

Tour Optimization in ITB Campus of Jatinangor Using Implementation of Chinese Postman Problem

Josh Reinhart Zidik - 13524048

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jalan Ganesha 10 Bandung

E-mail: joshrl710@gmail.com , 13524048@std.stei.itb.ac.id

Abstract—Campus tour is a common event to take place during new student orientation, usually its participants are divided into several groups guided by few tour guides. However, most campus is not designed with touring in mind, some groups will have to take the exact same route twice and at the same time colliding with another group. One such example is the ITB campus of Jatinangor which contains a lot of intersections and dead-ends. This paper explores inside the application of graph theory, specifically the Chinese Postman Problem, to optimize campus tour routes and minimize redundancy. We aim to identify the most efficient paths for campus tour in order to improve the overall experience and reduce congestion.

Keywords—Campus Tour, Graph Theory, Chinese Postman Problem

I. INTRODUCTION

Campus tour is an essential part to introduce the environment of a campus to new people, whether during student orientation or campus promotional program. Commonly, a campus tour isn't done with every participants in a single group, but instead divided into a number of groups with a tour guide leading it.

The goal of campus tour is to introduce every part of the campus to new people or students. In order to reach this goal effectively, a tour group must take every single route only once. It is to prevent a group from taking the same route twice, colliding with another group during its tour, or missing a spot to visit. Even though a tour can conclude in many endpoints, it is preferable to end in the same place as the starting point. This lets tour guides to be able to efficiently prepare and lead the next group.

At Institut Teknologi Bandung (ITB), the student organization Kabinet Mahasiswa hosts an annual orientation event known as Orientasi Studi Keluarga Mahasiswa (OSKM). The event lasts for about less than a week and mainly consists of new college students orientation which includes a campus tour inside the ITB campus of Jatinangor. In this context, it is crucial that all tour groups start and end at the same location, as the campus tour is directly followed by another orientation activities.

Since it is unlikely for a tour group to take exactly all route once in ITB Jatinangor campus, then some groups must take several routes twice. To find the most efficient route each

group must take, this paper will implement the use of graph theory specifically on the Chinese Postman Problem (CPP).

II. THEORETICAL BASIS

A. Graph Definition

A graph can be defined as a mathematical representation of a set of points (vertices) and the connections (edges) between them. A graph must have vertices, meaning that an empty set of points can't be defined as a graph. However, a graph may have no edges, meaning that an empty set of edges is allowed inside a graph [1].

B. Graph Terminology

1) Adjacency

Two vertices are adjacent to each other if both are connected with the same edge.

2) Incidency

A vertex is incident to an edge if the edge is connected to the vertex.

3) Degrees

Degree of a vertex is the number of edges which is connected to the vertex.

4) Isolated Vertex

An isolated vertex is a vertex that doesn't have any edges connected to it. An isolated vertex can also be defined as the vertex with 0 degree.

5) Path

A path is an alternating sequence of vertices and edges which starts from a vertex to another vertex.

6) Cycle or Circuit

Cycle is a path that starts and ends in the same vertex. As an example, on figure 1 a path that consists of vertices $3 - 2 - 1 - 0 - 6 - 3$ is a circuit because it starts and ends in vertex 3. [2]

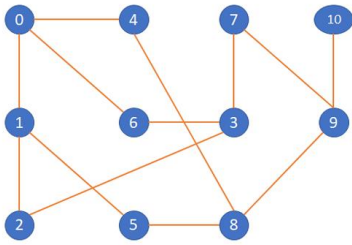


Fig. 1. An unweighted and undirected graph

Source: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf>, accessed on: 20 Jun. 2025

7) Weighted Graph

A weighted graph is a graph which has weight in every edges.

