INSTITUTO SUPERIOR TÉCNICO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Inteligência Artificial

LEIC-T

Inferência Exata em Redes Bayesianas

&

Aprendizagem por Reforço

Grupo 5

77906 — António Sarmento

83391 — Andreia Valente

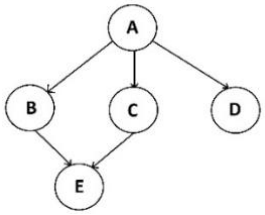
P1 – Inferência Exata em Redes Bayesianas

Introdução

Esta parte do projeto tem como objetivo desenvolver um **processo de inferência** para resolver uma rede de Bayes acíclica com o máximo de 10 nós. O processo desenvolvido baseou-se na **inferência por enumeração** (soma de termos de uma distribuição de probabilidade conjunta).

Resultados

Motivação: observar o impacto do nº de variáveis de inferência e da eliminação de variáveis irrelevantes na **computação de probabilidade conjuntas** e no **tempo de execução**.



Com uma rede de Bayes de 5 nós, alternou-se o número de variáveis conhecidas no problema. Os valores de probabilidades conjuntas obtidos representam apenas a variável pedida em *true* ou *false*. Os resultados apresentados para tempo de execução são a média dos 3 piores casos de 5 iterações teste.

Com o algoritmo de inferência por enumeração sem eliminação de variáveis irrelevantes à *query*, obtiveram-se os seguintes resultados:

Nº de variáveis desconhecidas	Tempo de execução (ms)	Probabilidades conjuntas calculadas
1	0.9995	[2]
2	0.9897	[4]
3	0.9901	[8]
4	1.9984	[16]

Com o algoritmo de inferência por enumeração com eliminação de variáveis irrelevantes à *query*, obtiveram-se os seguintes resultados:

Nº de variáveis desconhecidas	Tempo de execução (ms)	Probabilidades conjuntas calculadas
1	0.9001	[1,2]
2	0.9071	[1,2,4]
3	0.9851	[1,2,4,8]
4	0. 9999	[1,2,4,8,16]

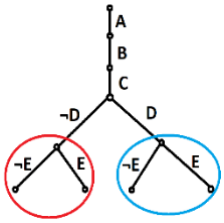
Métodos implementados

Para resolver a inferência exata de Bayes foi usado inferência por enumeração, em que os nós estão ordenados topologicamente.
Dado uma lista de evidências, a função recursiva *evaluation_tree* **constrói todas as listas de evidências que correspondem às probabilidades do fim de todos os ramos da árvore de avaliação**, calcula a probabilidade conjunta de cada lista de evidências, ignorando o cálculo para as variáveis irrelevantes à *query* (que o seu somatório equivale a 1) e soma todos os resultados.

A função *computePostProb* chama *evaluation_tree* para a variável pretendida como verdadeira e como falsa, faz a transformação em α das duas e retorna o resultado da inferência.

Descrição da eliminação de variáveis irrelevantes

O algoritmo de inferência por enumeração base avalia, em determinadas *queries*, a soma de ramos que são irrelevantes (que a soma equivale a 1). As variáveis relevantes para uma *query* são apenas as variáveis-pai das variáveis conhecidas na *query*.



No exemplo da rede apresentada na secção "Resultado", a *query* (-1, 1, 1, [], []) resulta no algoritmo computar P(¬E|B,C) e P(E|B,C) para fazer a sua soma (que equivale trivialmente a 1) e o mesmo acontece com D.

Este problema de otimização foi resolvido ao eliminar do somatório de inferência todos os nós que não correspondem a antecessores (de acordo com a ordem topológica) das variáveis dadas na *query*. No exemplo anterior, com esta otimização, o algoritmo já não calcularia os 4 ramos finais e calcularia a probabilidade conjunta diretamente, sem explorar ramos da árvore de avaliação.

Complexidade computacional

No pior caso, para uma rede com **n** variáveis booleanas, o algoritmo *evaluation_tree* tem de calcular a soma de todas as probabilidades conjuntas resultantes de uma recursão *depth-first search*, portanto a complexidade temporal do algoritmo é: **O(2^n)**. A complexidade espacial é **O(n)** por armazenar os valores das n recursões *depth-first search.*

Discussão de resultados

Para d variáveis desconhecidas, foi observada a relação constante de 2^d para as probabilidades conjuntas calculadas, **sem eliminação de variáveis.**

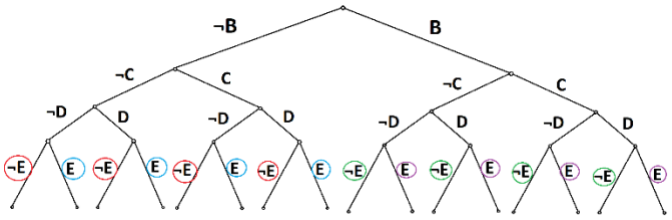
Com eliminação de variáveis irrelevantes, já não se observou a constante relação de 2^d e obtiveram-se resultados otimizados no número de probabilidades conjuntas a serem calculadas, como era esperado.

