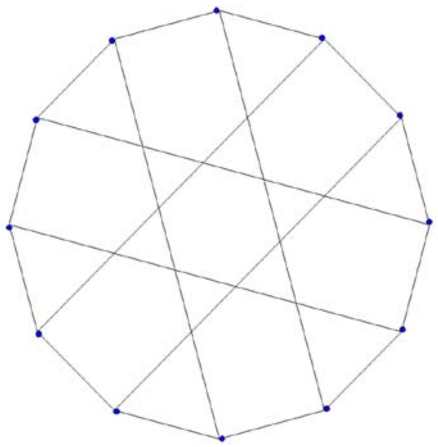

Figure 5: Franklin graph

After uploading the above graph to ChatGPT, we asked, “Based on the attached image/graph, could you develop a Python program to reproduce it?”

The Python program generated by ChatGPT produced the following image in Figure 5 (a).

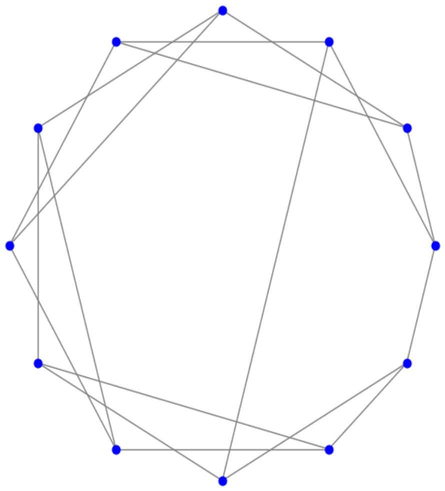

Figure 5 (a)

After uploading the above graph to ChatGPT, we asked, “Based on the attached image/graph, could you develop a Python program to reproduce it?”

The Python program generated by ChatGPT produced the following image in Figure 5 (b).

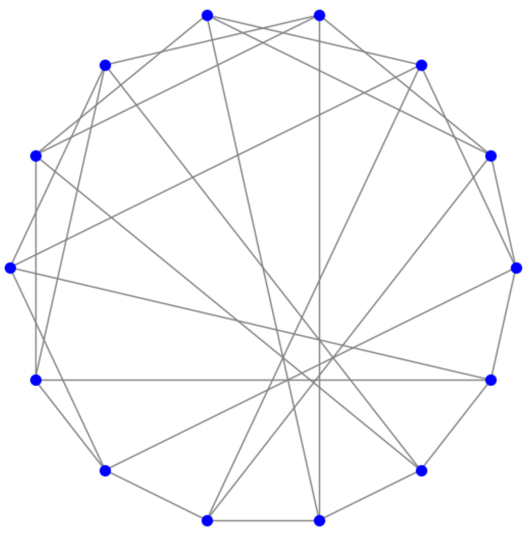

Figure 5 (b)

VIII. Truncated Icosahedral Graph

In geometry, the truncated icosahedron is a polyhedron that can be constructed by truncating all of the regular icosahedron's vertices. Intuitively, it may be regarded as footballs (or soccer balls) that are typically patterned with white hexagons and black pentagons. It can be found in the application of geodesic dome structures such as those whose architecture Buckminster Fuller pioneered are often based on this structure. It is an example of an Archimedean solid, as well as a Goldberg polyhedron (Weisstein, 2025).

According to Steinitz's theorem, the skeleton of a truncated icosahedron, like that of any convex polyhedron, can be represented as a polyhedral graph, meaning a planar graph (one that can be drawn without crossing edges) and 3-vertex-connected graph (remaining connected whenever two of its vertices are removed). The graph is known as *truncated icosahedral graph,* with 60 vertices and 90 edges. It is an Archimedean graph because it resembles one of the Archimedean solids. It is a cubic graph, meaning that each vertex is incident to exactly three edges. It is sometimes known as the Buckminster Fullerene graph (Weisstein, 2025; Wikipedia contributors, 2024).

