Graph Theory Application in Modeling Braess's Paradox for Urban Traffic Optimization

Kevin Wirya Valerian - 13524019
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
kevin.wirya.valerian@gmail.com *13524019@std.stei.itb.ac.id*

***Abstract*—In this modern era, urban traffic congestion remains a challenge in rapidly developing cities. Conventional approaches such as constructing new roads often overlook the counterintuitive phenomena described by Braess's Paradox, where adding new infrastructure can worsen overall traffic conditions. This paper investigates the application of Braess's Paradox within the context of urban traffic network. By modeling selected traffic scenarios through simplified graph-based networks, we will be able to analyze the effects of proposed urban road modifications on traffic flow and equilibrium. Using discrete mathematical modeling, identifying conditions under which network enhancements may degrade traffic efficiency is prominent. The study contributes a novel perspective on urban traffic strategy, offering insights into sustainable urban mobility solutions.**

***Keywords*—Braess's Paradox, urban traffic, graph theory, network optimization**

I. Introduction

Urban transportation systems are complex networks where individual decisions can collectively produce unexpected and suboptimal outcomes for the entire system. In densely populated and rapidly growing cities, traffic congestion has escalated into a critical concern. This situation can be further exacerbated by the city's geographical constraints, a high number of private vehicles, and a flourishing population, all putting immense strain on the existing road infrastructure.

The escalating congestion in cities lead the author to incorporate mathematical and informatics approach in solving this issue. After further research and literature reviews, I have decided to take Braess's Paradox as main focus. Moreover, since I am an undergraduate in Informatics Engineering who takes Discrete Mathematics course in this semester, I will try to implement my knowledge regarding graph theory in modeling this problem so that it can provide clear visualization and thorough explanation.

Braess's Paradox exemplifies how the rational behavior of individual drivers, aiming to minimize their own travel time, can lead to a state of User Equilibrium (UE) that is less efficient than the system's potential optimum. This paradox is deeply rooted in graph theory, where roads are modeled as edges and intersections as nodes. Through the lens of graph analysis, we can identify structural properties of road networks that influence traffic equilibrium and efficiency.

This study applies graph-theoretical methods to urban traffic network to explore the implications of Braess's Paradox. By modeling real-world traffic scenarios in some circumstances and modifying them, I aim to uncover whether certain proposed changes might alleviate overall traffic performance. The goal is to provide insights that inform urban transportation planning, ensuring that infrastructure investments are guided by systematic understanding rather than intuition alone.

II. Theoretical Framework

A. Graph Theory Fundamentals

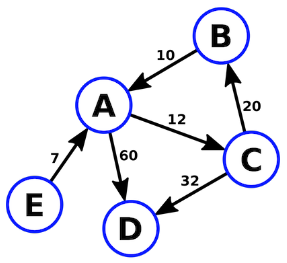


Fig. 1. Directed and Weighted Graph
(Source: https://study.com/academy/lesson/weighted-graphs-implementation-dijkstra-algorithm.html)

Graph theory is a branch of mathematics related to networks of points connected by lines. A graph can be defined as $G(V, E)$. V and E stand for the set of vertices and edges respectively. A graph cannot contain no vertices, but it might contain no edges. In this paper, vertices represent intersections while edges represent roads. Graphs can be classified in many ways. Two of them are directed graph and weighted graph.

- A directed graph is a graph which edges are direction-oriented, usually its edges are represented as arrows.
- A weighted graph is a graph which edges are assigned numerical values.

A graph might contain either paths or cycles. A path in a graph is a sequence of distinct vertices such that each consecutive pair of vertices in the sequence is connected by an edge. More formally, a path is a sequence of vertices $v_0, v_1, \ldots, v_k$ such that for every i from 0 to $k-1$, the edge (v_i, v_{i+1}) is in E. In a directed graph, edges have a direction. A path must follow the direction of the arrows. If you have an edge from A to B ($A \to B$), you have to traverse it in that direction.

A cycle (or circuit) in a graph is a path that starts and ends at the same vertex and does not repeat any other vertices or edges. More formally, a cycle is a sequence of vertices $v_0, v_1, \ldots, v_k$ such that these vertices make a path plus the edge (v_k, v_0) is also in E. The cycle of a graph must follow the same rules as the aforementioned rules for a path.

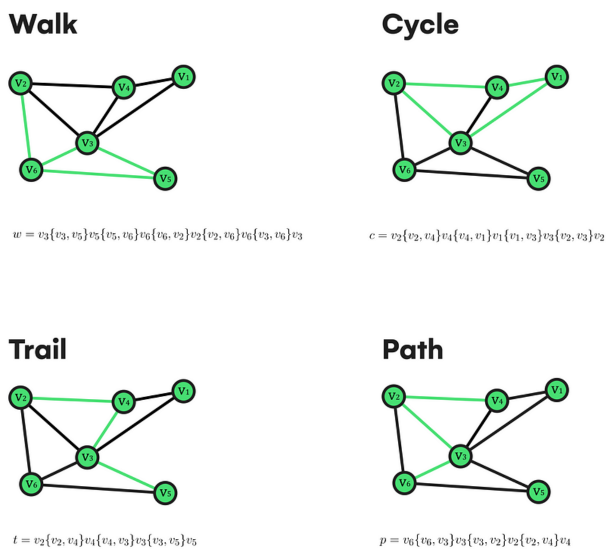

Walk
$w = v_3\{v_3, v_5\}v_5\{v_5, v_6\}v_6\{v_6, v_2\}v_2\{v_2, v_6\}v_6\{v_3, v_6\}v_3$
Cycle
$c = v_2\{v_2, v_4\}v_4\{v_4, v_1\}v_1\{v_1, v_3\}v_3\{v_2, v_3\}v_2$
Trail
$t = v_2\{v_2, v_4\}v_4\{v_4, v_3\}v_3\{v_3, v_5\}v_5$
Path
$p = v_6\{v_6, v_3\}v_3\{v_3, v_2\}v_2\{v_2, v_4\}v_4$

Fig. 2. Walk, Trail, Cycle and Path
(Source: https://www.freecodecamp.org/news/the-value-of-graph-theory-within-sustainability/)

Graph can be represented in many ways. One of them is by using adjacency matrix. In weighted and directed graph, for a directed graph $G = (V, E)$ with n vertices and m edges, where each edge (u, v) has an associated weight $w(u, v)$, we can represent the graph using a weighted adjacency matrix A of size $n \times n$. Each entry A_{ij} represents the weight of the edge from vertex i to vertex j.

