Relatório Projeto – 2ª Entrega
Inteligência Artificial

1. *Redes Bayesianas*

Descrição crítica dos resultados pedidos: A implementação desenvolvida, face aos testes disponibilizados, cumpriu todos os requisitos pedidos tendo resultados conforme os valores esperados.

Descrição crítica dos métodos implementados incluindo vantagens/desvantagens e limitações: A primeira classe implementada trata-se da classe "Node". Esta contém dois métodos: o método "init" e o método "computeProb". O primeiro método corresponde à inicialização da classe com os atributos "parents" e "prob" que se tratam, respetivamente, de um array contendo os índices dos parentes do nó e um array multidimensional contendo as probabilidades deste nó, tendo em conta as probabilidades dos parentes do mesmo. O segundo método recebe como argumento uma evidência correspondente a um tuplo de inteiros, indicando se a probabilidade de cada variável é negada ou não, e devolve uma lista com as probabilidades do nó ser false e de ser true. Este método começa por verificar o número de pais do nó, se este não tiver pais, devolve a probabilidade do nó ser true e de ser false, acedendo apenas ao atributo "self.prob" guardado na classe do nó. Caso o nó tenha um ou mais pais a função "calculateProb" é invocada, que está implementada de forma recursiva e analisa para cada parente desse nó a posição do tuplo evidência, determinando se probabilidade do parente se encontra a true ou a false. Caso a probabilidade do parente seja false, seleciona-se a posição 0 do array multidimensional de probabilidades do nó e caso seja true seleciona-se a posição 1 do mesmo array. A função "calculateProb" volta a ser invocada, recebendo agora como argumento a posição do array selecionada voltando a fazer as mesmas verificações e selecionando outra posição do array, de acordo com as verificações feitas anteriormente. A recursividade termina, quando não existirem mais posições do array a explorar, sendo a probabilidade da última posição a retornada.
A segunda classe implementada trata-se da classe "BN" e contém três métodos: "init", "computePostProb" e "computeJointProb". O método "init" refere-se ao método de inicialização da classe e recebe como argumentos um grafo da rede e um array de nodes, guardados na classe. O método "computeJointProb" recebe como argumento um tuplo evidência e devolve o valor da probabilidade da conjunta, calculando a probabilidade de cada nó através da função "computeProb" anteriormente referida e tendo em conta se a probabilidade de cada variável se encontra a true ou false. O método "computePostProb" recebe um tuplo evidência e calcula uma probabilidade condicionada. A função começa por identificar as variáveis cujo valor é desconhecido, bem como, a variável para a qual queremos calcular a probabilidade. Seguidamente, são calculadas todas as combinações possíveis para as variáveis desconhecidas e determinadas as probabilidades conjuntas para a variável para a qual se pretende encontrar a probabilidade, a variáveis cuja probabilidade é conhecida, variando apenas as combinações das variáveis desconhecidas.
Esta implementação tem como vantagem representar de forma simples e concisa dependências entre variáveis especificando assim qualquer distribuição de probabilidade conjunta total. Por outro

lado, tem como desvantagem o facto de ser ineficiente, pois a necessidade de somar todas probabilidades conjuntas de todas as combinações de variáveis desconhecidas aumenta consideravelmente a complexidade.

Discussão da complexidade computacional e possíveis métodos alternativos: A complexidade computacional do programa é $O(n2^n)$, em que n é o número de variáveis da rede. Tal como referido anteriormente esta implementação é ineficiente, pois utiliza o processo de inferência por enumeração que no pior caso calcula e soma as probabilidades conjuntas de todas as combinações possíveis do número de variáveis da rede menos uma. Um método mais eficiente alternativo ao utilizado seria o processo de inferência por eliminação. Este processo elimina a computação de cálculos repetidos guardando resultados intermédios assim evitando ter de os calcular outra vez, o que reduz a complexidade do algoritmo.

2. Aprendizagem por reforço

Trajetória 1 (exercise 1):

Ambiente: Na primeira trajetória temos um total de 7 estados e 2 ações possíveis para o agente: ação 0 e ação 1.

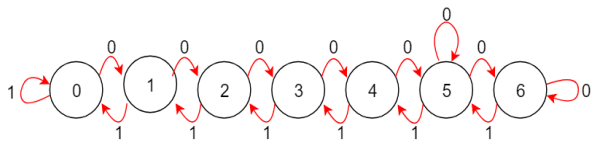

Figura 1 - representação gráfica do ambiente onde o agente se move (arcos são ações e nós são estados)

Função de recompensa: A função de recompensa, dá uma recompensa de 1 ao agente se este se encontrar nos estados 0 e 6 e dá uma recompensa de 0 caso contrário.

Politica óptima: a politica óptima escolhe por estado a ação que o leva a obter as probabilidades mais altas.