Python Program for Creating truncated icosahedral graph

```
import turtle
import math
t = turtle.Turtle()
t.speed("fastest")
#20-gons
for k in range(20):
   t.penup()
   t.goto(0,0)
   t.setheading(18*k)
   t.forward(300)
   t.pendown()
   t.fillcolor("black")
   t.begin_fill()
   t.circle(4)
   t.end_fill()
   t.penup()
   x1 = 300*math.cos(math.pi*((18*k)%360)/180)
   y1 = 300*math.sin(math.pi*((18*k)%360)/180)
   t.goto(x1,y1)
   t.setposition(x1, y1)
   t.pendown()
   letter = str(k+1)
   t.color('black')
   t.write(letter, align="right", font=("Verdana", 13,
"normal"))
   t.color('black')
   x2 = 300*math.cos\
(math.pi*((18*k+18)%360)/180)
   y2 = 300*math.sin(math.pi*((18*k+18)%360)/180)
   t.goto(x2,y2)
#20 red vertices
for k in range(20):
   t.penup()
   t.goto(0,0)
   t.setheading(18*k)
   t.forward(240)
   t.pendown()
   t.fillcolor("red")
   t.begin_fill()
   t.circle(4)
   t.end_fill()
   x1 = 240*math.cos(math.pi*((18*k)%360)/180)
   y1 = 240*math.sin(math.pi*((18*k)%360)/180)
   t.goto(x1,y1)
   t.setposition(x1, y1)
   t.pendown()
   letter = str(20+k+1)
   t.color('red')
   t.write(letter, align="right", font=("Verdana", 13,
"normal"))
   t.color('black')
#10 blue vertices
radius = 180
for k in range(10):
   t.penup()
   t.goto(0,0)
   t.setheading(36*k)
   t.forward(radius)
   t.pendown()
```

```
    t.fillcolor("blue")
    t.begin_fill()
    t.circle(4)
    t.end_fill()
    x1 = 180*math.cos(math.pi*((36*k)%360)/180)
    y1 = 180*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x1,y1)
    t.setposition(x1, y1)
    t.pendown()
    letter = str(40+k+1)
    t.color('blue')
    t.write(letter, align="right", font=("Verdana", 13,
"normal"))
    t.color('black')
# 10 green vertices
radius = 120
for k in range(10):
    t.penup()
    t.goto(0,0)
    t.setheading(36*k)
    t.forward(radius)
    t.pendown()
    t.fillcolor("green")
    t.begin_fill()
    t.circle(4)
    t.end_fill()
    x1 = 120*math.cos(math.pi*((36*k)%360)/180)
    y1 = 120*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x1,y1)
    t.setposition(x1, y1)
    t.pendown()
    letter = str(50+k+1)
    t.color('green')
    t.write(letter, align="right", font=("Verdana", 13,
"normal"))
    t.color('black')
#edges between black and red vertices
for k in range(0,19,2):
    t.penup()
    x1 = 300*math.cos(math.pi*((18*k)%360)/180)
    y1 = 300*math.sin(math.pi*((18*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    t.color('red')
    x2 = 240*math.cos(math.pi*((18*k+18)%360)/180)
    y2 = 240*math.sin(math.pi*((18*k+18)%360)/180)
    t.goto(x2,y2)
    t.color('black')
for k in range(0,19,2):
    t.penup()
    x1 = 240*math.cos(math.pi*((18*k)%360)/180)
    y1 = 240*math.sin(math.pi*((18*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    t.color('green')
    x2 = 300*math.cos\
(math.pi*((18*k+18)%360)/180)
    y2 = 300*math.sin(math.pi*((18*k+18)%360)/180)
    t.goto(x2,y2)
    t.color('black')
# edges between red vertices
for k in range(1,20,2):
    t.penup()
    x1 = 240*math.cos(math.pi*((18*k)%360)/180)
    y1 = 240*math.sin(math.pi*((18*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 240*math.cos\
(math.pi*((18*k+18)%360)/180)
    y2 = 240*math.sin(math.pi*((18*k+18)%360)/180)
    t.goto(x2,y2)
#edges between red and blue vertices
for k in range(10):
    t.penup()
    x1 = 180*math.cos(math.pi*((36*k)%360)/180)
    y1 = 180*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 240*math.cos(math.pi*((2*18*k)%360)/180)
    y2 = 240*math.sin(math.pi*((2*18*k)%360)/180)
    t.goto(x2,y2)
for k in range(10):
    t.penup()
    x1 = 180*math.cos(math.pi*((36*k)%360)/180)
    y1 = 180*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 240*math.cos\
(math.pi*((2*18*k+54)%360)/180)
    y2 = 240*math.sin(math.pi*((2*18*k+54)%360)/180)
    t.goto(x2,y2)
# edges between blue and green vertices
for k in range(10):
    t.penup()
    x1 = 180*math.cos(math.pi*((36*k)%360)/180)
    y1 = 180*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 120*math.cos(math.pi*((36*k)%360)/180)
    y2 = 120*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x2,y2)
# edges between green and green vertices
for k in range(0,10,2):
    t.penup()
    x1 = 120*math.cos(math.pi*((36*k)%360)/180)
    y1 = 120*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 120*math.cos\
(math.pi*((36*k+72)%360)/180)
    y2 = 120*math.sin(math.pi*((36*k+72)%360)/180)
    t.goto(x2,y2)
for k in range(1,10,2):
    t.penup()
    x1 = 120*math.cos(math.pi*((36*k)%360)/180)
    y1 = 120*math.sin(math.pi*((36*k)%360)/180)
    t.goto(x1,y1)
    t.pendown()
    x2 = 120*math.cos\
(math.pi*((36*k+72)%360)/180)
    y2 = 120*math.sin(math.pi*((36*k+72)%360)/180)
    t.goto(x2,y2)
t.hideturtle()
```

Using the previous algorithm we generate the truncated icosahedral graph as shown in Figure 6.

