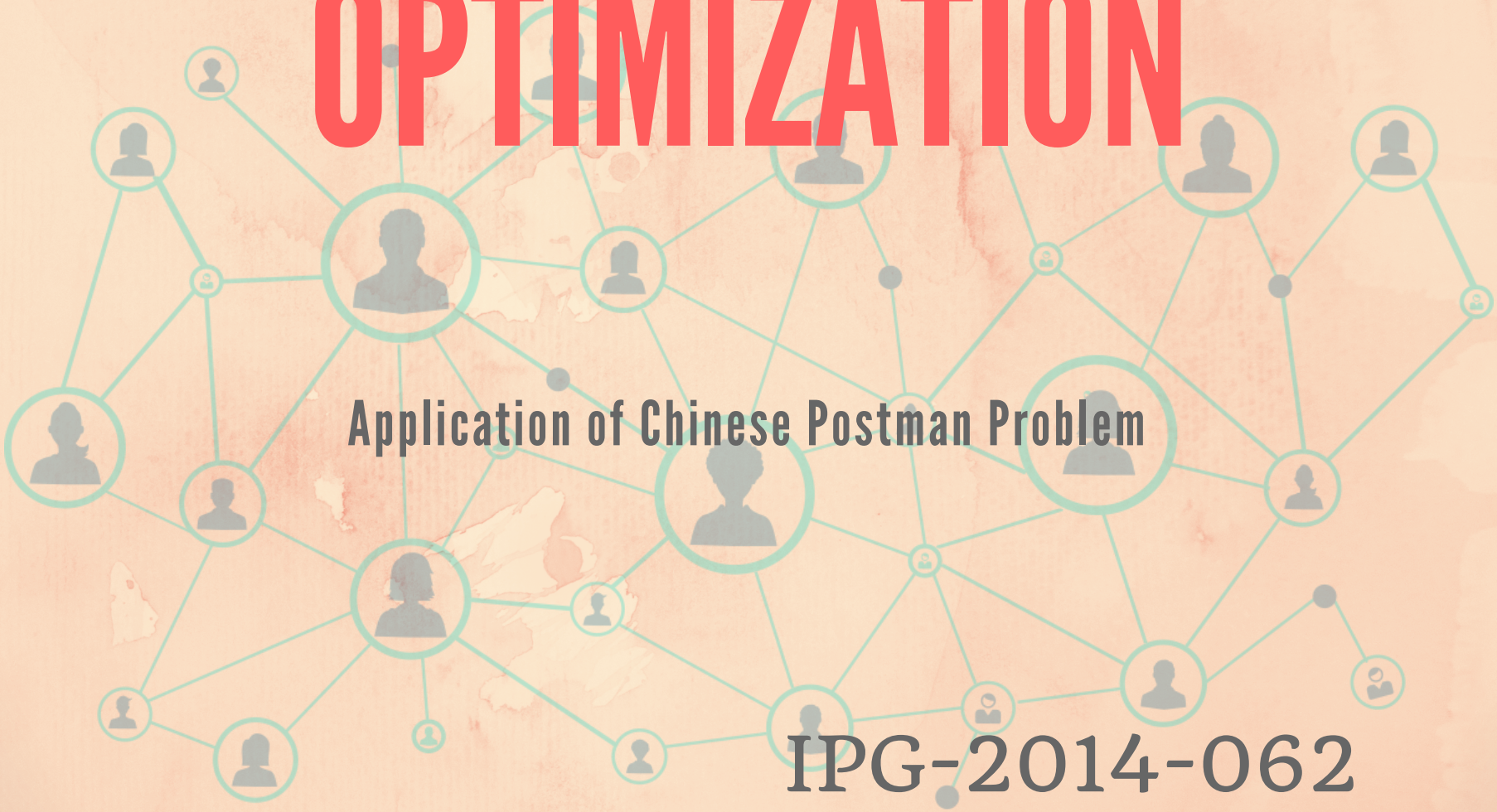
NETWORK GRAPH OPTIMIZATION



Application of Chinese Postman Problem

IPG-2014-062

NAME: PRAKHAR DEV GUPTA

SUBJECT: SPECIAL TOPICS IN NETWORK

MOTIVATING GRAPH OPTIMIZATION

The Problem

- The Chinese Postman Problem (CPP), also referred to as the Route Inspection or Arc Routing problem, is quite similar to well known Travelling Salesman Problem.
- The objective is to find the shortest path that covers all the links (roads) on a graph at least once.
- Possible without doubling back? Great! That's the ideal scenario and the problem is quite simple.

