A-PRIORI PATHFINDING ROBOT

Project Year 3

Kasia, Mekhola, Zosia, Eman

Research Question:

WHAT ALGORITHM IS OPTIMUM FOR A ROBOT WAITER?

TABLE OF CONTENTS

- Assigned Tasks - What does the robot do?
- Assumptions and considerations
- The robot in action - Live simulation
- Methodology:

  -Layout

  -Algorithms

  -Translation function
- Performance analysis and Results

Assumed tasks:

01 - COLLECTING FOOD

We assume the robot has a designated initial position where it can collect the orders that are ready to be served and the kitchen staff can enter the table numbers of each order.

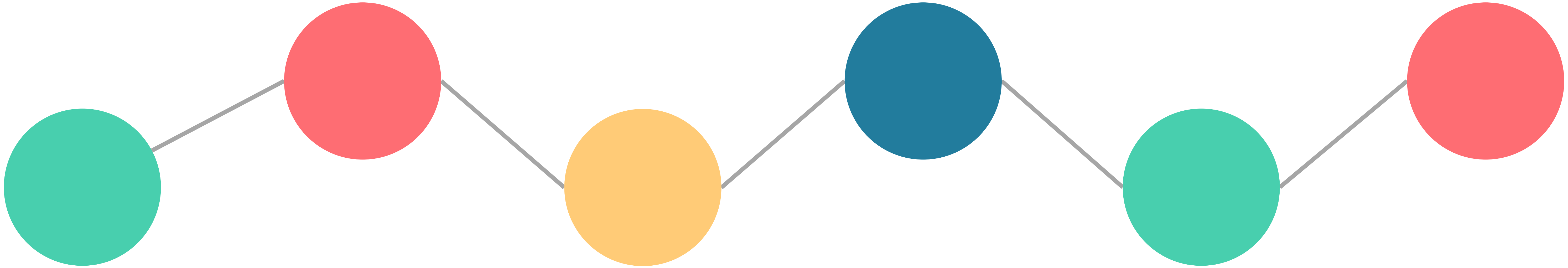
02 - PATHFINDING

Depending on the table numbers assigned for the trip, the robot must calculate its shortest path a-priori to actually moving, such that each table is visited once before returning to the initial position to collect the next round of orders.

03 - FOLLOWING PATH

Once the shortest path is calculated, the robot must simply follow that path at a constant speed in a straight line.

ASSUMPTIONS AND CONSIDERATIONS



The path of the robot is free of any obstacles.

The robot does not need to take orders from customers.

Different table orders are placed on different shelves assigned by staff for each trip.

The maximum carrying capacity is dependent on hardware so program mut be flexible to it.

Waiting time for each stop is dependent on weight sensors in the shelves.

A-priori pathfinding is essential due to time constraints.

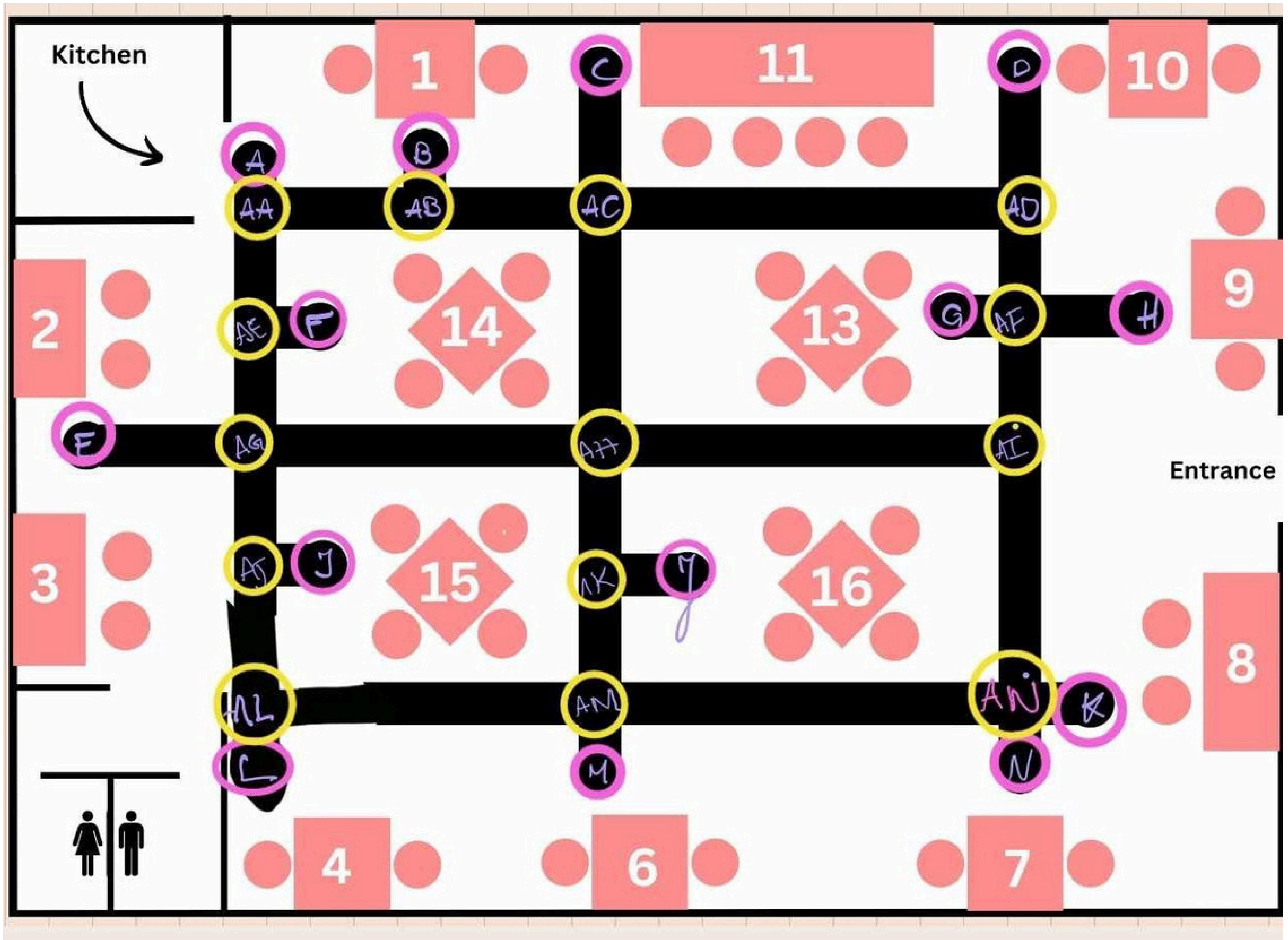
SIMULATION IN WEBOTS ENVIRONMENT

METHODOLOGY

HOW THE WAS SIMULATION WAS PROGRAMMED

LAYOUT REPRESENTATION

1) Design floorplan
2) Map coordinates
3) Calculate distances
4) Create dictionary



