TÉCNICO LISBOA Grupo 72	**86448 - João Coelho** **87658 - Francisco Santos** 07-DEC-2018
IA Projeto 2	

Parte 1

Descrição do Problema

A primeira parte do projeto teve como objetivo o desenvolvimento de uma rede *bayesiana* à qual fosse possível questionar sobre a probabilidade conjunta e sobre a probabilidade posterior de um dado elemento.

Na rede *bayesiana* as diferentes variáveis têm de ser representadas com a estrutura nó a qual armazena informação sobre as dependências e sobre como a probabilidade é alterada consoante as evidências das variáveis das quais depende. É necessário definir um método para calcular a probabilidade do nó dadas evidências sobre os outros nós.

Descrição da implementação

Para o cálculo da probabilidade de cada nó não há muito a dizer para além de que é necessário percorrer os nós dos quais este depende.

Para o cálculo da probabilidade conjunta dado uma configuração de variáveis (evidências) é necessário calcular o produto da probabilidade de todos os nós.

O cálculo da probabilidade posterior, é a probabilidade de uma variável dado um conjunto de evidências que neste contexto pode ter algumas variáveis desconhecidas ou sem valor de evidência.

É possível reduzir o problema ao cálculo da probabilidade conjunta de todas as configurações que são possíveis para esse conjunto de evidências, ou seja, considerar todas as probabilidades possíveis para cada membro desconhecido. Como cada valor desconhecido vai influenciar os seus nós filhos devido às dependências entre nós, de maneira a evitar alguns cálculos repetidos foi necessário encontrar a ordem topológica e fazer o cálculo da probabilidade de cada nó por essa ordem.

Discussão da complexidade computacional

O cálculo de cada nó é limitado temporalmente e espacialmente por O(n) uma vez que no pior caso o nó depende de todos os outros nós.

Como o cálculo da probabilidade conjunta é feito com um nó de cada vez, o espaço é limitado por O(n), como vão ser necessárias computar n operações de cálculo de probabilidade o tempo é limitado por $O(n^2)$
O cálculo da probabilidade é feito iterando sobre todas os os valores possíveis para variáveis desconhecidas de forma a limitar espacialmente por O(n), apesar dos esforços para reduzir a repetição de valores a solução encontrada tem o tempo limitado por O(2^n).

O fator limitante deste problema é o cálculo da probabilidade posterior que assumimos ter de ser calculada exata, para os 10 nós estamos a falar de 2^{10}=1k como unidade, não é intratável, mas torna impossível o tratamento de uma maior quantidade de nós.

Podemos considerar iterar em vez de para cada valor de evidência diferente iterar para cada nível diferente da rede *bayesiana*, no entanto em nada vai alterar o pior caso em que existirem n níveis na rede.

Ainda são repetidos alguns cálculos de probabilidades de nós, quanto mais próximo da raiz mais cálculos são repetidos, para combater isso poderíamos usar eliminação de variáveis que embora seja NP-difícil para as redes consideradas ao comprimir os nós em nós com o produto das probabilidades de forma a criar uma "*ploytree*" é possível alcançar um tempo de $O(n^2)$ para o cálculo da probabilidade posterior e O(n) para o cálculo da probabilidade conjunta.

Dito isto não seria praticável ir por esse caminho pois a memória teria crescimento exponencial com o número de nós e seria ainda mais intratável um problema desses pois $O(2^n)$ seria uma unidade de 1k*bytes de um nó normal.

Ambiente 1:

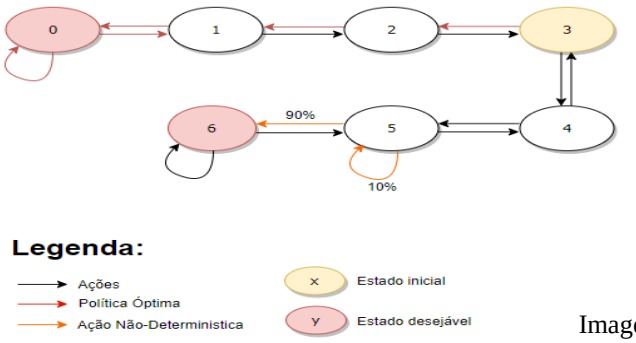

Imagem 1

Ambiente 2:

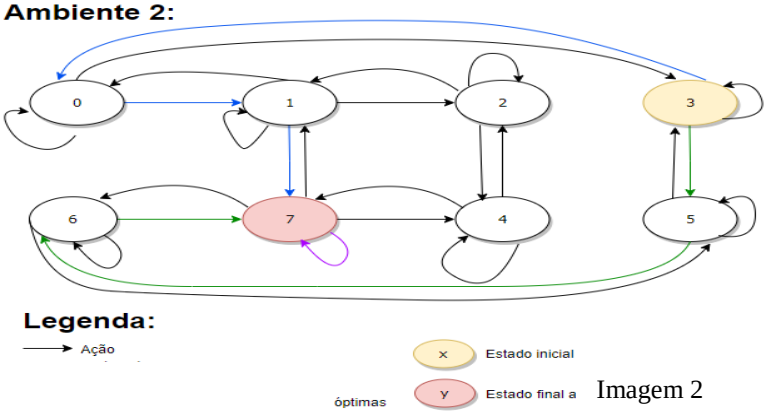

Imagem 2

Ações duplicadas não foram representadas.

Parte 2

Descrição do Problema

Na segunda parte foi pedido para desenvolver um algoritmo para auxiliar um robot a interagir com um mundo.

Esse algoritmo após aprender o ambiente através de transições treino teria de conseguir calcular a matriz dos valores esperados para cada ação e induzir uma política ótima.

Foi ainda pedido que fosse possível, ao receber algumas informações sobre o ambiente, gerar transições úteis para a aprendizagem do agente sobre o mundo.

Como não se conhece muito sobre as especificações dos problemas que o algoritmo tenta resolver assumimos que são problemas parecidos com os dos ambientes de teste: -pode haver algum não determinismo, mas num conjunto de ações muito reduzido - espaço de estados e ações relativamente pequenos

Descrição dos ambientes de teste

O ambiente do primeiro exercício está na imagem 1.

No ambiente exercício 1 qualquer ação tem uma recompensa associada de valor 0 exceto nos estados 0 e 6 em que a recompensa por cada ação é 1, após a qual o estado inicial é retomado.

A política ótima, descrita na imagem, é chegar a um dos estados com recompensa o mais rápido possível. Embora à primeira vista não seja óbvio que o caminho para o estado 6 que está a mesma distância não é ótimo, o não determinismo faz com que haja a chance de não chegar apenas em 3 transições, o que não existe para o caminho até 0.

O ambiente do segundo exercício está na imagem 2.

No ambiente exercício 2 qualquer ação tem uma recompensa associada de valor -1 exceto no estado 7 em que a recompensa por cada ação é 0.

A política ótima, descrita na imagem, é chegar a um dos estados com maior recompensa. As maiores diferenças com o exercício de teste 1 são desta vez existirem 2 políticas igualmente ótimas, não haver não determinismo e uma vez chegado ao estado desejado não é necessário recomeçar no estado inicial.

Descrição da implementação e sugestões de melhoramento

A pedido do enunciado foi completado o algoritmo *Q-learning*, para iterar sobre um conjunto de transições e gerar uma matriz com os valores esperados para as diferentes ações. Foi usada a fórmula (1) descrita no livro da cadeira na página 844 para atualizar a matriz:

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

A fórmula (1) permite ao agente aprender sobre o ambiente apenas com o resultado esperado das suas ações. É de referir também que o algoritmo tem uma grande tolerância para encontrar a política correta com casos suficientes mesmo que as ações escolhidas para aprendizagem não sejam as melhores uma vez que é um algoritmo *off-policy*. Embora seja muito flexível o método utilizado demora muito para convergir.

Utilizou-se o cálculo da norma de *Frobenius* entre as matrizes Q e nQ para verificar a convergência dos valores de Q. Não conseguimos simular convergência total, pois isto significaria ter iterações infinitas de modo a que a norma de *Frobenius* da diferença das matrizes fosse 0, no entanto, utilizamos 0.01 como um valor para o qual achamos razoável afirmar que próximas iterações não irão gerar resultados que afetem muito o valor de Q.

