Report for Problem 4

Yiran Xu

04/28/2019

1 Problem Formulation

We are asked to solve a deterministic shortest path problem. This problem can be solved by many approaches. A direct thought is that we can use a Markov Decision Process (MDP) to describe the problem.
Specifically, for this deterministic shortest path problem, our MDP is:

- Vertex space $\mathcal{V}$: including all possible positions.
- State space: The initial state x_0 is the start point s. The terminal state $x_T = \tau$. $x_t \in \mathcal{X}$, which is the position at time t for $t = 1, ..., T-1$.
- Control space: In this case, the control input u_t is which next state we decide to move. Then the control space $\mathcal{U}_{T-1} = \{\tau\}$ and $\mathcal{U}_t = \mathcal{V} \backslash \{\tau\}$.
- Motion Model: Given x_t and u_t, x_{t+1} is one adjacent point to x_t corresponding with u_t, i.e. $x_{t+1} = u_t$.
- Planning horizon: $T = |\mathcal{V}| - 1$
- The stage and the terminal costs: terminal cost $\mathfrak{q}(x) = 0$ and stage cost $l(x, u) = c_{x_t, u_t}$, where c_{x_t, u_t} is the arc length between x_t and $x_{t+1} = u_t$.

Usually, a node j is called a child of i if there is an edge from i to j and i is called a parent of j. Our goal is to find a shortest path from s to τ.

2 Technical Approach

There are plentiful approaches for the deterministic shortest path problem: Dynamic Programming (DP), Forward DP, Label Correcting (LC) algorithm, A* algorithm, etc. In this report, Label Correcting is implemented.

2.1 Label Correcting Algorithm

The idea of LC is to discover shorter path from the origin s to every other node i progressively. This is acquired by using a label g_i to keep track of the lowest cost from the origin node s. Each node i has a label g_i which represents an upper bound on the shortest path from s to i.
As the algorithm progresses, if a path from s to i is discovered and whose distance(cost) is smaller than g_i, then the label of i will be corrected as this new distance of the path. In order to manage all the nodes, the algorithm maintains a list of nodes called OPEN. OPEN includes the possible candidates to form the shortest path and will be examined as the process. When the OPEN list is empty, the algorithm terminates. The OPEN usually begins with the origin node s.
The pseudo-code of LC algorithm is shown in Algorithm 1.

Algorithm 1: Label Correcting Algorithm

```
Result: Parents
OPEN ← $\{s\}$, $g_s = 0$, $g_i = \infty$ for all $i \in \mathcal{V}\backslash\{s\}$ ;
while OPEN is not empty do
    Remove $i$ from OPEN;
    for $j \in$ Children(i) do
        if $(g_i + C_{i,j}) < g_j$ and $(g_i + C_{i,j}) < g_\tau$ then
            $g_j \leftarrow (g_i + C_{i,j})$
            Parent($j$) ← $i$
            if $j \neq \tau$ then
                OPEN ← OPEN ∪ $\{j\}$
            end
        end
    end
end
```

In "if" condition we want to check if the path from s to j through i is shorter than previous path from s to j (i.e. if $g_i + C_{i,j} < g_j$) and if it has a chance of leading to the shortest path to τ (i.e. if $g_i + C_{i,j} < g_\tau$). If so, then label g_i will be reduced and node j is placed into the OPEN list so that paths passing through j and reaching the children of j could be examined later. On the other hand, a node j will not enter OPEN list if it does not satisfies the condition. Hence, the number of nodes that enter the OPEN list can be pretty smaller than the total number of nodes. In particular, LC algorithm can be much more efficient than pure DP algorithm.

3 Results

3.1 Experiment Description

The data in this problem are six situations, including start point $s = x_0$, the terminal point $x_T = \tau$, cost matrix $\mathbf{C}$ with $C_{i,j}$ representing the cost from node i to node j.
The input of each instance includes the start point s, the terminal point τ, cost matrix $\mathbf{C}$ and the number of nodes n. The output via LC is the shortest path and the optimal cost-to-go values $(V^*(x_0), ..., V^*(x_T))$ along the path. The results will be shown in Section 3.

3.2 Results

There are totally 6 instances. The results of these instances are shown in Fig.1 and Table 1 and Table 2.

Table 1: Results of Paths

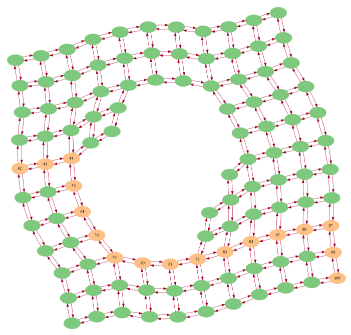
Instances	Paths
1	42-43-44-53-61-70-79-80-81-82-83-84-85-86-87-98-109
2	20-31-42-53-64-75-86-97-108-109-120
3	2-6-9-13-18-23
4	11-22-33-44-55-66-67-68-69-70-71-72-83-84-85-96-97-108-109-120
5	0-11-22-23-24-25-26-27-28-29-30-31-32-41-50-58-67-76-87-98-109
6	27-38-37-45-46-109-120-119-118