WEBOTS GPS MODULE

DICT: NODES & EDGES

NETWORKX LIBRARY

VISUALIZING THE MAP

1

WEIGHTS

THE APPROXIMATE DISTANCES BETWEEN TWO NODES

2

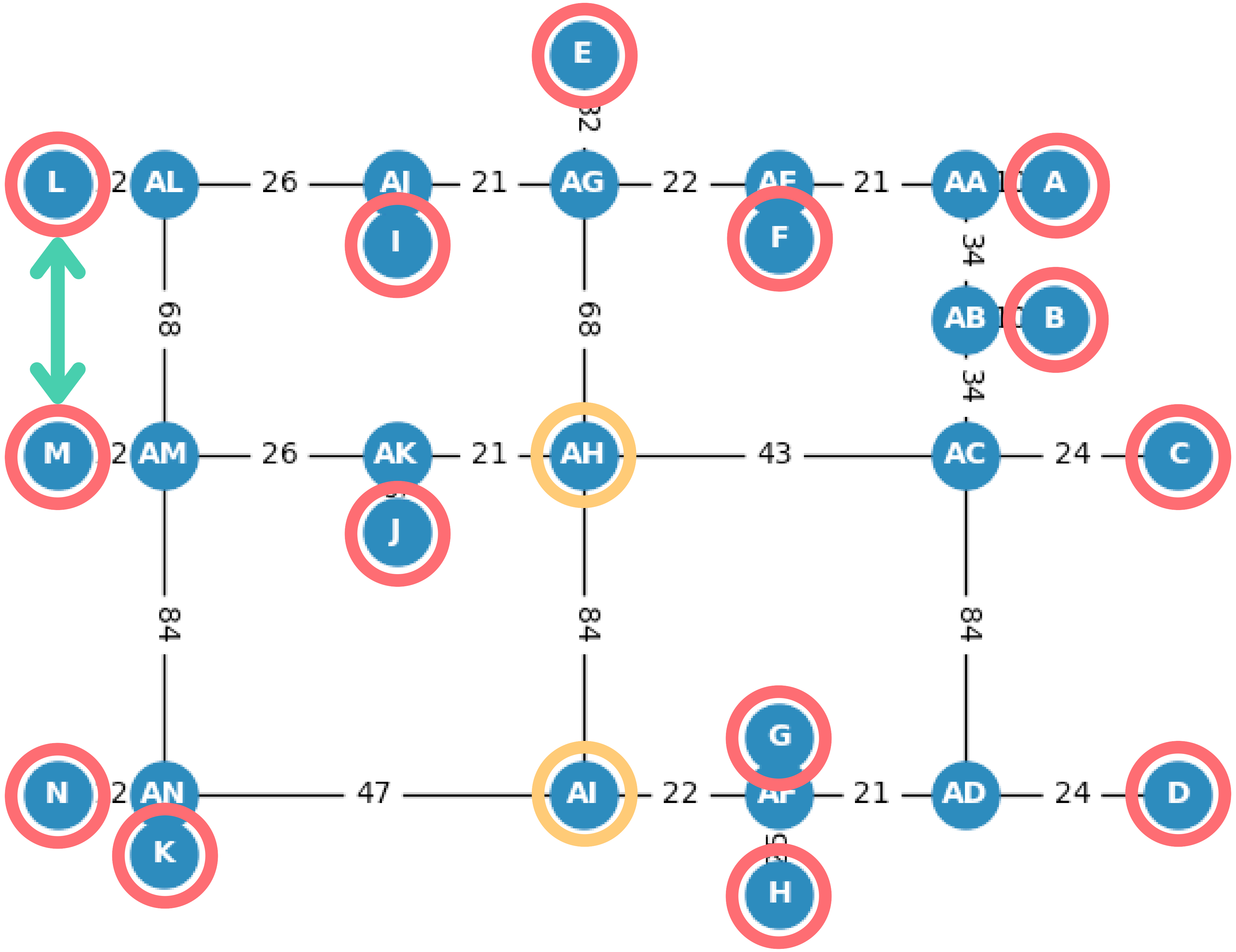
DESTINATION NODES

THE DESIGNATED STOPS THAT THE ROBOT NEEDS TO WAIT AT BEFORE THE CUSTOMERS COLLECT ORDER

3

INTERMEDIATE NODES

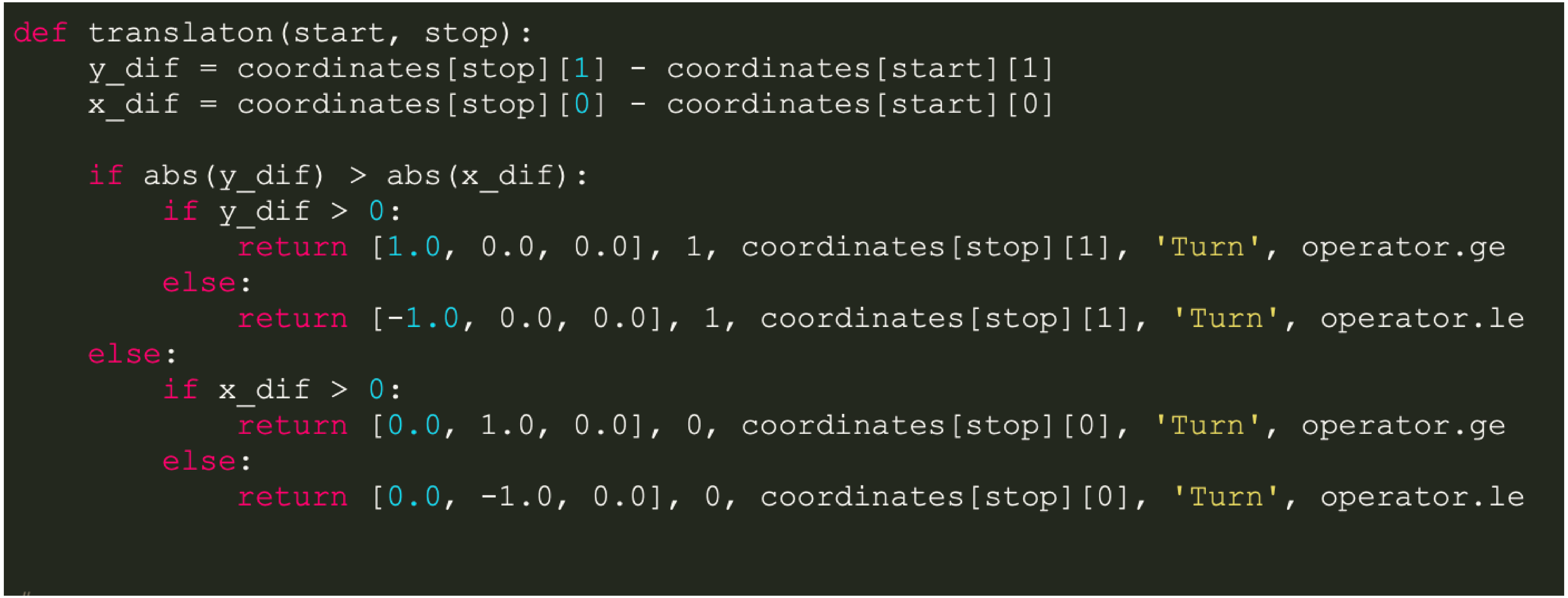
NODES THAT IT DOES NOT NEED TO WAIT AT BUT STILL NEEDS TO FOLLOW A CHOSEN PATH



A* AND DIJKSTRA'S ALGORITHMS

Feature	Dijkstra's Algorithm	A* Algorithm
Heuristic Use	None	Yes
Goal	Shortest path to all nodes	Shortest path to a target
Efficiency	Can be slower without a heuristic	Faster with a good heuristic
Complexity	$O(V + E \log V)$	$O(V + E \log V)$
Optimality	Always finds the shortest path	Finds the shortest path if the heuristic is admissible

