INSTITUTO SUPERIOR TÉCNICO

Inteligência Artificial

Relatório 2º Projeto (2018/2019)

Grupo 22 (Alameda) | Francisco Sena (86420) | João Almeida (86447)

Introdução

Neste projeto testaram-se alguns algoritmos que lidam com a incerteza no mundo recorrendo a modelação e inferência com redes *Bayesianas* (P1) e aprendizagem por reforço com o algoritmo *Q-learning* (P2).

P1 – Inferência exata em Redes Bayesianas

Na classe *Node* foi implementado o método *computeProb*, que calcula a probabilidade associada a um evento dado um conjunto de evidências. Para determinar essa probabilidade, basta percorrer a tabela de probabilidade condicional do nó, que no pior caso tem complexidade espacial $O(2^n)$ (n = nº de nós na rede). De facto, para modelar um problema com muitas variáveis, a memória torna-se uma limitação pois o tamanho das tabelas de cada variável cresce exponencialmente em função do número de pais desse nó. Com efeito, seria possível inspecionar a tabela integralmente em busca da linha onde o domínio das variáveis coincide com as evidências, o que seria ineficiente pois demoraria tempo exponencial. Com a estrutura de dados escolhida para a representação das tabelas de probabilidades condicionadas, e dada a implementação que foi optada para o método *computeProb,* tem-se uma complexidade temporal $O(n)$.

Na classe *Graph* foram implementados dois métodos:

- *computeJointProb*: Com este método podemos construir a tabela de distribuição conjunta, ou seja, a probabilidade conjunta de todas as variáveis da rede com todas as possíveis combinações de evidências, tais que o somatório de todas as entradas da tabela resulte em 1, como é observado nos testes do *mainBN*. Recorrendo ao exemplo do enunciado, o que é calculado neste método é a probabilidade do *Burglary* (b), *EarthQuake* (e), *Alarm* (a), *JohnCalls* (j) e *MaryCalls* (m) ocorrerem, onde as variáveis podem tomar o valor *True* ou *False*:

$$P(j, m, a, b, e) = P(j \mid a)\, P(m \mid a)\, P(a \mid b, e)\, P(b)\, P(e)$$

Este método tem complexidade temporal $O(n^2)$ pois são percorridos todos os nós da rede e, para cada um, calcula-se a sua probabilidade, recorrendo ao *computeProb*.

- *computePostProb*: Qualquer probabilidade a-posteriori pode ser calculada usando somas de produtos das probabilidades condicionadas das variáveis da rede, logo, recorre-se à tabela de probabilidades conjuntas. Neste contexto, optou-se pela implementação mais ingénua, que pode ser descrita por:

$$P(X \mid e) = \alpha\, P(X, e) = \alpha \sum_{y} P(X, e, Y) = \frac{\sum_{y \in \{0,1\}} P(X, e, y)}{\sum_{X, y \in \{0,1\}} P(X, e, y)}$$

Onde *X* é a variável da qual queremos inferir, *e* é o conjunto das evidências e *Y* é o conjunto das *hidden variables*.

Temos, portanto, de recorrer ao método *computeJointProb* para calcular cada parcela dos somatórios. Temos, para esta abordagem, uma complexidade temporal $O(n^2\,2^n)$.

Existem métodos mais eficientes que o utilizado, como a inferência por enumeração e o método de eliminação de variáveis, mas, para o problema em causa, não foram necessários tais métodos, pois a rede tem no máximo 10 nós.

P2: Aprendizagem por Reforço

Na classe *finiteMDP* foi implementada a função ***traces2Q***, que recebe com argumento uma trajetória, e retorna uma aproximação da função *Q*. Para isto é percorrida a trajetória, e consequentemente atualizada a matriz *Q*, até que esta deixe de variar significativamente, ou seja, até que atinja um limite de convergência.

Foi também implementada a função ***policy,*** que para dados estado atual e política de exploração, retorna a ação a executar. Se a política for de *exploitation*, a ação escolhida é a que maximiza o *Q-value* para o conjunto *{estado atual, ação}*. No caso da *exploration*, optou-se por selecionar a ação de forma aleatória, embora existam formas mais sofisticadas onde, num determinado estado, são privilegiadas ações que nunca (ou por menos vezes) foram executadas, recorrendo por exemplo a uma tabela de frequências absolutas.

