Laboratório 10 - Programação Dinâmica

Marcelo Buga Martins da Silva

CT-213 - Professor Marcos Ricardo Omena de Albuquerque Máximo

14/06/2021

1 Introdução

O propósito desse laboratório foi a implementar algoritmos de programação dinâmica no contexto de solução de um Processo Decisório de Markov (Markov Decision Process - MDP). Os algoritmos implementados foram avaliação de política (*policy evaluation*), iteração de política (*policy iteration*) e iteração de valor (*value iteration*), tendo como objetivo avaliar e determinar politicas ótimas em um grid world, sendo um problema de Aprendizado por Reforço (*Reinforcement Learning - RL*) no caso em que o modelo do MDP é conhecido.

O problema consiste em um grid world com 5 ações possíveis:

STOP: continua parado na mesma posição.

UP: move-se uma célula para cima no tabuleiro.

RIGHT: move-se uma célula para direita no tabuleiro.

DOWN: move-se uma célula para baixo no tabuleiro.

LEFT: move-se uma célula para esquerda no tabuleiro.

Considera-se que a ação STOP sempre é executada com perfeição, i.e. com probabilidade 1, o agente permanece na mesma posição após executar essa ação. Já as demais ações tem uma probabilidade p_c de serem

executadas corretamente. Se a ação não for executada corretamente, o resultado de uma das demais ações acontece com igual probabilidade, i.e. com $\frac{1-p_c}{4}$. No grid world, há obstáculos, que ocupam algumas células do grid. Se um determinado movimento for levar o agente para um obstáculo, então o agente permanece na sua posição. Ademais, os limites do grid são também barreiras. Além dessas questões, o MDP tem fator de desconto γ e a recompensa é -1 para cada instante que o agente passa em uma célula que não é a objetivo. Há uma única célula objetivo no grid, onde o agente recebe recompensa 0.

Esse relatório apresentará uma breve descrição da implementação dos algoritmos, bem como a comparação entre Grid Worlds distintos, com diferentes valores de probabilidade para a tomada correta de decisão e valores de conceito de desconto (γ).

2 Avaliação de Política

Para a avaliação de política, iterou-se por cada estado (posições do grid), cada ação e cada possível próximo estado para encontrar seu valor. Foi utilizado método de solução iterativa do sistema de equações lineares dados pela equação de Bellman (1) para que, dado um valor inicial qualquer, o algoritmo convergisse para o valor adequado, como na equação 2.

$$v_\pi(s) = \sum_{a\in A} \pi(a|s) r(s,a) + \gamma \sum_{a\in A} \sum_{s'\in S} \pi(a|s) p(s'|s,a) v_\pi(s') \tag{1}$$

$$v_{k+1}(s) = \sum_{a\in A} \pi(a|s) r(s,a) + \gamma \sum_{a\in A} \sum_{s'\in S} \pi(a|s) p(s'|s,a) v_k(s') \tag{2}$$

O algoritmo de avaliação de política foi implementado de forma síncrona não vetorizada, tendo critérios de parada um número máximo de iterações e um ε dado pela diferença entre o valor da iteração atual e o da iteração anterior. Seguiu-se um modelo de implementação muito similar ao da função *greedy_policy*, já disponibilizado no código base.

Esse algoritmo, quando testado, foi o de execução mais lenta, tendo sido percebida uma convergência lenta do valor quando testado para política aleatória, de forma que, no teste, todas as 10000 iterações foram feitas sem ter sido atingido o critério de parada para $\varepsilon = 10^{-5}$.

3 Iteração de valor

O algoritmo de iteração de valor se baseia em iterar diretamente sobre a função valor de acordo com a equação de otimalidade de Bellman, como mostra a equação 3. Ela foi implementada de forma similar ao que foi descrito para a Avaliação de Política, tendo como maior diferença que é tomado o valor máximo entre os

valores esperados para cada ação.

$$v_{k+1}(s) = max_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v_k(s') \right) \tag{3}$$

Esse algoritmo também foi implementado de forma síncrona não vetorizada, tendo critérios de parada um número máximo de iterações e um ε dado pela diferença entre o valor da iteração atual e o da iteração anterior. Esse algoritmo convergiu muito mais rapidamente, atendendo o critério de parada para $\varepsilon = 10^{-5}$ em poucas iterações para o caso teste (menos de 50).

4 Iteração de Política

O algoritmo de iteração de política basicamente avalia o valor de uma política sem necessariamente que esse valor convirja, mas que seja estimado. Depois, usa-se do algoritmo *greedy* para refinar a política. Essas ações são realizadas até o atendimento do critério de parada número máximo de iterações e um ε dado pela diferença entre o valor da iteração atual e o da iteração anterior.

Para essa função, foi utilizada a função *policy_evalution* descrita no item anterior para aproximar o valor da política e a função *greedy_policy* para o refinamento da política. Essa função também convergiu rapidamente para o caso de teste, sendo necessárias menos de 20 iterações.