BUT IS THE IDEAL SITUATION ALWAYS POSSIBLE?

- It may not always be possible to traverse every route without double backing any.
- If some roads must be traversed more than once, you need some math to find the shortest route that hits every road at least once with the lowest total mileage.

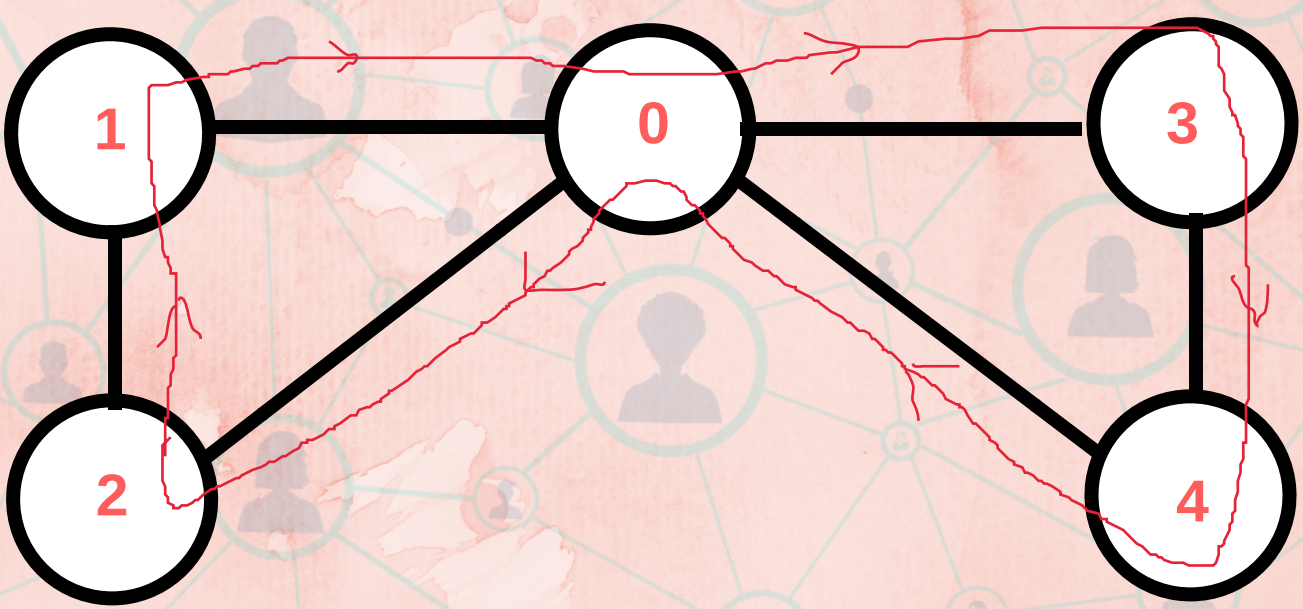
FORMAL PROBLEM STATEMENT

"Given an undirected labelled graph(weighted or unweighted), find the shortest closed path that visits every edge at least once where shortest refers to the the minimum total travelling cost."

Will make it more clear with a working example.