- **Como se move o agente?**

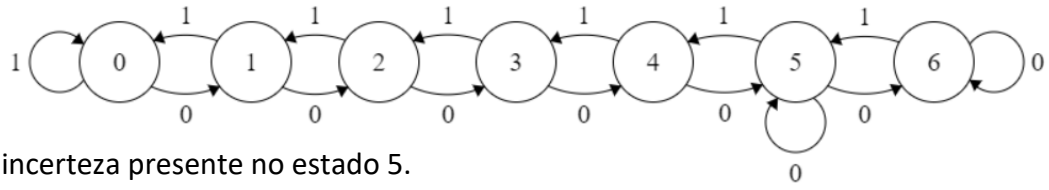
Usando como exemplo o exercício 1, o agente gera uma trajetória usando uma política de *exploration.* De seguida, usando a função *traces2Q*, percorre a trajetória gerada e consequentemente atualizada a matriz *Q*, até que esta deixe de variar significativamente, ou seja, até que atinja um limite de convergência. Com a matriz Q atualizada e usando a política de *exploitation,* espera-se que o agente se mova de forma a maximizar a recompensa.

- **Qual é o impacto de cada ação em cada estado?**

Uma ação tomada num estado afeta diretamente o *Q-value* desse mesmo estado e indiretamente o dos estados adjacentes. Ao iterar sobre a mesma trajetória várias vezes, este impacto no *Q-value* é propagado para estados cada vez mais distantes, até ter efeito em todo o ambiente.

- **Ex. 1**

O ambiente onde o agente se move pode ser representado pelo autómato não-determinista



devido à incerteza presente no estado 5.

Quando executada a ação 0 nesse estado o agente irá com probabilidade 0.9 para o estado 6 ou, com probabilidade 0.1, permanecerá no mesmo estado.

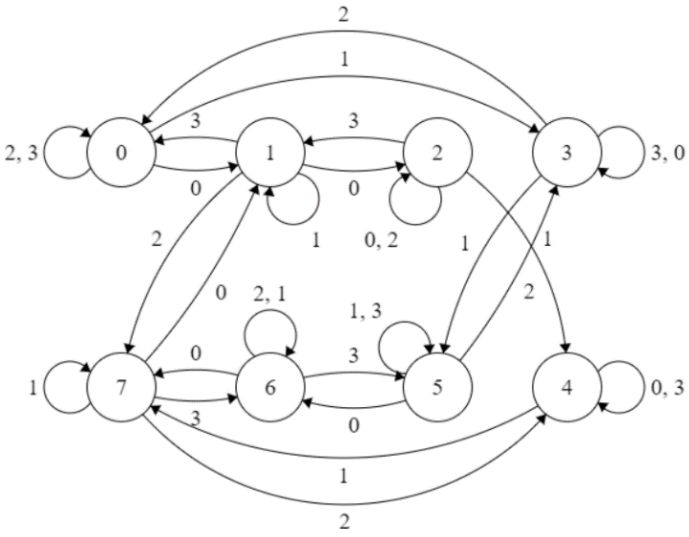
Pela análise da trajetória, verificou-se que a função recompensa é dada por:

$$R(s,s') = \begin{cases} 1, & s \in \{0,6\} \\ 0, & caso\ contrário \end{cases}$$

Neste ambiente, para obter o máximo de recompensa, a política ótima consiste em tentar atingir o estado 0 ou o estado 6 assim que possível e tentar permanecer lá.

- **Ex. 2**

Por análise da trajetória, verificou-se que o ambiente pode ser representado graficamente por:



A função recompensa é dada por:

$$R(s,s') = \begin{cases} 0, & s = 7 \\ -1, & caso\ contrário \end{cases}$$

Neste ambiente, para obter o máximo de recompensa, a política ótima consiste em tentar atingir o estado 7 assim que possível e tentar permanecer lá.

- **Análise crítica dos resultados e limitações**

Os dados utilizados para construir o gráfico apresentado de seguida foram obtidos com testes executados sobre o exercício 2 dos testes públicos. Observando o gráfico, conclui-se que quantas mais iterações forem executadas, menor são as diferenças entre a norma da matriz *Q* da iteração *k* e da iteração *k+1*. Daqui pode-se concluir que a matriz *Q* está a convergir para *Q**, como referido anteriormente.

Após alguns testes para valores de a*lfa,* conclui-se que 0.2 era um valor suficientemente bom para o problema em causa.