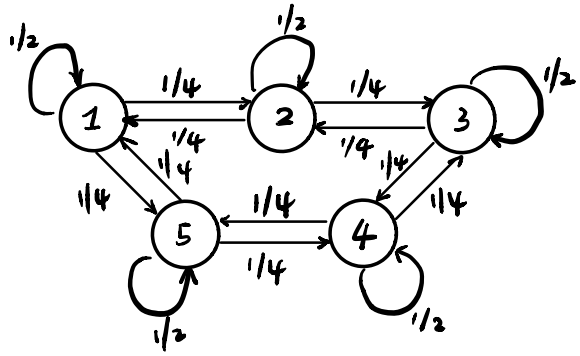
1.

(a) The state trasition diagram is as followed:



It is irreducible since we can access to every state from every state.

It is not periodic since $P_{jj} \neq 0$. For instance, if we start Markov Chain with $X_0 = 1$, $P_{0|0}^{(1)} = 1/2$. It is possible that the state keeps 1:

X	1	1	1	1	⋯
t	0	1	2	3	⋯

or

X	1	2	1	2	⋯
t	0	1	2	3	⋯

Hence this Markov Chain is not periodic.

(b) $P_{100} = P_0 P^{100}$.

According to eigenvalue decomposition. $P = VDV^{-1}$

where $D = \frac{1}{8}\,\mathrm{diag}(3-\sqrt{5},\ 3-\sqrt{5},\ 3+\sqrt{5},\ 3+\sqrt{5},\ 8)$

$$V = \begin{bmatrix} -1 & \frac{1}{2}(-1-\sqrt{5}) & -1 & \frac{1}{2}(-1+\sqrt{5}) & 1 \\ \frac{1}{2}(1+\sqrt{5}) & \frac{1}{2}(1+\sqrt{5}) & \frac{1}{2}(1-\sqrt{5}) & \frac{1}{2}(1-\sqrt{5}) & 1 \\ \frac{1}{2}(-1-\sqrt{5}) & -1 & \frac{1}{2}(-1+\sqrt{5}) & -1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$V^{-1} = \frac{1}{10}\begin{bmatrix} -1+\sqrt{5} & -1+\sqrt{5} & -1-\sqrt{5} & 4 & -1-\sqrt{5} \\ -1-\sqrt{5} & -1+\sqrt{5} & -1+\sqrt{5} & -1-\sqrt{5} & 4 \\ -1-\sqrt{5} & -1-\sqrt{5} & -1+\sqrt{5} & 4 & -1+\sqrt{5} \\ -1+\sqrt{5} & -1-\sqrt{5} & -1-\sqrt{5} & -1-\sqrt{5} & 4 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Then $P^n = (VDV^{-1})(VDV^{-1})\cdots(VDV^{-1})$

$= VD^nV^{-1}$

Since $\frac{1}{8}(3-\sqrt{5}) < 1$, $\frac{1}{8}(3+\sqrt{5}) < 1$,

$\lim_{n\to\infty} P^n = V \cdot \mathrm{diag}(0,0,0,0,1)\, V^{-1} = \frac{1}{5}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$

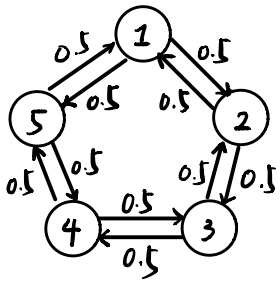
Thus $p_\infty^T = p_0^T \lim_{t\to\infty} P^t = p_0^T \cdot \frac{1}{5}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \left[\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right]$

If we simulate p for 100 times, the simulated value is

$$p_{100} = [0.2000,\ 0.2000,\ 0.2000,\ 0.2000,\ 0.2000]^T$$

which is close to p_∞.

(c) Consider this situation as a Markov Chain with 5 states distributed as $p_0 = [25\ \ 20\ \ 35\ \ 24\ \ 46]^T$.



Consider that talking to neighbors as a trasition between 2 states. Then, based on the observation of (a) & (c), we can estimate the average age:

(1) Randomly start from one person.

(2) Let person talk to one of his/her neighbors to tell him/her the current

0
iterations and all the ages he/she knows.

(3) Repeat (2) until $\frac{\text{all the ages}}{\#\text{ iterations}} < \epsilon$

By doing this, we're actually simulating the process of (a) and (b), just changing the probability to actual ages.

Prob1_sim

April 30, 2019

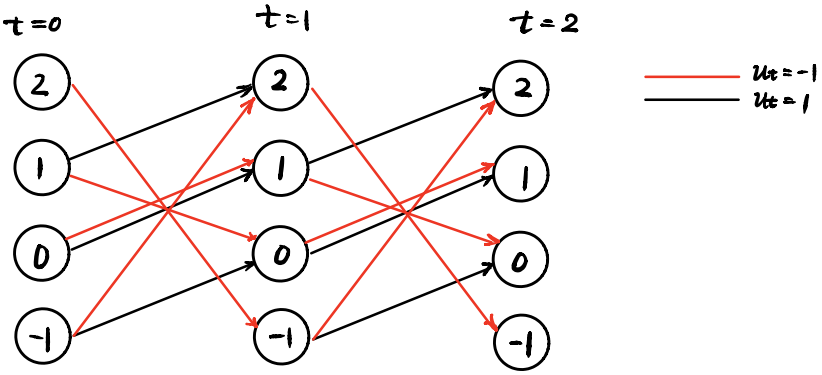
This the program for simulation.

