Solve the Deterministic Shortest Path Problem using Dynamic Programming

Jingpei Lu
Jacobs School of Engineering
University of California, San Diego
jil360@ucsd.edu

I. Problem Formulation

A. *Deterministic Shortest Path (DSP) Problem*

Consider a graph with a finite vertex space $\mathcal{V}$ and a weighted edge space

$$C := \left\{(i, j, c_{ij}) \in \mathcal{V} \times \mathcal{V} \times \mathbb{R} \cup \{\infty\}\right\} \tag{1}$$

where c_{ij} denotes the arc length or cost from vertex i to vertex j. A path from a start node $s \in \mathcal{V}$ to an end node $\tau \in \mathcal{V}$ can be represented by an ordered list $Q := (i_1, i_2, \ldots, i_q)$ of nodes $i_k \in \mathcal{V}$. The length of the path can be represented as

$$J^Q = \sum_{t=1}^{q-1} c_{t,t+1} \tag{2}$$

Our objective is to find the shortest path

$$Q^* = \underset{Q \in \mathbb{Q}}{\operatorname{argmin}} J^Q \tag{3}$$

that has the smallest length from a start node $s \in \mathcal{V}$ to an end node $\tau \in \mathcal{V}$.

B. *Formulate the DSP Problem as a Markov Decision Process (MDP)*

To model this problem as a Markov Decision Process, we should clearly define the states space $\mathcal{X}$, control spaces $\mathcal{U}$, the initial state x_o, the motion model $p_f(\cdot\,|x_t, u_t)$, the planning horizon T, the stage cost $l(x_t, u_t)$ and terminal costs $q(x_T)$. One way to define the Markov Decision Process is to use the vertex space $\mathcal{V}$ as our state space $\mathcal{X}$, and the arc length c to be our cost from one state to another. So, we can define the MDP as

- The state is $x_t \in \mathcal{X} := \{i \in \mathcal{V}\}$, where $\mathcal{V}$ is a set of vertexes on the graph.
- The control is $u_t = x_{t+1}$ for $u_t \in \mathcal{U}$,which is the movement of going from one vertex to another connected vertex.
- The initial state x_o can be defined as the start node s and the terminal state x_T is the end node τ.
- The planning horizon T is $|\mathcal{V}| - 1$ where $|\mathcal{V}|$ is the number of nodes in the graph.
- Since the graph is deterministic, so our motion model $p_f(x_{t+1}|x_t, u_t)$ should either be 1 or 0 depends on which control we choose to go with.
- The stage costs are the arc lengths between two vertexes. For example, if we are moving from $x_t = i$ to $x_{t+1} = j$, then $l(x_t, x_{t+1}) = c_{ij} \in \mathbb{R} \cup \{\infty\}$.
- The terminal cost $q(x_T) = 0$.

Therefore, finding the optimal policy

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmin}} V^\pi(x_o) \tag{4}$$

is equivalent to find the shortest path in the DSP problem as described in equation (3), where a policy π is equivalent to a path Q and the value function V^π is equivalent to the length of the path J^Q.

II. Technical Approach

Our dataset contains the number of nodes in the graph $n = |\mathcal{V}|$, the start node s, the goal node τ, and a matrix $C \in \mathbb{R}^{n \times n}$ specifying the cost c_{ij} of transitioning from node i to node j. If a transition is not possible, $c_{ij} = \infty$. We also have a strong assumption that there are no negative cycles in the graph and $c_{ii} = 0$ for all $i \in \mathcal{V}$.

A. *Label Correcting Algorithm*

The label correcting (LC) algorithm is a general algorithm for Shortest Path (SP) problems that does not necessarily visit every node of the graph. The LC algorithm prioritizes the visited nodes using the cost-to-arrive values. It introduces a concept of label g_i that keeps (an estimate of) the lowest cost from start node s to each visited node $i \in \mathcal{V}$. Each time g_i is reduced, the labels g_j of the children of i can be corrected as

$$g_j = g_i + c_{ij} \tag{5}$$

The LC algorithm also introduces a concept of OPEN list which stores a set of nodes that can potentially be part of the shortest path to τ. The implementation of the algorithm is demonstrated in the figure below.