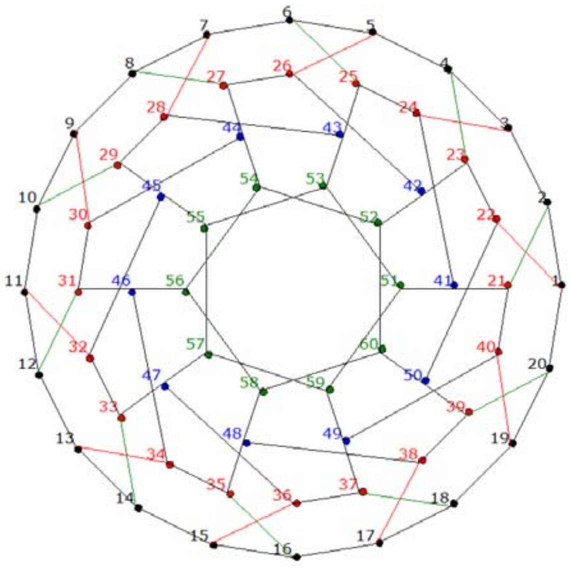

Figure 6: Truncated icosahedral graph

After uploading the above graph to ChatGPT, we asked, "Based on the attached image/graph, could you develop a Python program to reproduce it?"

The Python program generated by ChatGPT produced the following image in Figure 6 (a).

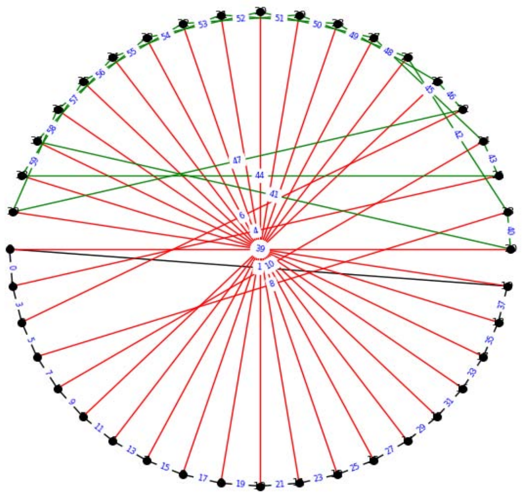

Figure 6 (a)

After uploading the above graph to ChatGPT, we asked, "Based on the attached image/graph, could you develop a Python program to reproduce it?"

The Python program generated by ChatGPT produced the following image in Figure 5 (b).

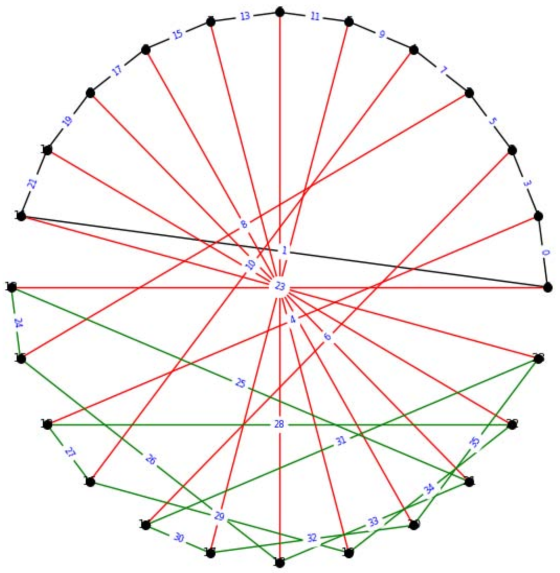

Figure 6 (b)

IX. Triangular Grid Graph

The triangular grid graph T_n is the lattice graph obtained by interpreting the order-(n+1) triangular grid as a graph, with the intersection of grid lines being the vertices and the line segments between vertices being the edges. Equivalently, it is the graph on vertices (i, j, k) with i, j, k being nonnegative integers summing to n such that vertices are adjacent if the sum of absolute differences of the coordinates of two vertices is 2.

The graph bandwidth of T_n is n+1. T_n is also the hexagonal king graph of order n, i.e., the connectivity graph of possible moves of a king chess piece on a hexagonal chessboard (West, 2000; Weisstein, 2025).