In [23]:
```
import numpy as np

        P = np.array([[1/2, 1/4, 0, 0, 1/4],
                      [1/4, 1/2, 1/4, 0, 0],
                      [0, 1/4, 1/2, 1/4, 0],
                      [0, 0, 1/4, 1/2, 1/4],
                      [1/4, 0, 0, 1/4, 1/2]])
        P = np.mat(P) # use matrix for power calculation
        p_0 = np.array([[25/150, 20/150, 35/150, 24/150, 46/150]]) # prior
        np.set_printoptions(precision=5)
        p_100 = p_0 @ (P**100)
        print(p_100)
```

```
[[0.2 0.2 0.2 0.2 0.2]]
```

2. State: $x_t \in \{-1, 0, 1, 2\}$, control input: $u_t \in \{-1, 1\}$

Motion model: $x_{t+1} = x_t u_t + u_t^2 = x_t u_t + 1$



(a) Cost: stage cost $l_t(x_t, u_t) = x_t u_t$, terminal cost $q(x) = x^2$.

When $t = T = 2$, $V_2 = q(x_2) = \begin{cases} 1, & x_2 = \pm 1 \\ 0, & x_2 = 0 \\ 4, & x_2 = 2 \end{cases}$

For all $x_t \in \{-1, 0, 1, 2\}$ and $t = 1, 0$:

$$V_t^*(x_t) = \min_{u_t \in \{-1,1\}} \{ l_t(x_t, u_t = -1) + V_{t+1}^{u_t=-1}(x_{t+1}), l_t(x_t, u_t = 1) + V_{t+1}^{u_t=1}(x_{t+1}) \}$$

$$= \min_{u_t \in \{-1,1\}} \{ -x_t + V_{t+1}^{u_t=-1}(x_{t+1}), x_t + V_{t+1}^{u_t=1}(x_{t+1}) \}.$$

For $t = 1$.

if $x_1 = 2$. $V_1^*(2) = \min_{u_t \in \{-1,1\}} \{ -2 + V_2(-1) \}$

$= -2 + 1 = -1$

$\pi_1^*(2) = -1$

if $x_1 = 1$. $V_1^*(1) = \min_{u_t \in \{-1,1\}} \{ -1 + V_2(0), 1 + V_2(2) \}$

$= \min \{ -1, 5 \} = -1$

$\pi_1^*(1) = -1$

if $x_1 = 0$. $V_1^*(0) = \min_{u_t \in \{-1,1\}} \{ V_2(1), V_2(1) \} = 1$

$\pi_1^*(0) = -1$ or 1

if $x_1 = -1$, $V_1^*(-1) = \min_{u_t \in \{-1,1\}} \{1 + V_2(2), -1 + V_2(0)\}$

$= \min\{5, -1\} = -1$

$\pi_1^*(-1) = 1$

For $t = 0$,

if $x_0 = 2$, $V_0^*(2) = \min_{u_t \in \{-1,1\}} \{-2 + V_1(-1)\}$

$= -3$

$\pi_0^*(2) = -1$

if $x_0 = 1$, $V_0^*(1) = \min_{u_t \in \{-1,1\}} \{-1 + V_1(0), 1 + V_1(2)\}$

$= \min_{u_t \in \{-1,1\}} \{-1 + 1, 1 - 1\} = 0$

$\pi_0^*(1) = -1$

if $x_0 = 0$, $V_0^*(0) = \min_{u_t \in \{-1,1\}} \{V_1(1), V_1(1)\}$

$= V_1(1) = -1$

$\pi_0^*(0) = -1$ or 1

if $x_0 = -1$, $V_0^*(-1) = \min_{u_t \in \{-1,1\}} \{1 + V_1(2), -1 + V_1(0)\}$

$= \min\{1 + (-1), -1 + 1\} = 0$

$\pi_0^*(-1) = -1$ or 1

(b) When $x_0 = 2$, $x_1 = -1$ with $u_0 = -1$

cost $\lambda_0(x_0 = 2, u_0 = -1) = l_0(x_0, u_0) = -2$.

$x_1 = -1$, based on (a),

$\pi_1^*(-1) = 1$, i.e. $u_1 = 1$. then

cost $\lambda_1(x_1 = -1, u_1 = 1) = \lambda_0 + l_1(x_1, u_1)$

$= (-2) + (-1) \times 1$

$= -3$

and $x_2 = 0$, with terminal cost $q(0) = 0$.