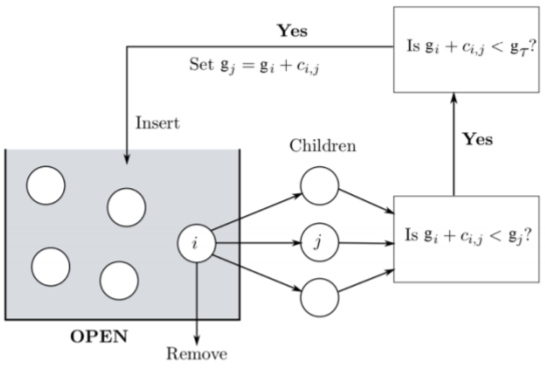
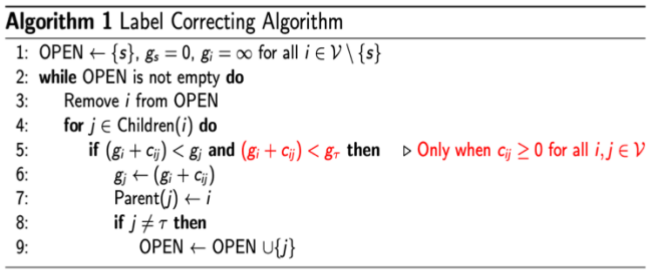

Yes
Set $g_j = g_i + c_{i,j}$
Is $g_i + c_{i,j} < g_T$?
Insert
Children
Yes
i
j
Is $g_i + c_{i,j} < g_j$?
OPEN
Remove

Figure 1

We start with the start node s. We set $g_s = 0$ and $g_i = \infty$ for all $i \in \mathcal{V} \backslash \{s\}$ and put s into the OPEN list. Every time we remove a node i from the OPEN list, we check two conditions of all its children as described in the Figure 1. If a node satisfied these two conditions, we insert this node to the OPEN list. We iterate this process until the OPEN list is empty.

The pseudocode implementation for the LC algorithm is described on the figure below.



```
Algorithm 1 Label Correcting Algorithm
1: OPEN ← $\{s\}$, $g_s = 0$, $g_i = \infty$ for all $i \in \mathcal{V} \setminus \{s\}$
2: while OPEN is not empty do
3:     Remove $i$ from OPEN
4:     for $j \in$ Children($i$) do
5:         if $(g_i + c_{ij}) < g_j$ and $(g_i + c_{ij}) < g_\tau$ then    ▷ Only when $c_{ij} \geq 0$ for all $i, j \in \mathcal{V}$
6:             $g_j \leftarrow (g_i + c_{ij})$
7:             Parent($j$) ← $i$
8:             if $j \neq \tau$ then
9:                 OPEN ← OPEN ∪$\{j\}$
```

Figure 2

If there exists at least one finite cost path from s to τ, then the Labe Correcting (LC) algorithm terminates with g_τ equal to the shortest path from s to τ. Otherwise, the LC algorithm terminates with $g_\tau = \infty$.

The optimal g-value satisfy the equation.

$$g_i = \min_{j \in \text{Parent(i)}} g_j + c_{ji} \tag{6}$$

Therefore, once the g-value is available, the least cost path $Q^* = q_0, q_1, \dots, q_\tau$ is greedy path computed starting from $q_T = \tau$ and back tracking using

$$q_i = \min_{j \in \text{Parent}(q_{t+1})} g_j + c_{j,q_{t+1}} \tag{7}$$

for $t = T - 1, \dots, 0$.

III. Result

The LC algorithm can output different shortest paths depending on the implementation of the OPEN list. For my implementation, I implemented the OPEN list as a stack. I ran my implementation on six given datasets and obtained the results as below.

Dataset 1

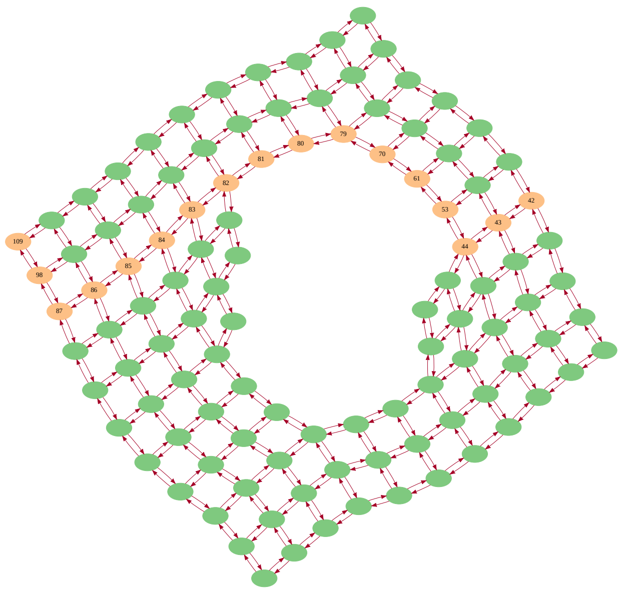


Figure 3

Minimum cost path: 42 -> 43 -> 44 -> 53 -> 61 -> 70 -> 79 -> 80 -> 81 -> 82 -> 83 -> 84 -> 85 -> 86 -> 87 -> 98 -> 109
Optimal cost-to-go values: 0.00 -> 1.00 -> 2.00 -> 3.00 -> 4.00 -> 5.00 -> 6.00 -> 7.00 -> 8.00 -> 9.00 -> 10.00 -> 11.00 -> 12.00 -> 13.00 -> 14.00 -> 15.00 -> 16.00

