Inteligência Artificial - TP2
Aprendizado por Reforço

Guilherme Torres
Departamento de Ciência da Computação - UFMG

1 Introdução

O objetivo do trabalho que segue foi implementar um algoritmo de aprendizado por reforço, para encontrar a política ótima para um modelo de decisões de Markov em um ambiente parcialmente observável. Para isso, foi usado o algoritmo Q-learning[1], que faz passeios aleatórios nos estados e atualiza a função qualidade $Q(s, a)$ para cada um dos estados.

O programa recebe como entrada um mapa como uma matriz de caracteres e o agente pode se movimentar em quatro direções. Existem posições terminais (situações onde o agente termina de se movimentar) e não-terminais. Ao terminar, o programa produz como saída um arquivo *q.txt* com as ações recomendadas e função qualidade para cada uma das posições não terminais, e *pi.txt*, um mapa com as direções recomendadas pelo algoritmo em cada situação.

O programa foi implementado na linguagem Rust (rustc versão 1.21.0, cargo versão 0.22.0). O código fonte está na pasta src/. Para compilar, segue o script *compila.sh* e o script *qlearning.sh* compila e executa o programa, tendo como parâmetros, respectivamente, o mapa de entrada, a taxa de aprendizado α, a taxa de desconto γ e o número de iterações do algoritmo.

As decisões tomadas na implementação e o algoritmo estão explicitados na seção seguinte.

2 Implementação e funcionamento

2.1 Algoritmo

O algoritmo de Q-learning implementado pode ser descrito nos seguintes passos:

1. É definido o valor inicial de $Q(s, a)$ para todas as ações a e cada estado s como zero
2. A posição do agente é marcada como uma posição não-terminal aleatória no mapa.
3. Uma ação aleatória é escolhida.
4. O valor de $Q(s, a)$ para o estado atual e a ação escolhida é atualizado.
5. O estado é atualizado.
6. Se o estado atual for um terminal, define novamente uma posição não-terminal aleatória.
7. Repete os itens 3 a 6 até atingir o número pré-definido de iterações.
8. Produz os arquivos de saída.

2.2 Atualização da função Q(s, a)

A função $Q(s, a)$ de um estado é atualizada segundo a equação:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r(s') + \max_{a'} Q(s', a'))$$

sendo α a taxa de aprendizado, γ a taxa de desconto do valor do próximo estado, a a ação escolhida aleatoriamente, s' o estado alcançado ao se executar a ação a no estado s e $max_{a'}$ a ação que tem o maior valor de $Q(s', a')$.

2.3 Os valores α e γ

Durante os testes, os parâmetros α e γ foram mantidos estáticos a cada execução, ou seja, eles não se alteraram durante o curso do programa.

α teve o valor inicial de 0.2, e esteve sujeito a variações entre os testes por ser um fator primordial no controle do *Exploration vs. exploitation*. Basicamente, um valor mais alto nesse parâmetro indica que haverá uma convergência maior do programa às tendências que ele encontrar de início. Um valor menor abre espaço para mais exploração.

γ também tém uma participação nesse aspecto do algoritmo, porém menor, por isso o seu valor durante os testes foi fixo em 0.9.

2.4 Número de iterações

O número de iterações deve ser o suficiente para o programa apresentar uma convergência. Como foi implementado em Rust, uma linguagem compilada para arquivos binários, e, portanto, muito rápida, pôde-se dar ao luxo de fazer 100000 iterações em tempo hábil. Porém, para que fosse posível notar a diferença da velocidade de convergência entre as diferentes taxas de aprendizado, os testes na seção seguinte foram feitos com 3000 iterações, mas reforça-se que, a longo prazo, todas as taxas de aprendizado tendem a convergir para uma política em particular.

3 Testes

Os testes a seguir foram realizados em uma máquina com OS Linux Debian 9, processador Pentium G4400 dual-core (3.3Ghz) e 4x2 Gb de memória RAM. Todos eles rodaram em tempo hábil (menos de um segundo), com os parâmetros especificados acima. Os arquivos de teste podem ser encontrados na pasta maps/.