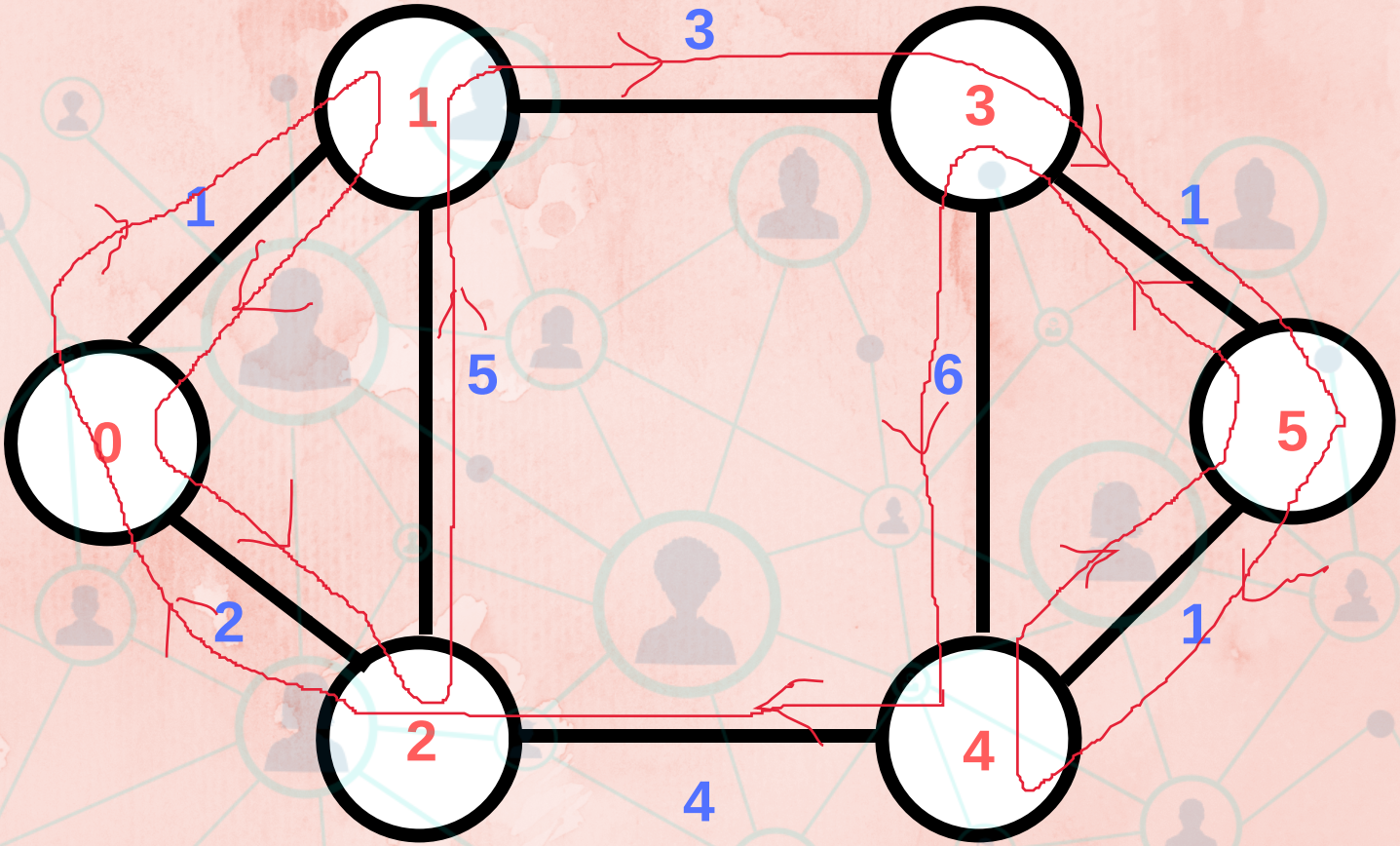
CASE 1: ALL NODES HAVE EVEN DEGREE

- If input graph contains Euler Circuit, then a solution of the problem is Euler Circuit