There are some rules for this representation

- **Matrix Dimensions:** If there are n vertices, the matrix will be of size $n \times n$. Each row and column corresponds to a vertex.
- **Matrix Entries (A_{ij}):**
  - If there is a directed edge from vertex i to vertex j: then $A_{ij} = w(i, j)$, the weight of that edge.
  - If there is no edge from vertex i to vertex j then A_{ij} is typically represented as ∞ (infinity), indicating the absence of a direct connection.
  - If there is a loop, then A_{ii} is the weight of that loop.

Here is an example to help you have a clear understanding about the weighted adjacency matrix representation. Suppose there are 4 vertices: A, B, C, D. The weighted edges are

- A $\rightarrow$ B = 4
- A $\rightarrow$ D = 2
- A $\rightarrow$ E = 6
- B $\rightarrow$ C = 2
- C $\rightarrow$ A = 5
- C $\rightarrow$ D = 0
- D $\rightarrow$ C = 1
- E $\rightarrow$ B = 3

Then the weighted adjacency matrix A is:

TABLE I
Weighted Adjacency Matrix for a Directed Graph with Vertices A, B, C, and D

	A	B	C	D	E
A	0	4	∞	2	6
B	∞	0	2	∞	∞
C	5	∞	0	0	∞
D	∞	∞	1	0	∞
E	∞	3	∞	∞	0

B. *Traffic Assignment Theory (Wardrop's Principles)*

Traffic Assignment Theory is a crucial theory in transportation engineering that models the traffic distribution across a road network, given a set of travel demands. It seeks to predict the equilibrium state that arises from individual driver choices. The foundational concepts were developed by J.G. Wardrop. There are two main Wardrop's Principles.

1) Wardrop's First Principle (Wardrop or User Equilibrium)
This principle describes a state where every driver, acting individually and rationally, chooses the route that minimizes their own travel time. At equilibrium, no single driver can unilaterally switch to a different route and achieve a shorter travel time. This means all routes actually used between an origin and a destination will have equal and minimum travel times, while any unused routes will have travel times equal to or greater than those used.
2) Wardrop's Second Principle (Optimal Flow)
This principle describes an ideal state where traffic is assigned to routes in a way that minimizes the total travel time for all vehicles in the entire network. This represents a socially optimal outcome, prioritizing the collective efficiency over individual travel time minimization.

Next, another important term related to Wardrop's Principle is The Price of Anarchy (PoA). It was first defined by Koutsoupias and Papadimitriou. PoA quantifies the inefficiency of User Equilibrium compared to System Optimal network. It is defined as the ratio of the total travel time in a User Equilibrium state to the total travel time in a System Optimal state. If a selfish routing network is given by a triple of the form (G, r, c) where G is a multi commodity flow network, r is a vector of traffic rates and c is a vector of cost functions, indexed by the edges of G. A flow f feasible for (G, r, c) is a Wardrop Equilibrium if and only if

$$\sum_{e \in \mathcal{E}} c_e(f_e) f_e \leq \sum_{e \in \mathcal{E}} c_e(f_e) f_e^* \tag{1}$$

Before jumping to PoA itself, let us define the objective function from min-cost network flow and define the cost $C(f)$ of a flow f in (G, r, c) as

$$C(f) = \sum_{P \in \mathcal{P}} c_P(f) f_p = \sum_{e \in \mathcal{E}} c_e(f_e) f_e \tag{2}$$

for every flow f feasible for (G, r, c). The overall cost $c_P(f)$ here incurred by the traffic on the path P in the flow f is defined as the sum of the costs of the edges.

FInally, the PoA can be defined as the ratio of the cost of a user equilibrium and of a system optimal flow.

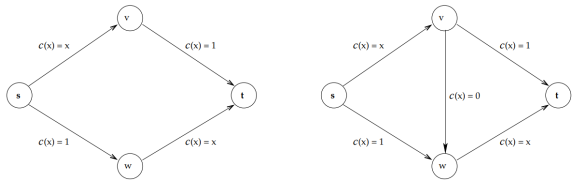
$$P(G, r, c) = \frac{C(f)}{C(f^*)} \tag{3}$$

where f is a Wardrop or User Equilibrium and f^* is the system optimal for (G, r, c). A PoA greater than 1 signifies that selfish routing leads to a worse outcome than cooperative routing.

C. Braess's Paradox

There exists a four-node network shown in Figure 3. There are two vertices s and t, each with combined cost $1 + x$, where x is the amount of traffic that uses the route. The routes are identical. If all drivers are selfish, they should split evenly between them. Assuming that there is one unit of traffic, all network users experience $3/2$ units of cost in the selfish routing outcome. In this case, cost is more or less their travel time.

Now suppose that, edge $v \to w$ exists with a constant cost function $c(x) = 0$. We therefore expect all network users to go to the new route. Nonetheless, the heavy congestion on the edges (s, v) and (w, t), as the consequence of a new edge built make all of the traffic now experiences two units of cost. Braess's Paradox thus shows that the seemingly helpful action of adding a new zero-cost edge can increase the cost experienced by all of the traffic!



(a) Initial network (b) Augmented network

Fig. 3. Braess's Paradox: Adding an edge $\neq$ best solution! (Source: https://timroughgarden.org/papers/optima.pdf)

D. Shortest Path Algorithm - Djikstra's Algorithm

There are a plenty of shortest-path finding algorithms out there. However, Djikstra's algorithm suits this work the best since it has been widespreadly used in analyzing weighed graphs nowadays. Dijkstra's Algorithm first was developed by Edsger W. Dijkstra in 1956. It stands as a cornerstone in graph theory for solving the shortest path problem. In the context of urban traffic optimization, it serves as a critical computational tool for simulating individual route choices within a network. Here is the pseudo-code for Djikstra's Algorithm.