Python Program for Creating T_n

```
import turtle
import math
t = turtle.Turtle()
t.speed("fastest")
def Triangular_Grid_Graph(n):
    size = 600//n
    for k in range(0, n+1):
        x_cor = -300+k*size
        for i in range(k+1):
            t.penup()
            t.goto(x_cor,300-i*size*math.sqrt(3)/2)
            t.pendown()
            t.fillcolor("red")
            t.begin_fill()
            t.circle(2)
            t.end_fill()
            x_cor =  x_cor - size/2
    t.color('black')
    for k in range(0, n):
        x_cor = -300+k*size
        for i in range(k+1):
            t.penup()
            t.goto(x_cor,300-i*size*math.sqrt(3)/2)
            t.setposition(x_cor,300-i*size*math.sqrt(3)/2)
            t.pendown()
            t.goto(x_cor + size,300-i*size*math.sqrt(3)/2)
            x_cor =  x_cor - size/2
    for k in range(0, n):
         x_cor = -300+k*size
         for i in range(k+1):
            t.penup()
            t.goto(x_cor,300-i*size*math.sqrt(3)/2)
            t.setposition(x_cor,300-i*size*math.sqrt(3)/2)
            t.pendown()
          t.goto(x_cor+size/2,300(i+1)*size*math.sqrt(3)/2)
            x_cor =  x_cor - size/2
    for k in range(0, n):
         x_cor = -300+(k+1)*size
         for i in range(k+1):
            t.penup()
            t.goto(x_cor,300-i*size*math.sqrt(3)/2)
            t.setposition(x_cor,300-i*size*math.sqrt(3)/2)
            t.pendown()
            t.goto(x_cor-size/2,300(i+1)*size*math.sqrt(3)/2)
            x_cor =  x_cor - size/2
    t.hideturtle()
```

Using the previous algorithm we generate the truncated icosahedral graphs T_{15} and T_{30} as shown in Figure 7-A and Figure 7-B, respectively.