The graph has Eulerian cycle "2 1 0 3 4 0 2"

CASE 2: NON-EULER WEIGHTED GRAPH



"1 3 5 4 5 3 4 2 0 1 0 2 1"

ALGORITHM:

- If the graph is Eulerian, return the sum of all edge weights.Else do following steps:
- We find all the vertices with odd degree
- List all possible pairings of odd vertices. For n odd vertices total number of pairings possible are, (n-1) * (n-3) * (n -5)... * 1
- For each set of pairings, find the shortest path connecting them.
- Find the pairing with the minimum shortest connecting pairs.
- Modify the graph by adding all the edges that have been found in step 5.
- The weight of Chinese Postman Tour is the sum of all edges in the modified graph.
- Print Euler Circuit of the modified graph.

DETAILS OF IMPLEMENTATION

DATASET DESCRIPTION:

- Edge list for the network of *Sleeping Giant State Park* trail map. Contains **123 tuples.**
- Each row represents a single edge of the graph with some edge attributes:

- **node1 & node2:** names of the nodes connected.
- **trail:** edge attribute indicating the abbreviated name of the trail for each edge. For example: rs = red square
- **distance**: edge attribute indicating trail length in miles.
- **color**: trail color used for plotting.
- **estimate**: unused attribute.

edgelist_sleeping_giant.csv

Raw

Search this file...

	node1	node2	trail	distance	color	estimate
2	rs_end_north	v_rs	rs	0.3	red	0
3	v_rs	b_rs	rs	0.21	red	0
4	b_rs	g_rs	rs	0.11	red	0
5	g_rs	w_rs	rs	0.18	red	0
6	w_rs	o_rs	rs	0.21	red	0
7	o_rs	y_rs	rs	0.12	red	0
8	y_rs	rs_end_south	rs	0.39	red	0
9	rc_end_north	v_rc	rc	0.7	red	0
10	v_rc	b_rc	rc	0.04	red	0
11	b_rc	g_rc	rc	0.15	red	0
12	g_rc	o_rc	rc	0.13	red	0

Figure: Few tuples of the dataset table

Node List

- There are some node attributes that are added: X, Y coordinates of the nodes (trail intersections) tp plot the graph with the same layout as the trail map.

- **id:** name of the node corresponding to node1 and node2 in the edge list.
- **X:** horizontal position/coordinate of the node relative to the top left.
- **Y** vertical position/coordinate of the node relative to the topleft.

nodelist_sleeping_giant.csv

Raw

Search this file...

	id	X	Y
2	b_bv	1486	732
3	b_bw	716	1357
4	b_end_east	3164	1111
5	b_end_west	141	1938
6	b_g	1725	771
7	b_gy2	2000	983
8	b_o	2039	1012
9	b_rc	1311	979
10	b_rd	268	1744
11	b_rd_dupe	268	1744
12	b_rh	525	1380
13	b_rs	1747	700

Figure: Few tuples of the nodelist table

MAIN LIBRARIES USED:

- **Networkx**: Popular Python package with the strongest graph algorithms.
- **Pandas**: an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- **Matplotlib**: A plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits.
- **Imageio** is a Python library that provides an easy interface to read and write a wide range of image data, including animated images, volumetric data, and scientific formats. Used for creating GIFs.

VISUALIZATION OF THE GRAPH

- The positions are manipulated by the **X and Y coordinates** in the nodelist dataset.
- Edglist is used to manipulate the edge colors.

