ECE4271 Homework Assignment #1 Robin Liu ZT455

Thought Exercises

2. a) Payoff Matrix

Player A \ Player B	C	D
C	(3,3)	(0,5)
D	(5,0)	(1,1)

TFT	All D-strict	Payoff for TFT
C	D	0
D	D	1
D	D	1
D	D	1
⋮		⋮
D	D	1

9 Games

∴ Total payoff for TFT is 9.

b)

TFT	All cooperate	payoff for TFT
C	C	3
C	C	3
C	C	3
⋮	⋮	⋮
C	C	3

10 games

∴ Total payoff is 30.

c)

TFT	Anti-TFT	Payoff for TFT	Payoff for Anti-TFT
C	D	0	5
D	D	1	1
D	C	5	0
C	C	3	3
C	D	0	5
D	D	1	1
D	C	5	0
C	C	3	3
C	D	0	5
D	D	1	1

∴ Total payoff for TFT is:

~~0×3+1×3+5×2+3×2~~ = 3+10+6

= 19

d) $E(X_{\text{payoff for TFT}}) = \sum_{i=1}^{N} x_i p_i$

where $x_1, \dots x_n$ are possible payoffs for TFT and

p_i is the probability of that payoff.

Since we play 10 games, there are: 2^{10} possible move combinations for random strategy (C or D at each game). Each has $1/2^{10}$ prob. of occurring.

Thus:

$N = 1024$ ($2^{10} = 1024$)

$E(X_{\text{payoff of TFT}}) = \sum_{i=1}^{N} x_i p_i$

Each x_i represents payoff of TFT when random strategy plays move combination m_i.

$= x_1 p_1 + x_2 p_2 + \cdots + x_{1024} p_{1024}$

$p_1 = p_2 \cdots = p_{1024} = 1/2^{10}$

Using a computer program to simulate this game, we get:

[Code included]

$E(X_{\text{payoff to TFT}}) = \boxed{21.75}$ $\boxed{\text{Jan 24}}$

$\therefore E(X_{\text{payoff}}) = \underline{21.75}$

e) Defect vs. TFT:

Using diagram for part (a): $\text{Payoff}_{\text{Defect}} = 5+9 = \boxed{14}$

Cooperate vs. TFT:

Using table in part (b): $\text{Payoff}_{\text{Coop}} = 3\times 10 = \boxed{30}$

Anti-TFT vs. TFT:

Using table in part (c): $\text{Payoff}_{\text{Anti-TFT}} = 3\times 5 + 2\times 3 + 3\times 1 + 3\times 0$

$= 15 + 6 + 3$

$= \boxed{24}$

Random vs. TFT:

Using same technique as part (d), we compute

$$E(X_{\text{payoff for RANDOM}}) = \sum_{i=1}^{N=1024=2^{10}} x_i p_i$$

$= x_{1,1} + x_{1,2} p_{1,2}$

Each x_i is payoff from prisoner & RANDOM w/ m_i move combo.

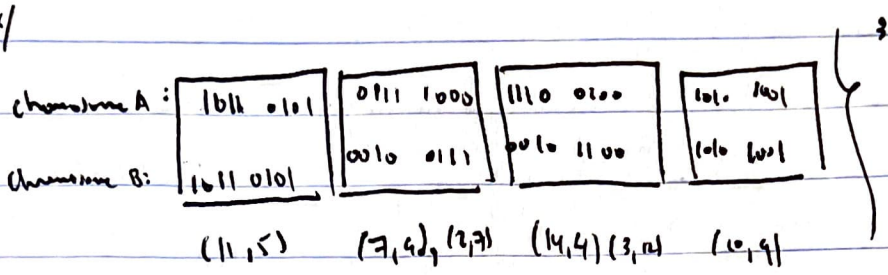
Using a computer program, we get:

