Planning and Decision Making Assignment 2

Zhuoyang Wang (6376770)

1. Graph Search

Answer 1.1

Denote a path p from a to b as $p : a \rightsquigarrow b$, and the length (distance) of path p is $l(p)$.

Suppose that there exists another path from s to v using edge e with a distance shorter than $P_s(u) + d(e) + P_t(v)$. Then there exist a path $p_1 : s \rightsquigarrow u$ and a path $p_2 : v \rightsquigarrow t$ with a total length shorter than $P_s(u) + P_t(v)$, i.e.

$$l(p_1) + l(p_2) < P_s(u) + P_t(v). \tag{1}$$

From the definition of $P_s(u)$ and $P_t(v)$, we have $l(p_1) \geq P_s(u)$ and $l(p_2) \geq P_t(v)$. By addition we have

$$l(p_1) + l(p_2) \geq P_s(u) + P_t(v), \tag{2}$$

which is contradictory to Equation 1. Therefore, the assumption does not hold, i.e., there doesn't exist any path from s to v using edge e with a distance shorter than $P_s(u) + d(e) + P_t(v)$ (which is obviously feasible).

So the shortest path from node s to node t that uses the edge e is $P_s(u) + d(e) + P_t(v)$.

Answer 1.2

The paths that are not exactly equal to Q should have at least one edge e that is not in Q. And from the property obtained in Question 1.1 we can calculate the shortest path using that edge e. So by trying all the edges that are not in Q and checking the shortest distance, we can find the second shortest path. Here is the algorithm:

Find the second shortest path ($G = (V, E)$, s, t):

1. Run shortest path algorithms (Dijkstra, etc.) on G to obtain the shortest paths (with edge sequences) from s to any other node v, denoted as $P_s(v)$.
2. Record the shortest path Q, with the edge set E_Q.
3. Run shortest path algorithms the reversed graph of G to obtain the shortest paths (with edge sequences) from any other node v to t, denoted as $P_t(v)$.
4. Iterate through each edge $e : u \to v$ in $E \setminus E_Q$, calculate $\tilde{l}(e) = P_s(u) + d(e) + P_t(v)$ and record the paths.
5. The path with the minimal $\tilde{l}(e)$ is (one of) the second shortest path.

2. Map to Graph

Put in all the vertices of the obstacles and the start and goal points, and then connect all the mutually visible points to obtain the shortest-path roadmap as shown in Figure 1.

Pick seven of the edges and calculate their costs (defined as the Euclidean distance between the two nodes):

- $1 \to 5$: $\sqrt{1^2 + (1+2)^2} = \sqrt{10} \approx 3.162$
- $5 \to 15$: $\sqrt{(2+4)^2 + (5-2)^2} = \sqrt{45} \approx 6.708$
- $15 \to 14$: $\sqrt{2.5^2 + 1^2} = \sqrt{7.25} \approx 2.693$

- $14 \to 16$: $\sqrt{(5-2.5)^2 + (2-1)^2} = \sqrt{7.25} \approx 2.693$
- $1 \to 3$: $\sqrt{1^2 + (1+2)^2} = \sqrt{10} \approx 3.162$
- $3 \to 15$: $\sqrt{4^2 + 5^2} = \sqrt{41} \approx 6.403$
- $1 \to 9$: $\sqrt{1^2 + (1+2+2+2)^2} = \sqrt{50} \approx 7.071$

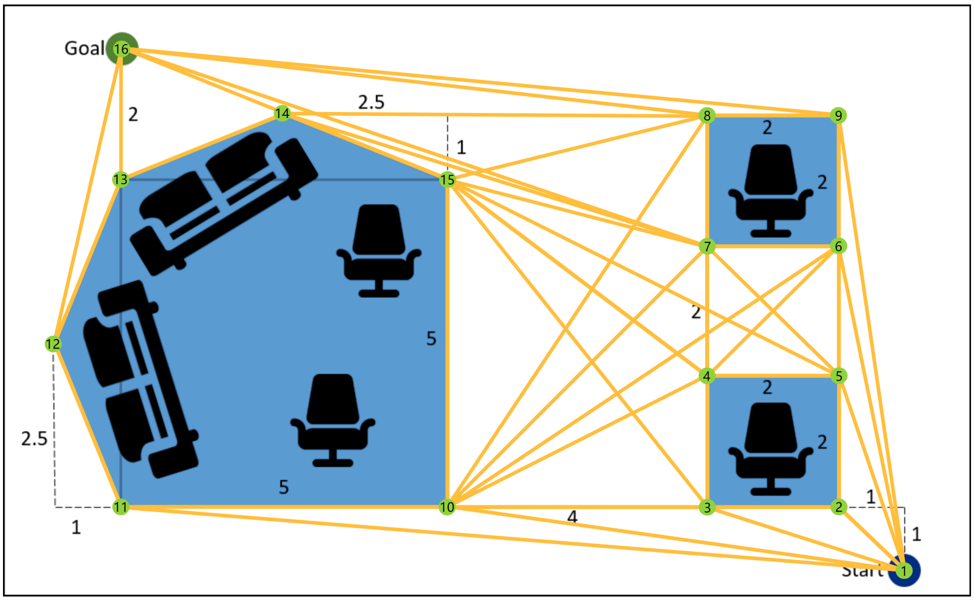

Figure 1: Shortest-path roadmap for Question 2.