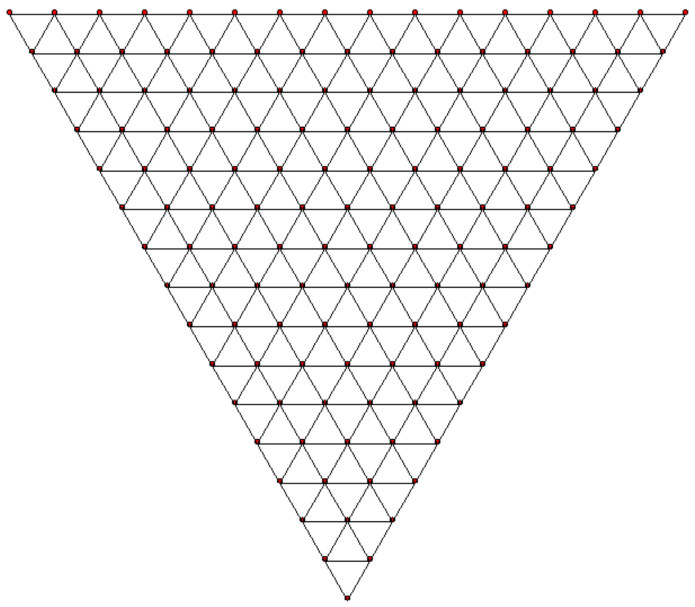

Figure 7-A: Truncated Icosahedral Graph T_{15}

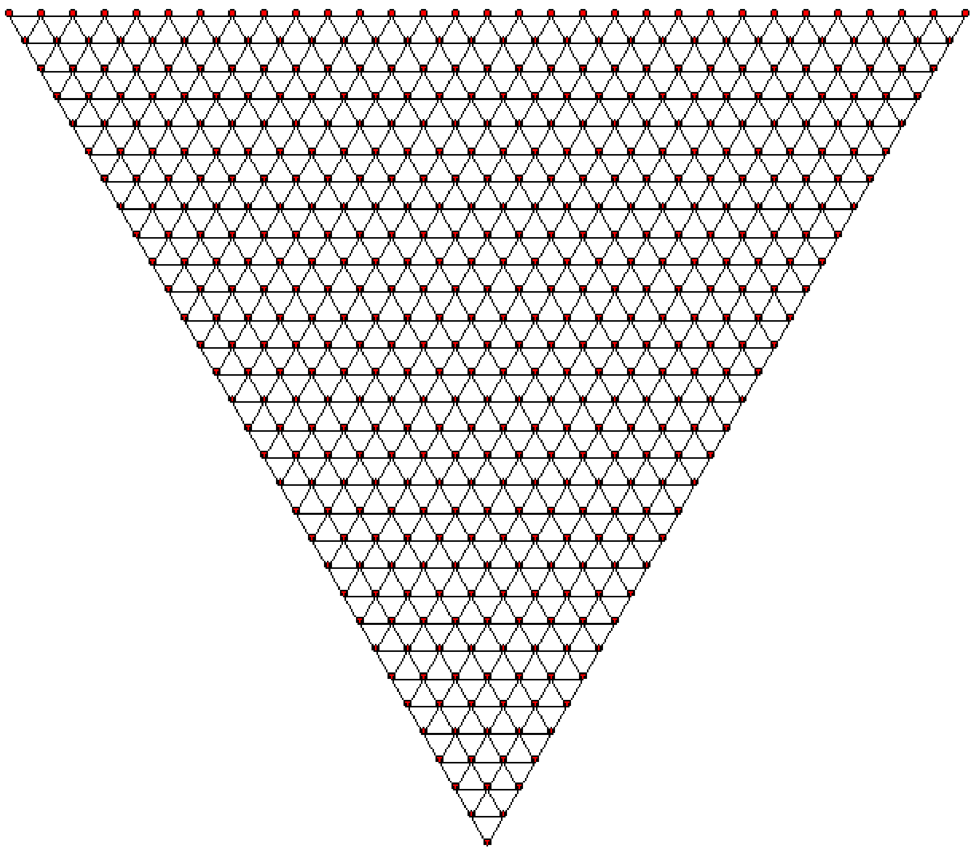

Figure 7-B: Truncated Icosahedral Graph T_{30}

After uploading the above graph to ChatGPT, we asked, “Based on the attached image/graph, could you develop a Python program to reproduce it?”

The Python program generated by ChatGPT produced the following image in Figure 7-B (a).

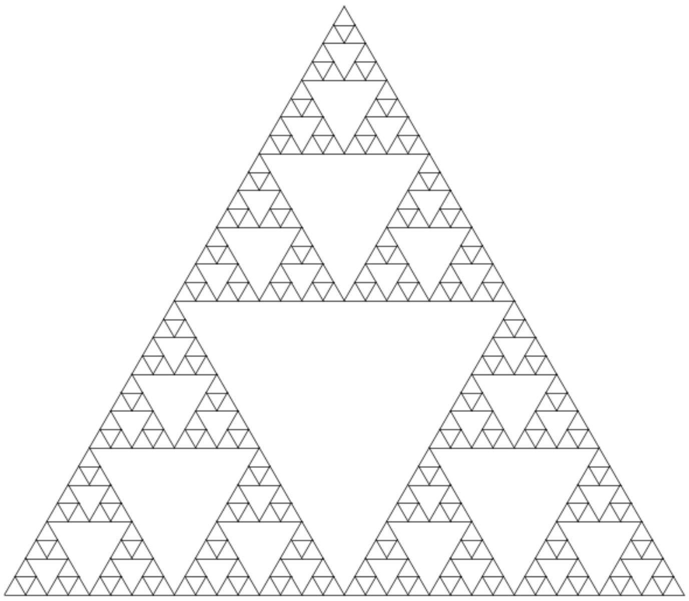

Figure 7-B (a)

After uploading the above graph to ChatGPT, we asked, "Based on the attached image/graph, could you develop a Python program to reproduce it?"

The Python program generated by ChatGPT produced the following image in Figure 7-B (b).

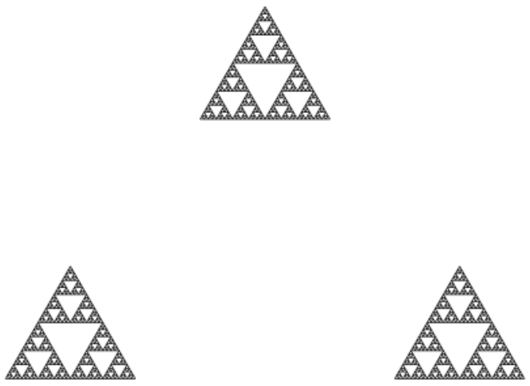

Figure 7-B (b)

X. Integrating Knowledge Management Into Computational Graph Theory Education

Knowledge management (KM) is pivotal in bridging theory and application in computational graph theory, particularly when using Python to model complex mathematical structures. As graph construction increasingly leverages algorithmic logic and programming, effective KM strategies are essential for facilitating interdisciplinary learning, enhancing educational outcomes, and optimizing research processes.

The Knowledge Management Mesosystem Model (Gao & Gao, in press) offers a structured framework that supports the integration of human expertise, algorithmic design, and AI-assisted discovery in educational environments. It comprises three interdependent layers: the Knowledge/Human Layer, the Yin-Yang Knowledge Development and Sharing Layer, and the Data/Machine Layer. These layers align well with the iterative process of coding, testing, and visualizing graphs, allowing students and researchers to transition seamlessly between theory development and practical application.