Algorithm 1 Dijkstra's Algorithm

```
Require: Graph $G = (V, E)$, source node $s$
Ensure: Shortest distances from $s$ to all other nodes
 1: for each vertex $v \in V$ do
 2:     $dist[v] \leftarrow \infty$
 3:     $prev[v] \leftarrow$ undefined
 4: end for
 5: $dist[s] \leftarrow 0$
 6: $Q \leftarrow V$ {Initialize priority queue (or min-heap)}
 7: while $Q$ is not empty do
 8:     $u \leftarrow$ vertex in $Q$ with smallest $dist[u]$
 9:     Remove $u$ from $Q$
10:     for each neighbor $v$ of $u$ do
11:         $alt \leftarrow dist[u] + weight(u, v)$
12:         if $alt < dist[v]$ then
13:             $dist[v] \leftarrow alt$
14:             $prev[v] \leftarrow u$
15:         end if
16:     end for
17: end while
18: return $dist$, $prev$
```

III. Methodology

A. Research Model

This work models urban transportation systems as directed and weighted graphs. In this representation, each intersection within the network is treated as a vertex. The connections between these intersections, representing road segments, will be illustrated as directed edges. Each edge will be assigned with a cost function, which defines the travel time required to traverse that specific road segment. Crucially, these travel times are dynamic, increasing as the traffic volume (demand) on the link due to congestion. This cost function can be perceived as the dynamic weight of an edge. Next, the entire network, including its connectivity and current travel times, is formally represented through a weighted adjacency matrix, where entries A_{ij} denote whether there exists an edge from a vertex to another. If no direct link exists, the entry is set to -1 (for ease, as ∞ is hard to be represented), indicating the absence of a connection. The primary goal of this model is to simulate traffic flow patterns under different conditions, particularly focusing on how individual routing decisions impact the overall system.

B. Case-Based Network Construction

Due to the restricted official urban traffic datasets, the study employs representative case graphs inspired by urban road configurations and well-known paradox scenarios. These graphs, described textually and simulated computationally, serve as projections of real-world networks to investigate the emergence of optimal routing.

To simulate the manifestation of Braess's Paradox, the author provides three hypothetical traffic network examples which were constructed based on realistic urban configurations. These graphs serve as synthetic datasets used to analyze equilibrium, system-optimal behaviors, and Braess-Paradox-related behaviors under different changes.

Example 1: Network with Braess's Paradox: Here is an example of network with Braess Link. It is a modification from the classical Braess's Paradox (using $1 + x$). Linear functions are implemented here to show the changes after.

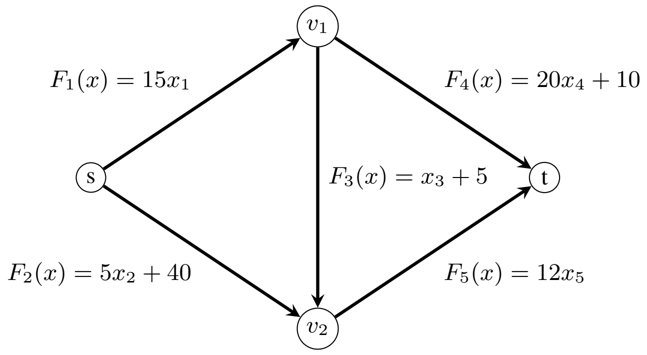


Fig. 4. Network with Braess Link (Source: Author)

Example 2: Network without Braaess's Paradox: Here is an example of network without Braess Link. It is identified that not all isomorphic networks will ensure the existence of Braess's Paradox. Linear functions are implemented here to show the changes after.

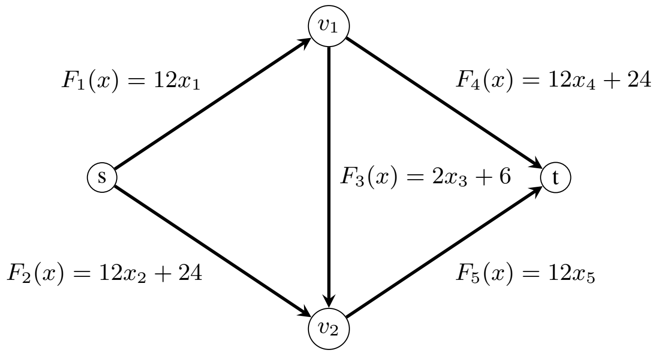


Fig. 5. Network without Braess Link (Source: Author)

Example 3: Complex Network: As classical Braess's Paradox can be modified, there is the second Braess graph. In Figure 6, we defined $A(x) = x$, $B(x) = 10$, and $C(x) = 10x$ for simplicity.

C. Data Processing

Once the network is represented as weighted adjacency matrix and all of the components, e.g. demand, nodes, source node, have been defined, it will be transformed into a graph using computational libraries such as NetworkX in Python. This allows for checking all possible paths from a defined origin-destination pair. Dijkstra's algorithm here is employed

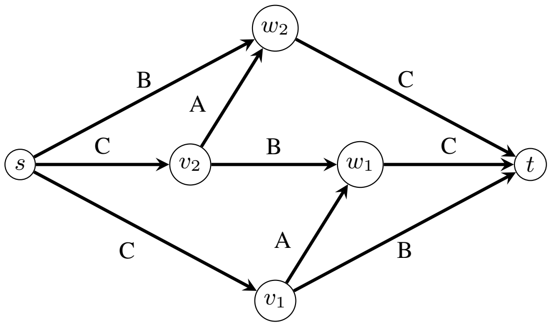


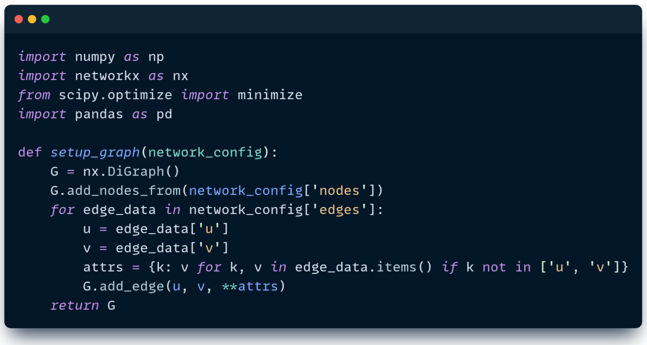
Fig. 6. Complex Network with Braess Link (Source: https://timroughgarden.org/papers/optima.pdf)

inside the libraries used in the program to compute shortest paths between intersections, supporting the individual route choice analysis. Each path is analyzed to compute its total cost based on the weights of its edges. This step is essential for identifying and comparing possible route choices that drivers may take.