TRANSLATING COORDINATES INTO INSTRUCTIONS



```python
def translaton(start, stop):
    y_dif = coordinates[stop][1] - coordinates[start][1]
    x_dif = coordinates[stop][0] - coordinates[start][0]

    if abs(y_dif) > abs(x_dif):
        if y_dif > 0:
            return [1.0, 0.0, 0.0], 1, coordinates[stop][1], 'Turn', operator.ge
        else:
            return [-1.0, 0.0, 0.0], 1, coordinates[stop][1], 'Turn', operator.le
    else:
        if x_dif > 0:
            return [0.0, 1.0, 0.0], 0, coordinates[stop][0], 'Turn', operator.ge
        else:
            return [0.0, -1.0, 0.0], 0, coordinates[stop][0], 'Turn', operator.le
```

STATE MACHINE

01 - TURNING

Turning until it riches the correct angle.
Usese **compass** - a webots base nodes

02 - GOING STRAIGHT

Goes straight until it riches the correct coordinates.
Usese **GPS** - a webots base nodes

03 - CALCULATION

Gets current position (node) and the next one
and does the “translation”

HOW TO MEASURE PERFORMANCE OF AN ALGORITHM IN THIS CONTEXT?

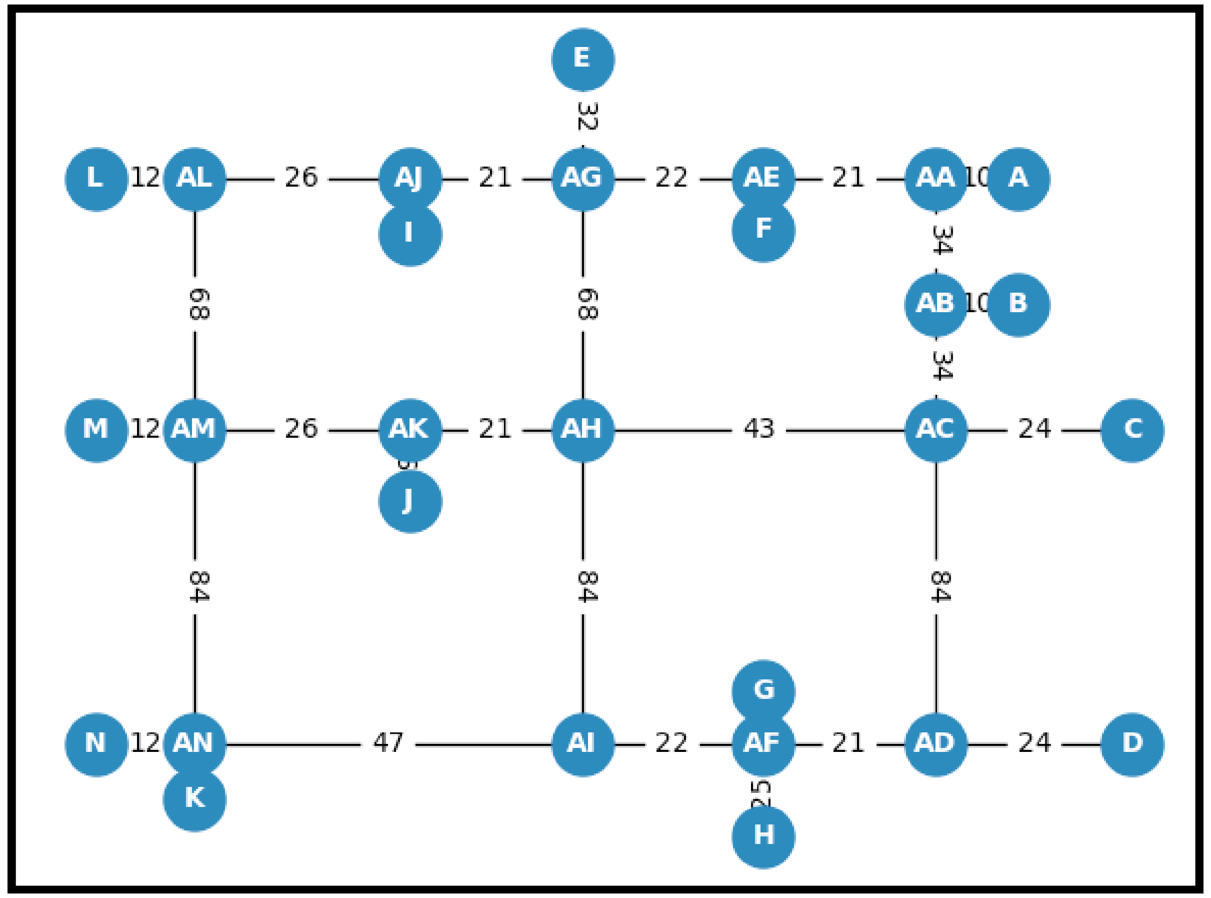
- 01 - RUNTIME
- 02 - COST MINIMIZATION
- 03 - LAYOUT COMPLEXITY