Instructors can cultivate higher-order thinking, collaboration, and computational creativity by incorporating KM strategies into graph theory education. Gao et al. (2025) emphasize the role of innovative teaching practices in business analytics, which mirror similar approaches in computational mathematics-where hands-on programming tasks and active learning deepen student engagement. Moreover, integrating AI into KM workflows allows for more dynamic interaction between human logic and machine-generated insights, facilitating advanced problem-solving and deeper conceptual understanding (Gao et al., 2024).

Russ (2021) further highlights the necessity of sustainable KM in technology-driven disciplines, underscoring how ethical AI usage and data governance must accompany algorithmic exploration. The symbiotic relationship between knowledge creation and dissemination within KM frameworks is crucial when teaching programming-based graph construction, where students learn from existing models and contribute to evolving digital knowledge ecosystems.

In summary, embedding KM principles into computational graph theory enriches the learning experience, encourages innovation, and ensures a sustainable, interdisciplinary approach to knowledge generation in the era of intelligent technologies.

XI. Responsible Integration of AI in Computational Research and Education

As artificial intelligence (AI) becomes more integrated into educational and research contexts, adopting a balanced and responsible approach to its use is essential. In computational fields such as graph theory, AI can support algorithm development, automate visualization, and even suggest code for complex graph structures. However, using AI wisely means recognizing its role as a complement to-not a replacement for-human logic, creativity, and critical thinking.

Gao et al. (2024) highlights that while AI can generate solutions and assist in mathematical reasoning, it must be tempered with human oversight to ensure accuracy, especially in domains requiring rigorous proofs and logical consistency. Misusing generative AI—such as uncritically accepting outputs without validation—can lead to erroneous conclusions and undermine academic integrity.

The Knowledge Management Mesosystem Model (Gao & Gao, in press) provides a helpful framework for guiding wise AI integration. Its Data/Machine Layer emphasizes AI-assisted learning while maintaining a strong role for human decision-making. Ethical considerations, data governance, and contextual understanding must be part of any AI-driven educational or research activity.

Furthermore, wise AI use aligns with Russ's (2021) model of sustainable knowledge management, which calls for thoughtful integration of technology to enhance-not replace-human cognitive processes. In a programming-rich environment like Python-based graph construction, students and researchers should use AI to augment their understanding: generating baseline code, debugging, or exploring design variations while still being actively involved in problem-solving and model evaluation.

Ultimately, using AI wisely means fostering an interdisciplinary mindset where machine intelligence supports but does not eclipse human reasoning. When guided by ethical principles and pedagogical goals, AI can significantly enhance the teaching, learning, and research of mathematical and computational topics.

XII. Prompt Engineering and its Role in Graph Construction

Prompt engineering, the art of crafting precise and effective instructions to guide large language models (LLMs), has become essential in leveraging generative artificial intelligence for diverse tasks, including mathematical problem-solving and programming support. In the context of this study, prompt engineering was pivotal in engaging tools like ChatGPT to recreate Python visualizations of classic graphs in graph theory. By formulating well-structured prompts-such as asking for a Python program to replicate a given graph image-researchers could derive functional code outputs that accurately reproduced complex structures like the Wagner and Desargues graphs.

The power of prompt engineering lies in its ability to direct AI toward high-quality, context-aware responses. As Hernández et al. (2024) described, successful interactions with LLMs depend heavily on clarity, specificity, and contextual cues within the prompt. Their work provides over 100 examples, demonstrating that prompt quality significantly impacts response effectiveness across domains. Similarly, Mastering Generative AI and Prompt Engineering underscores that prompt engineering enhances productivity and creativity by allowing users to customize AI output to specific goals, such as generating reproducible code or verifying mathematical properties (Data Science Horizons, 2024).

In graph theory education and computational research, prompt engineering bridges human intent and machine-generated assistance. It transforms LLMs from passive responders into collaborative problem-solvers capable of producing code that is not only syntactically correct but also aligned with theoretical graph attributes. As Python continues to serve as a primary medium for algorithmic exploration, prompt engineering empowers both students and researchers to interact more effectively with AI models, thus streamlining the process of constructing, analyzing, and visualizing graphs.

XIII. Conclusion

In the Information Age where information inflows and outflows are rapid, complex, and dynamically interspersed within highly uncertain environments, the imperative nature of the necessity for algorithmic learning in higher education has become increasingly clear. Not only is algorithmic learning considered to be integral within the Information and Communications Technology (ICT) domain (Byrka, Sushchenko, Luchko, Perun, & Luchko, 2024), but it is among the most sought after skill for millennials (Ananiadou & Claro, 2009). More importantly, the inculcation of algorithmic learning can help reify an otherwise esoteric way of thinking and, therefore, learning, by helping students organize their thoughts logically in a stepwise fashion.