To simulate routing behavior, the study applies Wardrop's Principles. Under the first principle (User Equilibrium) and the second principle (System Optimal), both equilibrium states are simulated using iterative adjustment techniques, allowing us to validate its efficiency. This processing framework enables the investigation of Braess's Paradox. By modifying the adjacency matrix to include or exclude specific edges and recomputing the equilibrium flows, the paradox can be observed and analyzed.

IV. Implementation

A. Imports and Graph Setup

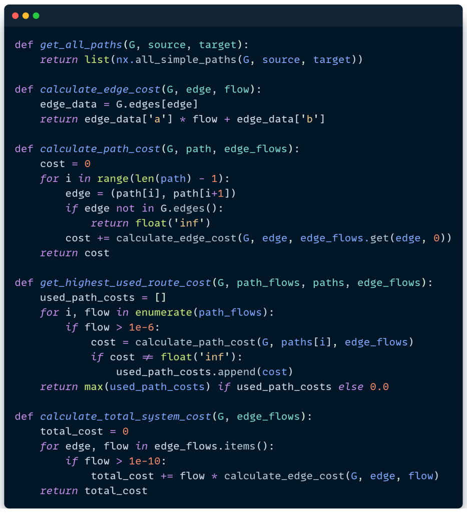


```python
import numpy as np
import networkx as nx
from scipy.optimize import minimize
import pandas as pd

def setup_graph(network_config):
    G = nx.DiGraph()
    G.add_nodes_from(network_config['nodes'])
    for edge_data in network_config['edges']:
        u = edge_data['u']
        v = edge_data['v']
        attrs = {k: v for k, v in edge_data.items() if k not in ['u', 'v']}
        G.add_edge(u, v, **attrs)
    return G
```

Fig. 7. Imports and Graph Setup
(Source: Author)

This script starts by importing a few important Python libraries. NumPy is used for handling numerical operations. NetworkX helps model the road network as a directed graph, allowing us to simulate different routing scenarios. The minimize function from SciPy is used to solve optimization problems to find the most efficient flow of traffic that minimizes overall system cost. Pandas is included to neatly present the results in tables. At first, the program will set an empty directed graph as the network at. Additional edges will be initialized as users give inputs to the program.

B. Calculation Helpers

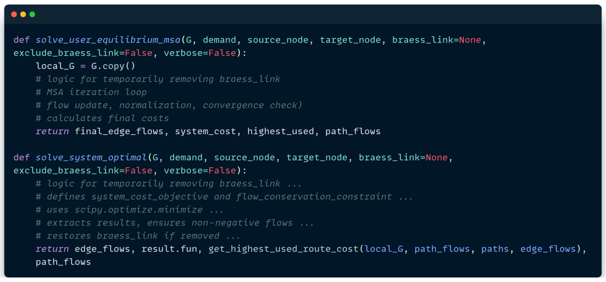


```python
def get_all_paths(G, source, target):
    return list(nx.all_simple_paths(G, source, target))

def calculate_edge_cost(G, edge, flow):
    edge_data = G.edges[edge]
    return edge_data['a'] * flow + edge_data['b']

def calculate_path_cost(G, path, edge_flows):
    cost = 0
    for i in range(len(path) - 1):
        edge = (path[i], path[i+1])
        if edge not in G.edges():
            return float('inf')
        cost += calculate_edge_cost(G, edge, edge_flows.get(edge, 0))
    return cost

def get_highest_used_route_cost(G, path_flows, paths, edge_flows):
    used_path_costs = []
    for i, flow in enumerate(path_flows):
        if flow > 1e-6:
            cost = calculate_path_cost(G, paths[i], edge_flows)
            if cost != float('inf'):
                used_path_costs.append(cost)
    return max(used_path_costs) if used_path_costs else 0.0

def calculate_total_system_cost(G, edge_flows):
    total_cost = 0
    for edge, flow in edge_flows.items():
        if flow > 1e-10:
            total_cost += flow * calculate_edge_cost(G, edge, flow)
    return total_cost
```

Fig. 8. Calculation Helpers
(Source: Author)

This part of the program handles the core computations for analyzing traffic behavior in the network. It first identifies all possible routes between the origin and destination, then calculates the cost of each segment and full path based on traffic flow. In this process, it takes care of invalid paths, for example, those affected by removed links, are excluded from consideration by assigning them infinite cost. For analyzing User Equilibrium, it determines the most expensive among the highest used route cost since UE assumes all used paths have equal minimal cost. Additionally, it can compute the total system cost by summing the cost contributions from every road segment, which is essential for optimizing the efficiency of the overall traffic network.