By rough observation we can find $1 \to 5 \to 15 \to 14 \to 16$ and $1 \to 3 \to 15 \to 14 \to 16$ are two candidates for the shortest path. Calculating their costs we have:

- $1 \to 5 \to 15 \to 14 \to 16$: $3.162 + 6.708 + 2.693 + 2.693 = 15.256$
- $1 \to 3 \to 15 \to 14 \to 16$: $3.162 + 6.403 + 2.693 + 2.693 = 14.951$

So the shortest path is $1 \to 3 \to 15 \to 14 \to 16$ with a distance of approximately 14.951.

3. Dijkstra and A*

Answer 3.1

The steps for Dijkstra's algorithm are shown in Figure 2.

Visited	Not visited Q (sorted in ascending order of cost)
	Start(., 0), A, B, C, D, E, F, G, H, Goal
Start(., 0)	**A(Start, 1)**, **B(Start, 1)**, **E(Start, 1.414)**, **C(Start, 2)**, D, F, G, H, Goal
Start, A(Start, 1)	B(Start, 1), E(Start, 1.414), C(Start, 2), D, F, G, H, Goal
Start, A, B(Start, 1)	E(Start, 1.414), C(Start, 2), **D(B, 3)**, F, G, H, Goal
Start, A, B, E(Start, 1.414)	C(Start, 2), D(B, 3), **G(E, 3.650)**, F, H, Goal
Start, A, B, E, C(Start, 2)	D(B, 3), **F(C, 3)**, G(E, 3.650), H, Goal
Start, A, B, E, C, D(B, 3)	F(C, 3), G(E, 3.650), **Goal(D, 8)**, H
Start, A, B, E, C, D, F(C, 3)	G(E, 3.650), Goal(D, 8), H
Start, A, B, E, C, D, F, G(E, 3.650)	**H(G, 4.650)**, **Goal(G, 5.886)**
Start, A, B, E, C, D, F, G, H(G, 4.650)	Goal(G, 5.886)

Figure 2: Steps of Dijkstra's algorithm for Question 3.1.

As a result, the shortest path found by Dijkstra's algorithm is Start $\to E \to G \to$ Goal with a cost of approximately 5.886.

Answer 3.2
First we need to define a heuristic function $h(v)$ that estimates the cost from node v to the goal. A common choice is the Euclidean distance between node v and the goal node which is always admissible. The heuristic function values for each node are shown in Table 1.

v	Start	A	B	C	D	E	F	G	H	Goal
$h(v)$	4.000	4.123	4.123	2.000	5.000	3.162	1.000	2.236	2.000	0.000

Table 1: The heuristic function values $h(v)$ for each node v.

The steps for A* algorithm are shown in Figure 3. The calculated cost to arrive at each node is denoted as $g(v)$, the vertices are denoted as $v(\text{parent}, g(v), f(v))$ in the table.

Visited	Not visited Q (sorted in ascending order of $f(v) = g(v) + h(v)$)
	Start(., 0, 4), A, B, C, D, E, F, G, H, Goal
Start(., 4)	**C(Start, 2, 4)**, **E(Start, 1.414, 4.576)**, **A(Start, 1, 5.123)**, **B(Start, 1, 5.123)**, D, F, G, H, Goal
Start, C(Start, 2, 4)	**F(C, 3, 4)**, E(Start, 1.414, 4.576), A(Start, 1, 5.123), B(Start, 1, 5.123), D, G, H, Goal
Start, C, F(C, 3, 4)	E(Start, 1.414, 4.576), A(Start, 1, 5.123), B(Start, 1, 5.123), **G(F, 5, 7.236)**, D, H, Goal
Start, C, F, E(Start, 1.414, 4.576)	A(Start, 1, 5.123), B(Start, 1, 5.123), **G(E, 3.650, 5.886)**, D, H, Goal
Start, C, F, E, A(Start, 1, 5.123)	B(Start, 1, 5.123), G(E, 3.650, 5.886), D, H, Goal
Start, C, F, E, A, B(Start, 1, 5.123)	G(E, 3.650, 5.886), **D(B, 3, 8)**, H, Goal
Start, C, F, E, A, B, G(E, 3.650, 5.886)	**Goal(5.886, 5.886)**, **H(G, 4.650, 6.650)**, D(B, 3, 8)

Figure 3: Steps of A* algorithm for Question 3.2.

Finally the path found by A* algorithm is Start $\to E \to G \to$ Goal with a cost of approximately 5.886.