Dataset 2

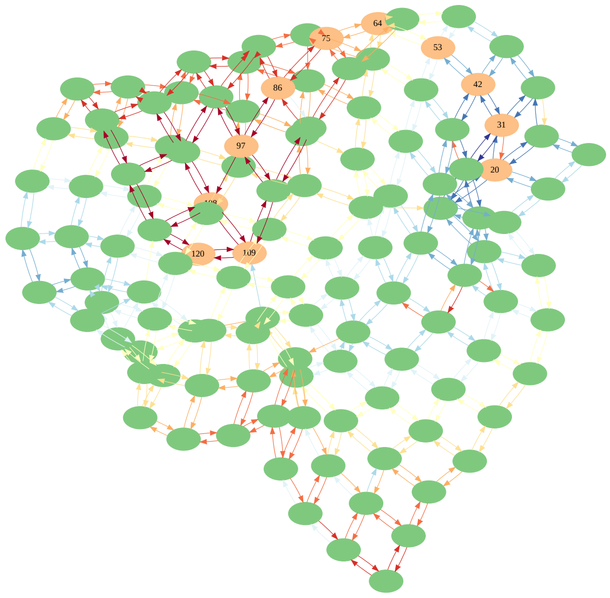


Figure 4

Minimum cost path: 20 -> 31 -> 42 -> 53 -> 64 -> 75 -> 86 -> 97 -> 108 -> 109 -> 120
Optimal cost-to-go values: 0.00 -> 16.00 -> 32.00 -> 46.00 -> 55.00 -> 61.00 -> 64.00 -> 66.00 -> 66.00 -> 67.00 -> 67.00

Dataset 3

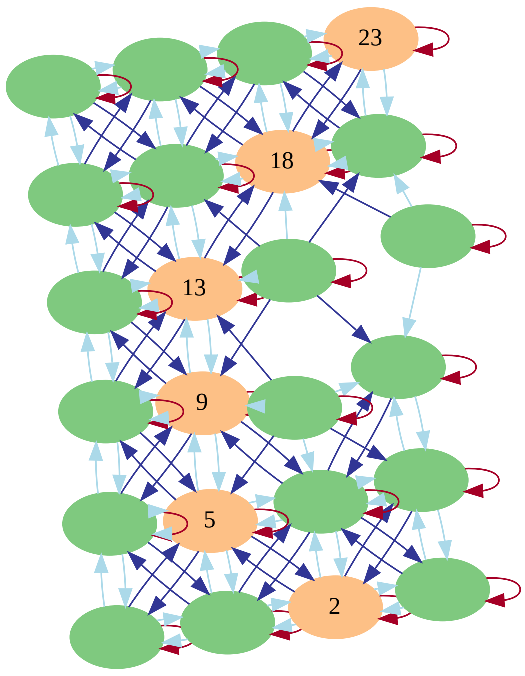


Figure 5

Minimum cost path: 2 -> 5 -> 9 -> 13 -> 18 -> 23
Optimal cost-to-go values: 0.00 -> 1.00 -> 2.00 -> 3.00 -> 4.00 -> 6.00

