HW1: Markov Processes and Dynamic Programming

Shuo Xu
Department of Computer Science and Engineering
s3xu@ucsd.edu

Abstract

1 Problem 1

(a)

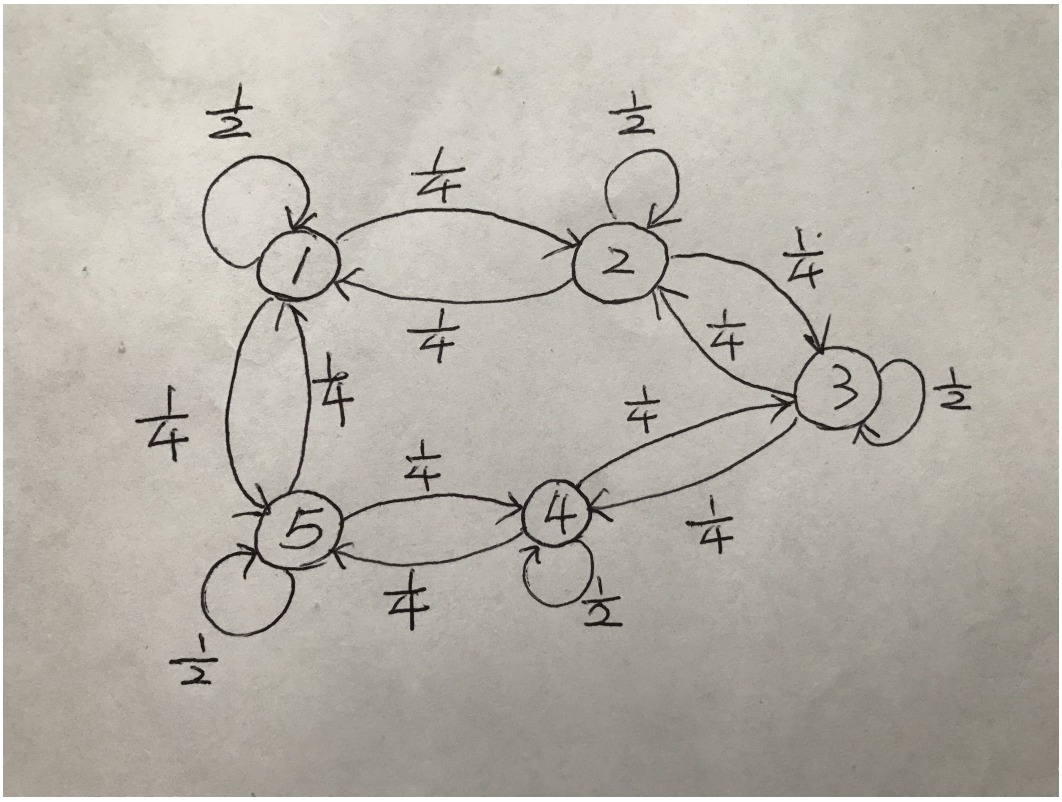

Figure 1: Transition diagram

This chain is irreducible. Intuitively speaking, from the diagram above, all states belong to a single communicating class, so it's possible to go from any state to every state.

Mathematically, since

$$P^2 = \begin{bmatrix} 0.375 & 0.25 & 0.0625 & 0.0625 & 0.25 \\ 0.25 & 0.375 & 0.25 & 0.0625 & 0.0625 \\ 0.0625 & 0.25 & 0.375 & 0.25 & 0.0625 \\ 0.0625 & 0.0625 & 0.25 & 0.375 & 0.25 \\ 0.25 & 0.0625 & 0.0625 & 0.25 & 0.375 \end{bmatrix},$$

all the entries of which are positive, thus it's possible to go from every state to every state. In other words, this is a regular chain, and regular chain is irreducible.

The chain is aperiodic. From the diagram, if we start from state 1, we can get back to state 1 in 1, 2, 3, 4... steps. The period, greatest common denominator, of these integers is obviously 1, thus the chain is aperiodic.

(b)

According to Theorem 11.7 and 11.8 in Grinstead-Snell,

$$\lim_{n\to\infty} P^n = W,$$

let w be the fixed row vector of W, then

$$wP = w.$$

The solution of the above equition is

$$w = [0.2 \quad 0.2 \quad 0.2 \quad 0.2 \quad 0.2],$$

which means p_t will approximate towards w as t increases.

According to Theorem 11.9 in Grinstead-Snell,

$$\lim_{n\to\infty} = vP^n = w$$

for arbitary probability vector v, so the theoretical value of p_∞ is $[0.2 \quad 0.2 \quad 0.2 \quad 0.2 \quad 0.2]$.