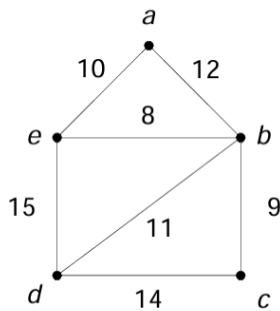


Fig. 2. An example of weighted graph

Source: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf>, accessed on: 20 Jun. 2025

C. Euler Circuit and Eulerian Graph

Euler circuit is a circuit of a graph that passes through every edges inside said graph. While Eulerian graph is a graph that contains a Euler circuit. The theorem was first introduced when Euler solved the problem of the Seven Bridges of Konigsberg. The problem was to find a path through the city that will cross every bridge at once. [3]

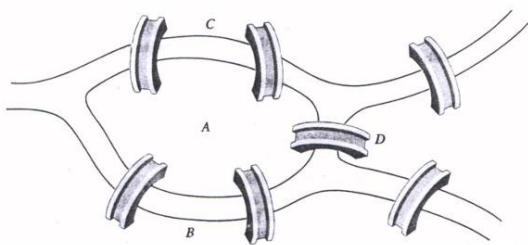


Fig. 3. The Seven Bridges of Konigsberg problem

Source: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/22-Graf-Bagian3-2024.pdf>, accessed on: 20 Jun. 2025

A graph must exactly only has even degrees in order to be Eulerian. This theorem comes from the fact that for every vertices, there should be n connected edges that goes inward and n connected edges that goes outward. If there exists at least

a pair of vertices that has odd degrees, then the graph will not have a Euler circuit.

D. Chinese Postman Problem

Chinese Postman Problem (CPP) is a subtopic of graph theory found by a mathematician called Kuan Mei-Ko. The premise of the problem involves finding the minimum distance needed from a circuit to travel every edges of the graph at least once. [4]

The algorithm to solve CPP involves these steps:

- Calculate the degree from every vertices, if there is no vertex with odd degree then the graph is Eulerian, return the sum of every weight inside the graph. However if there is one or more pairs of odd degree vertices, do the following steps.
- Search for every vertices with odd amount of degrees.
- List every possible pair of odd vertices.
- From the list, select several pairs which doesn't contain the same vertices inside to create a set of pairs.
- Calculate the shortest distance between every pair inside the set chosen earlier.
- Repeat step number 4 and 5 until the shortest distance possible from a set of pair is determined.
- Modify the graph by adding edges based on the set of pair determined in previous step.

With these steps, an Eulerian graph can be crafted out of a non-Eulerian graph. Thus, creating a graph that contains a Euler circuit which the sum of all edges is the solution of CPP [4].

III. ALGORITHM

This section discusses the implementation of the CPP algorithm using Dijkstra's algorithm to find the shortest path between two vertices. However, since Dijkstra's algorithm is not the primary focus of this paper, its implementation will not be included in this section. The CPP algorithm presented in this section is adapted from reference [5] with several modifications to fit our study case.

A. Initialize Graph

The first step of the algorithm is to create an undirected weighted graph using adjacency list. The row and column of the list represents a pair of vertices that is connected by an edge, while the value inside the list is the weight of the edge.



```

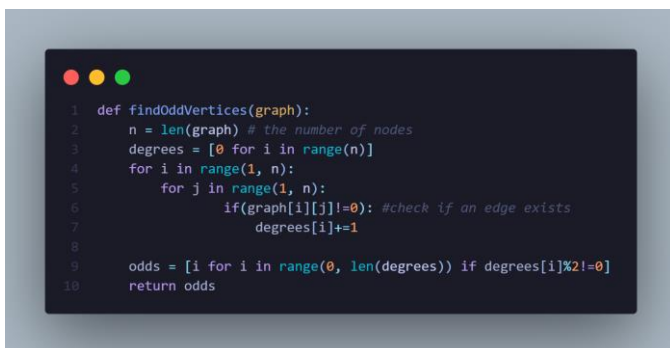
1 def makeGraph():
2     nodes = int(input()) # N nodes
3     edges = int(input()) # M edges
4     graph = [[0 for j in range(edges+1)] for i in range(nodes + 1)] # initialize graph
5
6     # Input edges
7     for _ in range(edges):
8         u, v, w = map(int, input().split())
9         graph[u][v] = w
10        graph[v][u] = w # undirected graph
11    return graph