Hence, the optimal cost is -3,

control sequences are $[u_0, u_1] = [-1, 1]$

state trajectory is $[x_0, x_1, x_2] = [2, -1, 0]$.

3. (a) Consider to use Mathematical Induction to prove the optimal value function $V_t^*(x)$ is the minimum of 2^{T-t} p.d. quadratic functions.

① $t = T$, $2^{T-t} = 2^0 = 1$

$$V_T^*(x_T) = q(x_T) = \frac{1}{2} x_T^T x_T = \min\left\{\frac{1}{2} x_T^T x_T\right\}$$

is the minimum of $2^{T-t} = 1$ p.d. quadratic function.

② When $t = k$. Suppose that $V_k^*(x_k)$ is the minimum of 2^{T-k} p.d. quadratic functions.

Then $t = k-1$.

$$V_{k-1}^*(x_{k-1}) = \frac{1}{2} x_{k-1}^T x_{k-1} + \min\left\{V_k(Ax_{k-1}), V_k(Bx_{k-1})\right\}$$

Because $V_k(Ax_{k-1})$ and $V_k(Bx_{k-1})$ are the minimum of 2^{T-k} p.d. quadratic functions (according to assumption).

i.e. $V_k = \min_i \{x_{k-1}^T C_i x_{k-1}\}$, $C_i \in \mathbb{R}^{n \times n}$

Then $V_{k-1}^*(x_{k-1}) = \min_i \left\{ x_{k-1}^T \left(\frac{1}{2} I + C_i\right) x_{k-1} \right\}$. with $\frac{1}{2} I + C_i > 0$.

quadratic functions $= 2^{T-k} \cdot 2 = 2^{T-(k-1)}$

Thus, $t = k-1$. $V_{k-1}^*(x_{k-1})$ is the minimum of $2^{T-(k-1)}$ p.d. quadratic functions.

Based on ①②. $V_t^*(x)$ is the minimum of 2^{T-t} p.d. quadratic functions.

(b) See below.

ECE276B-HW1-p3

Problem 3(b)

When T = 3, $A = \begin{bmatrix} 0.75 & -1 \\ 1 & 0.75 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$, we can consider the problem based on (a).

The optimal value function

$$V_t^*(\mathbf{x}_t) = min\{l(\mathbf{x}_t, u_t) + V_{t+1}^*(A\mathbf{x}_{t+1}), l(\mathbf{x}_t, u_t) + V_{t+1}^*(B\mathbf{x}_{t+1})\}, t = 0, 1, ..., T-1$$

$$V_T^*(\mathbf{x}) = l(\mathbf{x}_T) = \frac{1}{2}\mathbf{x}_T^T\mathbf{x}_T, t = T$$

Thus, the dynamic programming actually can be realized by a recursive function, which is Optimal_Search(x, T, t) below.

The code is shown as following:

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        from matplotlib import cm

        # Some useful functions
        def motion_model(xt, ut):
            """
                Inputs:
                    xt - current state
                    ut - control input.
                        ut = 0 <- policy 1
                        ut = 1 <- policy 2
                Output:
                    xt - next state
            """

            if ut == 0:
                A = np.array([[.75, -1],
                              [1, .75]])
                xt = A @ xt
            else:
                B = np.array([[1, .5],
                              [.5, .5]])
                xt = B @ xt
```