5 Comparação entre Grid Worlds diferentes

Foi executado o código para comparar a iteração de política, de valor e a avaliação de política entre um Grid World, sendo o primeiro com $p_c = 1$ e $\gamma = 1$ e o segundo com $p_c = 0,8$ e $\gamma = 0,98$. Os resultados são apresentados pelas Figuras 1 e 2:

A princípio, percebe-se que as implementações são coerentes, dado que os resultados de Policy Iteration e Value Iteration são idênticos em cada um dos Grid Worlds. Em termos da diferença entre eles, mais expressivamente nota-se a que o valor associado a política aleatória é substancialmente menor para o primeiro caso. Isso era esperado, visto que a tendência quando $\gamma < 1$ é diminuir muito o valor para estados muito além do atual, o que é expressivo quando a política é aleatória. O valor ser ligeiramente menor no caso de Value Iteration e Policy Iteration também era esperado, visto que quando se determina uma probabilidade p_c diferente de 1, acontecerão tomadas erradas de decisão que diminuirão o valor associado a cada casa.

```
Evaluating random policy, except for the goal state, where policy always executes stop:
Value function:
[  -384.09,  -382.73,  -381.19,    *    ,  -339.93,  -339.93]
[  -380.45,  -377.91,  -374.65,    *    ,  -334.92,  -334.93]
[  -374.34,  -368.82,  -359.85,  -344.88,  -324.92,  -324.93]
[  -368.76,  -358.18,  -346.03,    *    ,  -289.95,  -309.94]
[    *    ,  -344.12,  -315.05,  -250.02,  -229.99,    *    ]
[  -359.12,  -354.12,    *    ,  -200.01,  -145.00,     0.00]
Policy:
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[    *    ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,    *    ]
[  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ,    S    ]
-----------------------------------------------------------------


Value iteration:
Value function:
[   -10.00,    -9.00,    -8.00,    *    ,    -6.00,    -7.00]
[    -9.00,    -8.00,    -7.00,    *    ,    -5.00,    -6.00]
[    -8.00,    -7.00,    -6.00,    -5.00,    -4.00,    -5.00]
[    -7.00,    -6.00,    -5.00,    *    ,    -3.00,    -4.00]
[    *    ,    -5.00,    -4.00,    -3.00,    -2.00,    *    ]
[    -7.00,    -6.00,    *    ,    -2.00,    -1.00,     0.00]
Policy:
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    RD   ,    R    ,    D    ,    DL   ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    RD   ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,   SURD  ]
-----------------------------------------------------------------
```

```
Policy iteration:
Value function:
[   -10.00,    -9.00,    -8.00,    *    ,    -6.00,    -7.00]
[    -9.00,    -8.00,    -7.00,    *    ,    -5.00,    -6.00]
[    -8.00,    -7.00,    -6.00,    -5.00,    -4.00,    -5.00]
[    -7.00,    -6.00,    -5.00,    *    ,    -3.00,    -4.00]
[    *    ,    -5.00,    -4.00,    -3.00,    -2.00,    *    ]
[    -7.00,    -6.00,    *    ,    -2.00,    -1.00,     0.00]
Policy:
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    RD   ,    R    ,    D    ,    DL   ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    RD   ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,   SURD  ]
------------------------------------------------------------
```

Figura 1: Saída para $p_c = 1$ e $\gamma = 1$

```
Evaluating random policy, except for the goal state, where policy always executes stop:
Value function:
[   -47.19,   -47.11,   -47.01,    *    ,   -45.13,   -45.15]
[   -46.97,   -46.81,   -46.60,    *    ,   -44.58,   -44.65]
[   -46.58,   -46.21,   -45.62,   -44.79,   -43.40,   -43.63]
[   -46.20,   -45.41,   -44.42,    *    ,   -39.87,   -42.17]
[    *    ,   -44.31,   -41.64,   -35.28,   -32.96,    *    ]
[   -45.73,   -45.28,    *    ,   -29.68,   -21.88,     0.00]
Policy:
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[    *    ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,    *    ]
[  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ,    S    ]
-----------------------------------------------------------------


Value iteration:
Value function:
[   -11.65,   -10.78,    -9.86,    *    ,    -7.79,    -8.53]
[   -10.72,    -9.78,    -8.78,    *    ,    -6.67,    -7.52]
[    -9.72,    -8.70,    -7.59,    -6.61,    -5.44,    -6.42]
[    -8.70,    -7.58,    -6.43,    *    ,    -4.09,    -5.30]
[    *    ,    -6.43,    -5.17,    -3.87,    -2.76,    *    ]
[    -8.63,    -7.58,    *    ,    -2.69,    -1.40,     0.00]
Policy:
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    RD   ,    D    ,    D    ,    R    ,    D    ,    D    ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    D    ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,    S    ]
-----------------------------------------------------------------
```

```
Policy iteration:
Value function:
[   -11.65,   -10.78,    -9.86,    *    ,    -7.79,    -8.53]
[   -10.72,    -9.78,    -8.78,    *    ,    -6.67,    -7.52]
[    -9.72,    -8.70,    -7.59,    -6.61,    -5.44,    -6.42]
[    -8.70,    -7.58,    -6.43,    *    ,    -4.09,    -5.30]
[    *    ,    -6.43,    -5.17,    -3.87,    -2.76,    *    ]
[    -8.63,    -7.58,    *    ,    -2.69,    -1.40,     0.00]
Policy:
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    RD   ,    D    ,    D    ,    R    ,    D    ,    D    ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    D    ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,    S    ]
------------------------------------------------------------
```

Figura 2: Saída para $p_c = 0,8$ e $\gamma = 0,98$