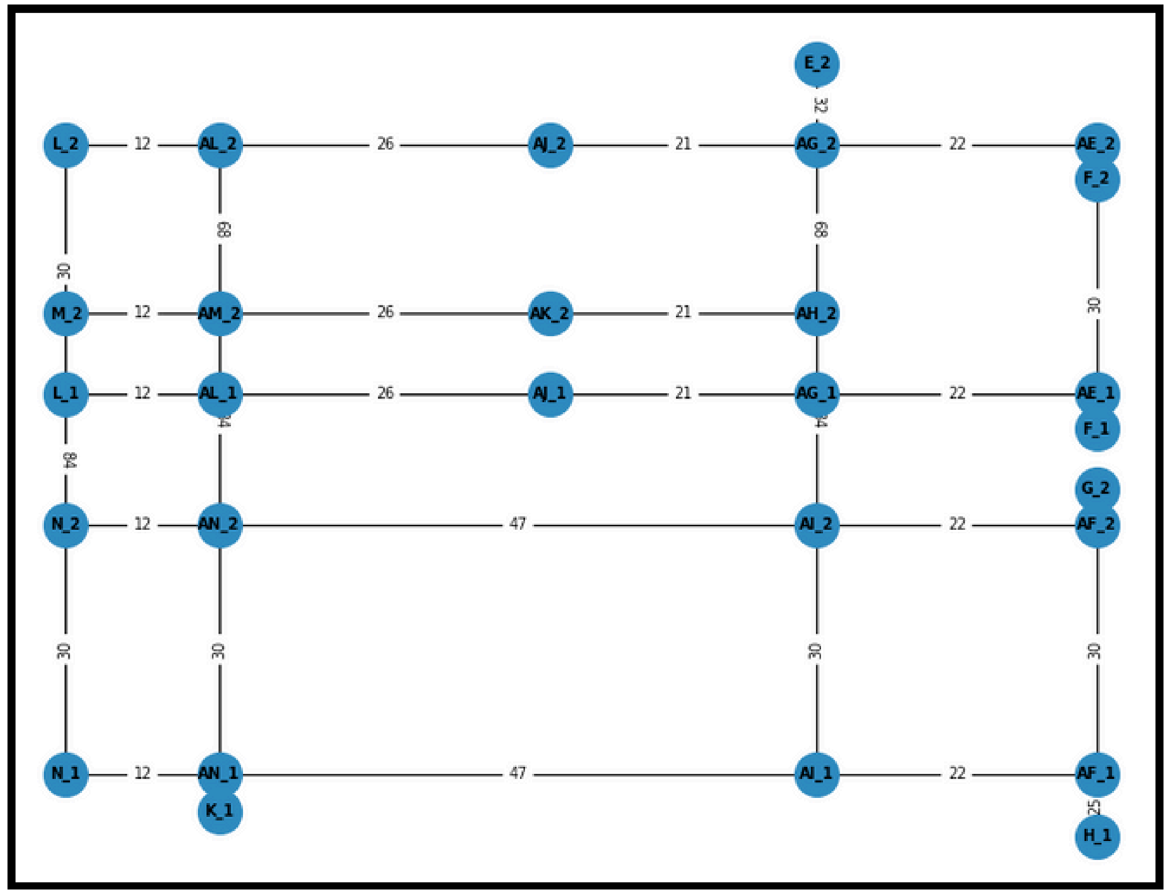
LAYOUT COMPLEXITY

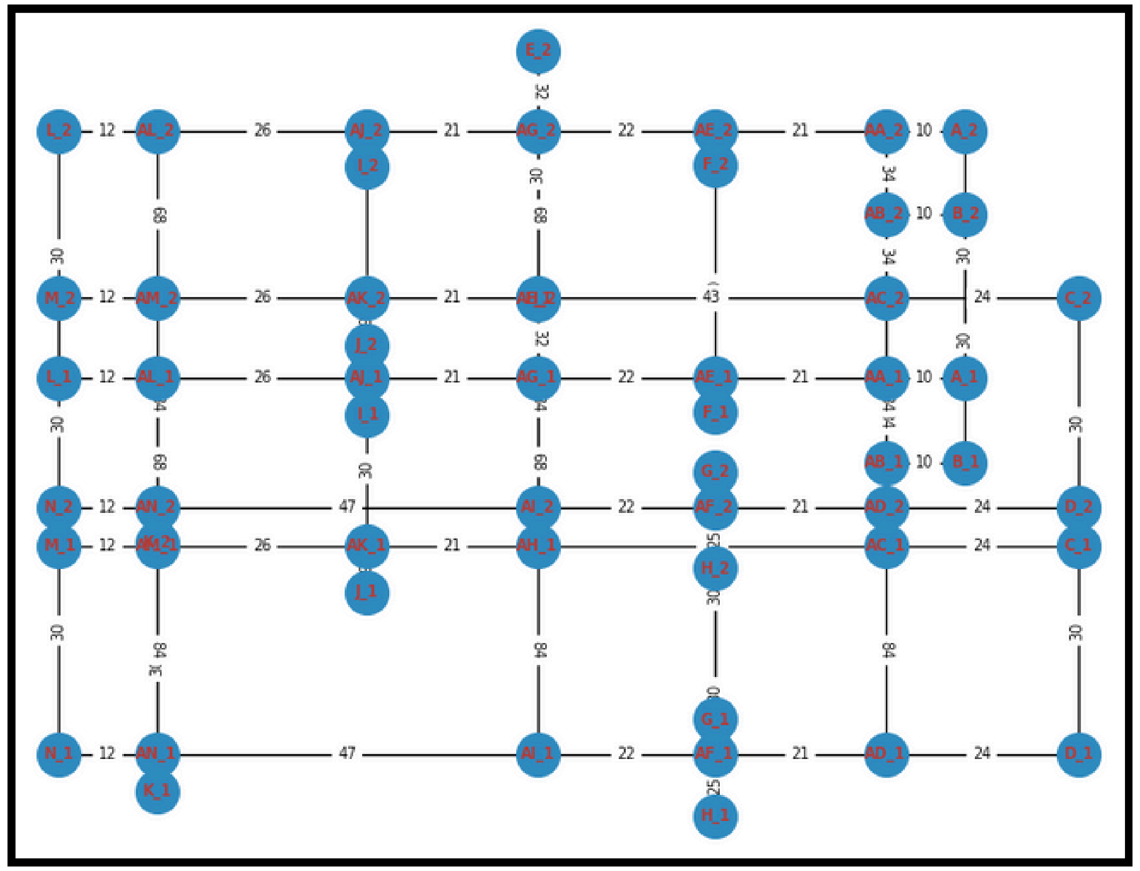
NUMBER OF NODES AND CONNECTIONS

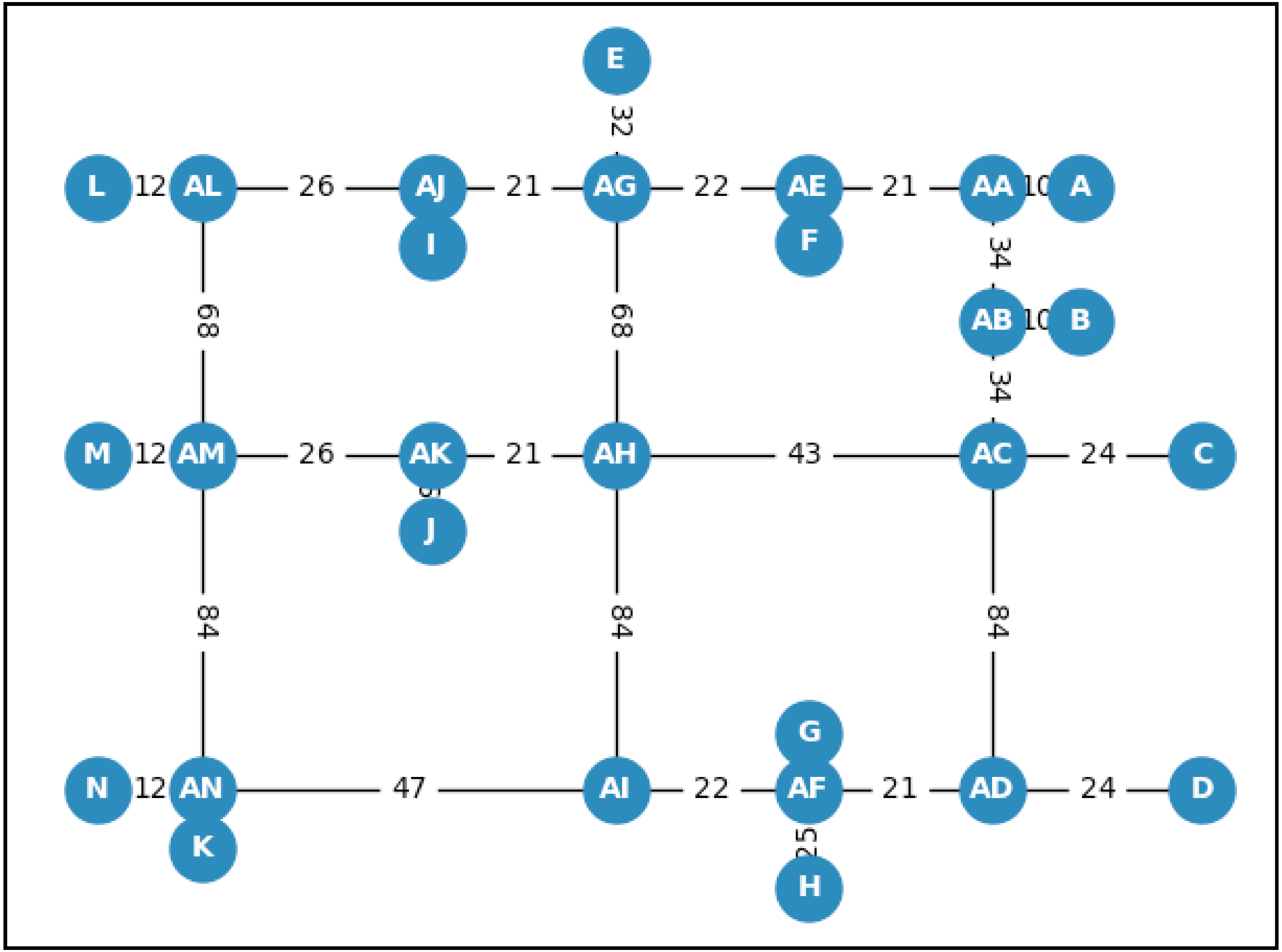