The simulated value after 100 periods is

$$p_{100} = [0.2 \quad 0.2 \quad 0.2 \quad 0.2 \quad 0.2],$$

which is the same to the theoretical value of p_∞.

(c)

To solve the problem, we construct a transition matrix as below.

$$P = \begin{bmatrix} 1/3 & 1/3 & 0 & 0 & 1/3 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 1/3 & 1/3 & 1/3 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 1/3 & 0 & 0 & 1/3 & 1/3 \end{bmatrix}.$$

The initial states are the ages of each person,

$$S_{init} = [25 \quad 20 \quad 35 \quad 24 \quad 46]$$

S_t denotes the averaged age known by every person at time t. The average age known by anyone at tth step is

$$S_t = S_{init} P^t$$

The averaged age should be

$$S_{avg} = \lim_{t\to\infty} S_t$$

According to Perron-Frobenius Theorem, $\lim_{t\to\infty} S_t = w$ where $wP = w$. The solution of the fixed row vector w is $w = [30 \quad 30 \quad 30 \quad 30 \quad 30]$.

Thus, the theoretical average age of all 5 people is 30, which is supported by the simulation result at 100th time step.

2 Problem 2

Here's the values of x_{t+1} given x_t and u_t.

u_t \ x_t	-1	0	1	2
-1	2	1	0	-1
1	0	1	2	-

as well as the costs,

u_t \ x_t	-1	0	1	2
-1	1	0	-1	-2
1	-1	0	1	-

Here's a graph of using dynamic programming over the system with horizon of 2. It is constructed from right to left. The number in each node denotes the state x_t at time t. c denotes the cost. u is the control signal, and $V_t(x)$ is the optimal cost of state x at time t.

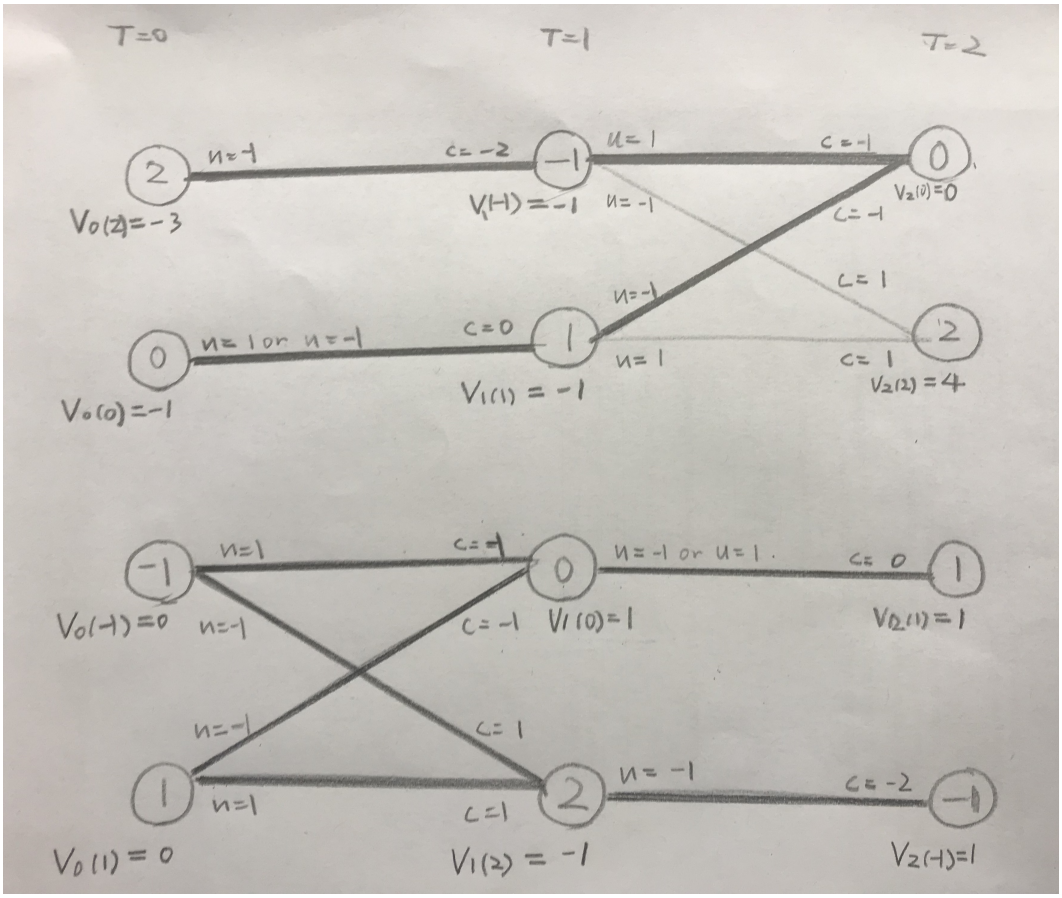

Figure 2: Dynamic programming solution

The bold line between nodes suggests the best policy.

