Unicamp - Instituto de Computação

MC886 B / MO416 A

Introdução à Inteligência Artificial

Prof. Dr. Jacques Wainer

Exercício 7

Aluno:
George Gigilas Junior - 216741

Introdução

Este exercício corresponde ao exercício 21.1 do livro texto, cujo enunciado está disponível juntamente com a atividade. Este arquivo corresponde às minhas soluções para o problema.

Solução

Enunciado - Implementar um agente de aprendizado passivo em um ambiente simples, como o mundo 4x3. Para o caso de um modelo de ambiente inicialmente desconhecido, compare o desempenho de aprendizado dos algoritmos direct utility estimation, TD e ADP. Faça a comparação para a política ótima e para diversas políticas aleatórias. Para qual delas a utilidade converge mais rápido?

Para resolver esse exercício, utilizei o simulador para um ambiente 4x3 feito no Exercício 6. Além disso, criei um código em Python que implementa os três agentes pedidos: o agente que utiliza o algoritmo Direct Utility Estimation, o que utiliza o TD e o que utiliza o ADP.

Para verificar quando o algoritmo convergiu, eu comparei se os elementos da matriz de utilidade estão próximos dos elementos da matriz de utilidade da iteração anterior. Se todos os elementos diferem em menos de um valor limite (definido no código), eu considerei que a matriz convergiu. Um detalhe importante é que esse valor limite não poderia ser muito pequeno, senão o direct utility estimation não convergia em tempo hábil, então eu o defini como sendo 0,0001 (definido empiricamente). Além disso, vale ressaltar que, pelo fato de a execução dos algoritmos ser não-determinística (até certo ponto), às vezes os primeiros resultados já parecem convergir. Para contornar isso, estipulei que os algoritmos deveriam fazer, no mínimo, 200 iterações. Assim como no último exercício, defini γ = 1,0.