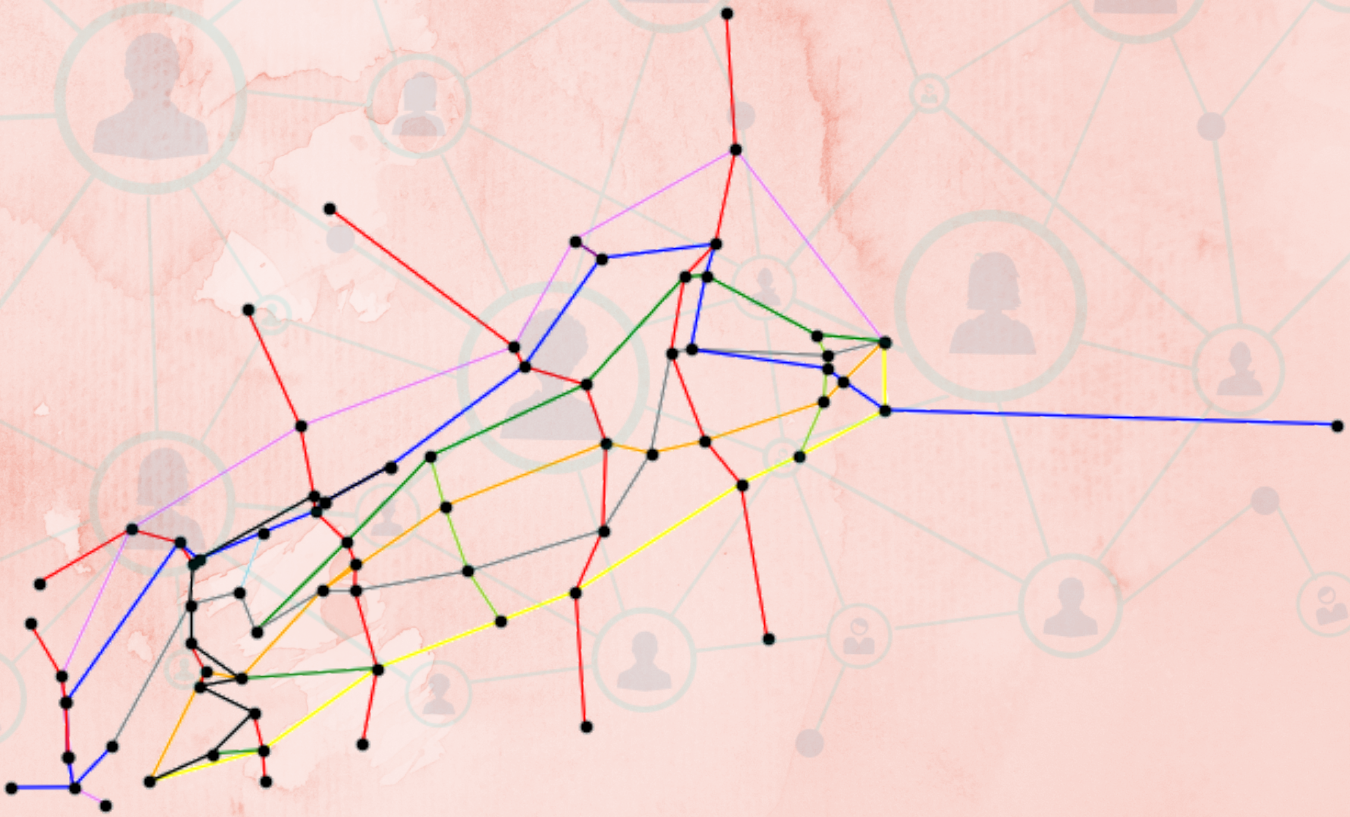


Figure:Graph representation of sleeping giant trail map

ASSUMPTIONS AND SIMPLIFICATIONS

ASSUMPTION 1: REQUIRED TRAILS ONLY:

- We ignore optional trails in this dataset and focus on required trails only.

ASSUMPTION 2: UPHILL == DOWNHILL:

- The cost of walking a trail is equivalent to its distance, regardless of which direction it is walked.

ASSUMPTION 3: NO PARALLEL EDGES:

- To avoid the computational complexity, the dataset is such that it does not have any parallel edges.