(a)

Table 1: Best policy of $u_t = \pi_t(x)$

t \ x	-1	0	1	2
0	-1 or 1	-1 or 1	-1 or 1	-1
1	1	-1 or 1	-1	-1

(b)

From the graph above, when $x_0 = 2$,

- The optimal cost is -3,
- The control sequence is -1, 1,
- The state trajectory is 2, -1, 0

3 Problem 3

(a)

Intuitively, from time t to time T, we have T - t chances to make decision of which control option should be taken. And since we have 2 options each time, the number of possible state functions, as well as corresponding value functions will be 2 times the scale of last time's. So the number of candidate functions will be 2^{T-t}.

Here's a deduction process in detail.

- When $time = t$, we are at the only one state x with 2 control options. The combinations of them are $2^1 = 2$ possible state functions: $\{Ax, Bx\}$, and $2^1 = 2$ corresponding value functions $\{\ell(x, 1), \ell(x, 2)\}$.
- When $time = t + 1$, we have 2 possible states and 2 control options, the combinations of which is a set of $2^2 = 4$ functions: $\{A^2x, ABx, BAx, B^2x\}$, and 4 corresponding value functions $\{\ell(Ax, 1) + \ell(x, 1), \ell(Ax, 2) + \ell(x, 1), \ell(Bx, 1) + \ell(x, 2), \ell(Bx, 2) + \ell(x, 2)\}$.
- ...
- When $time = t+k$, we have 2^k possible states functions and 2 control options, the combinations of which is a set of 2^{k+1} state functions, and 2^{k+1} corresponding value functions.
- ...
- When $time = T = t + (T - t)$, we have 2^{T-t} possible state functions. Since this is the last step, we should have 2^{T-t} corresponding value functions of $q(x)$.

Since all $\ell(x, u)$ and $q(x)$ are quadratic functions, and the summation of quadratic functions are quadratic, at time T, we have 2^{T-t} candidate positive definite quadratic functions. And $V_t^*(x)$ of which should be the minimum of them.

To select the optimal policy, we should keep track of the control selected at each time step when doing the forward dynamic programming. At time T, when the minimum of 2^{T-t} value function is picked out, we could use the tracing information to recover the control we selected.

To illustrate the ideas above better, here's an implementation of the algorithm in python.

```
import numpy as np
from collections import defaultdict

def cost(x):
    return x.T.dot(x) / 2
```

```
def find_best_policy(x, controls, T):

    states, values = defaultdict(list), dict()
    states[0].append(x)
    values[0] = np.array([cost(x)])
    for i in range(T):
        states[i + 1].extend([control.dot(state) for control in controls
                                                 for state in states[i]])
        values[i + 1] = np.array([cost(state) for state in states[i + 1]]
                                                ) + np.concatenate((values[i
                                                ], values[i]))

    policy = []
    T = len(values) - 1
    idx = np.argmin(values[T])
    for t in range(T, 0, -1):
        policy.append(1 if idx < len(values[t]) // 2 else 2)
        idx %= (len(values[t]) // 2)
    return policy[::-1]

if __name__ == "__main__":
    A = np.array([[0.75, -1], [1, 0.75]])
    B = np.array([[1, 0.5], [0.5, 0.5]])
    policies = []
    xs, ys, values = [], [], []
    X, Y = np.meshgrid(np.arange(-1, 1.01, 0.005), np.arange(-1, 1.01, 0.
                                                 005))
    values = np.zeros_like(X)

    for row in range(X.shape[0]):
        for col in range(X.shape[1]):
            xs.append(X[row][col])
            ys.append(Y[row][col])
            best_policy, best_value = find_best_policy(np.array([X[row][
                                                  col], Y[row][col]]), [A,
                                                   B], T=3)
            policies.append(best_policy)
            values[row][col] = best_value
```

(b)