C. User Equilibrium and System Optimal Algorithms

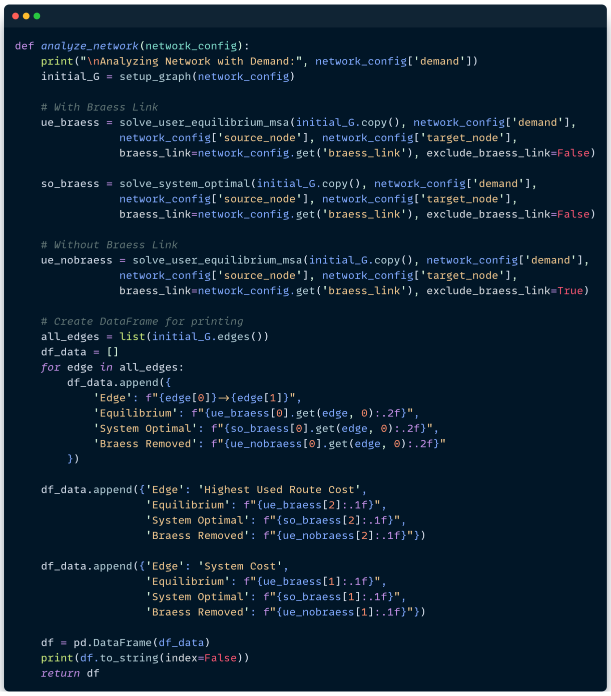


```python
def solve_user_equilibrium_msa(G, demand, source_node, target_node, braess_link=None,
exclude_braess_link=False, verbose=False):
    local_G = G.copy()
    # logic for temporarily removing braess_link
    # MSA iteration loop
    # flow update, normalization, convergence check)
    # calculates final costs
    return final_edge_flows, system_cost, highest_used, path_flows

def solve_system_optimal(G, demand, source_node, target_node, braess_link=None,
exclude_braess_link=False, verbose=False):
    # logic for temporarily removing braess_link ...
    # defines system_cost_objective and flow_conservation_constraint ...
    # uses scipy.optimize.minimize ...
    # extracts results, ensures non-negative flows ...
    # restores braess_link if removed ...
    return edge_flows, result.fun, get_highest_used_route_cost(local_G, path_flows, paths, edge_flows),
    path_flows
```

Fig. 9. User Equilibrium and System Optimal Algorithms
(Source: Author)

The code is designed to model and analyze traffic flow behavior under two distinct routing strategies, User Equilibrium (UE) and System Optimal (SO). On one's hand, the former method simulates how individual drivers selfishly choose the shortest route using the Method of Successive Averages (MSA). On the other hand, the latter method computes the most efficient overall traffic distribution by minimizing the total system cost through numerical optimization. Both approaches support the condition where of a Braess link is inserted or not to explore how network changes affect urban traffic.

D. Network Analysis and Output



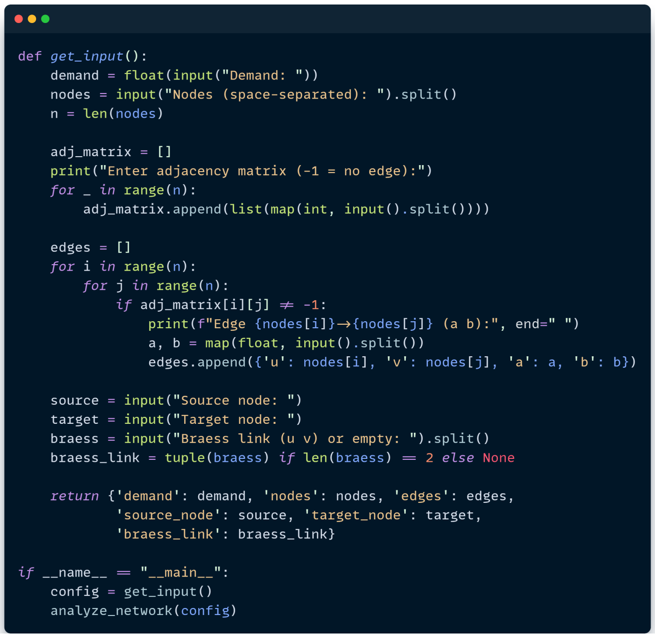
```python
def analyze_network(network_config):
    print("\nAnalyzing Network with Demand:", network_config['demand'])
    initial_G = setup_graph(network_config)

    # With Braess Link
    ue_braess = solve_user_equilibrium_msa(initial_G.copy(), network_config['demand'],
                network_config['source_node'], network_config['target_node'],
                braess_link=network_config.get('braess_link'), exclude_braess_link=False)

    so_braess = solve_system_optimal(initial_G.copy(), network_config['demand'],
                network_config['source_node'], network_config['target_node'],
                braess_link=network_config.get('braess_link'), exclude_braess_link=False)

    # Without Braess Link
    ue_nobraess = solve_user_equilibrium_msa(initial_G.copy(), network_config['demand'],
                network_config['source_node'], network_config['target_node'],
                braess_link=network_config.get('braess_link'), exclude_braess_link=True)

    # Create DataFrame for printing
    all_edges = list(initial_G.edges())
    df_data = []
    for edge in all_edges:
        df_data.append({
            'Edge': f"{edge[0]}->{edge[1]}",
            'Equilibrium': f"{ue_braess[0].get(edge, 0):.2f}",
            'System Optimal': f"{so_braess[0].get(edge, 0):.2f}",
            'Braess Removed': f"{ue_nobraess[0].get(edge, 0):.2f}"
        })

    df_data.append({'Edge': 'Highest Used Route Cost',
                    'Equilibrium': f"{ue_braess[2]:.1f}",
                    'System Optimal': f"{so_braess[2]:.1f}",
                    'Braess Removed': f"{ue_nobraess[2]:.1f}"})

    df_data.append({'Edge': 'System Cost',
                    'Equilibrium': f"{ue_braess[1]:.1f}",
                    'System Optimal': f"{so_braess[1]:.1f}",
                    'Braess Removed': f"{ue_nobraess[1]:.1f}"})

    df = pd.DataFrame(df_data)
    print(df.to_string(index=False))
    return df
```

Fig. 10. Network Analysis and Output
(Source: Author)

This part is responsible for displaying the results of different traffic flow scenarios in an organized format. It begins by printing headers to indicate which code is being executed first, such as User Equilibrium and System Optimal, both with and without the Braess link. After performing all computation, it resets the network to its original state. Once completed, it compiles the data into a structured results. Finally, the method prints a conclusion that checks for the presence of the classic Braess's Paradox.