STEPWISE DESCRIPTION OF IMPLEMENTATION

STEP 1: COMPUTING ODD NODE PAIRS:

- Finding all possible pairs of odd degree nodes. For undirected graph, (a,b) == (b,a)
- Computed by the formula given below:

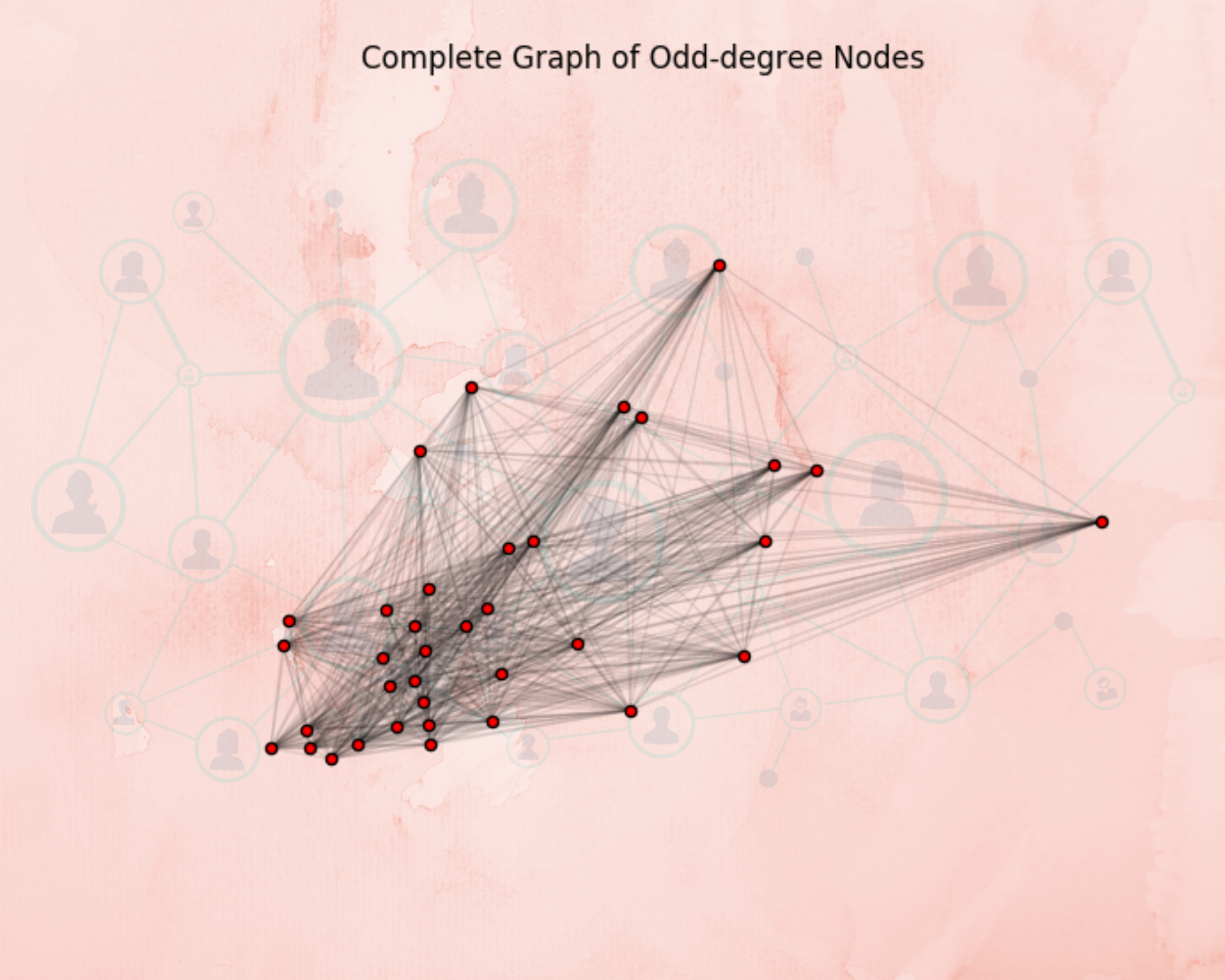
$$\#\ of\ pairs = n\ choose\ r = \binom{n}{r} = \frac{n!}{r!(n-r)!} = \frac{36!}{2!(36-2)!} = 630$$