```

Fig. 4. A python function to create a graph based on adjacency list

B. Search For All Odd Vertices

Next step, the algorithm checks if there exists an odd vertices inside the graph. If there is no odd vertices, then the graph is Eulerian and the program will return the sum of all weights. Otherwise, the odd vertices will be appended inside the array *odds* which will be used for CPP algorithm.



```

1 def findOddVertices(graph):
2     n = len(graph) # the number of nodes
3     degrees = [0 for i in range(n)]
4     for i in range(1, n):
5         for j in range(1, n):
6             if graph[i][j] != 0: #check if an edge exists
7                 degrees[i] += 1
8
9     odds = [i for i in range(0, len(degrees)) if degrees[i] % 2 != 0]
10    return odds

```

Fig. 5. A python function to find all odd vertices from a graph



```

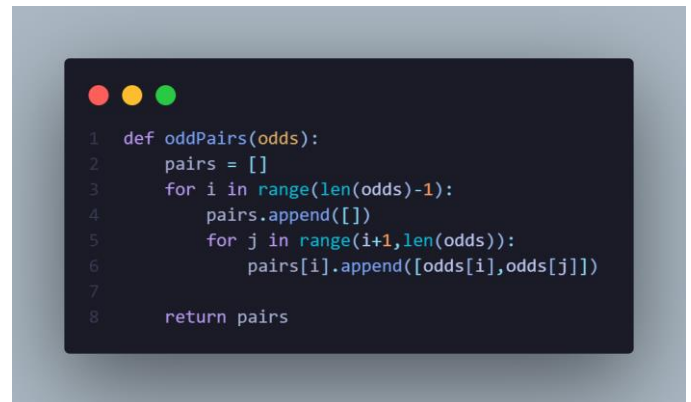
1 def chinesePostman(graph):
2     odds = findOddVertices(graph)
3     if len(odds) == 0:
4         return sumEdgesWeight(graph)

```

Fig. 6. An implementation of CPP if the graph is Eulerian

C. Generate All Odd Vertices Pair Combinations

All vertices with odd degrees are paired to find out the combination with the shortest sum of distance. Therefore, the program generates all possible pairing combinations using a nested loop.

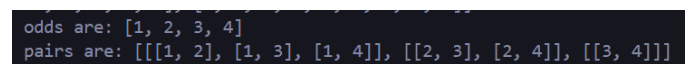


```

1 def oddPairs(odds):
2     pairs = []
3     for i in range(len(odds)-1):
4         pairs.append([])
5         for j in range(i+1, len(odds)):
6             pairs[i].append([odds[i], odds[j]])
7
8     return pairs

```

Fig. 7. A python function to generate every possible pairings of odd vertices



```

odds are: [1, 2, 3, 4]
pairs are: [[1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]]

```

Fig. 8. The example result of the function above

D. Search For The Pair Combination With Minimum Weight

After generating all possible pairing combinations, the program selects $n / 2$ pairs where n is the number of vertices with odd degree. Using Dijkstra's algorithm it will determine the shortest distance between each pair. This process will be iterated through all valid combinations, where a combination is valid if no vertex appears in more than one pair. As an example, the pairs (1-2) and (3-4) are valid, but pairs (1-3) and (3-4) are not because vertex 3 appears in both. After the iteration process is done, the selection with the minimum weight or shortest distance will be picked as the solution to CPP.

```

1 def chinesePostman(graph):
2     odds = findOddVertices(graph)
3     if(len(odds)==0):
4         return sumEdgesWeight(graph)
5     pairs = oddPairs(odds)
6     l = (len(pairs)+1)//2
7
8     selectedPairs = []
9
10    def getPairs(pairs, done = [], final = []):
11        if(pairs[0][0][0] not in done):
12            done.append(pairs[0][0][0])
13
14            for i in pairs[0]:
15                f = final[:]
16                val = done[:]
17                if(i[1] not in val):
18                    f.append(i)
19                else:
20                    continue
21
22            if(len(f)==1):
23                selectedPairs.append(f)
24                return
25            else:
26                val.append(i[1])
27                getPairs(pairs[1:],val, f)
28
29    else:
30        getPairs(pairs[1:], done, final)
31    getPairs(pairs)
32
33    return minAdditional(graph, selectedPairs)