E. Input Collector and Main Runner

This part is where the user can provide network configuration so that the program can analyze the Braess's Paradox inside the traffic network. The program will receive inputs of demand, nodes, adjacency matrix, cost functions, source and target node, and braess link. After the input has been received, the runner will do the work from the beginning part of the source code.



```python
def get_input():
    demand = float(input("Demand: "))
    nodes = input("Nodes (space-separated): ").split()
    n = len(nodes)

    adj_matrix = []
    print("Enter adjacency matrix (-1 = no edge):")
    for _ in range(n):
        adj_matrix.append(list(map(int, input().split())))

    edges = []
    for i in range(n):
        for j in range(n):
            if adj_matrix[i][j] != -1:
                print(f"Edge {nodes[i]}->{nodes[j]} (a b):", end=" ")
                a, b = map(float, input().split())
                edges.append({'u': nodes[i], 'v': nodes[j], 'a': a, 'b': b})

    source = input("Source node: ")
    target = input("Target node: ")
    braess = input("Braess link (u v) or empty: ").split()
    braess_link = tuple(braess) if len(braess) == 2 else None

    return {'demand': demand, 'nodes': nodes, 'edges': edges,
            'source_node': source, 'target_node': target,
            'braess_link': braess_link}

if __name__ == "__main__":
    config = get_input()
    analyze_network(config)
```

Fig. 11. Input Collector and Main Runner
(Source: Author)

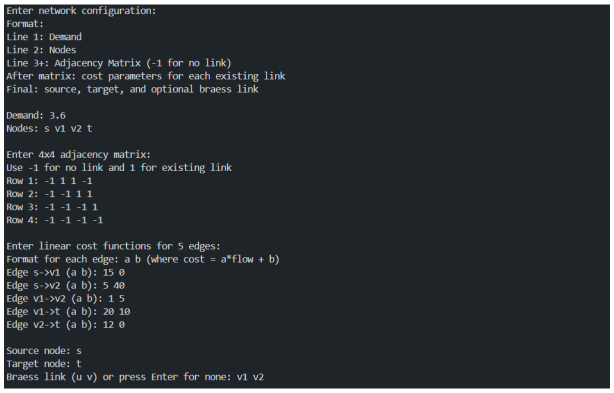
V. Case Analysis

A. Example 1: Network with Braess's Paradox

In Figure 13, the analysis of the traffic network with the modified cost functions and a demand of 3.6 units reveals the occurrence of the Classic Braess's Paradox. The User Equilibrium (UE) calculation for the network with the Braess link showed the highest used route cost of 73.5 and a total system cost of 264.4. Nevertheless, when the Braess link was excluded, the User Equilibrium resulted in a lower highest used route cost of 71.4 and a total system cost of 257.0. These numbers can be compared and as a result, they confirm the Classic Braess's Paradox that adding the link increased the highest used route cost (UE travel time) by 2.07 units, from 71.4 to 73.5. It shows that individual selfish routing decisions can lead to a worse overall outcome for all users. At quick glance, this statement can be strengthened by the PoA calculation (the ratio of UE and SO cost) that is greater than one.

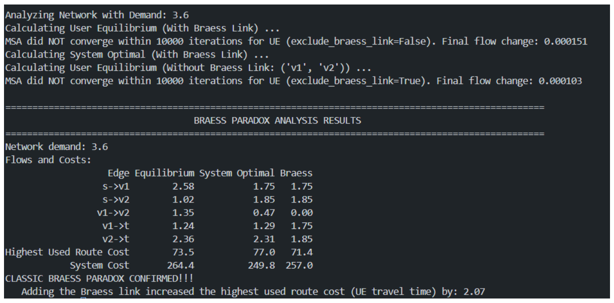
B. Example 2: Network without Braess's Paradox

In Figure 15, the analysis of the traffic network with the specified cost functions and a demand of 6.0 units reveals that the Classic Braess's Paradox does not occur in this scenario. The User Equilibrium (UE) with the Braess link resulted in a highest used route cost of 96.0 and a total system cost of 576.0. When the Braess link ($v1 \rightarrow v2$) was removed, the User Equilibrium still yielded the same highest used route with no improvement in system cost. This indicates that the presence of the Braess link had no impact on the efficiency in this configuration. The absence of any improvement confirms that the Classic Braess's Paradox is not observed here.



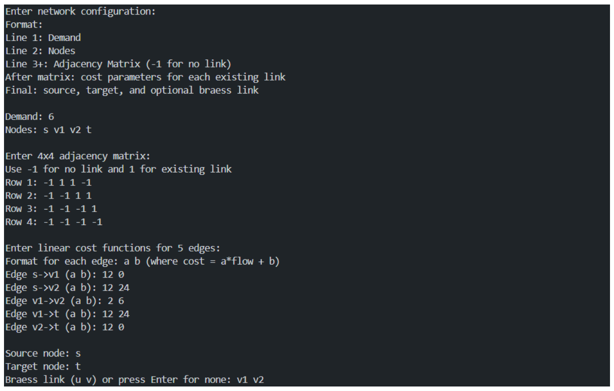
```
Enter network configuration:
Format:
Line 1: Demand
Line 2: Nodes
Line 3+: Adjacency Matrix (-1 for no link)
After matrix: cost parameters for each existing link
Final: source, target, and optional braess link

Demand: 3.6
Nodes: s v1 v2 t

Enter 4x4 adjacency matrix:
Use -1 for no link and 1 for existing link
Row 1: -1 1 1 -1
Row 2: -1 -1 1 1
Row 3: -1 -1 -1 1
Row 4: -1 -1 -1 -1

Enter linear cost functions for 5 edges:
Format for each edge: a b (where cost = a*flow + b)
Edge s->v1 (a b): 15 0
Edge s->v2 (a b): 5 40
Edge v1->v2 (a b): 1 5
Edge v1->t (a b): 20 10
Edge v2->t (a b): 12 0

Source node: s
Target node: t
Braess link (u v) or press Enter for none: v1 v2
```

Fig. 12. Input for Network with Braess's Paradox
(Source: Author)