Descrição do movimento do agente: Para todos os estados o agente avança por efeito da ação 0 e recua por consequência da ação 1. A única exceção encontra-se nos estados 0, 5 e 6, em que o agente permanece no estado 0 por efeito da ação 1, permanece no estado 5 por consequência da ação 0 e permanece no estado 6 também por efeito da ação 1.

Estado inicial	*Ação 0*		*Ação 1*	
	Estado final	*Recom-pensa*	*Estado final*	*Recom-pensa*
0	1	1	0	1
1	2	0	0	0
2	3	0	1	0
3	4	0	2	0
4	5	0	3	0
5	5,6	0	4	0
6	6	1	5	1

Trajetória 2 (exercise 2):

Ambiente: Na segunda trajetória temos um total de 8 estados e 4 ações possíveis para o agente: ação 0, 1, 2 e 3.

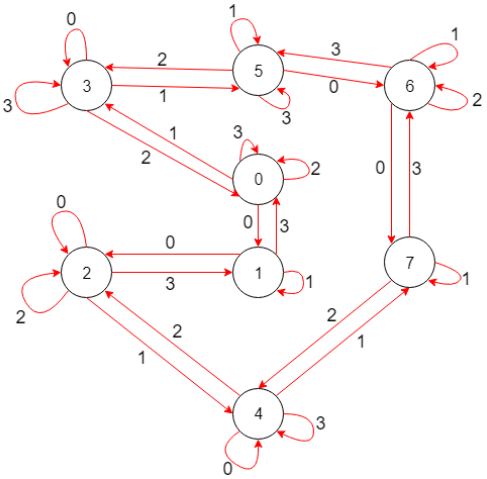

Figura 2 - representação gráfica do ambiente onde o agente se move (arcos são ações e nós são estados)

Função de recompensa: A função de recompensa, dá uma recompensa de 0 ao agente se este se encontrar no estado 7 e dá uma recompensa de -1 caso contrário.

Politica óptima: a politica óptima escolhe por estado a ação que o leva a obter as probabilidades mais altas.

Descrição do movimento do agente:

E.I	*Ação 0*		*Ação 1*		*Ação2*		*Ação 3*	
	E.F.	*R*	*E.F.*	*R*	*E.F.*	*R*	*E.F*	*R*
0	1	-1	3	-1	0	-1	0	-1
1	2	-1	1	-1	7	-1	0	-1
2	2	-1	4	-1	2	-1	1	-1
3	3	-1	5	-1	0	-1	3	-1
4	4	-1	7	-1	2	-1	4	-1
5	6	-1	5	-1	3	-1	5	-1
6	7	-1	6	-1	6	-1	5	-1
7	1	0	7	0	4	0	6	0

E.I.–Estado Inicial *E.F.* –Estado Final *R*- Recompensa

Descrição crítica dos resultados pedidos: A implementação desenvolvida, face aos testes disponibilizados, obtém os resultados conforme os valores esperados. Apesar disso, para os testes do ficheiro "mainRL.py" é possível obter trajetórias não óptimas para passos de entrada menores que 1500.

Descrição crítica dos métodos implementados incluindo vantagens/desvantagens e limitações: Os métodos implementados para a classe "finiteMDP" foram: o método "traces2Q" e "policy". Na função "traces2Q" é implementado o algoritmo Q-Learning que recebe uma trajetória e calcula os valores Q para cada ação através da formula: "$Q(s,a) \leftarrow Q(s,a) + \alpha\ (\ R(s) + \gamma\ max\ (Q(s',a') - \ Q(s,a)))$". O método "policy" define duas políticas que mais tarde serão usadas para gerar uma trajetória: a política "explore" e a política "exploit". No caso da política "explore" o agente percorre o ambiente de forma aleatória, começando pelo estado do argumento passado "x", explorando, assim, todas as possibilidades existentes. No caso da política "exploit" o agente percorre o ambiente respeitando a política óptima, começando também pelo estado do argumento passado "x" e selecionando a ação que corresponde à probabilidade mais elevada.

Uma desvantagem desta implementação deve-se ao facto de não ser possível obter os valores exatos da matriz retornada pela função "traces2Q". Esta implementação tem como vantagem o agente aprender a comportar-se com sucesso num ambiente desconhecido, decidindo ele próprio que ações tomar não tendo uma política fixa. Uma limitação é facto do agente aprender políticas determinísticas num ambiente não estacionário.

Discussão da complexidade computacional e possíveis métodos alternativos: A complexidade computacional é $O(n * m)$, sendo n o número de passos de trajetória e m a quantidade de vezes que é necessário executar o algoritmo Q-learning até que não haja uma diferença significativa entre os valores de "Q_{new}" e "Q_{old}", ou seja, até que o agente não tenha mais nada a aprender.