Analyzing Constellations using Graph Theory

Cluster Innovation Centre
University of Delhi

Kritika Verma
Siddhartha Mahajan

July 2022

Month Long Project submitted for the paper
Understanding real life situations through Discrete Mathematics

Certificate of Originality

The work embodied in this report entitled **"Analyzing Constellations using Graph Theory"** has been carried out by **Kritika Verma** and **Siddhartha Mahajan** for the paper **"Understanding real life situations through Discrete Mathematics"**. I declare that the work and language included in this project report is free from any kind of plagiarism.

_____

(Name and Signature)

Acknowledgement

With a deep sense of gratitude, we express our dearest indebtedness of Dr. Nirmal Yadav for their support throughout the duration of our project. We would like to thank them for giving us the opportunity to do this wonderful project. Their learned advice and constant encouragement has helped us to complete this project. It is a privilege for us to be their students.

Analyzing Constellations using Graph Theory

by

Kritika Verma
BTech ITMI 2021-25, Cluster Innovation Center,
University of Delhi

Siddhartha Mahajan
BTech ITMI 2021-25, Cluster Innovation Center,
University of Delhi

This report focuses on graph properties of the constellations including their Hamiltonian path and circuit,and Euler's path and circuit

Index

1 Introduction

1.1 Background and Context

Humans have been seeing and studying the stars since forever. During all that time, many of them have found apparent shapes in the distribution of the stars. These shapes or constellations have been a part of human culture and religion since forever. Its an ability of the human mind to find patterns everywhere, even if there are none. If we see the stars as vertices and the lines of the constellations as edges, the constellations are all graphs. We've tried to find the properties of these graphs.

1.2 Scope and Objectives

- Find an algorithm for Hamiltonian Path/Circuit.
- Find an algorithm for Eulerian Path/Circuit
- Figure out if the Adjacency Matrix is sufficient input in these Algorithms.
- Make a graph from the Adjacency Matrix and vice-versa.
- Convert all the useful constellations into graphs.
- Make a user friendly GUI to show all the work.
- To enhance our skills in LaTeX.
- Learn how to collaborate better on group projects.

1.3 Achievements

- We found algorithms for Hamiltonian Path/Circuit and worked with the most suitable one.

- We found algorithms for Eulerian Path/Circuit and worked with the most suitable one.
- The Adjacency Matrix was a sufficient input in these Algorithms.
- We made a graph from the Adjacency Matrix of it and vice-versa.
- We converted 32 Constellations into graphs and tested the algorithms on them.
- We made a user friendly GUI to show all our work.
- We have enhanced our skills in LaTeX.
- We have enhanced our skills in Python.
- We learnt how to collaborate better on group projects.
- We indulged in some recreational Mathematics.

2 Formulation of the Problem

2.1 Problem Statement

We worked on the following problem statements:

- The algorithm of finding the Hamiltonian Path by hand is very abstract. Find a better and reliable algorithm that computers can work with.
- To some it feels like representing a graph as its Adjacency Matrix is not useful in the case of finding the Eulerian or Hamiltonian path/circuit. Using only the adjacency matrix to find the circuits will give a better understanding to it all.
- The constellations are not Mathematical entities. Hence, using graph theory to see them gives us a new perspective and better understanding of our concepts of graph theory. This can be a good way to introduce graph theory to beginners.

2.2 Methodology

Hamiltonian Path/Circuit

We found a reliable and easy to understand algorithm for finding the Hamiltonian Path/Circuit. It works by backtracking. The steps of the algorithm are:

1. Put vertex 0 as the first vertex in the path. If there is a Hamiltonian Cycle, then the path can be started from any point of the cycle as the graph is not directed.
2. Move to the next vertex.

3. Check if this vertex is an adjacent vertex of the previously added vertex and is not included in the path earlier.
4. Try different vertices as a next candidate in Hamiltonian Cycle.
5. if all vertices are included in the path. Last vertex must be adjacent to the first vertex in path to make a cycle.
6. Remove current vertex if it doesn't lead to a solution and keep repeating step 4 and 3.

Eulerian Path/Circuit

The algorithm to find Eulerian Path/Circuit is:

1. Find out the degrees of all vertices. Add all the entries in a row of the Adjacency matrix to get the degrees of the vertex that row represents.
2. Check if all non-zero degree vertices are connected.
3. If all of them have an even degree, then we have an Eulerian Circuit.
4. If two of them have odd degrees and the rest have even degrees then we have a Eulerian Path.
5. Do an DFS traversal starting from node with non-zero degree. Create a recursive function that takes the index of the node and a visited array.
    - Mark the current node as visited and print the node.
    - Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

Using these Algorithms

We constructed a data-set of 32 constellations which consisted of the vertices in the specific constellations and the edges between the vertices. Making all these graphs into Adjacency Matrices we applied both the algorithms to them all. We also made a GUI in Python using tkinter for better user experience.