```
        return xt

    def ComputeCost(x):
        """
            Compute cost
        """
        x = x.reshape(2, 1)
        return 1/2 * (x.T @ x)
```

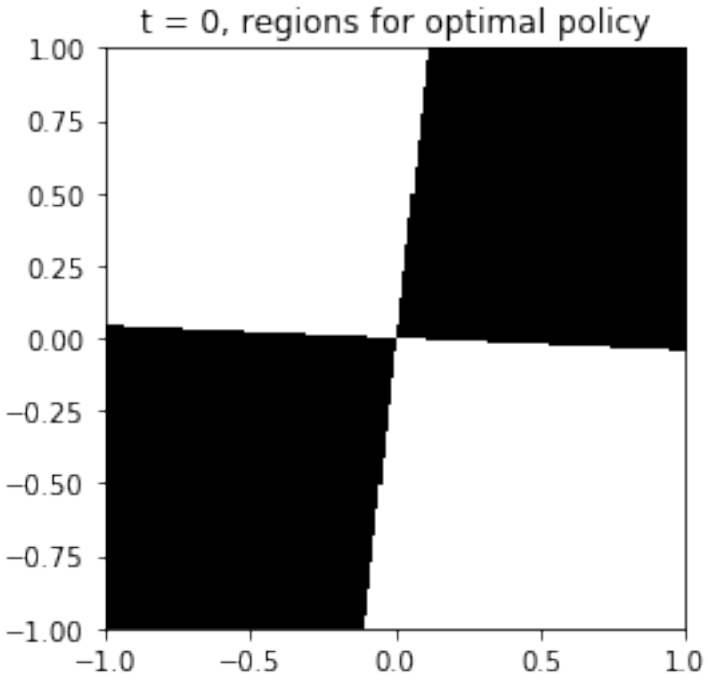
In [2]:
```
    def Optimal_Search(x, T, t):
        """
            Inputs:
                x is the initial condition, it should be a 2-by-1 array.
                T is the horizon.
                 start at t = 2, T = 1;
                       at t = 1, T = 2;
                       at t = 0, T = 3.
            Outputs:
                policy is the policy for every stage.
                V_opt is the optimal cost-to-go values.
        """
        if t == T:
            return np.argmin(ComputeCost(x)), np.min(ComputeCost(x))
        else:
            V1 = ComputeCost(x) + Optimal_Search(motion_model(x, 0), T, t + 1)[1]
            V2 = ComputeCost(x) + Optimal_Search(motion_model(x, 1), T, t + 1)[1]
            policy = np.argmin([V1, V2])
            V_opt = np.min([V1, V2])
            return  policy, V_opt
```

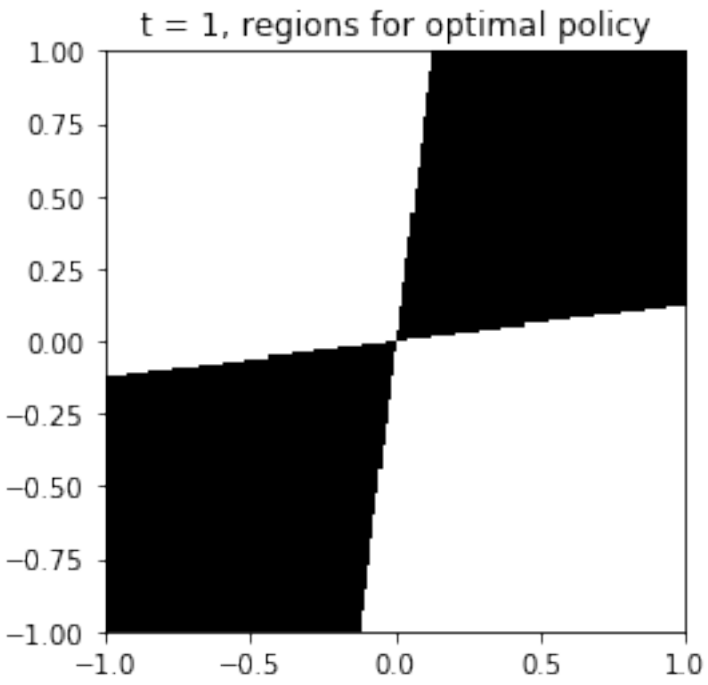
In [7]:
```
    # main function
    # create a grid on [-1, 1] x [-1, 1]
    N = 200
    x_range = np.linspace(-1, 1, N)
    y_range = x_range.copy()
    X, Y = np.meshgrid(x_range, y_range)
    pts_list = np.vstack([X.ravel(), Y.ravel()])# return 2-by-N matrix
    pts = np.stack((X, Y), axis = 0) # stack X,Y as 2xNxN

