# Relatório 2º Projeto de IA - IST @ 2018/19


| NÚMERO: | NOME: |
|---|---|
| 87641 | Carolina Carreira |
| 87691 | Miguel Barros |


- ## Descrição crítica dos resultados pedidos

Para testar o tempo de execução da implementação feita, foi usado o comando unix time, repetida a execução do programa vinte vezes e feita a média dos resultados obtidos. Para os testes públicos dados temos que:
O primeiro conjunto de testes públicos obteve um resultado correto em 0.19 segundos.
O segundo conjunto de testes públicos obteve um resultado correto em 0.03 segundos.
Estes resultados estão dentro do esperado para problemas de redes bayseanas desta dimensão (o primeiro conjunto de teste usa uma rede com cinco nós, o segundo uma rede com 4 nós).

Para testar o aumento do tempo de execução em relação ao aumento do número de nós da rede foi criado um teste extra com 10 nós. Cada nó tinha 3 ou menos pais. Nesta foram calculadas todas as joint probabilities possíveis e três probabilidades condicionadas tendo respetivamente 2, 3 e 6 variáveis desconhecidas. Foi obtido um tempo de execução de 0.2 segundos.

Daqui retira-se que o tempo de execução não varia significativamente com o número de nós da rede, ao nível descrito (numa rede com mais de 10 nós ou com mais pais em cada nó já poderia dar resultados menos rápidos). Para mais, foi testado com mais e menos variáveis desconhecidas e também não foi observada nenhuma variação significativa do tempo de execução (mantendo-se sempre aproximadamente nos 0.2 segundos).

- ## Descrição dos métodos implementados incluindo vantagens/desvantagens e limitações

Seja e a evidência da probabilidade condicionada representada, e X o valor que queremos inferir.
O método que nós utilizamos para calcular a probabilidade condicional com variáveis desconhecidas pode dividir-se em quatro partes:

0. Probabilidade na forma:
$P(X|e) = \alpha P(X, e)$, em que o valor de $\alpha$ é desconhecido

1. Calcular valor de P(X,e), transformá-lo numa soma de joint probabilities possíveis com a evidência dada (ou seja, é efetuado o somatório sobre as variáveis que não estão na evidencia)(Marginalizar)

2. Calcular o valor de cada uma das joint distributions fazendo uma query na rede, usando a fórmula do teorema de Bayes:
$P(X_1, \ldots, X_N) = \prod_i^n P(X_i)|parents(X_i))$

3. Recuperar $\alpha$. É repetido o processo descrito para o caso em que X = 0. A soma dos dois resultados multiplicada por alpha deve dar 1

Desvantagens da nossa implementação:
– Só funciona se as variáveis tiverem valores booleanos de 1 e 0 ou true e false.
– Vão ser calculados valores repetidos ou óbvios, e alguns dos cálculos vão ser repetidos.
– Não podemos ter incerteza acerca de nenhum valor de nenhum dos nós

Vantagens:
– Fácil de implementar
– Não tem nenhuma aproximação para além dos arredondamentos dos cálculos
– São obtidos resultados razoavelmente rápido (ver tempos na seção anterior).

- ## Discussão da complexidade computacional e possíveis métodos alternativos

Seja *k* o número máximo de pais de um nó na rede.
Para cada nó temos que a complexidade espacial para o representar será de $O(2^k)$ pois cada pai tem dois estados possíveis (*true* ou *false*) e para cada nó temos de representar todas as combinações de valores possíveis dos nós dos pais. Seja a rede constituída por *n* nós (tal que $n > k$) temos, que para representar a rede teremos uma complexidade espacial de $O(n \times 2^k)$. No pior caso temos que $k \approx n$ e temos que a complexidade espacial será $O(n \times 2^n)$

Complexidade temporal
O método implementado para determinar qualquer probabilidade condicionada na rede foi inferência por enumeração.
Para cada nó da rede cujo valor é desconhecido temos dois valores possíveis (*true* e *false*) temos de somar dois termos, cada um multiplicando n valores. No pior de casos o número de variáveis desconhecidas (k) vai ser tal que $k \approx n$. Cada um destes tem dois valores possíveis, logo no total temos $2^n$ valores possíveis de combinações entre eles. Para cada uma destas combinações temos de fazer o produto de n operandos, um por cada nó da rede. Por causa disto vamos ter uma complexidade de $O(n \times 2^n)$

Este resultado pode ser melhorado tendo em atenção que:

Variáveis cujo valor seja conhecido e que não tenham pais no gráfico têm valores constantes durante todo o algoritmo, portanto só temos de efetuar uma multiplicação com eles

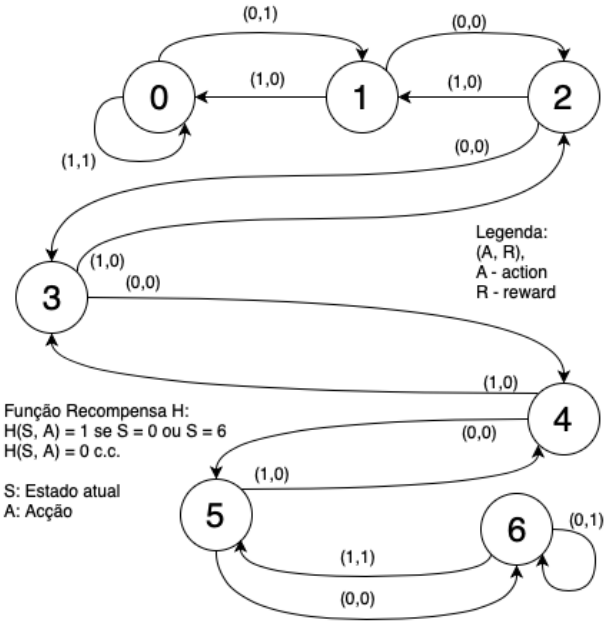
Variáveis cujo valor seja desconhecido e que não tenham pais só têm dois valores possíveis, portanto só precisamos de multiplicar estas variáveis duas vezes, uma por cada valor possível dela (em vez de $2n$ vezes).

Para além disto existem métodos que permitem acelerar a procura ainda mais, mas que não foram implementados, como o algoritmo de eliminação de variáveis, que permite acelerar a procura eliminando cálculos repetidos, usando programação dinâmica em que cada vez que um valor é calculado o seu valor é guardado para ser usado mais tarde.

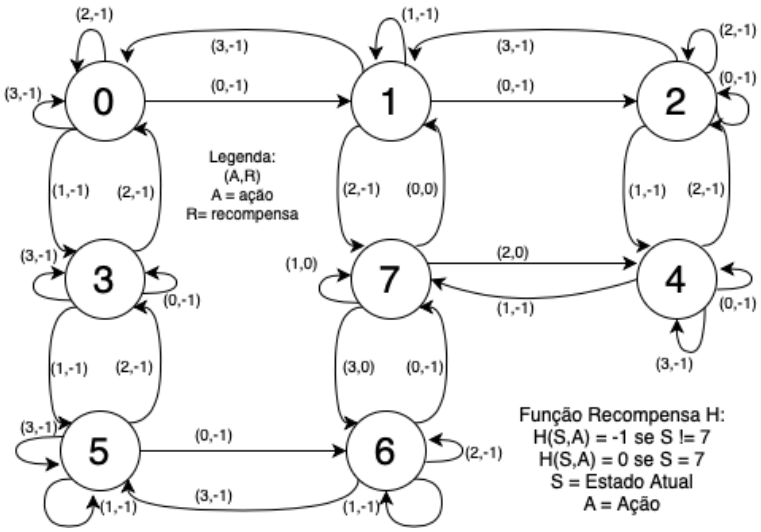
Para redes mais complexas e interligadas seria imprático fazer inferência exata, logo devem-se considerar 'Algoritmos de Monte Carlo' que aproximam os valores, e que dependem de uma coleta de amostras acerca do problema a ser representado na rede, como por exemplo os métodos *direct sampling* e *Markov chain sampling*

- Para cada um dos ambientes, e por inspeção das trajetórias uma representação gráfica do ambiente no qual o agente se move e função de recompensa

Ambiente 1:




Ambiente 2:




- Descrição da forma como o agente se move ( Qual é o impacto de cada Acão em cada estado )

O agente segue a política ótima, logo o impacto de cada ação é o melhor possível.

Ambiente 1:

$$\prod = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Ambiente 2:

$$\prod = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Cada linha é uma ação e cada coluna é um estado.

- Descrição critica dos resultados pedidos

O método usado foi o mesmo que anteriormente. Para cada um dos testes públicos disponibilizados foi escolhido um alpha de 0.1 (note-se que eventualmente poderia ser possível obter um resultado mais rápido escolhendo um alpha maior para certos problemas, mas este foi o recomendado). Os testes foram corridos 10 vezes para garantir resultados fidedignos e que não se originam na aleatoriedade usada no programa.