PATH LENGTH VARIABILITY

INTERSECTION POINTS AND CONGESTION



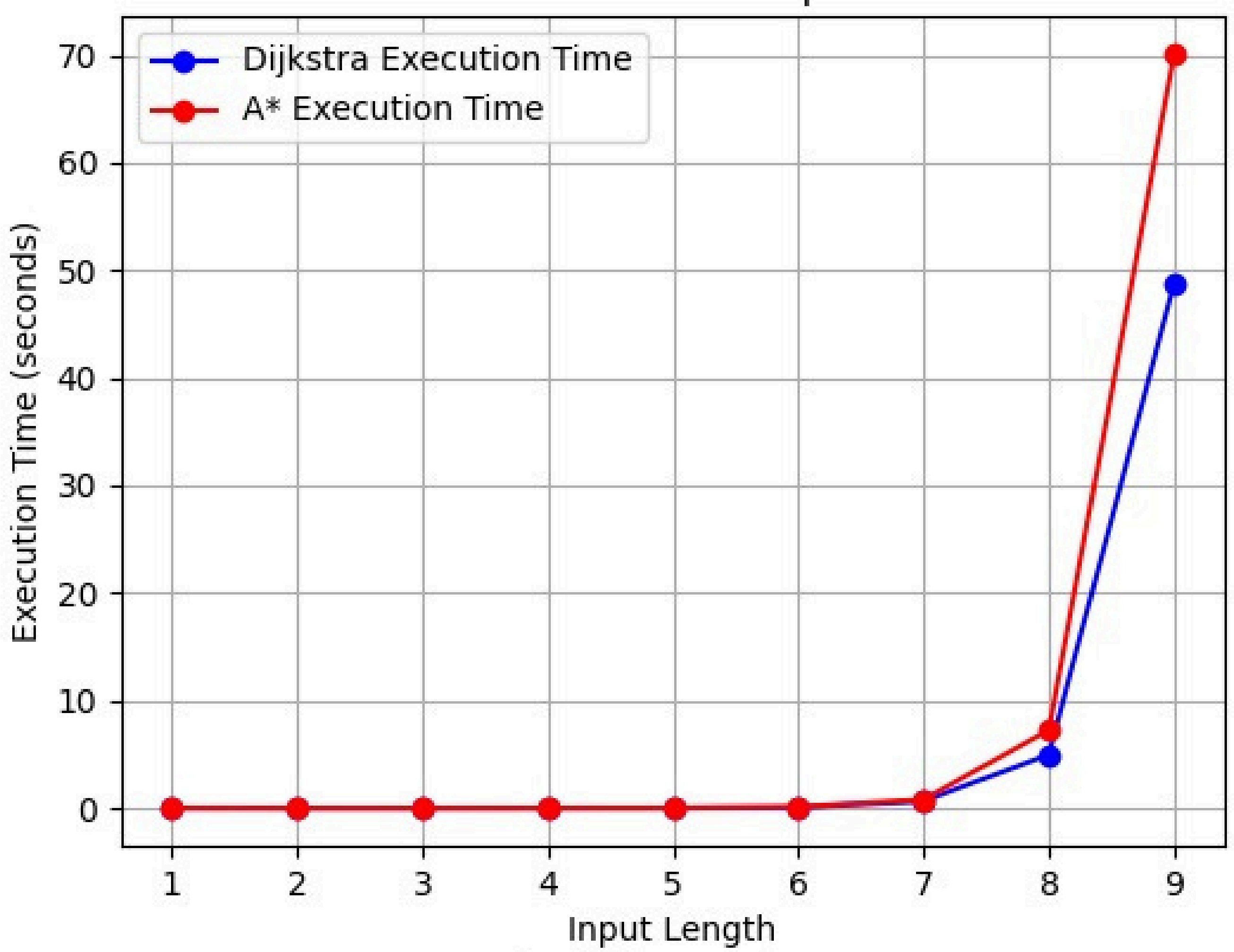




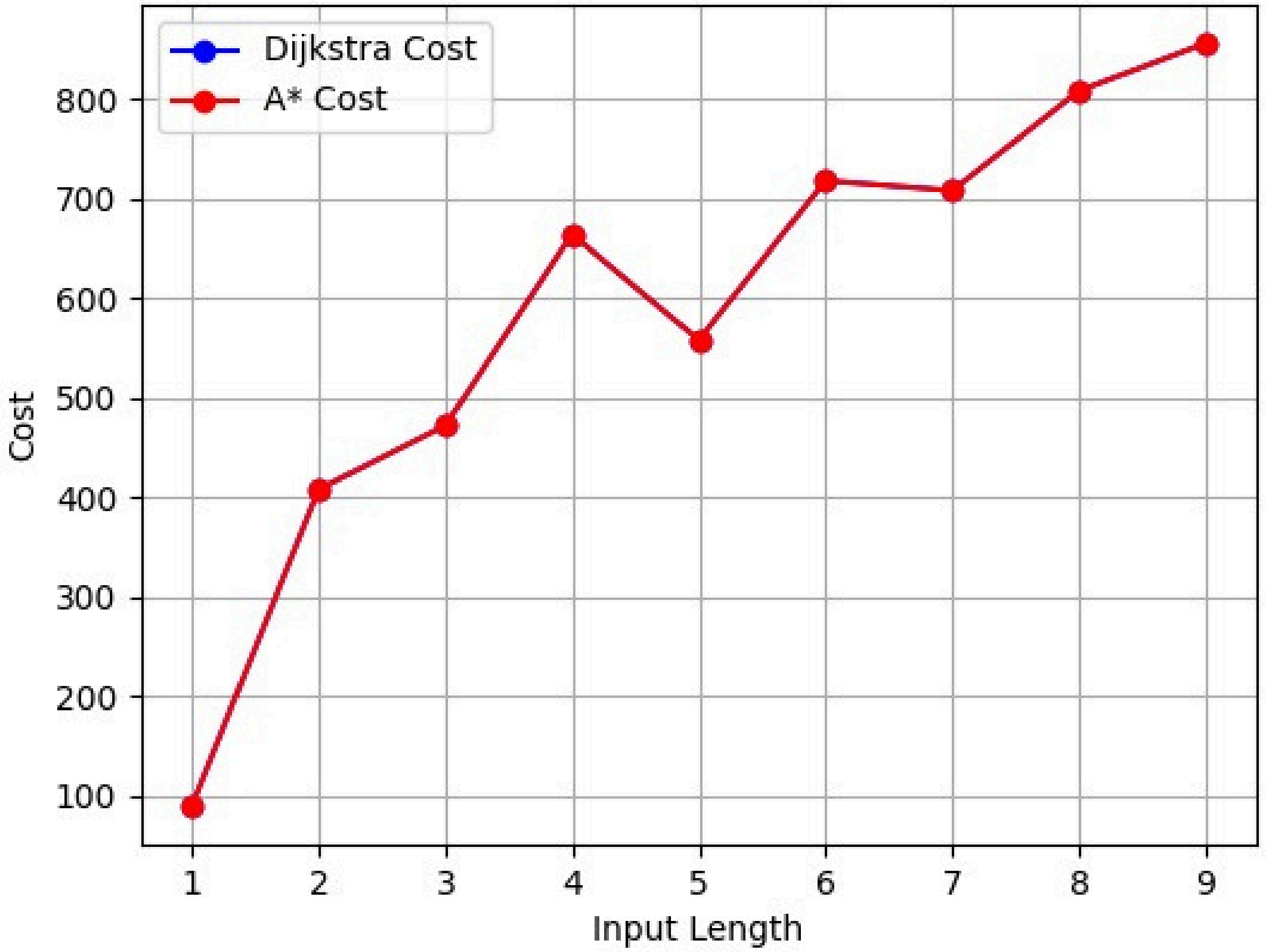

E
32
L
12
AL
26
AJ
21
AG
22
AE
21
AA
10
A
I
F
34