Os **tempos de execução** registados só mostraram uma maior discrepância para as *queries* com 4 variáveis desconhecidas. Nestas *queries*, existiu apenas uma situação - [[], [], [], [], -1] - em que o algoritmo teria de calcular todas as 16 probabilidades conjuntas e, portanto, sem a eliminação de variáveis irrelevantes, são quase sempre (menos na situação mencionada antes) usados ciclos de computação de probabilidade desnecessários.

Foi também observado que, para determinadas *queries*, o algoritmo permite o mesmo cálculo da probabilidade de um nó mais do que uma vez (exemplo na secção seguinte). Este facto representa ciclos de computação desnecessários logo a **implementação é ineficaz.**

Métodos alternativos

Computações repetidas: O algoritmo de inferência por enumeração avalia sub-expressões repetidas. Por exemplo, para o problema **(-1, [], [], [], [])**, são calculadas 16 probabilidades conjuntas para a variável A=*true* (ramos finais da árvore na imagem seguinte), como visto na secção "Resultados". A imagem seguinte representa, circundados pela



mesma cor, os cálculos de probabilidade repetidos:

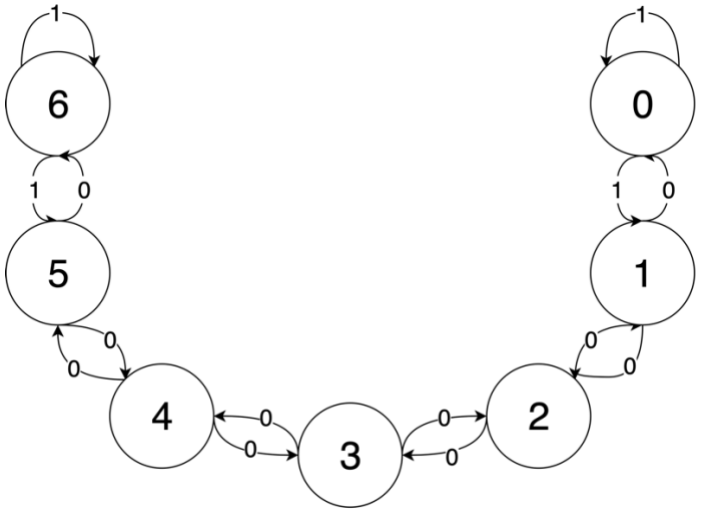
Solução: O **algoritmo de eliminação de variáveis** avalia a expressão do problema da direita para a esquerda, armazenando resultados intermédios e as somas de cada variável são feitas apenas sobre a expressão que depende dessa variável. No exemplo anterior, o algoritmo já não calcularia, por exemplo, P(E|B) quatro vezes, mas apenas uma.

P2 – Aprendizagem por Reforço

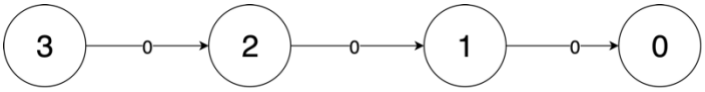
1º Ambiente

Representação gráfica do ambiente

Exploration (agente aprende ao acaso):



Exploitation (agente é ganancioso):



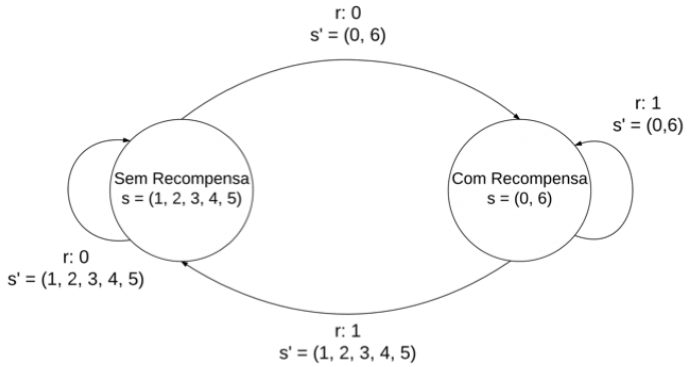
Função de recompensa

Sejam $\begin{matrix} s = \text{State} \\ a = \text{Action} \end{matrix}$; $R(s,a) = \begin{cases} 0 \text{ se } s = (1,2,3,4,5), a = (0,1) \\ 1 \text{ se } s = (0,6), a = (0,1) \end{cases}$

