TÉCNICO LISBOA

Inteligência Artificial 2018/2019– Grupo: 36
João Rafael Soares - 87675
Maria Catarina Duarte - 87681

Redes Bayesianas

Descrição das funções:

(Nota: n refere-se ao número de nós existente na rede)

computeProb:

Utilizando o tuplo de provas, navega a lista de probabilidades de modo a encontrar a probabilidade correspondente ao nó, tendo em conta os valores das provas dos pais do mesmo.

Retorna uma lista contendo as probabilidades do seguinte modo:

$$[\,P(\bar{A}|Pais), P(A|Pais)]$$

Complexidade: $O(n)$

computeJointProb:

Utilizando o tuplo de provas "*evid*", calcula o valor da probabilidade conjunta seguindo a seguinte formula geral:

$$P(A_1, A_2, \dots, A_n) = \prod_{i=1}^{n} P(A_i|A_{i+1,} \dots, A_n)$$

Para o nosso caso específico, utilizamos a função *computeProb* para obter os valores de cada passo da multiplicação, tendo assim só em conta os pais de cada nó, ficando assim, tendo em conta o tuplo "evid":

$$P(A_1, A_2, \dots, A_n) = \prod_{i=1}^{n} self.prob[i].computeProb(evid)[evid[i]]$$

Nesta função também poderíamos ter utilizado um algoritmo de eliminação, evitando assim a repetição de cálculos desnecessários e tornando a função mais eficiente.

Complexidade: $O(n * n) = O(n^2)$

computeEvidences:

Esta função recursiva auxiliar calcula todas as combinações possíveis de uma prova que contenha valores desconhecidos, retornando uma lista com todos esses valores.

Complexidade: $O(2^n)$

computePostProb:

Utilizando o tuplo de provas evid, calculamos a probabilidade *a-posteriori* para um dado nó, contendo provas com valores desconhecidos.

Utilizamos a fórmula geral:

$$P(A_1|a_2, \dots, a_n) = \alpha P(A_1, a_2, \dots, a_n)$$

$$\alpha = \frac{1}{P(a_2, \dots, a_n)}$$

Onde o cálculo de probabilidades com valores desconhecidos é dado por:

$$P(a_2, \dots, a_n) = \sum_{i=1}^{2} \dots \sum_{z=1}^{2} P(a_i, \dots, a_z)$$

Onde o somatório alterna entre o valor booleano da variável (ou seja, se é verdadeira ou falsa).

Para o nosso caso específico, utilizamos as funções *computeEvidences* e *computeJointProb* para calcular as combinações possíveis de provas e com essas calculamos as probabilidades conjuntas necessárias como mostrado na fórmula matemática anteriormente

Complexidade:

$$O(n + 2^n + n^2 + 2^n + n^2) = O(2^n)$$

Vantagens:

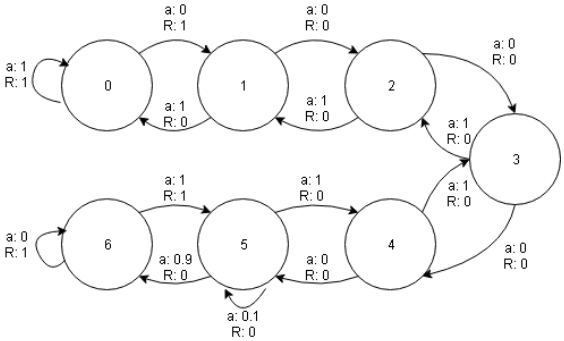
- Agente consegue tomar decisões racionais mesmo não tendo informação suficiente;
- Permite uma coexistência de várias fontes, independentemente do seu tipo ou ligação

Desvantagens:

- É computacionalmente ineficiente (Probabilidade a-posteriori tem complexidade exponencial)

Aprendizagem por Reforço:

Ambiente 1 (Exercício 1):