68
68
AB
10
B
34
M
12
AM
26
AK
21
AH
43
AC
24
C
5
J
84
84
84
G
N
12
AN
47
AI
22
AF
21
AD
24
D
K
25
H

RESULTS

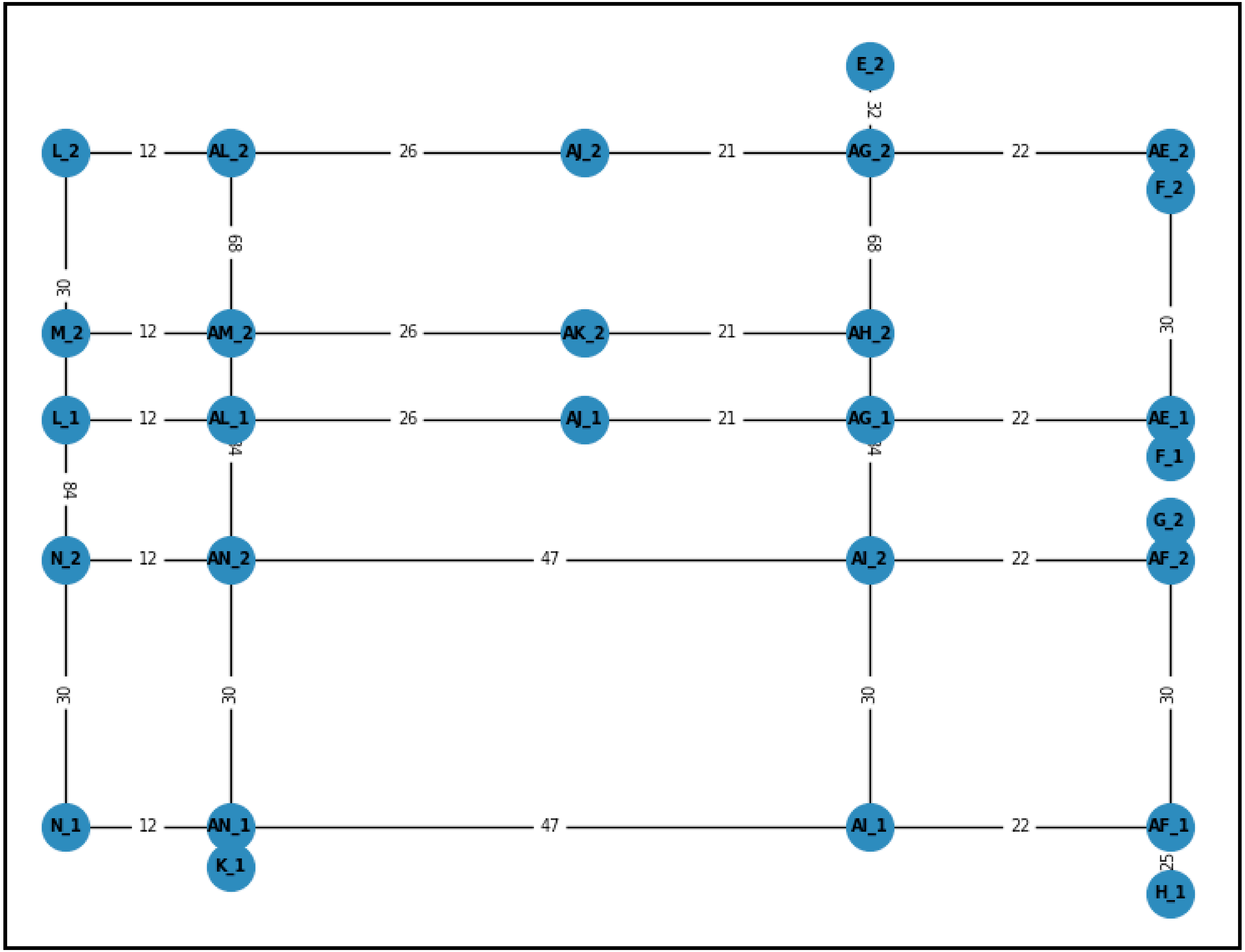
layout 1
(used in simulation)