```

Fig. 9. The algorithm to solve CPP by creating several sets of vertices pairs

```

1 def minAdditional(graph, selectedPairs):
2     # The selected pairs with minimum additional weight
3     minPairs = selectedPairs[0]
4     minWeight = 0
5
6     for i in selectedPairs:
7         s = 0
8         for j in range(len(i)):
9             s += dijkstra(graph, i[j][0], i[j][1])
10        if i == 0:
11            minWeight = s
12        elif s < minWeight:
13            minWeight = s
14            minPairs = selectedPairs[i]
15
16    print(minPairs)
17    addedDistance = minWeight
18    solution = addedDistance + sumEdgesWeight(graph)
19    return solution

```

Fig. 10. A python function to determine the set of pairs with shortest distance

The function minAdditional is added so the program will look for the set of pairs with shortest distance. The chosen set of pairs later will be used as the solve the CPP.

The line print(minPairs) in the code will print out the pair of vertices with minimum additional weight. This output will be our guide to add new edges inside the graph to make it Eulerian, hence solving the CPP.

IV. IMPLEMENTATION

A. Map Simplification into Weighted Graphs

First, the map of ITB campus of Jatinangor must be simplified into a weighted graph in order to correctly choose the path.



Fig. 11. Map of the ITB Jatinangor campus

Source: <https://x.com/oskmitb/status/1687787144201596928>, accessed on: 18 Jun. 2025

The layout of the ITB Jatinangor campus can be categorized into two separate networks: transportation routes and pedestrian pathways primarily used by students.

We begin by simplifying the transportation routes, representing each road as an edge, each intersection as a vertex, and each distance as the weight in the graph. This simplification focuses only on routes that pass through the campus's main facilities. Therefore, smaller routes leading to parking areas such as building 22 and 24 are excluded. Using yFiles Graph Editor, we constructed an undirected weighted graph consisting of 11 vertices and 14 edges.

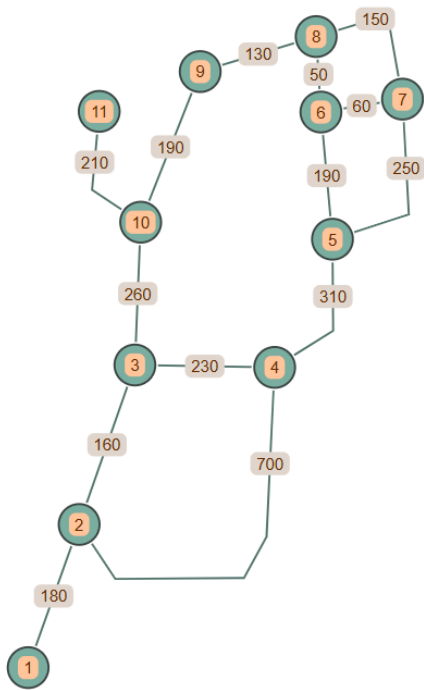


Fig. 12. The transportation route of ITB Jatinangor campus simplified into a graph

We also simplify the pedestrian pathways for the main campus tour with similar technique for transportation route. Using yFiles Graph Editor, we constructed an undirected weighted graph consisting of 11 vertices and 17 edges.



Fig. 13. Layout of the inner ITB Jatinangor campus

Fig. 14. Layout of the inner ITB Jatinangor campus' main route highlighted
Source: <https://x.com/oskmitb/status/1687787144201596928>, accessed on: 18 Jun. 2025