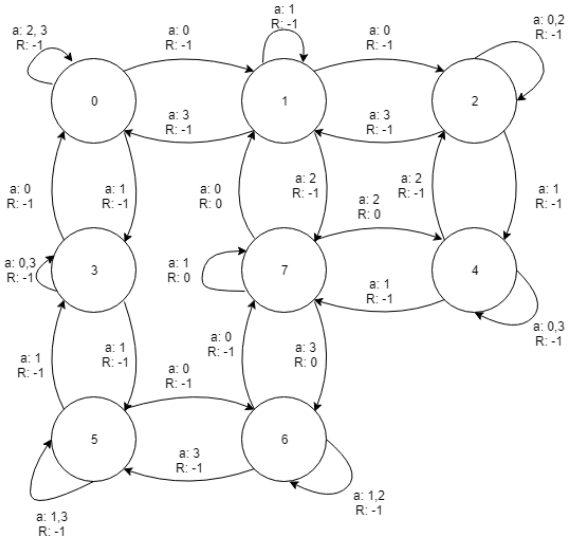


$$R(s,a) = \begin{cases} 1, & se\ s = 0\ \vee s = 6 \\ 0, & caso\ contrario \end{cases}$$

$$\pi^* = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$

Determinístico: Não

Ambiente 2 (Exercício 2):



$$R(s,a) = \begin{cases} 0, & se\ s = 7 \\ -1, & caso\ contrario \end{cases}$$

$$\pi^* = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Determinístico: Sim

Descrição das funções:

(Nota: a representa o número de ações possíveis, n é o número de trajetos calculados)

Policy:

Retorna a ação a realizar estando num estado X e dependendo da política ("*poltype*").

Se for uma política de "*exploitation*", escolhe o índice com o maior valor de Q.

Caso seja "exploration", escolhe uma ação *random*.

Complexidade: $O(a)$

Traces2Q:

Recebendo uma trajetória, esta função calcula os valores Q quando aplicado essa trajetória.

Utilizamos a função:

$$Q(s,a) =$$

$$Q(s,a) + \alpha(R + \gamma \max_{a' \epsilon A} Q(s',a') - Q(s,a))$$

Calculamos várias vezes os valores de Q para a mesma trajetória, terminando apenas quando a diferença entre iterações de cálculo for um valor bastante baixo (no nosso caso específico, com uma diferença menor de 10^{-7}).

runPolicy:

Esta função cria uma trajetória a partir de um estado inicial, escolhendo uma ação dependendo da política que se esteja a impor ("*exploitation*" ou "exploration").

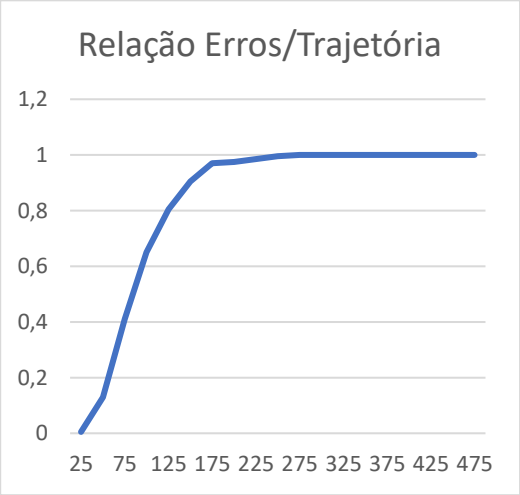
Complexidade: $O(n * a) = O(na)$

Limitações:

Quando o agente aprende utilizando uma política de exploração, criamos uma trajetória com a função *runPolicy*, onde fornecemos como input o número de trajetória a criar.

Caso este valor seja baixo demais, existe a possibilidade de não haver trajetórias com valores suficientes para o agente aprender.

Podemos ver no gráfico representado em baixo a maneira com que esta função evolui.



Como podemos ver, precisamos de no mínimo a volta de 225 trajetórias de modo a garantir que o agente aprende da maneira correta.

Vantagens:

- Não necessita de um modelo do mundo;

Desvantagens:

- Não consegue atingir uma solução exata com exploração, obtendo só uma aproximação;