Execution Time Comparison

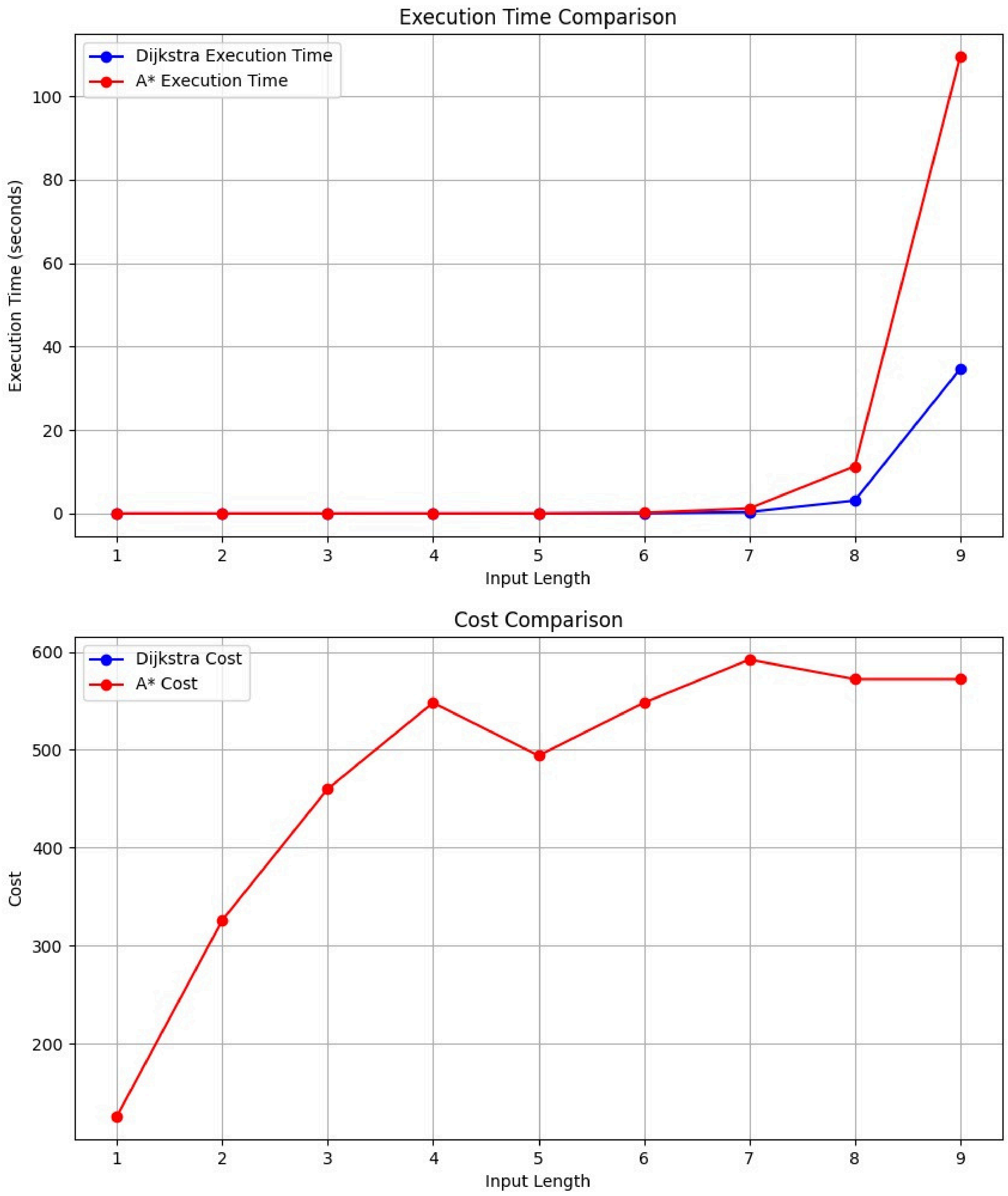


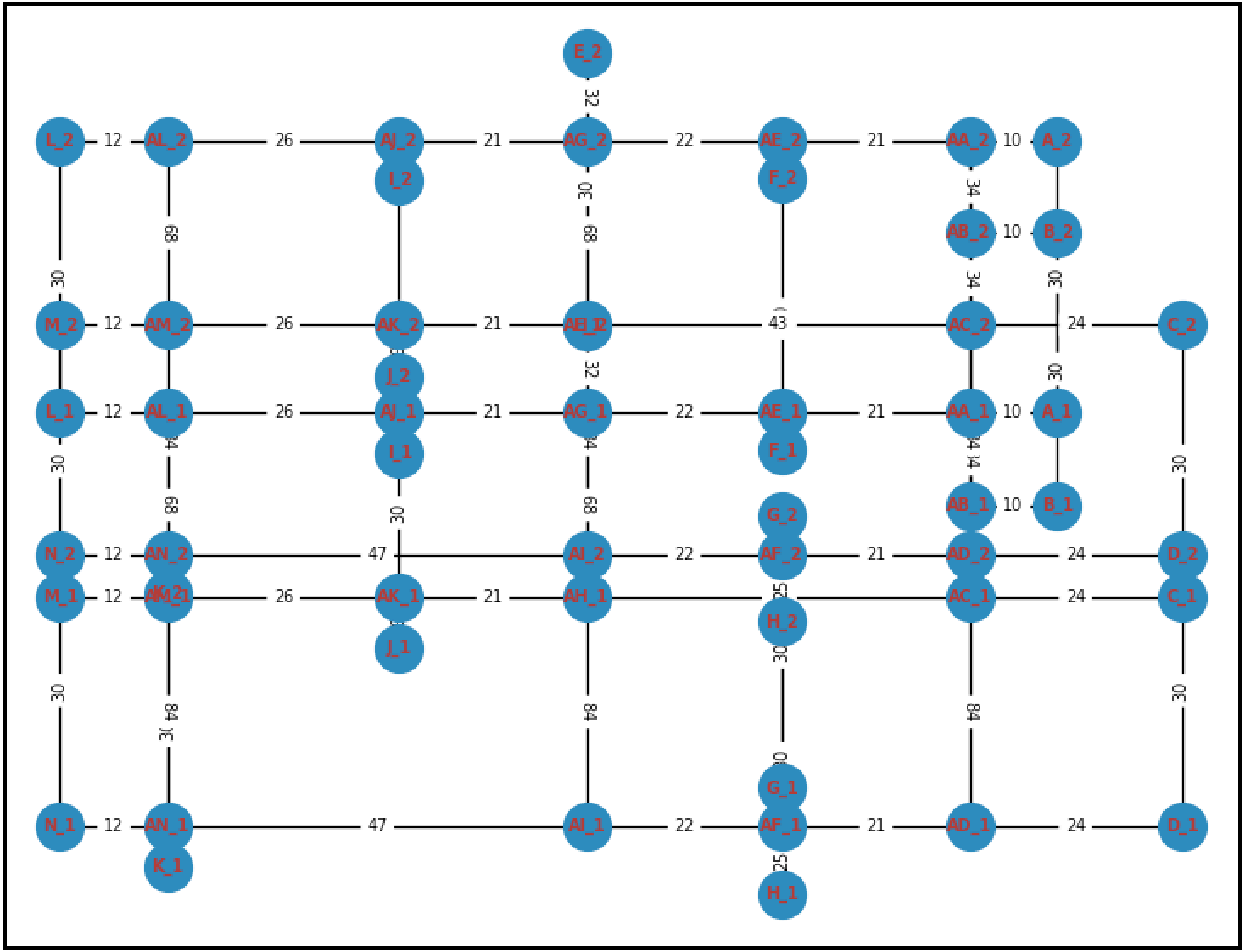
Cost Comparison


E_2
32
L_2
12
AL_2
26
AJ_2
21
AG_2
22
AE_2
F_2
68
68
30
30
M_2
12
AM_2
26
AK_2
21
AH_2
L_1
12
AL_1
26
AJ_1
21
AG_1
22
AE_1
F_1
84
G_2
N_2
12
AN_2
47
AI_2
22
AF_2
30
30
30
30
N_1
12
AN_1
47
AI_1
22
AF_1
K_1
25
H_1