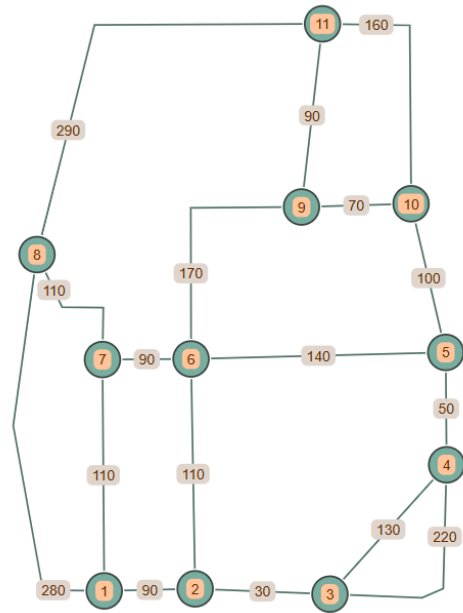


Fig. 15. Inner layout of ITB Jatinangor campus simplified into a graph

B. Program Implementation

First, the graph's data is converted to a table in order to fit the program's format. The column consists of a pair of vertices connected by an edge and the weight value of the edge. Table I shows every edges inside the graph of transportation route, while Table II shows every edges inside the graph of pedestrian pathway route.

TABLE I. TRANSPORTATION ROUTE EDGES

	Node 1	Node 2	Weight
1	1	2	180
2	2	3	160
3	2	4	700
4	3	4	230
5	3	10	260
6	4	5	310
7	5	6	190
8	5	7	250
9	6	7	60
10	6	8	50
11	7	8	150
12	8	9	130
13	9	10	190
14	10	11	260

TABLE II. PEDESTRIAN PATHWAYS EDGES

	Node 1	Node 2	Weight
1	1	2	90
2	1	7	110
3	1	8	280
4	2	3	30
5	2	6	110
6	3	4	130
7	3	4	220
8	4	5	50
9	5	6	140
10	5	10	100
11	6	7	90
12	6	9	170
13	7	8	110
14	8	11	290
15	9	10	70
16	9	11	90
17	10	11	160

Next, the data for both the first and second graph is inputted inside the program made earlier. The result of the program is a list of vertices pairs which is needed to be traversed twice to solve the CPP. Using said list, we can add new edges into the graph to make it Eulerian with the minimum additional weight.

```
The selected pairs of vertices are: [[1, 2], [3, 4], [5, 6], [7, 8], [10, 11]]
The minimum distance is: 3070
```

Fig. 16. The result for the transportation route graph showing a set of vertices pairs needed to solve the CPP

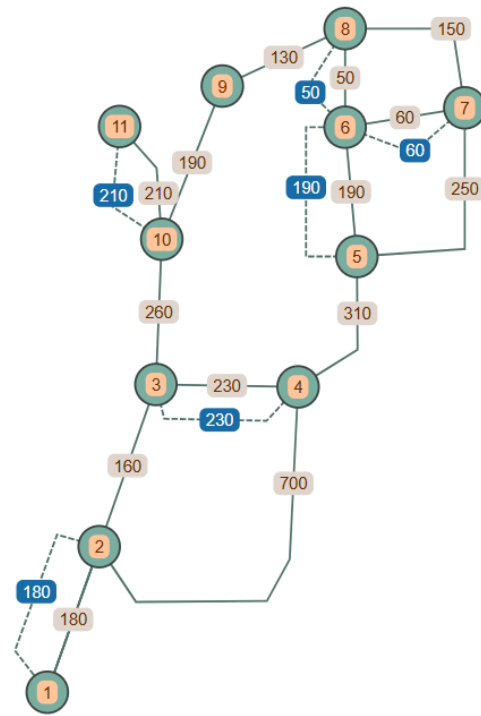


Fig. 17. Modified transportation route graph based on the result from previous figure

There are six new edges to the graph based on the solution offered by the program. Edges with blue colored weight are the new edges as the solution to the CPP. Since the program only outputs pairs of vertices, the actual path for each new edge must be determined manually. As an example, one of the pairs is the pair of vertices 7 and 8, even though there exists a direct edge connecting them, the chosen edges are the ones passing through vertex 6 because of the lower total distance.