This research has demonstrated how classic graphs in graph theory can be effectively constructed, visualized, and analyzed using Python programming. By focusing on historically significant and mathematically rich graphs such as the Wagner, Desargues, Herschel, Möbius–Kantor, Franklin, truncated icosahedral, and triangular grid graphs, this study bridges the gap between abstract mathematical theory and tangible computational implementation.

Python's turtle module proved to be a valuable tool for graph rendering, offering a visually intuitive means of exploring structural properties such as regularity, symmetry, Hamiltonicity, and chromatic characteristics. Using trigonometric and geometric reasoning in these Python scripts encourages learners to connect theoretical graph definitions with algorithmic design, deepening mathematical understanding and programming skills.

A significant contribution of this study is the integration of generative AI, particularly ChatGPT, through prompt engineering to reproduce and verify Python code for graph construction. This dual approach validates the manually written code and introduces learners to collaborative human-AI workflows in computational mathematics. Prompt engineering emerged as a vital skill in effectively guiding AI tools, enabling the generation of meaningful and accurate programming solutions aligned with graph-theoretic goals.

Moreover, this paper underscores the educational potential of combining coding with visual mathematics. Students and researchers gain a deeper appreciation for graph properties and computational logic by implementing classic graphs in Python. The methodology presented here is replicable and scalable, making it ideal for classroom use, student research, and interdisciplinary applications across science, engineering, and computer science.

Ultimately, this work contributes a practical and pedagogically sound approach to teaching and exploring graph theory. Hands-on programming and responsible AI integration fosters computational literacy and inspires further innovation at the intersection of mathematics, computer science, and education.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare no conflicts of interest.

Funding Statement

This study did not receive any funding in any form.

Acknowledgment

This paper was inspired and initiated by the talks, posters, and conversations at PyCon US 2024, held in Pittsburgh, Pennsylvania, from May 17 to 19, 2024. The first author attended the conference with financial support from the organizers. The work also drew upon his teaching materials for Introduction to Computing at the University of Richmond and Computer Programming I & II at Virginia State University. Part of the research was conducted during his visit to Elizabeth City State University (ECSU) from July 14 to 19, 2024, supported by an ECSU mini-grant. Additional insights were gained from two recent publications: Gao and Donald (2024) and Gao, Gao, Allagan, and Su (2025).

References Références Referencias

1. Ananiadou, K., Claro, M. (2009). 21st century skills and competences for new millennium learners in OECD countries. *OECD Working Papers, 41.* Paris: OECD Publishing.
2. Byrka, M., Sushchenko, A., Luchko, V., Perun, G., & Luchko, V. (2024). Algorithmic thinking in higher education: Determining observable measurable content. *Information Technologies and Learning Tools, 104*(6), 1-13.
3. Bondy, J. A., & Murty, U. S. R. (2007). *Graph theory* (pp. 275–276). Springer.
4. Wikipedia contributors. (2024, January 27). *Wagner graph. In Wikipedia, the Free Encyclopedia*. Retrieved March 21, 2025, from https://en.wikipedia.org/w/index.php?title=Wagner_graph&oldid=1199505619
5. Soifer, A. (2008). *The mathematical coloring book* (p. 245). Springer-Verlag.
6. Jakobson, Dmitry; Rivin, Igor (1999). "On some extremal problems in graph theory". *arXiv:math.CO/9907050.*
7. Bodlaender, H. L. (1998). A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science, 209(1–2), 1–45. https://doi.org/10.1016/S0304-3975(97)00228-4
8. Bodlaender, H. L., & Thilikos, D. M. (1999). Graphs with branchwidth at most three. *Journal of Algorithms*, 32(2), 167–194. https://doi.org/10.1006/jagm.1999.1011
9. Wagner, K. (1970). *Graphentheorie* (B.J. Hoch schul taschenbücher 248/248a, p. 61). Mannheim.
10. Lovász, L. (2006). Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1), 75–86. https://doi.org/10.1090/S0273-0979-05-01088-8