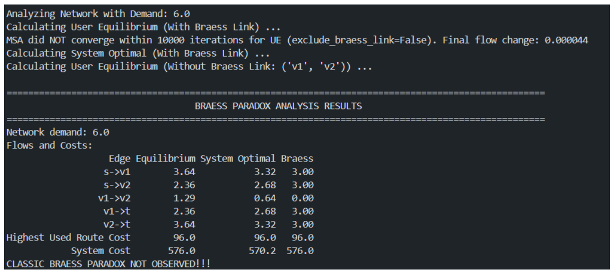
```
Analyzing Network with Demand: 3.6
Calculating User Equilibrium (With Braess Link) ...
MSA did NOT converge within 10000 iterations for UE (exclude_braess_link=False). Final flow change: 0.000151
Calculating System Optimal (With Braess Link) ...
Calculating User Equilibrium (Without Braess Link: ('v1', 'v2')) ...
MSA did NOT converge within 10000 iterations for UE (exclude_braess_link=True). Final flow change: 0.000103

================================================================================================
                                  BRAESS PARADOX ANALYSIS RESULTS
================================================================================================
Network demand: 3.6
Flows and Costs:
                   Edge Equilibrium System Optimal Braess
                  s->v1        2.58           1.75   1.75
                  s->v2        1.02           1.85   1.85
                 v1->v2        1.35           0.47   0.00
                  v1->t        1.24           1.29   1.75
                  v2->t        2.36           2.31   1.85
Highest Used Route Cost        73.5           77.0   71.4
            System Cost       264.4          249.8  257.0
CLASSIC BRAESS PARADOX CONFIRMED!!!
   Adding the Braess link increased the highest used route cost (UE travel time) by: 2.07
```

Fig. 13. Result for Network with Braess's Paradox
(Source: Author)



```
Enter network configuration:
Format:
Line 1: Demand
Line 2: Nodes
Line 3+: Adjacency Matrix (-1 for no link)
After matrix: cost parameters for each existing link
Final: source, target, and optional braess link

Demand: 6
Nodes: s v1 v2 t

Enter 4x4 adjacency matrix:
Use -1 for no link and 1 for existing link
Row 1: -1 1 1 -1
Row 2: -1 -1 1 1
Row 3: -1 -1 -1 1
Row 4: -1 -1 -1 -1

Enter linear cost functions for 5 edges:
Format for each edge: a b (where cost = a*flow + b)
Edge s->v1 (a b): 12 0
Edge s->v2 (a b): 12 24
Edge v1->v2 (a b): 2 6
Edge v1->t (a b): 12 24
Edge v2->t (a b): 12 0

Source node: s
Target node: t
Braess link (u v) or press Enter for none: v1 v2
```

Fig. 14. Input for Network without Braess's Paradox
(Source: Author)

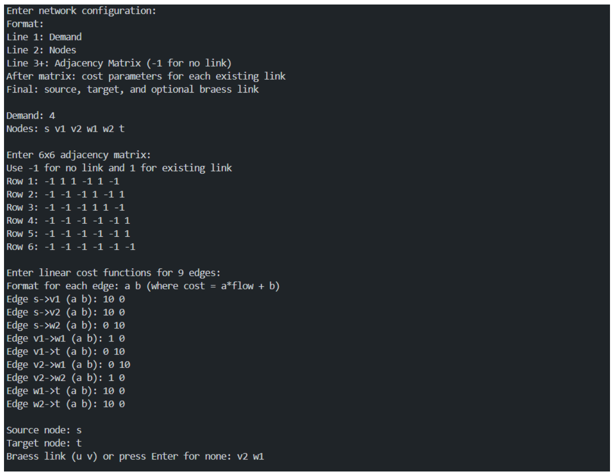


```
Analyzing Network with Demand: 6.0
Calculating User Equilibrium (With Braess Link) ...
MSA did NOT converge within 10000 iterations for UE (exclude_braess_link=False). Final flow change: 0.000044
Calculating System Optimal (With Braess Link) ...
Calculating User Equilibrium (Without Braess Link: ('v1', 'v2')) ...

================================================================================================
                                  BRAESS PARADOX ANALYSIS RESULTS
================================================================================================
Network demand: 6.0
Flows and Costs:
                   Edge Equilibrium System Optimal Braess
                  s->v1        3.64           3.32   3.00
                  s->v2        2.36           2.68   3.00
                 v1->v2        1.29           0.64   0.00
                  v1->t        2.36           2.68   3.00
                  v2->t        3.64           3.32   3.00
Highest Used Route Cost        96.0           96.0   96.0
            System Cost       576.0          570.2  576.0
CLASSIC BRAESS PARADOX NOT OBSERVED!!!
```

Fig. 15. Result for Network without Braess's Paradox
(Source: Author)

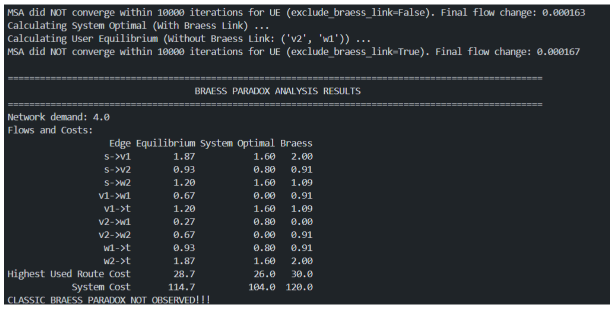
C. Example 3: Complex Network

In Figure 17, it is given a complex Braess network configuration with nodes $s, v1, v2, w1, w2,$ and t, and a total travel demand of 4.0 units. It reveals that the Classic Braess's Paradox does not occur. In the User Equilibrium (UE) scenario, the highest used route cost was 28.7 and the total system cost was 114.7. However, when the Braess link was removed, the UE resulted in a higher highest used route cost of 30.0 and a total system cost of 120.0. This indicates that removing the Braess link did not improve the overall traffic conditions. In fact, it implies that if we remove the Braess link, it will deteriorate the traffic condition since the highest used route cost is when the Braess link is not introduced.



```
Enter network configuration:
Format:
Line 1: Demand
Line 2: Nodes
Line 3+: Adjacency Matrix (-1 for no link)
After matrix: cost parameters for each existing link
Final: source, target, and optional braess link

Demand: 4
Nodes: s v1 v2 w1 w2 t

Enter 6x6 adjacency matrix:
Use -1 for no link and 1 for existing link
Row 1: -1 1 1 -1 1 -1
Row 2: -1 -1 -1 1 -1 1
Row 3: -1 -1 -1 1 1 -1
Row 4: -1 -1 -1 -1 -1 1
Row 5: -1 -1 -1 -1 -1 1
Row 6: -1 -1 -1 -1 -1 -1

Enter linear cost functions for 9 edges:
Format for each edge: a b (where cost = a*flow + b)
Edge s->v1 (a b): 10 0
Edge s->v2 (a b): 10 0
Edge s->w2 (a b): 0 10
Edge v1->w1 (a b): 1 0
Edge v1->t (a b): 0 10
Edge v2->w1 (a b): 0 10
Edge v2->w2 (a b): 1 0
Edge w1->t (a b): 10 0
Edge w2->t (a b): 10 0

Source node: s
Target node: t
Braess link (u v) or press Enter for none: v2 w1
```