Answer 3.3
The fundamental difference between Dijkstra's algorithm and A* algorithm lies in how they sort the open set (the set of nodes to be explored) Q. Dijkstra's algorithm sorts Q based on the actual cost from the Start node to the current node $g(n)$, while A* algorithm sorts Q based on the estimated total cost from the Start node to the Goal node through the current node $f(n) = g(n) + h(n)$, by introducing a heuristic function $h(n)$ that estimates the cost from the current node to the Goal node.

This difference affects their performance in terms of efficiency and optimality, i.e., sometimes A* can find the path more efficiently than Dijkstra's algorithm, especially when a good heuristic is used. Both algorithms are guaranteed to find the optimal path if the heuristic used in A* is admissible (never overestimates the true cost to reach the goal).

4. Dijkstra

Answer 4.1
Similar to Question 3.1, the steps for Dijkstra's algorithm are shown in Figure 4.

Visited	Not visited Q (sorted in ascending order of cost)
	S(., 0), A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, GO
S(., 0)	**B(S, 1)**, **A(S, 3)**, **D(S, 4)**, C, E, F, G, H, I, J, K, L, M, N, O, GO
S, B(S, 1)	**H(B, 2)**, A(S, 3), D(S, 4), C, E(B, 7), F, G, I, J, K, L, M, N, O, GO
S, B, H(B, 2)	A(S, 3), D(S, 4), **O(H, 4)**, **G(H, 6)**, E(B, 7), I(H, 8), C, F, J, K, L, M, N, GO
S, B, H, A(S, 3)	D(S, 4), O(H, 4), **C(A, 5)**, G(H, 6), E(B, 7), I(H, 8), F, J, K, L, M, N, GO
S, B, H, A, D(S, 4)	O(H, 4), C(A, 5), **L(D, 6)**, G(H, 6), E(B, 7), I(H, 8), F, J, K, M, N, GO
S, B, H, A, D, O(H, 4)	C(A, 5), **L(D/O, 6)**, G(H, 6), E(B, 7), I(H, 8), F, J, K, M, N, GO
S, B, H, A, D, O, C(A, 5)	L(D/O, 6), **F(C, 6)**, G(H, 6), E(B, 7), I(H, 8), **GO(C, 14)**, J, K, M, N
S, B, H, A, D, O, C, L(D/O, 6)	F(C, 6), G(H, 6), E(B, 7), I(H, 8), GO(C, 14), J, K, M, N
S, B, H, A, D, O, C, L, F(C, 6)	G(H, 6), E(B, 7), I(H, 8), GO(C, 14), J, K, M, N
S, B, H, A, D, O, C, L, F, G(H, 6)	E(B, 7), I(H, 8), **K(G, 9)**, GO(C, 14), J, M, N
S, B, H, A, D, O, C, L, F, G, E(B, 7)	I(H, 8), K(G, 9), **J(E, 11)**, GO(C, 14), M, N
S, B, H, A, D, O, C, L, F, G, E, I(H, 8)	K(G, 9), **N(K, 10)**, J(E, 11), GO(C, 14), M
S, B, H, A, D, O, C, L, F, G, E, I, K(G, 9)	**N(K, 10)**, J(E, 11), **M(N, 12)**, GO(C, 14)
S, B, H, A, D, O, C, L, F, G, E, I, K, N(K, 10)	J(E, 11), M(N, 12), GO(C, 14)
S, B, H, A, D, O, C, L, F, G, E, I, K, N, J(E, 11)	M(N, 12), **GO(C/J, 14)**
S, B, H, A, D, O, C, L, F, G, E, I, K, N, J, M(N, 12)	**GO(C/J/M, 14)**

Figure 4: Steps of Dijkstra's algorithm for Question 4.1.

As a result, there are several shortest paths found by Dijkstra's algorithm, one of which is $S \to A \to C \to GO$ with a cost of 14.

Answer 4.2

The screenshot of the implemented part and the result is shown below:

Your contribution

Please implement the following method to obtain the node with the lowest f_score. Remember that f_score is defined as

$$f(n) = g(n) + h(n)$$

where h(n) estimates the cost to reach goal from node n.

```python
def get_node_with_lowest_f_score(open_set, g, stop_node):
    lowest_f_score = float('inf')
    lowest_node = None
    for node in open_set:
        f_score = g[node] + heuristic(node, stop_node)
        if f_score < lowest_f_score:
            lowest_f_score = f_score
            lowest_node = node

    return lowest_node
```

[9] ✓ 0.0s Python

Testing your implementation

Now you should get the optimal path for the graph above. Please add the obtained path in your deliverable along with the cost associated.

```python
path = aStarAlgo(start_node, stop_node, heuristic)
if path:
    print('Shortest Path:', path)
else:
    print('No path found.')
```

[10] ✓ 0.0s Python

```
... Shortest Path: ['S', 'A', 'C', 'GO']
```