11. Wikipedia contributors. (2024, August 3). *Desargues graph*. In *Wikipedia*, *the Free Encyclopedia*. Retrieved March 20, 2025, from https://en.wikipedia.org/w/index.php?title=Desargues_graph&oldid=1238340280
12. Balaban, A. T., Fărcaşiu, D., & Bănică, R. (1966). Graphs of multiple 1, 2-shifts in carbonium ions and related systems. *Revue Roumaine de Chimie*, 11, 1205.
13. Mislow, K. (1970). Role of pseudorotation in the stereochemistry of nucleophilic displacement reactions. *Accounts of Chemical Research*, 3(10), 321–331. https://doi.org/10.1021/ar50034a001
14. Brouwer, A. E., Cohen, A. M., & Neumaier, A. (1989). *Distance-regular graphs*. Springer-Verlag.
15. Wikipedia contributors. (2023, December 6). *Herschel graph*. In *Wikipedia, the Free Encyclopedia*. Retrieved March 21, 2025, from https://en.wikipedia.org/w/index.php?title=Herschel_graph&oldid=1188570810
16. Lawson-Perfect, C. (2013, October 13). *An enneahedron for Herschel*. *The Aperiodical*. https://aperiodical.com/2013/10/an-enneahedron-for-herschel/
17. Bondy, J. A., & Häggkvist, R. (1981). Edge-disjoint Hamilton cycles in 4-regular planar graphs. *Aequationes Mathematicae, 22*(1), 42–45. https://doi.org/10.1007/BF02190157
18. Wikipedia contributors. (2024, July 24). *Möbius–Kantor graph. In Wikipedia, the Free Encyclopedia*. Retrieved March 22, 2025 from https://en.wikipedia.org/w/index.php?title=M%C3%B6bius%E2%80%93Kantor_graph&oldid=1236308906
19. Coxeter, H. S. M. (1950). Self-dual configurations and regular graphs. *Bulletin of the American Mathematical Society*, *56*(5), 413–455. https://doi.org/10.1090/S0002-9904-1950-09407-5
20. Wikipedia contributors. (2022, March 14). *Franklin graph*. In *Wikipedia, the Free Encyclopedia*. Retrieved March 23, 2025, from https://en.wikipedia.org/w/index.php?title=Franklin_graph&oldid=1077059121
21. Franklin, P. (1934). A six color problem. *Journal of Mathematics and Physics, 13*, 363–379. https://doi.org/10.1002/sapm1934131363
22. Wikipedia contributors. (2024, July 28). *Truncated icosahedron*. In *Wikipedia, the Free Encyclopedia*. Retrieved March 23, 2025, from https://en.wikipedia.org/w/index.php?title=Truncated_icosahedron&oldid=1237263616
23. Weisstein, E. W. (2025). *Truncated icosahedral graph*. From MathWorld-A Wolfram Web Resource. https://mathworld.wolfram.com/TruncatedIcosahedralGraph.html
24. West, D. B. (2000). Introduction to graph theory (2nd ed., pp. 390–392). Prentice Hall.
25. Weisstein, E. W. (2025). Triangular grid graph. MathWorld-A Wolfram Web Resource. https://mathworld.wolfram.com/TriangularGridGraph.html
26. Gao, S., Gao, W., Allagan, J., & Su, J. (2025). Innovative teaching in business analytics: Bridging theory, practice, and student engagement. *Journal of Technology Research*, 12. Retrieved from https://www.aabri.com/jtr.html
27. Gao, S., Gao, W., Malomo, O., Allagan, J., Eyob, E., Challa, C., & Su, J. (2024). Exploring the interplay between AI and human logic in mathematical problem-solving. *Online Journal of Applied Knowledge Management*, 12(1), 73--93. https://doi.org/10.36965/OJAKM.2024.12(1)73-93
28. Gao, S., & Gao, W. (in press). Enhancing business education with knowledge management meso system model. In M. Russ & M. Lytras (Eds.), *AI-driven knowledge management: Strategies for the modern business landscape*. Emerald Publishing.
29. Russ, M. (2021). Knowledge management for sustainable development in the era of continuously accelerating technological revolutions: A framework and models. *Sustainability*, 13(6), 3353. https://doi.org/10.3390/su13063353
30. Data Science Horizons. (2024). *Mastering generative AI and prompt engineering: A practic*al guide for data scientists.
31. Hernández, J. A., Conde, J., Querol, B., Martínez, G., & Reviriego, P. (2024). *ChatGPT: Learning prompt engineering with 100+ examples*. Madrid.
32. Gao, S., Gao, W., Allagan, J., & Su, J. (2025). Integrating Python and Generative AI for graph theory visualization and problem-solving. In *Proceedings of the 2025 International Conference on the AI Revolution: Research, Ethics, and Society* (AIR-RES 2025: April 14--16, 2025, Las Vegas, USA). Springer Nature (forthcoming).
33. Gao, J. M., & Donald, A. M. (2024). Blending Computational Thinking and Creativity: Algorithmic Art with Python. In *2024 International Conference on Computational Science and Computational Intelligence (CSCI)*. Springer Nature (forthcoming).