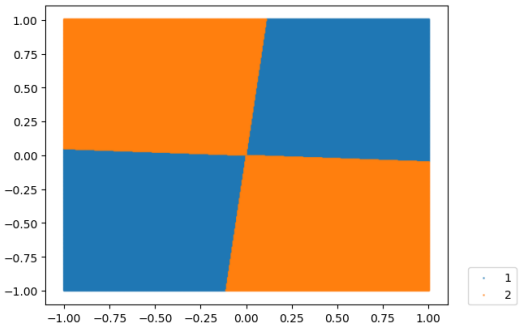

Figure 3: π_0^*

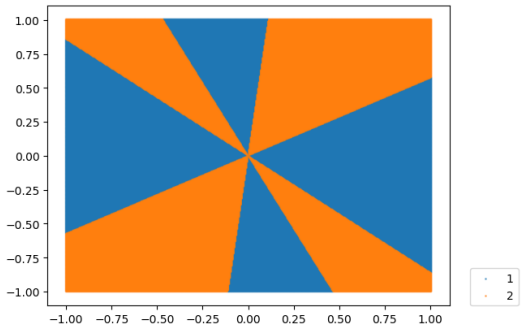

Figure 4: π_1^*

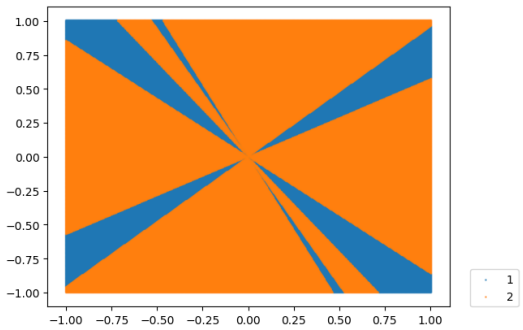

Figure 5: π_2^*

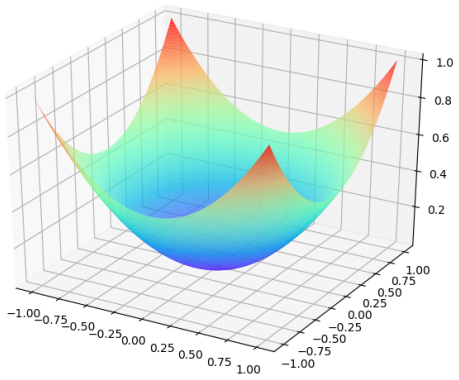

Figure 6: $V_0^*(x)$

Here's my code for simulation.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def plot2d(x, y, labels, time):

    controls, xs, ys = defaultdict(list), defaultdict(list), defaultdict(
                                        list)
    for i, label in enumerate(labels):
        controls[label].append(label)
```

```
        xs[label].append(x[i])
        ys[label].append(y[i])

    fig, ax = plt.subplots()
    for type in sorted(controls.keys()):
        ax.scatter(xs[type], ys[type], s=[1]*len(xs[type]), label=f'${
                                                type}$', alpha=0.5)
    plt.legend(bbox_to_anchor=(1.05, 0), loc=3, borderaxespad=0)
    plt.savefig(f"t{time}", bbox_inches='tight')
    plt.close()


def plot3d(x, y, z, name="v0"):
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot_surface(np.array(x), np.array(y), np.array(z), rstride=1,
                                            cstride=1, cmap=plt.get_cmap('
                                            rainbow'))
    plt.savefig(f"{name}")
    plt.close()


if __name__ == "__main__":
    A = np.array([[0.75, -1], [1, 0.75]])
    B = np.array([[1, 0.5], [0.5, 0.5]])
    policies = []
    X, Y = np.meshgrid(np.arange(-1, 1.01, 0.005), np.arange(-1, 1.01, 0.
                                            005))
    values = np.zeros_like(X)

    for row in range(X.shape[0]):
        for col in range(X.shape[1]):
            best_policy, best_value = find_best_policy(np.array([X[row][
                                                    col], Y[row][col]]), [A,
                                                     B], T=3)
            policies.append(best_policy)
            values[row][col] = best_value

    plot3d(X, Y, values)

    controls = [p for p in zip(*policies)]
    for time, control in enumerate(controls):
        plot2d(X.reshape(-1), Y.reshape(-1), list(control), time=time)
```

4 Problem 4

(a) Problem Formulation

Consider a DSP problem with vertex space V, weighted edge space C, start node $s \in V$ and terminal node $\tau \in V$.

- State: $\mathcal{X}_0 := \{s\}, \mathcal{X}_T := \{\tau\}, \mathcal{X}_t := \mathcal{V} \backslash \{\tau\}$
- Horizon: $T := |\mathcal{V}| - 1$
- Control spaces: $\mathcal{U}_{T-1} := \{\tau\}$ and $\mathcal{U}_t := \mathcal{V} \backslash \{\tau\}$
- Motion model: $x_{t+1} == u_t$ for $u_t \in \mathcal{U}_t := \mathcal{V} \backslash \tau$ for $t = 0, 1, \ldots, T-2$
- Costs: $\mathfrak{q}(\tau) := 0$ and $\ell_t(x_t, u_t) = c_{x_t, u_t}$ for $t = 0, \ldots, T-1$

(b) Technical Approach

I selected Dijkstra algorithm to solve this problem, a specialized version of label correcting method, using priority queue to select the next node from open set.

Here's the pseudocode.

Algorithm 1 Dijkstra algorithm