3.1 pacmaze-01-tiny.txt

Formato do teste:

```
7 14
##############
#-----0------#
#######-##-#-#
#-------&--#-#
#-##########&#
#------------#
##############
```

$\alpha = 0.2$

```
##############
#>>>>>0<<^^^<#
#######^##>#<#
#^^^vvv^&vv#<#
#<##########&#
#<v^vv^v^^v^v#
##############
```

$\alpha = 0.5$

```
##############
#>>>>>0<<<^^>#
#######^##v#<#
#^^vvv>^&>>#>#
#>##########&#
#v^^^^^^^^v^v#
##############
```

$\alpha = 0.8$

```
##############
#>>>>>0<<^^^^#
#######^##^#^#
#^^^^>>^&^^#^#
#>##########&#
#v^^^^^^^^^^>#
##############
```

$\alpha = 1.0$

```
##############
#>>>>>0<<<<<^#
#######^##>#>#
#^^^^^>^&^>#v#
#>##########&#
#v^^^^^^^^^^>#
##############
```

3.2 pacmaze-02-mid-sparse.txt

Formato do teste:

```
14 20
####################
#-------------#---#
#---&--&--&----#---#
#-------------#-#-#
#-###############-#-#
#----------------#-#
###################-#
#--------&---------#
#-&--########---&--#
#-----------------#
#-################-#
#-#-----#0#--------#
#----#----#--------#
####################
```

$\alpha = 0.2$

```
####################
#<<^>^^^^^v^^>>#^^>#
#v>>&>^&^^&<v<<#<v>#
#>^^>>^^^^^^<<>#>#>#
#>##############<#>#
#<v^^^vv^vv^^^^v>#v#
###################<#
#>^^^^^^^&^<v^>><^^#
#v&^^########^>v&^v#
#v^^>^>>^^^^v<^>^^>#
#>#################^#
#v#>^>>^#0#<^v^^^^>#
#>^<v#>>^^#v>v<v^vv#
####################
```

$\alpha = 0.5$

```
####################
#^^^^^^<>>^<^^^#^^^#
#<^^&v^&^^&v<^>#<v>#
#<vvvv>vvv<vvvv#>#>#
#<##############>#>#
#v^^^^^^^^^^^^^^>#>#
#################>#
#<^^vv^<^&^v^^^^^^>#
#<&>>########<<v&^>#
#<v^vv^^v^v^^vvvvv>#
#<################>#
#>#<^>vv#0#^^^^^^^>#
#v^v>#>>>^#vvvvvvv>#
####################
```

$\alpha = 0.8$

```
####################
#^^^^^^^^<^^^^>#^^^#
#<^v&^v&>^&v^^>#<v>#
#<vvvvvv<<vvvv>#>#>#
#>##############>#>#
#v^^^^^^^^^^^^^^>#>#
#################>#
#^^^^vvvv&^^^^^^^>#
#<&^>########<^^&^>#
#<vvv^^^^^^^^vvvvv>#
#<################>#
#>#>>>vv#0#^^^^^^^>#
#v^v^#>>>^#vvvvvvv>#
####################
```

$\alpha = 1.0$

```
####################
#^^^^^^^<^^^^^^#^^^#
#<^^&v^&><&v^^>#<v>#
#<vvvv>vvvvvvv>#>#>#
#>##############>#>#
#v^^^^^^^^^^^^^^>#>#
#################>#
#^^^^^v^v&^vv^^^^^>#
#<&v>########<^^&^>#
#<>vv^^^^^^^^vvvvv>#
#>################>#
#>#>>>>v#0#^^^^^^^>#
#v^v^#>>>^#vvvvvvv>#
####################
```

3.3 pacmaze-03-tricky.txt

Formato do teste:

```
11 20
####################
#----&--------&----#
#-&&&&&&&&-&&&&&&&&-#
#-------&-&--------#
#---&---&-&---&----#
#-------&-&--------#
#-----&-----&------#
#-------&-&--------#
#---&---&-&---&----#
#-------&0&--------#
####################
```

$\alpha = 0.2$

```
####################
#^^^^&^>>>>v<^&^><>#
#<&&&&&&&&v&&&&&&&&>#
#<>v<>><&^&>>><><>>#
#<>^&>>^&^&><^&^^>>#
#<<>v^>v&^&><<>^^^>#
#<>><<&v^<<^&v<<<>>#
#<<>><>^&v&^>^<<<<>#
#<>v&^>^&v&v><&><v>#
#vvvvvv^&0&v<vvvvv>#
####################
```

$\alpha = 0.5$

```
####################
#^^^^&^^^v>v^^&^><^#
#<&&&&&&&&>&&&&&&&&>#
#<>>^<>v&^&>v>^<<>>#
#<<^&><v&^&>^<&^^>>#
#<<>>>>>&v&>^<<^>>>#
#<<>^v&^>>>^&v<<>>>#
#<<v>><v&v&^vv<<>>>#
#<<v&v<v&v&^>v&vvv>#
#<vvv>vv&0&vvvvvvv>#
####################
```