Segue o código (a discussão está abaixo dele):
OBS: o código também pode ser encontrado no meu repositório no seguinte link: [MC886/Projeto 7 at main · GeorgeJuniorGG/MC886 (github.com)]

```
from collections import defaultdict
from environment_simulator import mdp

# Calcula pelo metodo do ponto fixo
def policyEvaluation(policy, utility, gamma, rewards, Prob, states,
reverseStates):
    n = 25                          # Numero de iteracoes que
determinei empiricamente para a aproximacao
    for i in range (n):
        for j in range(11):
```

```
            if(policy[j] == 0):
                direction = "left"
            elif(policy[j] == 1):
                direction = "right"
            elif(policy[j] == 3):
                direction = "down"
            else:
                direction = "up"
            utilityAux = rewards[states[j]]

            for dic in Prob[states[j]][direction]:
                newS = dic
                prob = Prob[states[j]][direction][newS]
                utilityAux += gamma * prob *
utility[reverseStates[newS]]

            utility[j] = utilityAux

    return utility

def directUtilityEstimation (policy, environment, states,
reverseStates, limite):

    utilidade = [0, 0, 0,
                 0, 0,
                 0, 0, 0,
                 0, 0, 0]

    happened =  [1, 1, 1,     # Vetor que indica se ja foi encontrado
algum valor de utilidade para os estados
                 1, 1,
                 1, 1, 1,
                 1, 1, 1]
    convergiu = [0, 0, 0,
                 0, 0,
                 0, 0, 0,
                 0, 0, 0]
    conv = False

    for i in range (200000):
        count = 0
        hist_estados = []         # Guarda os estados visitados
        hist_recompensas = []     # Guarda as recompensas recebidas
```

```
        utilidadeAux = [10, 10, 10, # Valores dummies para indicar que
nao foi setado nenhum valor
                10, 10,
                10, 10, 10,
                10, 10, 10]

        ocorrencias =  [0, 0, 0,
                        0, 0,
                        0, 0, 0,
                        0, 0, 0]          # Conta se algum estado foi
alcancado mais de uma vez

        estado = (1,1)
        recompensa = -0.04

        hist_estados.append(estado)
        hist_recompensas.append(recompensa)
        ocorrencias[reverseStates[estado]] += 1

        while (estado not in environment.terminalStates):
            aux = policy[reverseStates[estado]]
            if(aux == 0):
                direction = "left"
            elif (aux == 1):
                direction = "right"
            elif (aux == 3):
                direction = "down"
            else:
                direction = "up"

            estado, recompensa = environment.getDestination(estado,
direction)
            hist_estados.append(estado)
            hist_recompensas.append(recompensa)
            ocorrencias[reverseStates[estado]] += 1
            count += 1
            if count > 1000:
                return -1

        for j in range(len(hist_estados)):
            if(utilidadeAux[reverseStates[hist_estados[j]]] == 10):
```

```
                utilidadeAux[reverseStates[hist_estados[j]]] =
sum(hist_recompensas[j:])
            else:
                utilidadeAux[reverseStates[hist_estados[j]]] +=
sum(hist_recompensas[j:])

        for j in range(11):
            if(utilidadeAux[j] == 10):
                utilidadeAux[j] = 0
            elif(ocorrencias[j] > 1):
                utilidadeAux[j] = utilidadeAux[j] / ocorrencias[j]

        for j in range(11):
            if(utilidadeAux[j] != 0):
                if(happened[j] == 0):
                    utilidade[j] = (utilidade[j] + utilidadeAux[j]) /2
                else:
                    utilidade[j] = utilidadeAux[j]
                    happened[j] = 0
        conv = True
        for j in range(len(utilidade)):
            if(abs(convergiu[j] - utilidade[j]) > limite):
                conv = False
            convergiu[j] = utilidade[j]
        if (conv == True and i>200):
            print("DUE convergiu em: " + str(i))
            break

    return utilidade

def TD (policy, environment, states, reverseStates, gamma, limite):
    utilidade = [0, 0, 0,
                 0, 0,
                 0, 0, 0,
                 0, 0, 0]

    N_estado = [0, 0, 0,
                0, 0,
                0, 0, 0,
                0, 0, 0]
    convergiu = [0, 0, 0,
                 0, 0,
                 0, 0, 0,
```

```
                0, 0, 0]
    conv = False

    # De acordo com o livro
    alfa = lambda n: 60./(59+n)

    for i in range(200000):
        count = 0
        estado = (1,1)
        recompensa = -0.04
        prevRecompensa = 10
        prevEstado = -1

        if(N_estado[reverseStates[estado]] == 0):
            utilidade[reverseStates[estado]] = recompensa

        if(prevEstado != -1):
            N_estado[reverseStates[prevEstado]] += 1
            utilidade[reverseStates[prevEstado]] +=
alfa(N_estado[reverseStates[prevEstado]]) * (prevRecompensa + gamma *
utilidade[reverseStates[estado]] -
utilidade[reverseStates[prevEstado]])

        prevEstado = estado
        prevRecompensa = recompensa

        while(estado not in environment.terminalStates):

            aux = policy[reverseStates[estado]]
            if(aux == 0):
                direction = "left"
            elif (aux == 1):
                direction = "right"
            elif (aux == 3):
                direction = "down"
            else:
                direction = "up"

            estado, recompensa = environment.getDestination(estado,
direction)

            if(N_estado[reverseStates[estado]] == 0):