Política ótima: Exploration

Pelo material substantivo proposto para testar as duas políticas de exploração (*Exploration* & *Exploitation*), a política ótima será *Exploration* por terminar certamente no final de N experiências de treino pois a política *Exploitation* arriscava-se a ficar preso numa rotina, onde N é um inteiro suficientemente grande (500 tendo sido o valor usado para os nossos testes).

Como é que o agente se move?



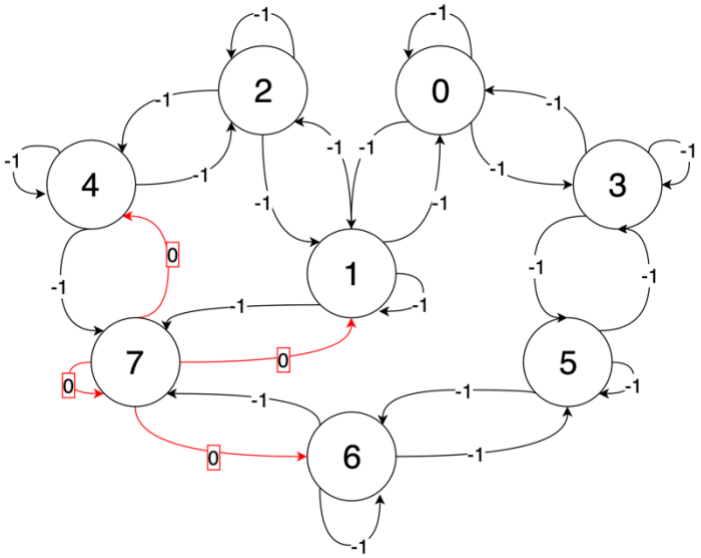
Durante a análise da trajetória do agente foi possível observar-se um padrão; representando o ambiente sob a forma simplificada de um grafo de dois nós, o agente move procura um estado em que possa permanecer em ciclo a receber uma recompensa = 1.

P2 – Aprendizagem por Reforço

2º Ambiente

Representação gráfica do ambiente

Trajetória de entrada:



Função de recompensa

$$R(s,a) = \begin{cases} -1 \text{ se } s = (0,1,2,3,4,5,6), a = (0,1,2,3) \\ 0 \text{ se } s = 7, a = (0,1,2,3) \end{cases}$$

Métodos implementados

Para o desenvolvimento correto de Q-Learning, foi necessário:

1. Implementar o algoritmo que processa um trajeto e retorna a qualidade de ações tomadas em dados estados (denominado `traces2Q`). Este algoritmo baseia-se na fórmula seguinte:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left(r + \gamma * max\left(Q(s_{t+1}, a)\right) - Q_{prev}(s_t, a_t)\right)$$

2. Implementar a política de *Exploration*, que procura pelas ações com probabilidades de sucesso > 0 num dado estado e escolher uma ação ao acaso.
3. Implementar a política de *Exploitation*, que dado um espaço de qualidade de ações e num dado estado, procura pela maior recompensa.

Vantagens:
Desvantagens:

Complexidade computacional

Para correr a rotina de política dado um número arbitrário **n** de experiências, quer de treino para aprendizagem, quer de pesquisa por ganância, será necessário recorrer a múltiplas funções auxiliares de manipulação das estruturas de dados usadas para o âmbito deste projeto.
Uma pesquisa não exaustiva na documentação aponta para complexidades na ordem de O(n log n), ficando aproximadamente **$O(n^2 \log n)$**.

Discussão de resultados

Em ambos ambientes foram obtidos resultados favoráveis, na potencial exceção do uso da política de *Exploitation* que raramente falha.

Para o treino do agente com *Exploration*, notou-se que a um certo número de experiências bastante grande (3000) não haverá diferença notável no resultado da qualidade da trajetória, demorando apenas mais tempo desnecessariamente. Para contrariar este defeito, pode-se recorrer a uma condição auxiliar que verifica se a diferença entre duas experiências é suficientemente pequena, travando o processo antes do tempo.