Dataset 4

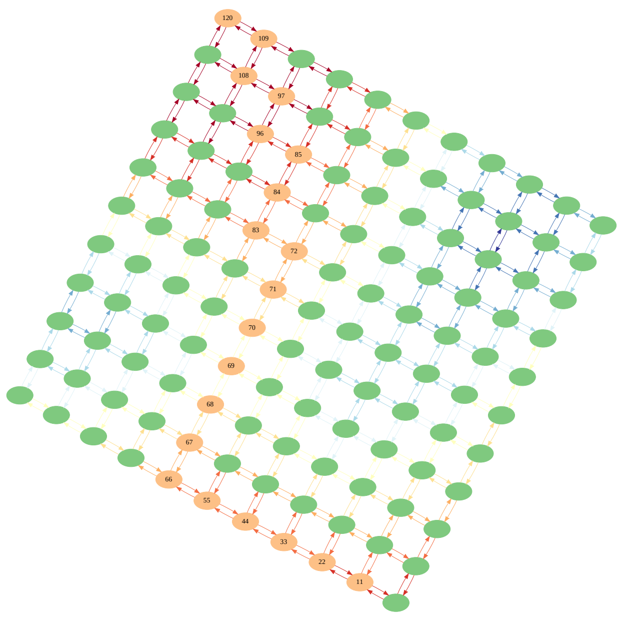


Figure 6

Minimum cost path: 11 -> 22 -> 33 -> 44 -> 55 -> 66 -> 67 -> 68 -> 69 -> 70 -> 71 -> 72 -> 83 -> 84 -> 85 -> 96 -> 97 -> 108 -> 109 -> 120
Optimal cost-to-go values: 0.00 -> 3.00 -> 6.00 -> 10.00 -> 14.00 -> 18.00 -> 24.00 -> 32.00 -> 41.00 -> 50.00 -> 58.00 -> 64.00 -> 70.00 -> 74.00 -> 77.00 -> 79.00 -> 80.00 -> 81.00 -> 82.00 -> 82.00

Dataset 5

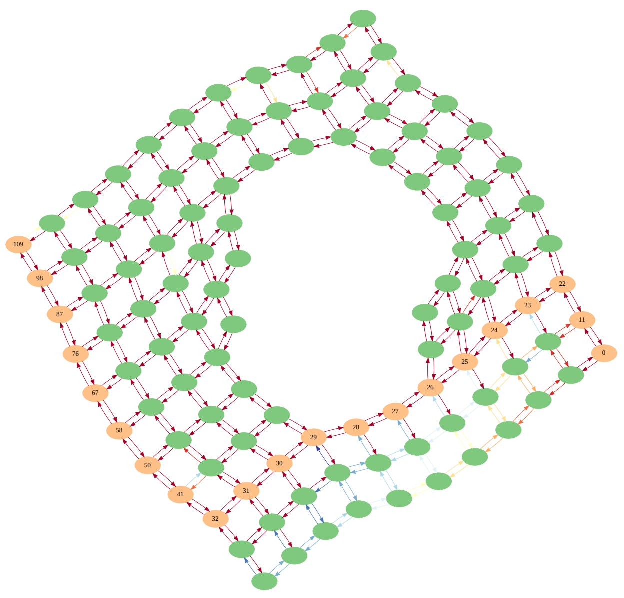


Figure 7

Minimum cost path: 0 -> 11 -> 22 -> 23 -> 24 -> 25 -> 26 -> 27 -> 28 -> 29 -> 30 -> 31 -> 32 -> 41 -> 50 -> 58 -> 67 -> 76 -> 87 -> 98 -> 109
Optimal cost-to-go values: 0.00 -> 2.00 -> 5.00 -> 7.00 -> 9.00 -> 11.00 -> 13.00 -> 15.00 -> 17.00 -> 19.00 -> 21.00 -> 23.00 -> 25.00 -> 27.00 -> 29.00 -> 31.00 -> 33.00 -> 35.00 -> 37.00 -> 39.00 -> 41.00

Dataset 6

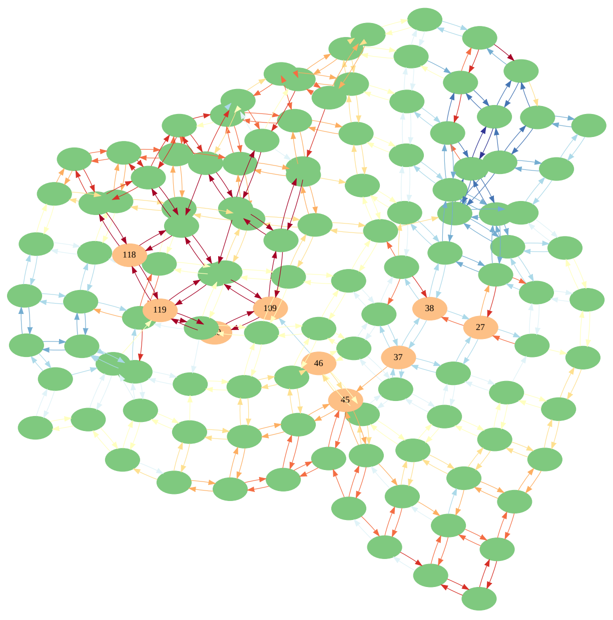


Figure 8

Minimum cost path: 27 -> 38 -> 37 -> 45 -> 46 -> 109 -> 120 -> 119 -> 118
Optimal cost-to-go values: 0.00 -> 4.00 -> 16.00 -> 22.00 -> 29.00 -> 41.00 -> 42.00 -> 42.00 -> 43.00