                utilidade[reverseStates[estado]] = recompensa
```

```
            if(prevEstado != -1):
                N_estado[reverseStates[prevEstado]] += 1
                utilidade[reverseStates[prevEstado]] +=
alfa(N_estado[reverseStates[prevEstado]]) * (prevRecompensa + gamma *
utilidade[reverseStates[estado]] -
utilidade[reverseStates[prevEstado]])

            if (estado not in environment.terminalStates):
                prevEstado = estado
                prevRecompensa = recompensa

            else:
                prevEstado = -1
                prevRecompensa = 10

            count += 1
            if count > 1000:
                return -1

        conv = True
        for j in range(len(utilidade)):
            if(abs(convergiu[j] - utilidade[j]) > limite):
                conv = False
            convergiu[j] = utilidade[j]
        if (conv == True and i>200):
            print("TD convergiu em: " + str(i))
            break

    return utilidade

def ADP (policy, environment, states, reverseStates, gamma, limite):
    utilidade = [0, 0, 0,
                 0, 0,
                 0, 0, 0,
                 0, 0, 0]

    convergiu = [0, 0, 0,
                 0, 0,
                 0, 0, 0,
                 0, 0, 0]
    conv = False
```

```
    visitados = set()
    N_prevEstado_prevDir = defaultdict(int)
    N_estado_prevEstado_prevDir = defaultdict(int)

    P = {(1,1):{"left": {},
                "right": {},
                "up": {},
                "down": {}},
         (1,2):{"left": {},
                "right": {},
                "up": {},
                "down": {}},
        (1,3): {"left": {},
                "right": {},
                "up": {},
                "down": {}},
        (2,1): {"left": {},
                "right": {},
                "up": {},
                "down": {}},
        (2,2): {"left": {},
                "right": {},
                "up": {},
                "down": {}},
        (2,3): {"left": {},
                "right": {},
                "up": {},
                "down": {}},
        (3,1): {"left": {},
                "right": {},
                "up": {},
                "down": {}},
        (3,2): {"left": {},
                "right": {},
                "up": {},
                "down": {}},
        (3,3): {"left": {},
                "right": {},
                "up": {},
                "down": {}},
        (4,1): {"left": {},
                "right": {},
                "up": {},
```

```
                "down": {}},
        (4,2): {"left": {},
                "right": {},
                "up": {},
                "down": {}},
        (4,3): {"left": {},
                "right": {},
                "up": {},
                "down": {}}}

    R = {(1,1): 0, (1,2): 0, (1,3): 0,
         (2,1): 0, (2,2): 0, (2,3): 0,
         (3,1): 0, (3,2): 0, (3,3): 0,
         (4,1): 0, (4,2): 0, (4,3): 0}

    for i in range(200000):
        count = 0
        estado = (1,1)
        prevEstado = -1
        recompensa = -0.04
        prevDirecao = -1

        if (estado not in visitados):
            visitados.add(estado)
            utilidade[reverseStates[estado]] = recompensa
            R[estado] = recompensa

        if(prevEstado != -1):
            N_prevEstado_prevDir[(prevEstado, prevDirecao)] += 1
            N_estado_prevEstado_prevDir[(estado, prevEstado,
prevDirecao)] += 1
            # Para cada t tal que N estado|prevEstado prevDir [t,
prevEstado, prevDir] seja diferente de 0
            for j in [estAtual for (estAtual, estAnt, dirAnt),
ocorrencias in N_estado_prevEstado_prevDir.items() if ocorrencias != 0
and (estAnt, dirAnt) == (prevEstado, prevDirecao)]:
                P[prevEstado][prevDirecao][j] =
N_estado_prevEstado_prevDir[(j, prevEstado, prevDirecao)] /
N_prevEstado_prevDir[(prevEstado, prevDirecao)]


        utilidade = policyEvaluation(policy, utilidade, gamma, R, P,
states, reverseStates)
```

```
        prevEstado = estado
        aux = policy[reverseStates[estado]]
        if(aux == 0):
            prevDirecao = "left"
        elif (aux == 1):
            prevDirecao = "right"
        elif (aux == 3):
            prevDirecao = "down"
        else:
            prevDirecao = "up"

        while(estado not in environment.terminalStates):

            aux = policy[reverseStates[estado]]
            if(aux == 0):
                direction = "left"
            elif (aux == 1):
                direction = "right"
            elif (aux == 3):
                direction = "down"
            else:
                direction = "up"

            estado, recompensa = environment.getDestination(estado,
direction)

            if (estado not in visitados):
                visitados.add(estado)
                utilidade[reverseStates[estado]] = recompensa
                R[estado] = recompensa

            if(prevEstado != -1):
                N_prevEstado_prevDir[(prevEstado, prevDirecao)] += 1
                N_estado_prevEstado_prevDir[(estado, prevEstado,
prevDirecao)] += 1
                # Para cada t tal que N estado|prevEstado prevDir [t,
prevEstado, prevDir] seja diferente de 0
                for j in [estAtual for (estAtual, estAnt, dirAnt),
ocorrencias in N_estado_prevEstado_prevDir.items() if ocorrencias != 0
and (estAnt, dirAnt) == (prevEstado, prevDirecao)]:
```

```
                    P[prevEstado][prevDirecao][j] =
N_estado_prevEstado_prevDir[(j, prevEstado, prevDirecao)] /
N_prevEstado_prevDir[(prevEstado, prevDirecao)]


            utilidade = policyEvaluation(policy, utilidade, gamma, R,
P, states, reverseStates)
            if (estado not in environment.terminalStates):