RESULTS

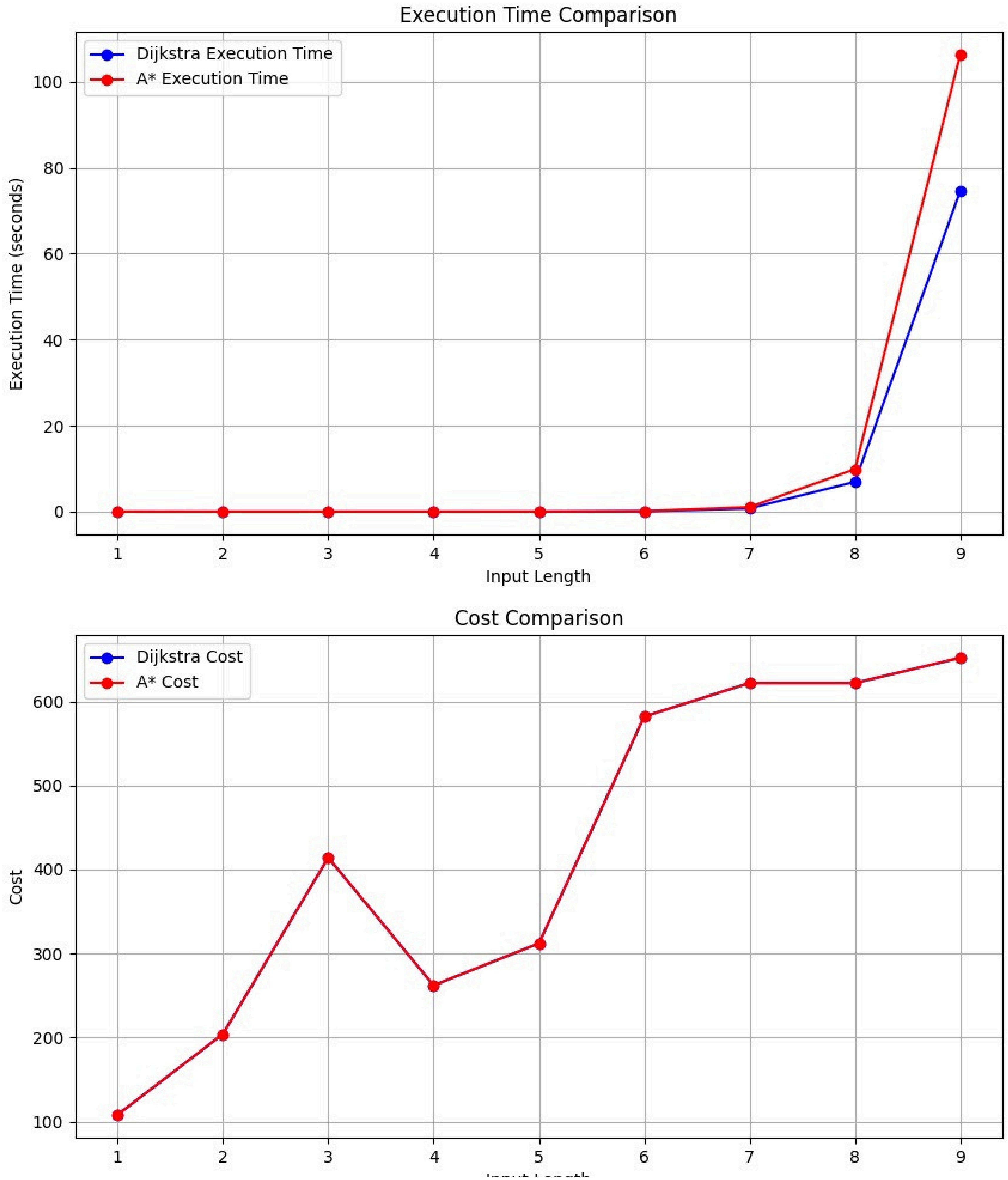
layout 2




E_2
32
L_2
12
AL_2
26
AJ_2
21
AG_2
22
AE_2
21
AA_2
10
A_2
I_2
F_2
30
34
68
68
AB_2
10
B_2
30
34
30
M_2
12
AM_2
26
AK_2
21
AH_2
43
AC_2
24
C_2
J_2
32
30
L_1
12
AL_1
26
AJ_1
21
AG_1
22
AE_1
21
AA_1
10
A_1
I_1
F_1
30
34
34
34
68
68
30
AB_1
10
B_1
G_2
N_2
12
AN_2
47
AI_2
22
AF_2
21
AD_2
24
D_2
M_1
12
AM_1
K_2
26
AK_1
21
AH_1
25
AC_1
24
C_1
H_2
J_1
30
30
84
84
84
30
30
G_1
N_1
12
AN_1
47
AI_1
22
AF_1
21
AD_1
24
D_1
K_1
25
H_1

RESULTS

layout 3



Execution Time Comparison

Cost Comparison

LIMITATIONS

1. Constraints of Digital Simulations

2. Limitations on Real-World Transferability

3. Inadequacies in Sensor Evaluation

References:

Akdogan, A. (2021). Understanding A* path algorithms and implementation with python. Medium. https://towardsdatascience.com/understanding-a-path-algorithms-and-implementation-with-python-4d8458d6ccc7

Alamri, H., Alamri, S. A., Alshehri, S., Alshehri, W., & Alhmiedat, T. (2021). Autonomous Maze Solving Robotics: Algorithms and systems. 10.18178/ijmerr.10.12.668-675

Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics. 4 (2): 100–107.

Kulaç, N., & Engin, M. (2023). Developing a machine learning algorithm for Service Robots in industrial applications. MDPI. 10.3390/machines11040421

Özkan, M., Tabakoğlu, A., Uygun, E. G. & Anbaroğlu, Berk (2022). A Computationally Reproducible Approach to Dijkstra's Shortest Path Algorithm. Advanced Geomatics, 2(1), 01-06

Shrinivaas, B. (2023). Dijkstra-the person & algorithm. Medium. https://medium.com/@balajeraam/dijkstra-the-person-algorithm-5016ebe9468#62c9