Para o primeiro teste foi obtido um resultado correto em 0.95s.

Para o segundo teste foi obtido um resultado correto em 0.64s

É de referir também uma particularidade da nossa implementação que o número de vezes

que o conjunto de teste é usado para aproximar o valor do Q é aproximado ao inverso do alpha. A lógica por detrás desta decisão é: valores de alpha mais pequenos precisam de explorar o mundo mais tempo para inferir corretamente o que fazer, este foi um compromisso para funcionar com alfas baixos, mas não abrandar a aprendizagem com alfas elevados.

Para explorar o ambiente escolhemos em cada ponto no tempo uma ação aleatória

Para obter uma exploração completa do ambiente do primeiro teste público foram precisos 3000 passos de exploração. No segundo teste publico foram precisos 200 passos de exploração. Nota-se então que o número de passos, como esperado, aumenta com o número de estados e ações possíveis (o segundo teste público tem menos estados e ações logo precisa de menos passos para ser explorado).

- Descrição dos métodos implementados incluindo vantagens/desvantagens e limitações

O método de *reinforcement learning* usado foi o *Q-learning*, em que o agente procura encontrar a política ótima para maximizar a recompensa recebida em cada estado do ambiente. O agente aprende explorando o ambiente e usando a informação que recebe (para este estado nesta ação recebo esta recompensa e vou para aquele estado) como input para a função de Q ($Q(S,A) = Q^*(S,A) + \alpha(r + \gamma max_b(Q(S',b)) - Q(S,A))$, que atualiza a matriz Q (matriz que associa um valor a cada par (estado, ação), valores mais pequenos são mais desejáveis para agente). Após explorar o ambiente o agente toma decisões de modo a maximizar a recompensa que recebe (ou seja, escolhendo a ação com o valor máximo de Q para o estado em que está.
Quando o agente está a explorar o ambiente escolhe em cada estado uma ação aleatória. Passado algum tempo o agente terá explorado todo o ambiente de maneira a saber qual melhor ação a tomar em cada estado.

As desvantagens deste método são que as ações que o agente toma podem não ser as mais interessantes. Por exemplo, um agente pode escolher repetir sempre uma ação que mantem o agente no mesmo estado e lhe dá uma recompensa em vez de ir à procura de uma recompensa melhor noutro estado. Também só produz resultados corretos se o agente tem uma ideia completa do mundo, algo que pode não acontecer se a exploração não for suficientemente exaustiva.

O agente, na nossa implementação não sabe lidar com mudanças no mundo, pois quando está na política *exploitation*, a decisão que ele toma quando está a escolher maximizar a recompensa pode levá-lo a não explorar o mundo e descobrir as alterações. Também tem uma elevada complexidade espacial (para cada estado temos de guardar os valores Q de cada ação possível) e certos ambientes são difíceis de representar desta maneira e neste caso têm de haver aproximações (por exemplo ambientes não discretos).

A principal vantagem do Q-learning é aprender a política ótima de um ambiente enquanto explora o ambiente.

- Discussão da complexidade computacional e possíveis métodos alternativos

Métodos alternativos podem passar por:
- *utility based agent*, que aprende uma função de utilidade nos seus estados e que usa essa função para selecionar ações e maximizar a utilidade da ação.
- *reflex agent* aprende uma política que mapeia diretamente estados para ações

Pode-se ainda referir que o método *exploitation* do Q-learning podia ser outro para além do Greedy:
- *Σ-greedy*, escolhe a melhor ação apenas Σ% das vezes, permite haver exploração;
- *Boltzman*, usa a fórmula $\prod(x,a) = \frac{e^{Q(x,a)/z}}{\sum_b e^{Q(x,b)/z}}$ para determinar a ação a tomar num dado estado. Útil para o agente não ficar preso a efetuar sempre a mesma ação;
- Otimista, começa com valores de Q altos explorando o ambiente sem escolher uma ação aleatória.

Complexidade espacial

Temos que representar o ambiente corresponde a ter uma matriz (estado, ação). Seja N o número de ações e M o número de estados a complexidade espacial será O(NM).

Complexidade temporal

Para aprender sobre o mundo o agente tem de explorá-lo todo.

Durante a exploração o agente gera uma ação aleatória. O número de passos que o agente vai ter de efetuar para explorar o ambiente não é calculável.

Para a política de exploitation escolher a ação a tomar vai ser O(N) (ele vê todas as ações que pode tomar e escolhe a que tiver o valor do Q mais alto).