$\alpha = 0.8$

```
####################
#^^^^&^^^^^<^<&^^^^#
#<&&&&&&&&v&&&&&&&&>#
#<<>>v>>&v&v^<<<v>#
#<<^&>^^&v&v^<&^<>>#
#<<>>^^<&v&^<<<>>>>#
#<<<^^&><<v^&vv^>>>#
#<<v<>>v&v&vvv<<>>>#
#<vv&>vv&v&><&vv>>#
#vvvvvv<&0&vv^vvvv>#
####################
```

$\alpha = 1.0$

```
####################
#^^^^&>^^^^^^^&>^^^#
#<&&&&&&&&^&&&&&&&&>#
#<<>>>vv&v&>>^<<<>>#
#<<^&^^<&^&>>^&^^>>#
#<<>>^^^&v&><^<^>>>#
#<<<^^&>>v><&^^<>>>#
#<<v^>>^&v&>>^<v>>>#
#<vv&vv^&v&^v^&vv>>#
#vvvvvvv&0&vv^vvvv>#
####################
```

Comparando os resultados dos testes, é notável o quanto o fator de aprendizado influencia na saída. Apesar de todos esses testes serem resultados intermediários, percebe-se que para taxas de aprendizado mais baixas, as decisões prioritárias são menos sólidas (em alguns momentos, andando em direção a recompensas mais baixas deliberadamente).

Isso não significa, porém, que uma taxa de aprendizado maior significa resultados melhores. Como é possível observar nas saídas com a taxa igual a 1, podem surgir tendências erradas para o agente e uma convergência prematura. Nesse caso, o agente ficaria preso às suas decisões erradas.

3.4 Resultados após 100000 iterações

Estes testes foram realizados com o valor da taxa de aprendizado 0.5.

pacmaze-01-tiny.txt -

```
##############
#>>>>>0<<<<<<#
#######^##^#^#
#^>>>>>^&>^#^#
#>##########&#
#v^^^^^^^^^^>#
##############
```

pacmaze-02-mid-sparse.txt -

```
####################
#^^^^^^^^^^^^^^#^^^#
#<^^&^v&v<&v^^>#<v>#
#<vvvvvvv<vvvv>#>#>#
#>###############>#>#
#v^^^^^^^^^^^^^^>#>#
##################>#
#^^^^vv^v&v^^^^^^^>#
#<&^>########<^^&^>#
#<vvv^^^^^^^^vvvvv>#
#>#################>#
#>#>>>>v#0#^^^^^^^>#
#v^v^#>>>^#vvvvvvv>#
####################
```

pacmaze-03-tricky.txt -

4 Conclusão e possíveis melhorias

O trabalho deixou bem evidente vários dilemas que surgem com a otimização de um espaço de estados parcialmente observável e, apesar de ser um método eficiente para este modelo de problema em particular, pode não ser o melhor para outras situações. O aprendizado por reforço implica que o agente precisa "errar" muito para abstrair uma política favorável e tornar-se viável. Se esse fosse o caso de um carro autónomo, por exemplo, significa que ele deve bater muito para finalmente aprender a dirigir. Apesar disso, há possíveis aplicações desse método para jogos eletrônicos, especialmente com ambientes não totalmente observáveis (ex. com *fog of war*).

Um possível aprimoramento do programa, inclusive já sendo uma técnica aplicada nesse tipo de aprendizado e em outros (programação genética, inteligência de colônias, etc.) seria favorecer o *exploitation* em vez do *exploration* com o passar do tempo. No caso, isso seria feito incrementando gradualmente o valor de α com o passar do tempo.

Além disso, analogamente ao que ocorre em modelos de decisão de Markov para ambientes totalmente observáveis, caso os próximos estados além de s' sejam observáveis, o valor deles pode ser considerado na atualização de $Q(s, a)$ (com o valor sendo descontado por γ^n). Isso ajudaria o agente a "enxergar mais um passo a frente" e pode auxiliar na convergência de estados mais distantes dos terminais, uma situação que ficou evidente neste trabalho.

Por fim, para esse problema, poderiam ter sido usados outros métodos como a programação genética (sendo a política ótima π a função representada pelos indivíduos e a fitness representada pelo somatório das recompensas para agentes seguindo tal política).

5 Referências

1. Russel, Norvig. *Artificial Intelligence, A modern approach, 3rd edition*