```
procedure DIJKSTRA(start, end, distance)
    parents ← {node : node for node in graph}
    queue ←all nodes in graph
    while queue is not empty do
        prev_node ← node in queue with smallest distance to start node
        for each neighbour n of prev_node do
            new_dis = distance[start][prev_node] + distance[prev_node][n]
            if new_dis < distance[start][n] then
                distance[start][n] ← new_dis
                parents[n] ← prev_node
    shortest_path ← [target]
    while start != target do
        target ← parents[target]
        shortest_path.append(target)
     return shortest_path
```

(c) Results

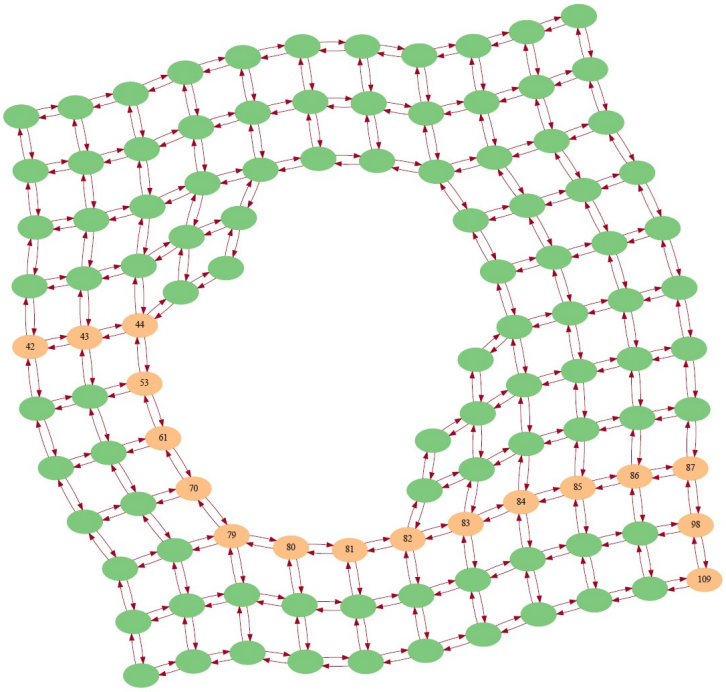

Figure 7: Transition diagram

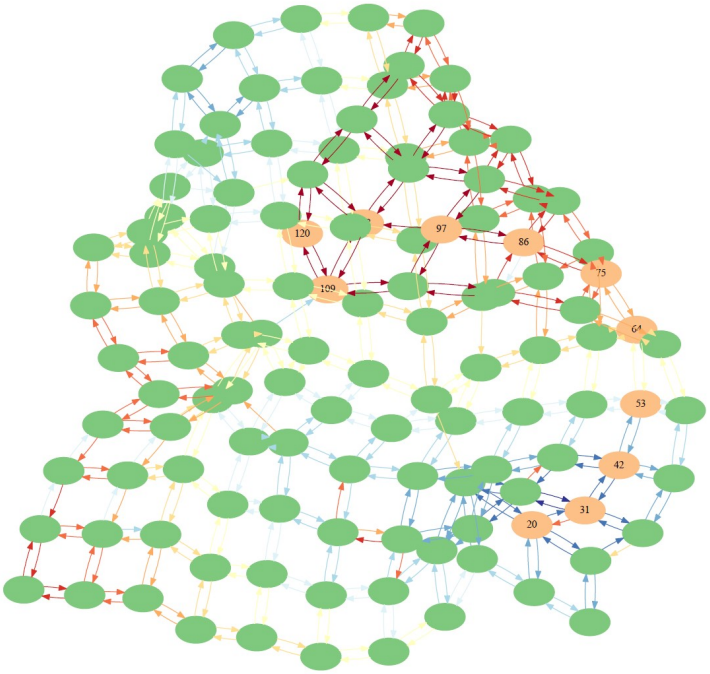

Figure 8: Transition diagram

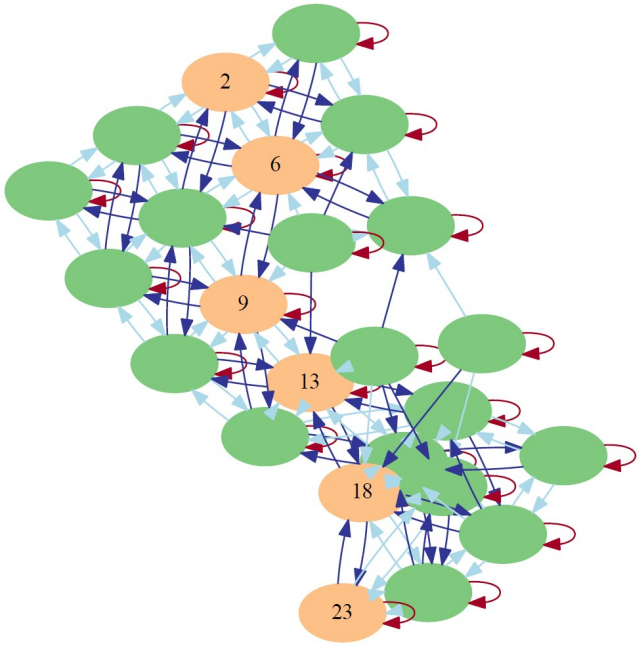

Figure 9: Transition diagram

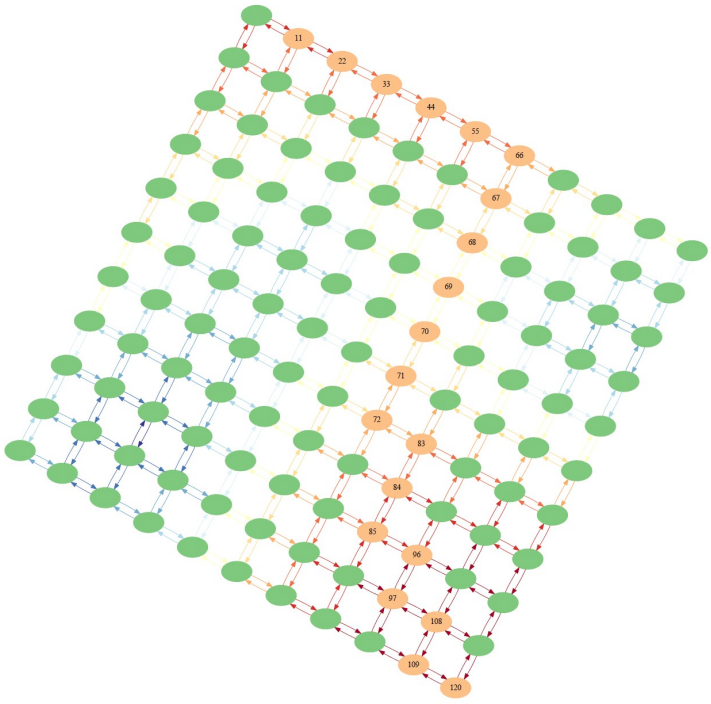

Figure 10: Transition diagram

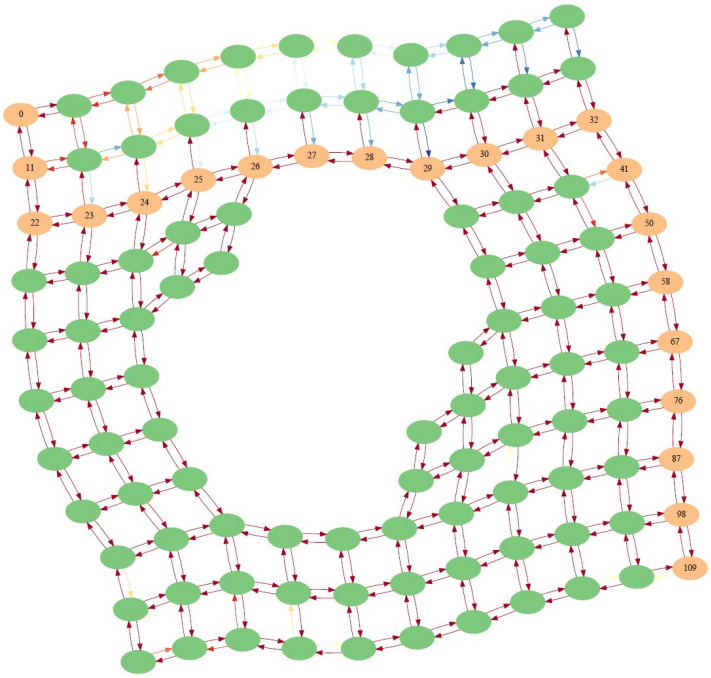

Figure 11: Transition diagram

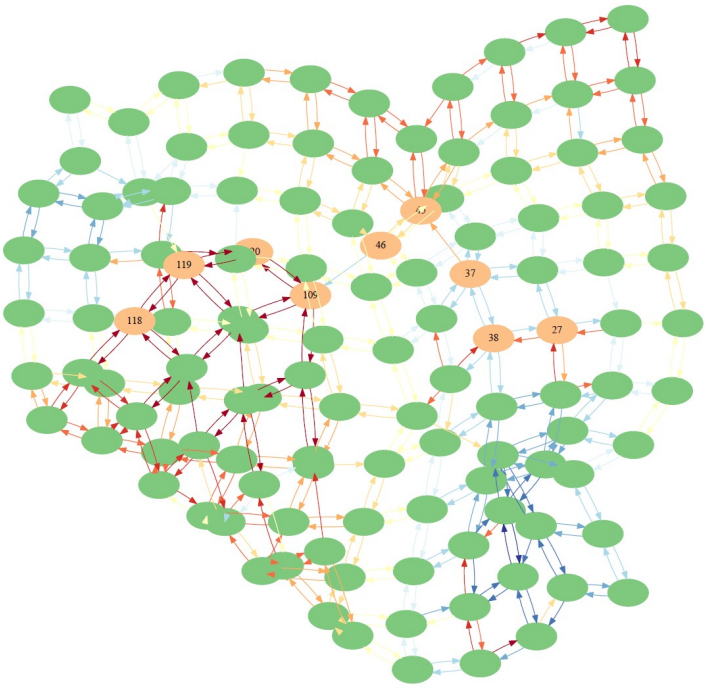

Figure 12: Transition diagram

5 Problem 5

(I could barely work out this solution without the inspiration from professor's and TA's hints. Special thanks to them!)