$E(X_{\text{payoff for RANDOM}}) = \boxed{14.05}$

```python
def get_payoff_for_A(move_a: int, move_b: int) -> int:
    assert move_a in ["c", "d"] and move_b in ["c", "d"]
    if (move_a, move_b) == ("c", "c"):
        return 3
    if (move_a, move_b) == ("c", "d"):
        return 0
    if (move_a, move_b) == ("d", "c"):
        return 5
    if (move_a, move_b) == ("d", "d"):
        return 1


def get_payoff_for_B(move_a: int, move_b: int) -> int:
    assert move_a in ["c", "d"] and move_b in ["c", "d"]
    if (move_a, move_b) == ("c", "c"):
        return 3
    if (move_a, move_b) == ("c", "d"):
        return 5
    if (move_a, move_b) == ("d", "c"):
        return 0
    if (move_a, move_b) == ("d", "d"):
        return 1


def get_TFT_move(current_move: int, opp_move_history: str) -> str:
    if current_move == 0:
        return "c"

    return opp_move_history[current_move - 1]


def compute_average_TFT_payoff(games: int) -> int:
    total_payoff = 0

    def helper(current_move: int, move_history: str, run_payoff: int) -> None:
        nonlocal total_payoff

        if current_move == games:
            total_payoff += run_payoff
            return

        # Recursive call for `cooperate`
        TFT_coop_move = get_TFT_move(current_move, move_history)
        helper(
            current_move + 1,
            move_history + "c",
            run_payoff + get_payoff_for_A(TFT_coop_move, "c"),
        )

        # Recursive call for `defect`
        TFT_defect_move = get_TFT_move(current_move, move_history)
        helper(
            current_move + 1,
            move_history + "d",
            run_payoff + get_payoff_for_A(TFT_coop_move, "d"),
        )

    helper(0, "", 0)

    return total_payoff / (2 ** games)


def compute_average_RANDOM_payoff(games: int) -> int:
    total_payoff = 0

    def helper(current_move: int, move_history: str, run_payoff: int) -> None:
        nonlocal total_payoff

        if current_move == games:
            total_payoff += run_payoff
            return

        # Recursive call for `cooperate`
        TFT_coop_move = get_TFT_move(current_move, move_history)
        helper(
            current_move + 1,
            move_history + "c",
            run_payoff + get_payoff_for_B(TFT_coop_move, "c"),
        )

        # Recursive call for `defect`
        TFT_defect_move = get_TFT_move(current_move, move_history)
        helper(
            current_move + 1,
            move_history + "d",
            run_payoff + get_payoff_for_B(TFT_coop_move, "d"),
        )

    helper(0, "", 0)

    return total_payoff / (2 ** games)


def main():
    print(f"Average TFT Payoff for {10} games: {compute_average_TFT_payoff(10)}")
    print(f"Average RANDOM Payoff for {10} games: {compute_average_RANDOM_payoff(10)}")


if __name__ == "__main__":
    main()
```

3. 15 pairs of 32-bit chromosomes

Ex/



Each pair of chromosomes (32-bit) has 4 blocks composed of pairs of 8-bit sequences. Each block can either yield one composition (homozygous) or two compositions (heterozygous). A given 8-bit bitstring has 2 values. Each bitstring $a_0 a_1 \ldots a_7$ can yield $\binom{16}{2}$ distinct compositions. Thus, each block can result in:

(1) bitstring in Chromosome A is the same as bitstring in chromosome B.

$\binom{16}{2}$ phenotype segments

Assume $(a,b) = (b,a)$ same composition

Monozygous

(2) bitstring in Chromosome A is different as bitstring in Chrom. B.

$$\binom{16}{2} \cdot \left[\binom{16}{2} - 1\right]$$

Heterozygous

- Each block has $\binom{16}{2} + \binom{16}{2}\left[\binom{16}{2} - 1\right]$ distinct phenotype segments.

There are 4 blows per 32-bit chromosome pair. There are 15 pairs. Thus, there are $15 \times 4 = 60$ blows in total. Each blow is independent of any other blow.

Thus, if each blow has $\binom{16}{2} + \binom{16}{2}\left[\binom{16}{2} - 1\right]$ distinct ~~genotypic~~ phenotype sequences, the total # of possible genotypes is:

$$\left[\binom{16}{2} + \binom{16}{2}\left[\binom{16}{2} - 1\right]\right]^{60}.$$

We are assuming that the same combination can happen multiple times in the sequence.

$$\therefore \left[\binom{16}{2} + \binom{16}{2}\left[\binom{16}{2} - 1\right]\right]^{60}$$

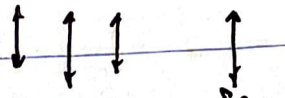
valid networks.

4. Prove that any string of l-length is an instance of 2^l different schemas.

String of length l:

Each position of String must match the corresponding position in the schema.

String of length l: $b_0\ b_1\ b_2 \cdots b_{l-1}$



Schema: $S_0\ S_1\ S_2 \cdots S_{l-1}$

Every String position can be 0, 1. Every schema position can be 0, 1, or *.

Thus, let's take a look at 2 situations:

(1) For a given position b_i in String:

$b_i = 0$

Thus, S_i must be either 0 or *.

(2) For a given position b_i in string:

$b_i = 1$

Thus, S_i must be either 1 or *.

For every position S_i, there can be 2 options: $(0/1)$ | *.

Thus, there are 2^l possible schemas that can match the string, where l is the length.

∴ Every string of length l has 2^l corresponding schemas. QED.

5. 1***:

1***:

f(1000) = 8
f(1001) = 9
f(1010) = 10
f(1011) = 11
f(1100) = 12
f(1101) = 13
f(1110) = 14
f(1111) = 15

Avg f: $\frac{8+9+10+11+12+13+14+15}{8}$

$= \boxed{11.5}$

0***:

f(0000) = 0
f(0001) = 1
f(0010) = 2
f(0011) = 3
f(0100) = 4
f(0101) = 5
f(0110) = 6
f(0111) = 7

Avg f: $\frac{0+1+2+3+\cdots+7}{8}$

$= \boxed{3.5}$

6. $f(s) = \#$ of 1's in s.