    policy = np.zeros((3, N, N))
    V = np.zeros((3, N, N))
    for i in range(N):
        for j in range(N):
            x0 = pts[:, i, j]
            x0 = x0.reshape(2, 1)
            for t in range(3):
                policy[t, i, j], V[t, i, j] = Optimal_Search(x0, 3, t)
```

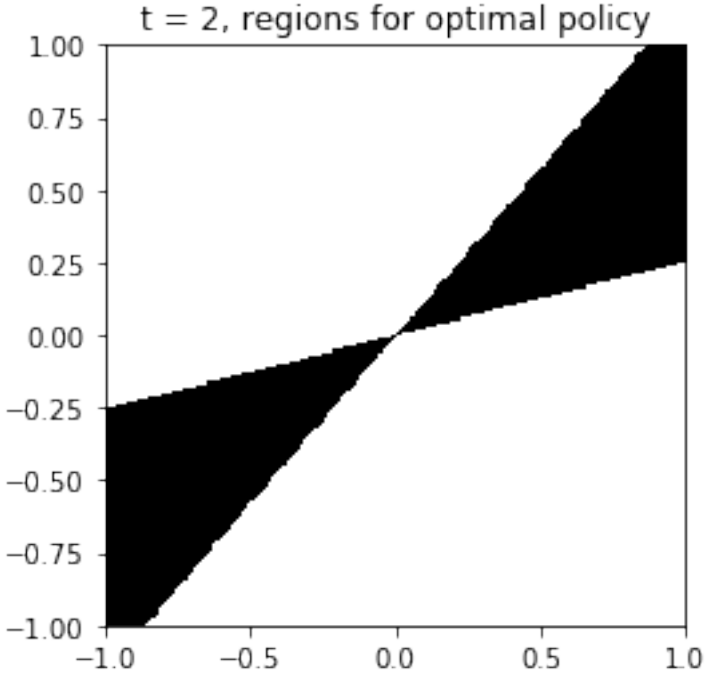
In [15]:
```
# display the results
```

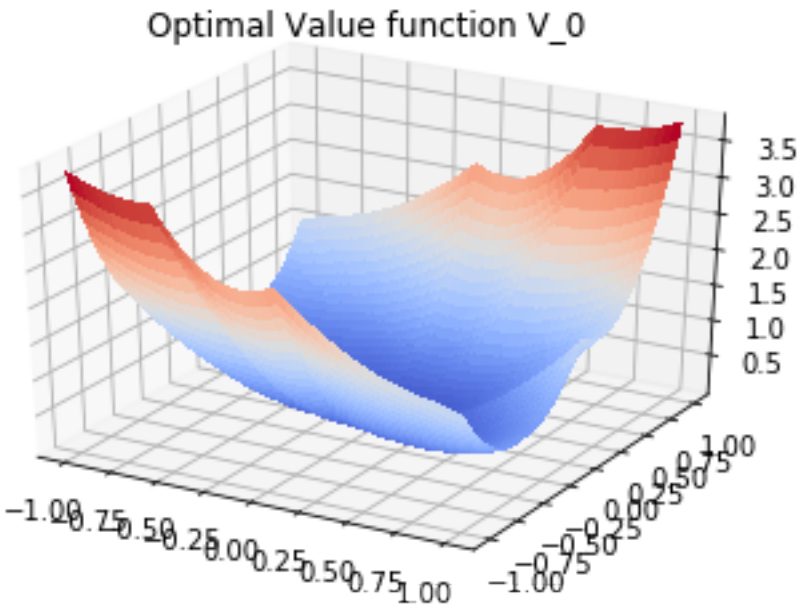
```
plt.figure()
plt.imshow(policy[0, :, :].T, cmap='gray', origin='lower', extent=[-1,1,-1,1])
plt.title("t = 0, regions for optimal policy")

plt.figure()
plt.imshow(policy[1, :, :].T, cmap='gray', origin='lower', extent=[-1,1,-1,1])
plt.title("t = 1, regions for optimal policy")

plt.figure()
plt.imshow(policy[2, :, :].T, cmap='gray', origin='lower', extent=[-1,1,-1,1])
plt.title("t = 2, regions for optimal policy")