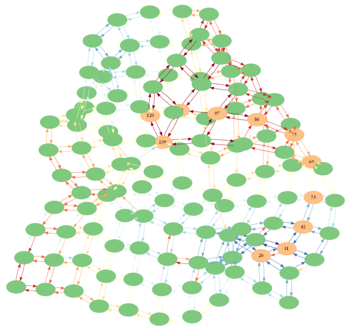
Table 2: Results of Cost Values

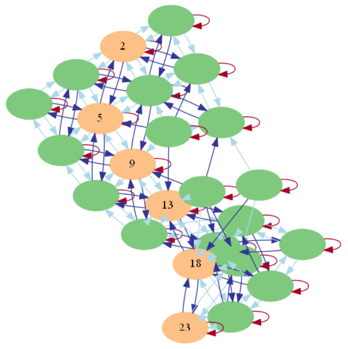
Instances	Cost-to-go Values
1	16.00 15.00 14.00 13.00 12.00 11.00 10.00 9.00 8.00 7.00 6.00 5.00 4.00 3.00 2.00 1.00 0.00
2	67.7 51.1 35.0 21.7 12.1 6.3 3.1 1.6 0.7 0.3 0.00
3	6.24 5.24 3.83 2.83 1.41 0.00
4	82.52 79.31 75.63 71.83 67.97 63.69 57.90 50.22 41.35 32.40 24.33 17.59 12.19 8.04 4.99 2.91 1.55 0.73 0.25 0.00
5	41.66 39.21 36.00 34.00 32.00 30.00 28.00 26.00 24.00 22.00 20.00 18.00 16.00 14.00 12.00 10.00 8.00 6.00 4.00 2.00 0.00
6	43.18 39.14 26.51 20.57 13.21 1.22 0.97 0.69 0.00

3.3 Discussion

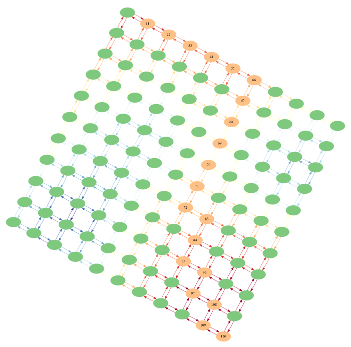
As a matter of fact, there is not only "shortest" path for each instance. For example, in Instance #1 and Instance #5, as shown in Fig.2 and Fig. 3, both

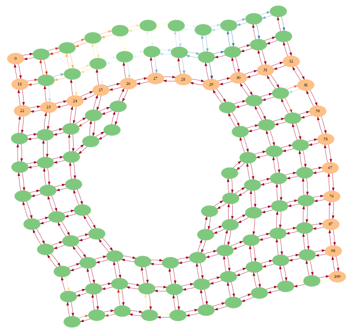

(a) Instance #1


(b) Instance #2



(c) Instance #3


(d) Instance #4


(e) Instance #5

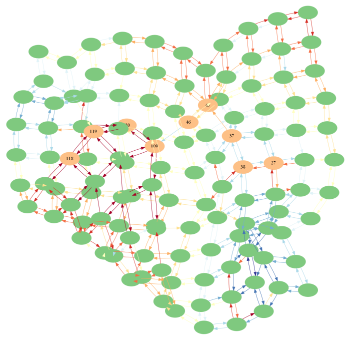
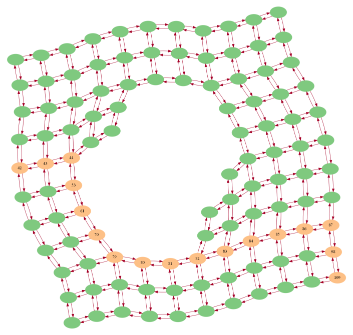

(f) Instance #6

Figure 1: The Optimal Paths

two paths in each instance cost the same values (distances). The difference is when we implement LC algorithm, which element in the list should be removed. In Fig. 2 and Fig. 3, (a) is removing the last element in the list while (b) is removing the first one.

This implies LC will only return one of the shortest paths for every situation. This also indicates that the implementation of LC algorithm is correct since two paths lead to one same cost.


(a) The Optimal Path 1

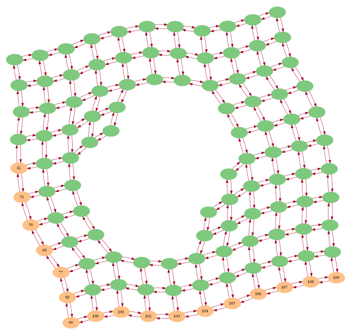
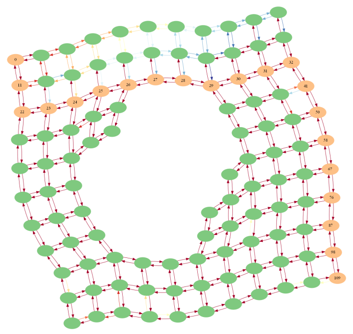

(b) The Optimal Path 2

Figure 2: Multiple Shortest Paths, Instance #1


(a) The Optimal Path 1

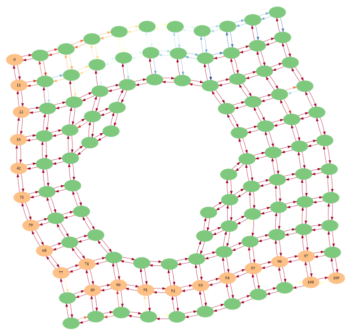

(b) The Optimal Path 2

Figure 3: Multiple Shortest Paths, Instance #5