Fig. 16. Input for Complex Network
(Source: Author)



```
MSA did NOT converge within 10000 iterations for UE (exclude_braess_link=False). Final flow change: 0.000163
Calculating System Optimal (With Braess Link) ...
Calculating User Equilibrium (Without Braess Link: ('v2', 'w1')) ...
MSA did NOT converge within 10000 iterations for UE (exclude_braess_link=True). Final flow change: 0.000167

====================================================================================================
                                   BRAESS PARADOX ANALYSIS RESULTS
====================================================================================================
Network demand: 4.0
Flows and Costs:
                  Edge Equilibrium System Optimal Braess
                 s->v1        1.87           1.60   2.00
                 s->v2        0.93           0.80   0.91
                 s->w2        1.20           1.60   1.09
                v1->w1        0.67           0.00   0.91
                 v1->t        1.20           1.60   1.09
                v2->w1        0.27           0.80   0.00
                v2->w2        0.67           0.00   0.91
                 w1->t        0.93           0.80   0.91
                 w2->t        1.87           1.60   2.00
Highest Used Route Cost       28.7           26.0   30.0
           System Cost       114.7          104.0  120.0
CLASSIC BRAESS PARADOX NOT OBSERVED!!!
```

Fig. 17. Result for Complex Network
(Source: Author)

VI. Conclusion

Through the use of adjacency matrices and algorithmic modeling, this work can simulate how individual route choices influence broader system behaviors. Moreover, adding new links unconsciously worsen the traffic conditions. By looking at overall network in a structured mathematical format, graph theory provides a comprehensive way to evaluate the effects of proposed infrastructure changes before they are implemented.

The computational tools enabled by graph theory allow for precise calculation of key traffic states such as User Equilibrium and System Optimal flow. This level of analysis is critical in large, complex urban environments where small changes can have far-reaching effects. It enables governments to model multiple scenarios, consider the trade-offs of different projects, and prioritize improvements that results in system-wide benefits rather than only shifting congestion.

In a broader context, graph theory assists policymakers with a data-driven consideration for future planning rather than relying solely on engineering intuition. Their decisions can be guided by insights into network behavior. This is not only reducing the cost of planning errors, but also enhancing accountability in public infrastructure investment. Therefore, cities can design transportation networks that are not just larger, but smarter for the public.

Attachment

Github: Source Code of ”Graph Theory Application in Modeling Braess's Paradox for Urban Traffic Optimization”
Here
Youtube Video: Demonstration on The Source Code ”Graph Theory Application in Modeling Braess's Paradox for Urban Traffic Optimization” using Python
Here

Acknowledgment

The author gratefully acknowledges the blessings and strength granted by God Almighty, which enabled the successful completion of this paper. Sincere appreciation is also extended to Dr. Ir. Rinaldi, M.T., lecturer of the IF1220 Discrete Mathematics course, for his continuous support, guidance, and encouragement throughout the semester and in the writing of this paper. Next, the author would thank both of the author's parents who always give the author supports during ups and downs during the process of this work. Last but not least, the author would also like to thank Mr. Arthur Pigou as the first inventor of this Braess's Paradox, which significantly inspired this work.

References

[1] Braess, D. (1968). ”Über ein Paradoxon aus der Verkehrsplanung.” Unternehmensforschung, 12, 258-268. (Accessed on June 16, 2025) https://homepage.rub.de/Dietrich.Braess/Paradox-BNW.pdf
[2] Dijkstra, E. W. (1959). ”A Note on Two Problems in Connexion with Graphs.” Numerische Mathematik, 1(1), 269-271. (Accessed on June 16, 2025) https://ir.cwi.nl/pub/9256/9256D.pdf
[3] Hagstrom, J. N., & Abrams, R. A. (2001). Characterizing Braess's Paradox for Traffic Networks. In 2001 IEEE Intelligent Transportation Systems Conference Proceedings (ITSC 2001). IEEE, 836 - 841. (Accessed on June 17, 2025) https://ieeexplore.ieee.org/document/948769
[4] Munir, R. (2024). IF1220 Matematika Diskrit - Semester II Tahun 2024/2025: Graf (Bagian 1) (Versi Update 2024). (Accessed on June 17, 2025) https://informatika.stei.itb.ac.id/rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf
[5] Munir, R. (2024). IF1220 Matematika Diskrit - Semester II Tahun 2024/2025: Graf (Bagian 2) (Versi Update 2024) . (Accessed on June 17, 2025) https://informatika.stei.itb.ac.id/ rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf
[6] Roughgarden, T. (2005). Selfish Routing and the Price of Anarchy. MIT Press. (Accessed on June 17, 2025). https://timroughgarden.org/papers/optima.pdf

[7] Wardrop, J. G. (1952). ”Some Theoretical Aspects of Road Traffic Research.” Proceedings of the Institution of Civil Engineers, 1(3), 325-362. (Accessed on June 17, 2025) https://people.irisa.fr/Nicolas.Markey/PDF/Papers/pice1(3)-War.pdf

Statement

Hereby I declare that this paper that I have written is my own work, not a reproduction or translation of someone else’s work and not plagiarized.

Bandung, 20 Juni 2025

Kevin

Kevin Wirya Valerian
13524019