Z = V[0, :, :].reshape(N, N)
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
plt.title("Optimal Value function V_0")
```

Out[15]: Text(0.5,0.92,'Optimal Value function V_0')




t = 1, regions for optimal policy
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0


t = 2, regions for optimal policy
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0



For region figures, *black* region presents policy 1, where $u_t = 1$ and *white* region represents policy 2, where $u_t = 2$.

The boundary between policy 1 and policy 2 is actually the Intersection of 2 P.D. quadratic functions. For example, if t = 2, the boundary is to solve:

$$\mathbf{x}_2^T(A^TA)\mathbf{x}_2 = \mathbf{x}_2^T(B^TB)\mathbf{x}_2$$

which is

$$\mathbf{x}_2^T(A^TA - B^TB)\mathbf{x}_2 = \mathbf{x}_2^T D\mathbf{x}_2 = \mathbf{0}.$$

which also is

$$0.3125x_1^2 - 1.5x_1x_2 + 1.0625x_2^2 = 0.$$

Solve for x_2 we will have

$$x_2 = \frac{12 \pm \sqrt{59}}{17}x_1$$

This is the equations of 2 lines, which are consistent with our figures.

As t = 2 to t = 0, assume the action matrix (optimal) we take is $C_t = \{A, B\}$ (left-multiply A or B), we can always find the boundary is to solve

$$\mathbf{x}_t^T C_t^T A^T A C_t \mathbf{x}_t = \mathbf{x}_t^T C_t^T B^T B C_t \mathbf{x}_t$$

$$\mathbf{x}_t^T C_t^T (A^TA - B^TB) C_t \mathbf{x}_t = \mathbf{0},$$

where

$$A^TA - B^TB = \begin{bmatrix} 0.3125 & -0.75 \\ -0.75 & 1.0625 \end{bmatrix}.$$

Therefore, the boundary will change but still keeps linear.

As for optimal value, it is a P.D quadratic function as shown in figure.

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 = ∞ for all i ∈ V\{s} ;
while OPEN is not empty do
    Remove i from OPEN;
    for j ∈ Children(i) do
        if (g_i + C_{i,j}) < g_j and (g_i + C_{i,j}) < g_τ then
            g_j ← (g_i + C_{i,j})
            Parent(j) ← i
            if j ≠ τ 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

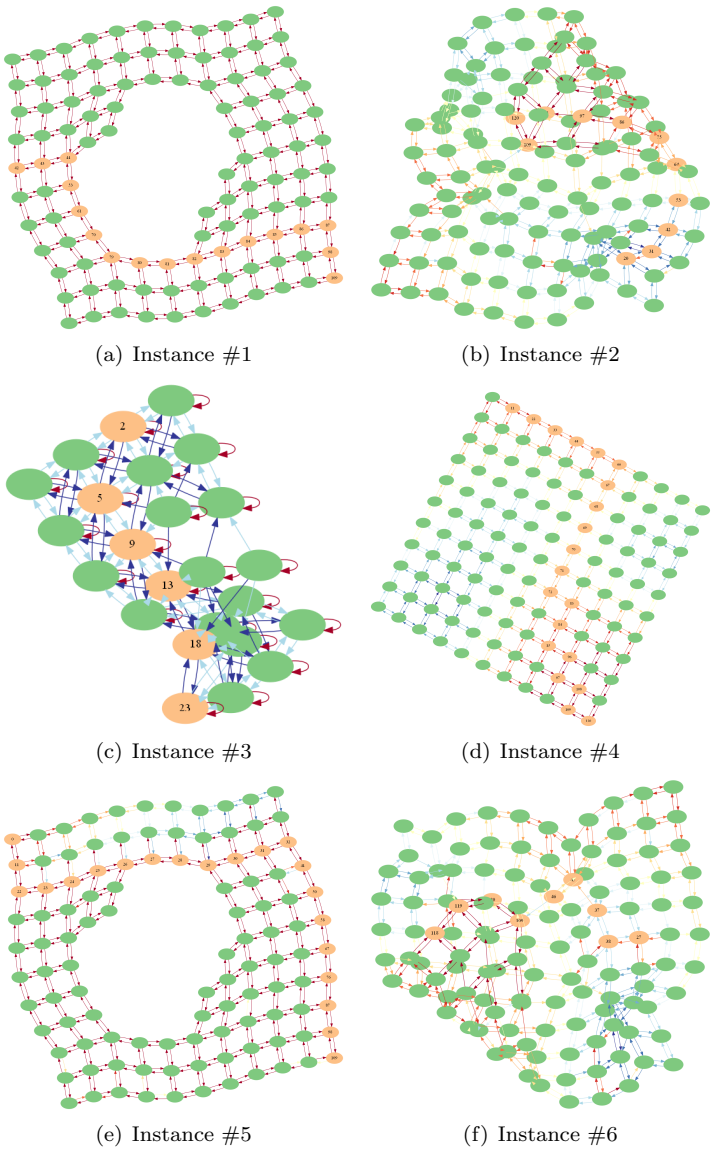
Instances	Paths
1	42-43-44-53-61-70-79-80-81-82-83-84-85-86-87-98-109
2	20-31-42-53-64-75-86-97-108-109-120
3	2-6-9-13-18-23
4	11-22-33-44-55-66-67-68-69-70-71-72-83-84-85-96-97-108-109-120
5	0-11-22-23-24-25-26-27-28-29-30-31-32-41-50-58-67-76-87-98-109
6	27-38-37-45-46-109-120-119-118

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

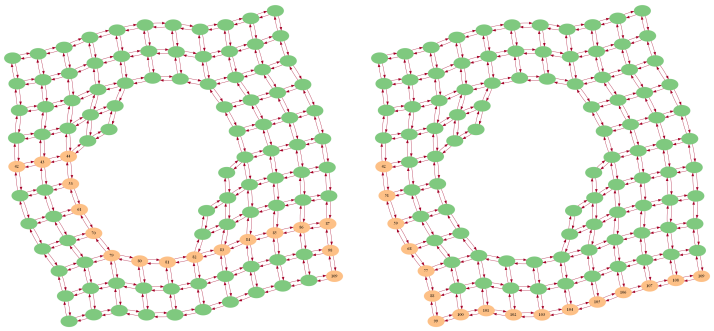
(c) Instance #3 (d) Instance #4

(e) Instance #5 (f) Instance #6

Figure 1: The Optimal Paths

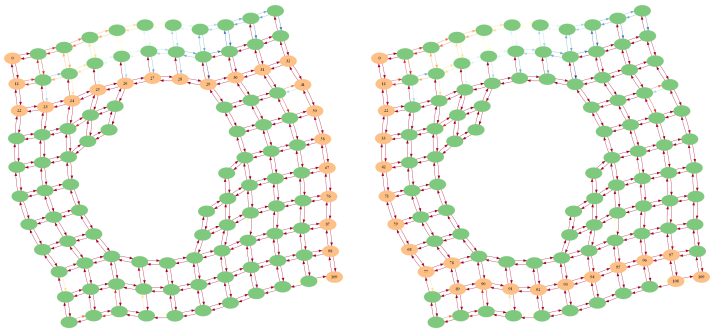
two paths in each instance cost the same values (distances). The difference is when we implement LC algorithm, which element in the list should be removed. In Fig. 2 and Fig. 3, (a) is removing the last element in the list while (b) is removing the first one.
This implies LC will only return one of the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

Report for Problem 5

Yiran Xu

04/28/2019

1 Problem Formulation

We are asked to play rock-paper-scissors(RPS) and win our opponent, i.e. the times of our victories should be larger than our opponent. Our opponent plays with a preferred move 50% of the time and each of the other two 25%. However, we do not know exactly which move the opponent prefers. Our goal is to win our opponent in 100 times of game. If we denote the score for "win", "draw" and "lose" as {1, 0, -1}, then our goal is to outperform than our opponent's score, i.e. $s - s_o > 0$, where s represents the score of ours while s_o is the score of the opponent.