2.3 Results

Graph Properties Of the Constellations		
Constellations	Hamiltonian Circuit	Eulerian Circuit
Antlia	False	False
Apus	False	False
Aquila	False	False
Ara	False	False
Aries	False	False
Camelpardalis	False	False
Centaurus	False	False
Chameleon	True	True
Circinus	False	False
CoronaAus	False	False
CoronaBor	False	False
Corvus	False	False
Crater	False	False
Delphinus	False	False
Equuleus	False	False
Fornax	True	True
Horlogium	False	False
Indus	True	True
Lacerta	False	False
Leo Minor	False	False
Lynx	False	False
Mensa	False	False
Microscopium	True	True
Musca	False	False
Pavo	False	False
Sculptor	True	True
Sextans	True	True
Telescopium	True	True
UrsaMinor	False	False
Vela	True	True
Volans	False	False

Table 1: Hamiltonian and Eulerian Circuits in Constellations.

3 Conclusion

We arrived at the following conclusions:

- We can find the Hamiltonion Circuit of any graph.
- We can find the Eulerian path of any graph.
- We can use backtracking to find Hamiltonion Circuit.
- We can use breadth first search to find Eulerian Path.
- We can take the edges of the graph and finds its adjacency matrix and vice versa.
- We can find the patterns in the constellations.
- We can draw graphs from their adjacency matrix.

4 References

- https://journal-ems.com/index.php/emsj/article/view/135/129
- http://www.seasky.org/constellations/constellation-centaurus.html
- https://www.udacity.com/wiki/creating-network-graphs-with-python
- https://networkx.org/documentation/stable/tutorial.html
- https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
- https://www.youtube.com/watch?v=Ta1uhGEJFBE
- https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

5 Appendix

Hamiltonian Cycle

```
class Graph():
    def __init__(self, vertices):
        self.graph = [[0 for column in range(vertices)]
                          for row in range(vertices)]
        self.V = vertices

    def isSafe(self, v, pos, path):
        if self.graph[ path[pos-1] ][v] == 0:
            return False
        for vertex in path:
            if vertex == v:
                return False
        return True
    def hamCycleUtil(self, path, pos):
        if pos == self.V:
            if self.graph[ path[pos-1] ][ path[0] ] == 1:
                return True
            else:
                return False
        for v in range(1,self.V):
            if self.isSafe(v, pos, path) == True:
                path[pos] = v
                if self.hamCycleUtil(path, pos+1) == True:
                    return True
                path[pos] = -1
        return False
    def hamCycle(self):
        path = [-1] * self.V
        ''' Let us put vertex 0 as the first vertex
            in the path. If there is a Hamiltonian Cycle,
            then the path can be started from any point
            of the cycle as the graph is undirected '''
        path[0] = 0
        if self.hamCycleUtil(path,1) == False:
            print ("Solution does not exist\n")
            return False
        self.printSolution(path)
        return True
    def printSolution(self, path):
        print ("Solution Exists: Following",
                "is one Hamiltonian Cycle")
        for vertex in path:
            print (vertex, end = " ")
        print (path[0], "\n")
''' Let us create the following graph
    (0)--(1)--(2)
    | / \ |
    | / \ |
    | /  \ |
    (3)-------(4) '''
g1 = Graph(5)
g1.graph = [ [0, 1, 0, 1, 0], [1, 0, 1, 1, 1],
             [0, 1, 0, 0, 1,],[1, 1, 0, 0, 1],
             [0, 1, 1, 1, 0], ]
g1.hamCycle();
```

```
''' Let us create the following graph
    (0)--(1)--(2)
    | / \ |
    | / \ |
    | /  \ |
    (3)  (4) '''
g2 = Graph(5)
g2.graph = [ [0, 1, 0, 1, 0], [1, 0, 1, 1, 1],
        [0, 1, 0, 0, 1,], [1, 1, 0, 0, 0],
        [0, 1, 1, 0, 0], ]
g2.hamCycle();
```

Eulerian Path

```
def findpath(graph):
    n = len(graph)
    numofadj = list()
    for i in range(n):
        numofadj.append(sum(graph[i]))
    startpoint = 0
    numofodd = 0
    for i in range(n-1, -1, -1):
        if (numofadj[i] % 2 == 1):
            numofodd += 1
            startpoint = i
    if (numofodd > 2):
        print("No Solution")
        return
    stack = list()
    path = list()
    cur = startpoint
    while(stack != [] or sum(graph[cur]) != 0):
        if (sum(graph[cur]) == 0):
            path.append(cur + 1)
            cur = stack.pop(-1)
        else:
            for i in range(n):
                if graph[cur][i] == 1:
                    stack.append(cur)
                    graph[cur][i] = 0
                    graph[i][cur] = 0
                    cur = i
                    break
    for ele in path:
        print(ele, "-> ", end = '')
    print(cur + 1)
graph1 = [[0, 1, 0, 0, 1],
          [1, 0, 1, 1, 0],
          [0, 1, 0, 1, 0],
          [0, 1, 1, 0, 0],
          [1, 0, 0, 0, 0]]
findpath(graph1)
graph2 = [[0, 1, 0, 1, 1],
          [1, 0, 1, 0, 1],
          [0, 1, 0, 1, 1],
          [1, 1, 1, 0, 0],
          [1, 0, 1, 0, 0]]
findpath(graph2)
graph3 = [[0, 1, 0, 0, 1],
          [1, 0, 1, 1, 1],
```

```
47            [0, 1, 0, 1, 0],
48            [0, 1, 1, 0, 1],
49            [1, 1, 0, 1, 0]]
50  findpath(graph3)
```