                prevEstado = estado
                aux = policy[reverseStates[estado]]
                if(aux == 0):
                    prevDirecao = "left"
                elif (aux == 1):
                    prevDirecao = "right"
                elif (aux == 3):
                    prevDirecao = "down"
                else:
                    prevDirecao = "up"
            else:
                prevEstado = -1
                prevDirecao = -1

        conv = True
        for j in range(len(utilidade)):
            if(abs(convergiu[j] - utilidade[j]) > limite):
                conv = False
            convergiu[j] = utilidade[j]
        if (conv == True and i > 200):
            print("ADP convergiu em: " + str(i))
            break

    return utilidade

def printPolicy (policy, reverseStates):
    for i in range(3,0,-1):
        for j in range(1,5):
            if (i, j) == (2,2):
                char = '#'
            elif (i,j) == (3,4) or (i,j) == (2,4):
                char = "T"
            else:
                aux = policy[reverseStates[(j, i)]]
                if(aux == 0):
```

```
                    char = "L"
                elif (aux == 1):
                    char = "R"
                elif (aux == 3):
                    char = "D"
                else:
                    char = "U"
            print(char, end=" ")
        print("")
    print("L = Left, R = Right, U = Up, D = Down, # = Hole, T =
Terminal State")

def printUtility (utility, reverseStates):
    for i in range(3,0,-1):
        for j in range(1,5):
            if (i, j) == (2,2):
                print("  #  ", end="")
            else:
                char = utility[reverseStates[(j,i)]]
                print("%.2f" %char,end=" ")
        print("")

environment = mdp((1,1))

# Valor de gamma definido com base no que foi feito no ultimo exercicio
gamma = 1

# Mapeamento de identificadores de estados
# OBS: o estado (2,2) nao esta presente pois ele corresponde a uma
parede
states =   {0: (1,1), 1: (1,2), 2: (1,3),
            3: (2,1), 4: (2,3),
            5: (3,1), 6: (3,2), 7:(3,3),
            8: (4,1), 9: (4,2), 10: (4,3)}
reverseStates ={(1,1): 0, (1,2): 1, (1,3):2,
                (2,1): 3, (2,3): 4,
                (3,1): 5, (3,2): 6, (3,3): 7,
                (4,1): 8, (4,2): 9, (4,3): 10}

# Vetor de acoes
# 0 corresponde a ir para a esquerda, 1 para a direita, 2 para cima e 3
para baixo
actions = [0, 1, 2, 3]
```

```
# Vetor de política ótima (encontrado no exercicio anterior)
# Em cada posição, temos a politica para cada estado: 0 corresponde a
ir para a esquerda, 1 para a direita, 2 para cima e 3 para baixo
# OBS: Nao existe politica para os estados terminais porque eles ja sao
o final do problema
optimalPolicy = [2, 2, 1,
                 0,    1,
                 0, 2, 1,
                 0, 0, 0]

# Vetores de políticas aleatórias (apenas 2, como definido no
enunciado)
randomPolicy1 = [1, 3, 3,
                 1,    0,
                 2, 2, 1,
                 0, 0, 0]
randomPolicy2 = [1, 2, 1,
                 1,    1,
                 2, 1, 3,
                 0, 0, 0]

# Vetores de utilidade: um para cada algoritmo, para cada politica
usada
utility1a = [0, 0, 0,
           0, 0,
           0, 0, 0,
           0, -1, 1]
utility1b = [0, 0, 0,
           0, 0,
           0, 0, 0,
           0, -1, 1]
utility1c = [0, 0, 0,
           0, 0,
           0, 0, 0,
           0, -1, 1]

utility2a = [0, 0, 0,
           0, 0,
           0, 0, 0,
           0, -1, 1]
utility2b = [0, 0, 0,
           0, 0,
```

```
           0, 0, 0,
           0, -1, 1]
utility2c = [0, 0, 0,
           0, 0,
           0, 0, 0,
           0, -1, 1]

utility3a = [0, 0, 0,
           0, 0,
           0, 0, 0,
           0, -1, 1]
utility3b = [0, 0, 0,
           0, 0,
           0, 0, 0,
           0, -1, 1]
utility3c = [0, 0, 0,
           0, 0,
           0, 0, 0,
           0, -1, 1]

# Chamadas dos metodos para as 3 politicas
limite = 0.0002
utility1a = directUtilityEstimation(optimalPolicy, environment, states,
reverseStates, limite)
utility2a = TD(optimalPolicy, environment, states, reverseStates,
gamma, limite)
utility3a = ADP(optimalPolicy, environment, states, reverseStates,
gamma, limite)

utility1b = directUtilityEstimation(randomPolicy1, environment, states,
reverseStates, limite)
utility2b = TD(randomPolicy1, environment, states, reverseStates,
gamma, limite)
utility3b = ADP(randomPolicy1, environment, states, reverseStates,
gamma, limite)

utility1c = directUtilityEstimation(randomPolicy2, environment, states,
reverseStates, limite)
utility2c = TD(randomPolicy2, environment, states, reverseStates,
gamma, limite)
utility3c = ADP(randomPolicy2, environment, states, reverseStates,
gamma, limite)
```

```
# Saída
print("Política Ótima")
printPolicy(optimalPolicy, reverseStates)
print("")
print("Direct Utility Estimation")
printUtility(utility1a, reverseStates)
print("TD")
printUtility(utility2a, reverseStates)
print("ADP")
printUtility(utility3a, reverseStates)
print("==================================")
print("Política Aleatória 1")
printPolicy(randomPolicy1, reverseStates)
print("")
print("Direct Utility Estimation")
printUtility(utility1b, reverseStates)
print("TD")
printUtility(utility2b, reverseStates)
print("ADP")
printUtility(utility3b, reverseStates)
print("==================================")
print("Política Aleatória 2")
printPolicy(randomPolicy2, reverseStates)
print("")
print("Direct Utility Estimation")
printUtility(utility1c, reverseStates)
print("TD")
printUtility(utility2c, reverseStates)
print("ADP")
printUtility(utility3c, reverseStates)
```