5.1 Problem Formulation

- State: $X_t = [R_t, P_t, S_t, SD_t]$, where $R_t, P_t, S_t \in \{i | 0 <= i <= 100, i \in \mathbb{Z}\}$, $SD_t \in \{i | -100 <= i <= 100, i \in \mathbb{Z}\}$. R_t, P_t, S_t stands for the number of rock, paper, scissors played by the opponent at time t, SD is score differential. Specially, $S_0 = [0, 0, 0, 0]$.
- Control space: $U = \{R, P, S\}$. R, P, S stands for rock, paper, scissors, separately.
- Stage cost: $\ell(x_t, u_t) = 0$
- Terminal cost: $q(X_t) = q([R_t, P_t, S_t, SD_t]) = SD_t$
- Horizon: $T = 100$
- Models:

  M_t is the opponent's move at time t. $M_t \in \{R, P, S\}$.

  F stands for opponent's favored move, which is not observable to us. $F \in \{R, P, S\}$.

  Given state at time t, the probability of the opponent's next move is

$$P_f = P(M_{t+1}|X_t) \quad = \sum_{F \in \{R,P,S\}} P(M_{t+1}|F)P(F|X_t)$$

  As for $P(M_t|F)$, we evaluate it as

$$P(M_t|F) = \begin{cases} 0.5, & M_t = F \\ 0.25, & M_t \neq F. \end{cases}$$

For $P(F|X_t)$, we can use Bayes rule to solve it.

$$\begin{aligned} P(F|X_t) &= P(F|R_t = r, P_t = p, S_t = s) \\ &= \frac{P(R_t = r, P_t = p, S_t = s|F)P(F)}{P(R_t = r, P_t = p, S_t = s)} \end{aligned}$$