STEP 2: SHORTEST PATH BETWEEN NODE PAIRS

- Networkx has a convenient implementation of Dijkstra's algorithm to compute the shortest path between two nodes.
- This function is applied to every pair (all 630) calculated above in **odd_node_pairs**

STEP 3: CREATE COMPLETE GRAPH

- A complete graph of all odd node is created and among them, we need to choose the **minimum matching pairs**.
- Since NetworkX supports the max_weight_matching, we associate a negative of edge distance as weight and then find the max_weight_matching on the **flipped** values.


Complete Graph of Odd-degree Nodes

- Since the graph is undirected, therefore the ordering of matching pairs does not matter. Hence each target pair is counted twice.

- Therefore the double counting need to be removed. The blue lines may not be the actual trail lines therefore we need to find them.

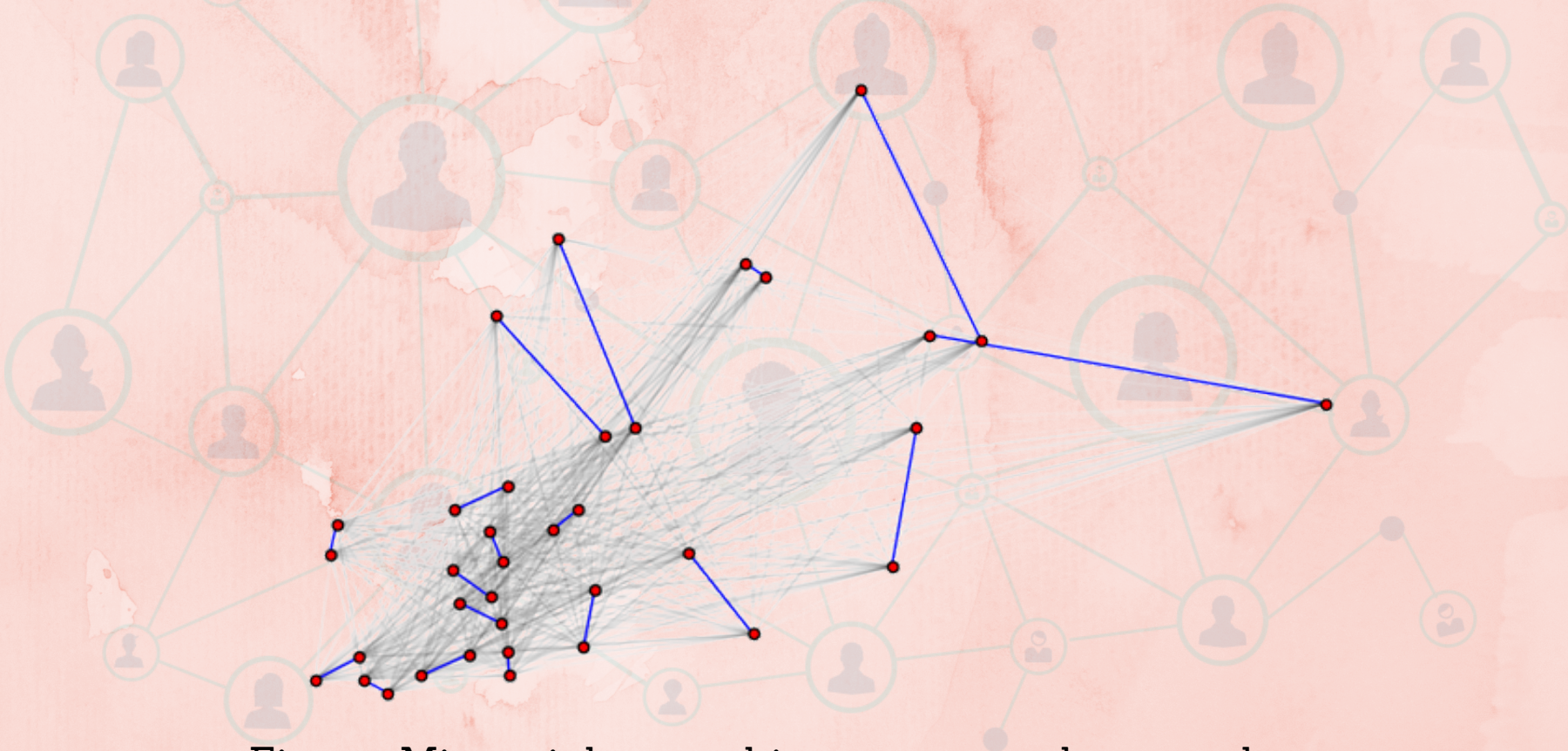


Figure: Min weight matching on a complete graph

STEP 4: AUGMENT THE ORIGINAL GRAPH:

- The original graph is augmented with the edges found in the minimum matching (step 3).
- This is utilised later for creating the Eulerian graph.
- The augmented graph adds 18 more edges in the original graph.
- Every node now has an even degree.

COMPUTING THE EULERIAN CIRCUIT

- As Euler famously postulated in 1736 with the Seven Bridges of Königsberg problem, there exists a path which visits each edge exactly once if all nodes have even degree.

- There are many Eulerian circuits with the same distance that can be constructed with the NetworkX eulerian_circuit function. However there are some limitations which need to be tackled.

LIMITATIONS WHICH ARE FIXED:

- The augmented graph contains edges that didn't exist on the original graph. To get the circuit (without bushwhacking), we break down these augmented edges into the shortest path through the edges that actually exist.

- eulerian_circuit only returns the order in which we hit each node. It does not return the attributes of the edges needed to complete the circuit. This is necessary because you need to keep track of which edges have been walked already when multiple edges exist between two nodes.