Even though this transportation route is not part of the main tour, it can also help for drivers who hope to make a tour inside ITB Jatinangor campus using transportation. The modified graph can help drivers efficiently save energy and tour every possible facilities.

```
The selected pairs of vertices are: [[1, 2], [5, 7], [8, 9], [10, 11]]
The minimum distance is: 2110
```

Fig. 18. The result for the pedestrian pathways graph showing a set of vertices pairs needed to solve the CPP

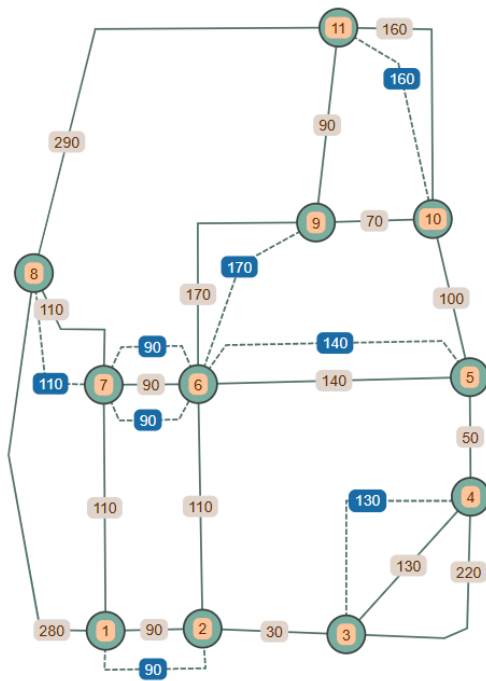


Fig. 19. Modified graph of pedestrian pathways based on the result of previous figure

By connecting the pair of vertices with new edges, we have successfully added eight new edges into the graph. Edges with blue colored weight are the new edges as the solution to the CPP. The edge between vertices 6 and 7 is added twice because it is part of the route between vertices pair 5-7 and 8-9.

Every edges now have even number of degree, which makes the graph Eulerian. Since the graph is Eulerian, a group can walk through every edges and return at the same point. With this, a tour can be done efficiently with the same starting and ending point.

V. CONCLUSION

This study implements the use of Eulerian graph to make a tour route that can visit every single facility in a campus. By solving the Chinese Postman Problem on the graph, an Euler circuit can be crafted inside the graph using the minimum amount of weight. However, since Chinese Postman Problem only determines the minimum weight needed, we modified the algorithm to instead outputs several pairs of vertices. The pairs of vertices are later used to determine new edges to solve Chinese Postman Problem. The result of this modified algorithm is six new edges on the first graph, which represents transportation route around ITB Jatinangor campus and also eight new edges on the second graph, which represents pedestrian pathway inside the ITB Jatinangor campus.

Using this modified graph, we hope that future tours can use it as a guide to tour efficiently through the entire campus. We also hope that the modified algorithms crafted for this case can also be used for other cases, especially one that involves the crafting of an Eulerian graph.

VI. APPENDIX

Github Repository

<https://github.com/Achideon/MakalahMatdisCPP>

VII. ACKNOWLEDGMENT

The author of this paper would like to express his gratitude to God for His blessings that help the writing of this paper. The author would also like to express his gratitude to his parents and family who have been supportive during author's study in the college and also during the writing of this paper. Additionally, the author would like to thank the lecturer Arrival Dwi Sentosa, S.Kom., M.T. for all the subject he has taught the author during the second semester. Last but not least, the author would like to express his deepest gratitude to his friends who's always around him during the brightest or even the darkest of times.

VIII. REFERENCES

- [1] Wilson, Robin J. "Introduction to Graph Theory", 4th edition
- [2] <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf>, [Accessed: 20 Jun.2025]
- [3] S. Manoneet Mahesh, "Euler Graphs and Euler Circuits" available online: <https://medium.com/@manoneetsikhwal2k1/euler-graphs-and-euler-circuits-a54e1f9d2140>, [Accessed: 20 Jun. 2025].
- [4] C. Gautam, "The Chinese Postman Problem" available online: <https://medium.com/@gautamsir076/the-chinese-postman-problem-2760799a2b4a>, [Accessed: 20 Jun. 2025].
- [5] S. Araz, "Chinese Postman in Python" available online: <https://towardsdatascience.com/chinese-postman-in-python-8b1187a3e5a>, [Accessed: 19 Jun. 2025].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Juni 2025

Josh Reinhart Zidik