$P(R_t = r, P_t = p, S_t = s|F)$ means given opponent's favored move, what his / her probability of having r, p, s number of rock, paper, scissors, separately. Specially, at the beginning of the game, we set $P(F) = \frac{1}{3}$ for $F \in \{R, P, S\}$.

5.2 Technical Approach

I used dynamic programming for the optimal policy.

We define function $Q_t^*(X_t, U_t)$ as the optimal value at time t given state and control, and $V_t^*(X_t)$ as the optimal value given state at time t.

$$\begin{aligned} Q_t^*(X_t, U_t) &= Q_t^*(X_t = [R_t, P_t, S_t, SD_t], U_t) \\ &= \sum_{M_{t+1} \in \{R,P,S\}} P(M_{t+1}|X_t) V_{t+1}^*([Update(R_t, P_t, S_t, M_{t+1}), SD_t + Judge(U_t, M_{t+1})]) \end{aligned}$$

$Judge(my\_move, opponent\_move)$ is defined as below

$$Judge(my\_move, opponent\_move) = \begin{cases} 1, & \text{my\_move beats opponent\_move} \\ 0, & \text{opponent\_move = my\_move} \\ -1, & \text{opponent\_move beats my\_move} \end{cases}$$

$Update(R, P, S, M)$ is defined as

$$Update(R, P, S, M) = \begin{cases} \text{[R + 1, P, S]}, & M = R \\ \text{[R, P + 1, S]}, & M = P \\ \text{[R, P, S + 1]}, & M = S \end{cases}$$

The optimal value at time t is