Specifically, we can formulate this problem as a Partially Observable Markov Decision Process (POMDP) problem.

- State space $\mathcal{X}$: $x_t \in \mathcal{X} = \{R_{ot}, P_{ot}, S_{ot}\} \times y$, which is *what your opponent plays* $\times$ *score differential between the opponent and me* at time t for $t = 1, ..., T-1$, where $\times$ is a Cartesian Product operator, score s is defined as -1 as our score is less than our opponent's, 0 as draw and 1 as our score is more than our opponent's.
- Control space $\mathcal{U}$: In this case, the control input u_t is what we decide to play at time t. Then the control space $\mathcal{U}_t = \{R_t, P_t, S_t\}$.
- Observation space $\mathcal{Z}$: This is based on how our opponent plays at time t, i.e. $\mathcal{Z}_t = \{R_{ot}, P_{ot}, S_{ot}\}$.
- Motion Model: Given x_t and u_t, x_{t+1} is what our opponent will play at next time. However, in this case, it seems that we can assume that x_{t+1} is independent of u_t since what we played at time t have nothing to do with our opponent's play because he has his own pattern. Thus $p_f(x_{t+1}|x_t, u_t) = p_f(x_{t+1}|x_t)$.
- Planning horizon: $T = 100$
- The stage and the terminal costs: Therefore we can set stage cost

$$l(x) = \begin{cases} 1 & \text{if win} \\ 0 & \text{if draw} \\ \text{-1} & \text{if lose} \end{cases} \quad (1)$$

.

Then we can convert this POMDP into MDP for DP algorithm implementation.

- State space $\mathcal{B} = \{\mathbf{b} \in [0,1]^3 \times y | \mathbf{1}^T\mathbf{b} = 1\}$, where $y \in \mathbb{R}$ is the cumulative score difference, $\mathbf{b}$ is the preference distribution of rock, paper and scissors of the opponent, i.e.

$$\mathbf{b}_t[0] = Pr(X_t = R|X_{1:t-1})$$
$$\mathbf{b}_t[1] = Pr(X_t = P|X_{1:t-1})$$
$$\mathbf{b}_t[2] = Pr(X_t = S|X_{1:t-1}).$$

  Actually,

$$\begin{aligned} Pr(X_t = R|X_{1:t-1}) = &Pr(X_t = R|prefer = R, X_{1:t-1})Pr(prefer = R|X_{1:t-1}) \\ &+ Pr(X_t = R|prefer = P, X_{1:t-1})Pr(prefer = P|X_{1:t-1}) \\ &+ Pr(X_t = R|prefer = S, X_{1:t-1})Pr(prefer = S|X_{1:t-1}) \end{aligned} \tag{2}$$

  The other two hold the same structure. For initialization $\mathbf{b_0} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]^T$.

- Control space $\mathcal{U}_t = \{R_t, P_t, S_t\}$. It is the same as the POMDP setting.

- Motion model: By using Bayes Rule,

$$Pr(prefer = R|X_{1:t-1}) = \frac{Pr(X_{1:t-1}|prefer = R)Pr(prefer = R)}{Pr(X_{1:t-1})}$$
$$Pr(prefer = P|X_{1:t-1}) = \frac{Pr(X_{1:t-1}|prefer = P)Pr(prefer = P)}{Pr(X_{1:t-1})}$$
$$Pr(prefer = S|X_{1:t-1}) = \frac{Pr(X_{1:t-1}|prefer = S)Pr(prefer = S)}{Pr(X_{1:t-1})}$$

  where

$$Pr(prefer = R) = Pr(prefer = P) = Pr(prefer = S) = \frac{1}{3},$$
$$Pr(X_{1:t-1}) = \sum_{prefer'} Pr(prefer'|X_{1:t-1})Pr(X_{1:t-1}|prefer')$$

$$Pr(X_{1:t-1}|prefer = R) = \prod_{X_i=R} Pr(X_i = R) \prod_{X_j=S} Pr(X_j = S) \prod_{X_k=P} Pr(X_k = P)$$

  with

$$Pr(X_i = R) = 0.5, Pr(X_j = S) = Pr(X_k = P) = 0.25.$$

  $Pr(X_{1:t-1}|prefer = P)$ and $Pr(X_{1:t-1}|prefer = S)$ are similar to this.