O algoritmo *tentar "olhar para a frente" e determinar a utilidade de um estado futuro com a informação disponível agora,* tem a desvantagem de, para sistemas complexos em que o estado do sistema se altera de maneira não óbvia, o valor esperado da ação não ser calculado corretamente. É o caso do exemplo 1 em que embora seja indiferente uma vez no estado zero aplicar uma ação ou outra porque vai voltar para a posição inicial, como não tem acesso a essa informação calcula o valor esperado da ação de voltar para o estado 1 com valor menor do que a de continuar em 0 embora sejam indiferentes.

Para controlar tais casos teria de ser imposto o fornecimento a matriz *absorv* ao *trace2Q* ou podia ser utilizado uma variação que permitisse "olhar para trás", ou seja, calcular o valor da ação esperada passada tendo em conta o ganho no presente.

Na implementação de (1) *gamma* é um parâmetro obrigatório, mas é necessário definir um *alpha* (*learning rate*). Na nossa implementação decidimos usar um *alpha* baixo para diminuir a chance de o ruído afetar a política escolhida.

O ruído nestes problemas é originado tanto pela incerteza que é ter ações não deterministas como pela incerteza associada com as ações de aprendizagem com as quais nem sempre temos controlo.

Foi ainda pedido que fosse criada uma política de *exploration* - para ser usada para aprender a matriz de utilidades de ações (Q) - outra de *exploitation* - para uma vez que se tenha Q poder encontrar a política que maximiza a utilidade.

A política de *exploration* apenas devolve uma ação aleatória dentro das possíveis. Como nos foi pedido para convergir a matriz de Q para todas as ações, tivemos de explorar sem descriminação de ações, pelo que o método aleatório é adequado.

A política de *exploitation* verifica, para uma posição, qual das ações tem um valor de Q maior, retornando-a. Como tal, é necessária uma sólida exploração do ambiente para a matriz de utilidades de ações ter resultados ótimos.

Valia a pena explorar a ideia da política de *exploration* retornar em vez de uma ação puramente aleatória, dar alguma prioridade a ações que foram experimentadas poucas vezes, de forma a reduzir o ruido.

Nos testes que nos foram fornecidos, foi-nos pedido para escolher um valor para o número de trajetórias a gerar de forma para o robot do problema aprender com elas. Infelizmente não conhecemos suficiente do problema inicial para decidir o *tradeoff* entre precisão da política, tempo de iteração sobre os valores e número de estímulos necessários gerar. Se considerarmos o agente como atuador no mundo real vemos que é impraticável aprender com muita precisão a trajetória ótima para um dado ambiente, devido ao volume de trajetórias necessário, atribui-se esta desvantagem ao algoritmo Q-*learning*. se alterarmos o *alpha* para um valor menor obtemos menos ruído, maior chance de acertar na política correta mas também mais tempo de computação do algoritmo de aprendizagem. Consideramos que um *alpha* de 0.05 e um número de transições de 3000 tem um bom *tradeoff* entre tempo e precisão de política, no entanto volto a referir que não sabemos o impacto de um erro e depende muito do contexto do problema ao qual não temos acesso

Iterações	N	Alfa	Tempo (s)	Erro Transições
500	3000	0.25	72	12
500	3000	0.1	125	2
500	3000	0.05	198	0
500	3000	0.01	198	0
500	6000	0.1	175	0
500	6000	0.25	112	0

Complexidade:

O agente tem de iterar sobre um espaço de estados para fazer convergir todas as ações, para realizar todas as ações pelo menos uma vez significa conseguir chegar a todos os estados o que no pior caso custa n, o custo total da aprendizagem é então O (n * iterações até aprendizagem) que variam de problemas para problemas mas em geral é bastante lento com o *Q-learning* e impraticável para conjuntos com muitos estados e/ou poucas transições teste. Esta complexidade poderia ser reduzida no caso médio se não nos fosse pedido para fazer convergir todas as ações de Q, seria então possível fazer uma exploração mais guiada a caminhos que nos levem a uma trajetória ótima.