H : k-defined bits of l.

Then, H: $\underbrace{b_1 b_2 b_3 \cdots b_l}_{k \text{ positions are } 1}$

For sake of Convenience, we can group the k's to one side.

This is okay because we only care about the # of 1's in Strings, not the positions.

H: $\underbrace{1 1 \cdots 1}_{k}\ \underbrace{b_1^* b_2^* b_3^* \cdots b_{l-k}^*}_{* \text{ (non-defined)}}$.

H has: minimum k 1's and maximum l 1's.

Then, any # of 1's (any string of H):

$$\sum_{i=k}^{i=l} i \cdot P_i = k \cdot P_k + (k+1) P_{k+1} + (k+2) \cdot P_{k+2} + \cdots + l \cdot P_l.$$

P_i: Probability of i 1's in string belong to Schema H

For the non-defined positions:

$b_1^* b_2^* \cdots b_{l-k}^* = * * * \cdots *$ ($l-k$)

($l-k$)

total # of distinct strings of H: $\underbrace{1 1 \cdots 1}_{k}\ \underbrace{* * \cdots *}_{l-k} = 2^{l-k}$.

0 1's: $\binom{l-k}{0} = 1$

1 1's: $\binom{l-k}{1} = l-k$

2 1's: $\binom{l-k}{2}$

3 1's: $\binom{l-k}{3}$

$\vdots$

$l-k$ 1's: $\binom{l-k}{l-k} = 1$

$$\therefore \sum_{i=k}^{i=l} i \cdot P_i = \left| \sum_{i=k}^{i=l} i \cdot \frac{\binom{l-k}{i-k}}{2^{l-k}} \right|$$

$$\therefore \mathbb{E}(f(H)) = \sum_{i=k}^{l} i \cdot \frac{\binom{l-k}{i-k}}{2^{l-k}}$$

7. When is union of two schemas also a schema?

Union of schemas:

Ex/ $\{0*\} \cup \{1*\} \Rightarrow \{01, 00\} \cup \{10, 11\} \Rightarrow$ **

$\{01\} \cup \{10\} \Rightarrow \{01, 10\} \not\Rightarrow$ ** s/t

** includes 11 and 00 as well

(S_1 and S_2)

i) Union of two schemas is also a schema if $S_1 \subset S_2$ or $S_2 \subset S_1$. If $S_1 \not\subset S_2$ and $S_2 \not\subset S_1$, then S_1 and S_2 can differ at most at one position in the schemas:

S_1: $a_0 a_1 \cdots a_n$

S_2: $b_0 b_1 \cdots b_n$.

Can only exist at most one i

i.e. $\boxed{a_i \neq b_i}$.

Ex: S_1: 0101 ← one position differs

② S_2: 0100

$S_1 \cup S_2 \rightarrow$ 010*

S_1: 0**1*

② S_2: 0**0*

$S_1 \cup S_2 \rightarrow$ 0***

(S_1, S_2)

ii) Intersection of two schemas is the a schema when there does not exist a position in S_1 and S_2 where S_1 and S_2 are both defined in that position but differ in value.

S_1: $a_0 a_1 \cdots a_n$

S_2: $b_0 b_1 \cdots b_n$

There exists a_i and b_i where $a_i \neq b_i$ and $a_i \neq *$ and $b_i \neq *$.

If there doesn't exist a position where S_1, S_2 both defined and do not differ, then:

For each position i in the schemas:

① $a_i = b_i$; $a_i, b_i \in \{0, 1\}$

The resulting schema has that value at position i.

② $a_i = *$, $b_i \neq *$ or $a_i \neq *$, $b_i = *$:

The resulting schema has defined value at pos. i.

③ $a_i = *$, $b_i = *$:

The resulting schema has $*$ at position i.

(S_1, S_2)

iii) The difference of two schemas is also a schema if $S_1 \subset S_2$ and there exists only one position in S_1 and S_2 where they differ:

S_1: $a_0 a_1 \cdots a_n$ } $S_1 \subset S_2$

S_2: $b_0 b_1 \cdots b_n$ }

- only one position a_i and b_i where $a_i \neq b_i$.

8. Yes. Take the case of $n=1$ 3-bit string.

$n=1$ { 010 ⟶ schemes: ① 010 ② *10 ③ *** ④ 0*0 ⑤ 01* ⑥ 0** ⑦ *1* ⑧ **0

∴ 8 - different schemes

∴ $8 = 1 \times 2^3 = 8$.

∴ $n \times 2^l$ different schemes when $n=1$, $l=3$ for string 010.