  Then we can plug those equations into Eq. 2 to update $\mathbf{b}_t$.

- Horizon $T = 100$.
- Costs: we can still use the same costs in POMDP as Eq.1.

Therefore, this problem is actually a simultaneous estimation and playing problem. Each round, we update our information about our opponent's preference and then use DP to decide the optimal policy. The state transition diagram is shown as Fig.1.

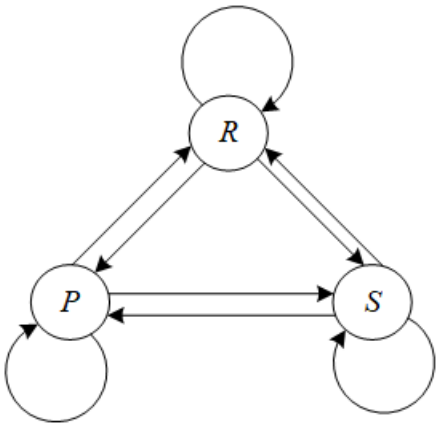


Figure 1: Transition Diagram for "Rock-Paper-Scissors"

2 Technical Approach

For each round, based on estimation of b_t, we can use Forward Dynamic Programming (DP) to determine the optimal policy. The algorithm for Forward DP is shown as Algorithm 1.

Algorithm 1: Forward Dynamic Programming Algorithm

Result: Optimal policy $\pi_t^*(x_t)$
T = 100, play set x_0, cost matrix $\mathbf{C}$
$V_0(x_0) = 0$
for $t = 1, 2, ..., T$ **do**
  $\mathrm{V}_t(\mathbf{b}_t) = \max_{u \in \mathcal{U}}(l(x_t, u_t) + \mathbb{E}_{x_{t+1} \sim p_f(*|x,u)}[V_{t+1}(x_{t+1})])$
  $\pi_t^*(\mathbf{b}_t) = argmax_{u \in \mathcal{U}}(l(x_t, u_t) + \mathbb{E}_{x_{t+1} \sim p_f(*|x,u)}[V_{t+1}(x_{t+1})])$
end

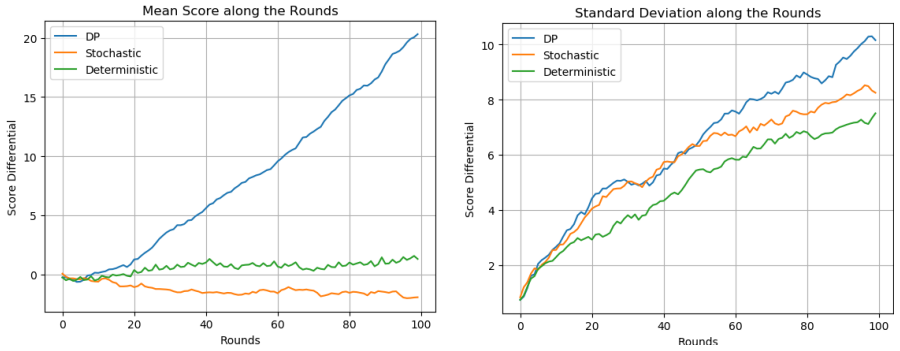
3 Results

3.1 Experiment Description

Since our opponent has a bias, we can assume he likes to play Paper. In order to explore the advantage using DP, three strategies - *deterministic, stochastic, and optimal (DP)*, are used. Here, *deterministic* strategy iterates *rock, paper, scissors, rock, paper, scissors, ...* Without losing generality, 50 times 100-game matches are played. Score y_t is the score differential at time t.

3.2 Results and Discussion

The results of 50 100-game matches are shown in Fig. 2. These figures show the mean values and standard deviation (std) as the number of rounds increases using three strategies. It is clear that DP outperforms the other two strategies as number of rounds increases from mean score. "Stochastic" strategy and "deterministic" nearly performs the same. For "optimal" strategy, it increases the score because as the game is going on we obtain more more information to estimate $\mathbf{b}_t$. Plus, std. of DP is the greatest in general, this is because the worst result will have big difference with the good results. This big deviation increases the std. Nevertheless, the other two strategies result in nearly same result: low score differential, so the stds are smaller.



(a) Mean (b) Standard Deviation

Figure 2: 50-Time Play: Mean and Std.

There are two more interesting things in this problem.
First of all, it's that the belief $\mathbf{b_t}$ in Eq. 2 is actually the same as following equations according to the results from my program:

$$
\begin{aligned}
\mathbf{b}_t[0] &= \frac{\#opponent\ played\ rock}{\#rounds} \\
\mathbf{b}_t[1] &= \frac{\#opponent\ played\ paper}{\#rounds} \\
\mathbf{b}_t[2] &= \frac{\#opponent\ played\ scissors}{\#rounds}
\end{aligned} \tag{3}
$$

This is the result followed by Maximum Likelihood Estimation (MLE), which indicates our optimal policy is actually the same as MLE.
Another one is the expectation in Algorithm 1. In this case, we know the belief $\mathbf{b}_t$, our optimal policy is always against our opponent's preference. This sounds intuitive but it is based on DP's choice.