NAIVE CIRCUIT:

- Provides simple but incomplete solution.
- Expected length of naive circuit is given by the formula

$$Circuit\ length = length\ of\ original\ graph + number\ of\ augmented\ edges$$

$$= 123 + 18 = 141$$

CORRECT CIRCUIT:

- Loop through each edge in the naive Eulerian circuit (naive_euler_circuit).
- Wherever we encounter an edge that does not exist in the original graph, we replace it with the sequence of edges comprising the shortest path between its nodes using the original graph.
- The shortest path is calculated using Dijkstra's algorithm. This comes predefined in NetworkX.
- The length of the corrected circuit is longer than the naive circuit's which is as expected.

VISUALIZATION OF THE SOLUTION

CREATING A MOVIE:

- For each path traversed in the final graph, we create a PNG image.
- The path traversed for the first time is shown in **black** and the path traversed for the second time is shown in **red.**
- Finally, all the PNG images are stitched together to give it a GIF feel
- The image stitching is easily handled by imageio library of Python.

APPLICATIONS OF CHINESE POSTMAN PROBLEM

- Various **combinatorial problems** are reduced to the Chinese Postman Problem, including finding a maximum cut in a planar graph and a minimum-mean length circuit in an undirected graph.

- Finds applications in building paths for the **sightseeing** in a city.

- The architecture of the **shopping malls** so that the customer could travel all the lanes at the least cost.

- One obvious application in the mail and product **delivery system**.

EXPECTED UPCOMING VARIANTS

- CPP for **partially directed** graphs.
- **Time-constrained** CPP.
- CPP for graphs with **parallel edges with different costs**.
- CPP with **different** start and end points.
- CPP for **stochastic networks.**

THAT'S ALL!

Sources:
GeeksForGeeks,
Github,
TechCrunch,
LifeHacker,
TechiDelight

ANY QUESTIONS ARE WELCOMED