As políticas utilizadas foram:
OBS: vale ressaltar que eu defini as políticas como sendo um caminho do estado final para um estado terminal, evitando loops. Caso contrário, o agente ficava preso trocando entre dois estados ou até mesmo entre mais estados, sem chegar a um estado terminal.

- Política Ótima:
  R R R T
  U # U T
  U L L L
  L = Left, R = Right, U = Up, D = Down, # = Wall, T = Terminal State

- Política Aleatória 1:
  D L R T
  D # U T
  R R U L
  L = Left, R = Right, U = Up, D = Down, # = Wall, T = Terminal State

- Política Aleatória 2:
  R R D T
  U # R T
  R R U L
  L = Left, R = Right, U = Up, D = Down, # = Wall, T = Terminal State

A partir desse código, obtive os seguintes resultados:

- Política Ótima:
  - Direct Utility Estimation - convergiu em 14085 iterações
    0.88 0.92 0.96 1.00
    0.84 # 0.90 -1.00
    0.80 0.74 0.00 0.00

  - TD - convergiu em 3933 iterações
    0.81 0.86 0.90 1.00
    0.76 # 0.71 -1.00
    0.71 0.65 0.00 0.00

  - ADP - convergiu em 214 iterações
    0.82 0.88 0.93 1.00
    0.77 # 0.72 -1.00
    0.72 0.67 0.00 0.00

- Política Aleatória 1:
  - Direct Utility Estimation - convergiu em 25169 iterações
    0.00 0.00 0.96 1.00
    0.62 # 0.92 -1.00
    0.80 0.84 0.88 0.71

  - TD - convergiu em 21203 iterações
    0.00 0.00 0.92 1.00
    0.43 # 0.72 -1.00
    0.48 0.54 0.60 0.29

  - ADP - convergiu em 1320 iterações
    0.00 0.00 0.92 1.00
    0.40 # 0.64 -1.00
    0.45 0.51 0.56 0.32

- Política Aleatória 2:
    - Direct Utility Estimation - convergiu em 9402 iterações
      -1.16 -1.12 -1.08 1.00
      -1.21 # -1.04 -1.00
      -1.16 -1.12 -1.08 -1.08

    - TD - convergiu em 3510 iterações
      -0.90 -0.86 -0.87 1.00
      -0.96 # -1.03 -1.00
      -1.16 -1.13 -1.08 -1.10

    - ADP - convergiu em 208 iterações
      -0.98 -0.91 -0.86 1.00
      -1.03 # -1.03 -1.00
      -1.18 -1.15 -1.10 -1.12

A partir desses resultados, eu concluí que o ADP é o algoritmo que converge mais rápido, seguido pelo TD (que consome menos memória). O Direct Utility Estimation é, na maioria das vezes, o que demora mais para convergir e, dependendo do limite considerado para convergir, às vezes ele nem converge em tempo hábil. Além dessa execução do programa, eu fiz algumas outras para testar e essa tendência se confirmou. Lembrando que, a cada execução, o número de iterações até convergir muda, porque o estado para o qual o agente vai nem sempre é o estado para o qual ele fez a ação de ir.

Por fim, em relação aos valores encontrados, notam-se algumas peculiaridades. De modo geral, os resultados obtidos pelo ADP e pelo TD são relativamente próximos. Porém, parte dos valores obtidos pelo Direct Utility Estimation diferem consideravelmente dos valores obtidos pelos outros algoritmos. A principal exceção a isso se trata dos estados mais próximos do estado terminal ao qual a política visa alcançar. Nos estados próximos desse estado terminal, os valores são relativamente próximos para os três algoritmos.