$$V_t^*(X_t) = max_{U_t \in \{R,P,S\}} Q_t^*(X_t, U_t).$$

The optimal policy at time t given state X_t is

$$\pi_t^*(X_t) = argmax_{U_t \in \{R,P,S\}} Q_t^*(X_t, U_t).$$

The dynamic programming algorithm runs backwards from T=100.

Algorithm 2 Dynamic programming algorithm

procedure DP(X, U, P_f, l_t, q, T)
  $V_T(X_T) = q(X_T), \forall X_T \in \{x_T | x_T = [R_T, P_T, S_T, SD_T] \in X, R_T + P_T + S_T = T\}$
  for $t = (T-1) \dots 0$ **do**
    $Q_t^*(X_t, U_t) \leftarrow \sum_{M_{t+1} \in \{R,P,S\}} P(M_{t+1}|X_t) V_{t+1}^*([Update(R_t, P_t, S_t, M_{t+1}), SD_t + Judge(U_t, M_{t+1})]), \forall X_t \in \{x_t | x_t = [R_t, P_t, S_t, SD_t] \in X, R_t + P_t + S_t = t\}, \forall U_t \in U$
    $V_t^*(X_t) = max_{U_t \in \{R,P,S\}} Q_t^*(X_t, U_t), \forall x \in X$
    $\pi_t^*(X_t) = argmax_{U_t \in \{R,P,S\}} Q_t^*(X_t, U_t), \forall x \in X$
  return policy $\pi_{0:T-1}$ and value function V

5.3 Results

Here're the comparison of three strategies. Assume here that my friend has a bias towards paper and generate 5000 plays from his strategy.

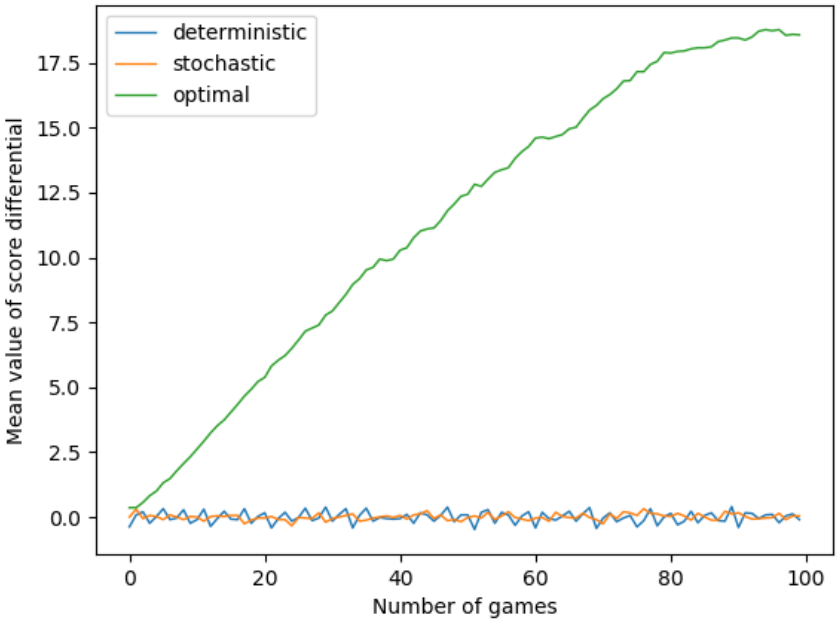

Figure 13: Averaged mean

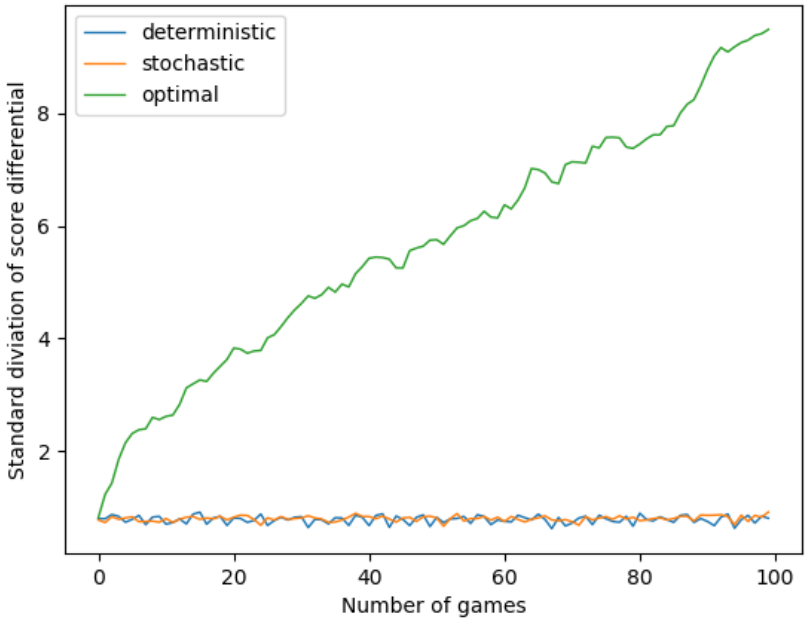

Figure 14: standard diviation