# ARTIFICIAL INTELLIGENCE

## PART 1

## BAYESIAN NETWORKS

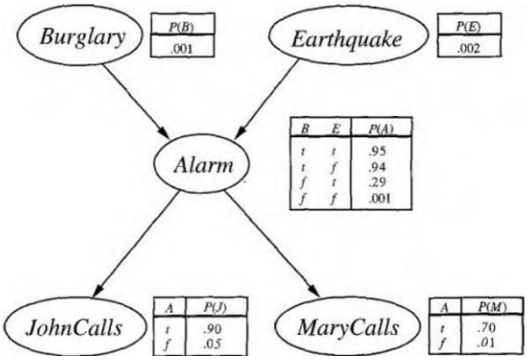


**Figura 1:** Representação da rede bayesiana utilizada no projeto

Uma rede bayesiana consiste num modelo que representa a relação probabilística entre um conjunto de varíaveis. As redes bayesianas tem por base o Teorema de Bayes, em que dado dois eventos A e B, P(A|B) = $\frac{P(B|A)P(A)}{P(B)}$. Esta probabilidade, em teoria, representa o grau de confiança de um agente que um dado evento vai ocorrer, baseado na observação prévia.

As redes bayesianas deste projeto são acíclicas e constituídas por nodes, em que para cada node, é possível calcular a probabilidade desse node ocorrer (e não ocorrer) a partir de uma lista de probabilidades e parents – a probabilidade de cada node é calculada através da função computeProb, com base numa lista de probabilidades e numa lista de parents.

Dentro das redes bayesianas, a partir de uma evidência, é possível calcular a probabilidade conjunta (joint probability). A partir da probabilidade conjunta, é possível chegarmos à probabilidade a-posteriori de uma varíavel (post probability), que corresponde à seguinte fórmula para calcular essa probabilidade: dadas duas varíaveis aleatórias discretas, X e Y, então:

P(X = x e Y = y) = $\frac{P(X=x,Y=y)}{P(X=x,Y=y)+P(\neg X=x,Y=y)}$.

Esta probabilidade é calculada, dada uma evidência, através das combinações possíveis para a variável desconhecida da evidência.

Após executar o ficheiro mainBN.py, observou-se que o código que foi produzido obteve resultados iguais aos pretendidos no ficheiro. Isto aplica-se tanto na função de computeProb, computeJointProb e computePostProb.

Face às desvantagens do uso de redes bayesianas, elas restringem a previsão apenas aos próximos n+1 acontecimentos dados os n acontecimentos previamente observado. Além disso, a complexidade das redes bayesianas – são computacionalmente caras.

Em termos de vantagens, permite a atualização desses valores de probabilidade sempre que uma nova observação é realizada e obtém-se novos dados. Além disso, face aos métodos normais, permite uma compactação dos dados em relações de independência condicional – sabendo os parentes de um node, permite-nos saber de que eventos ele depende para ocorrer.

A complexidade do código depende do número de combinações máximas que existem entre [0,1] e o número de valores desconhecidos na evidência que a função computePostProb recebe como argumento. Quanto mais valores desconhecidos existem, mais combinações possíveis existem, e mais lento será o programa. Ou seja, sendo o array [0,1]( (valores possíveis dentro da evidência) identificado como A e o número de valores desconhecidos [] representado por n, a complexidade do programa é O(C(A,n)), sendo C o símbolo de combinações.

É possível melhorar o algoritmo da seguinte forma: como na função

computePostProb usamos todas as combinações possíveis para calcular o resultado – o que torna o programa pouco eficiente – podemos reduzir o número de combinações possíveis, analisando se existissem variáveis independentes.

---

# PART 2
# BAYESIAN NETWORKS

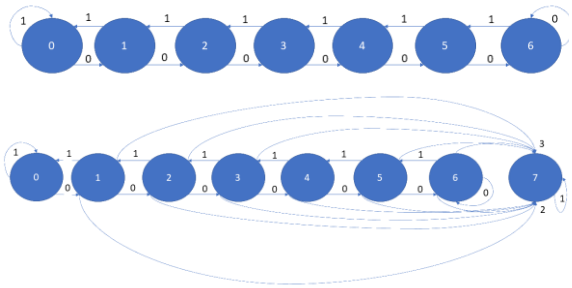

**Figura 2:** Autómatos correspondentes, respetivamente, ao teste 1 e 2 do ficheiro "mainRL.py"

O código produzido no ficheiro "RL.py" permitiu passar todos os testes executados com o ficheiro "mainRL.py". Apesar disso, nem sempre a aproximação de Q está dentro do previsto – isto é devido ao valor gerado pela função "policy" quando o agente está em modo "exploration" e o seu próximo estado é escolhido aleatoriamente e não garante que a ação escolhida seja a com maior recompensa ou mais eficiente. Apesar disso, a trajetória gerada é sempre óptima.
A base do projeto é no conceito de Aprendizagem por Reforço (Reinforcement Learning), mais concretamente, em Q-Learning, que é um algoritmo de aprendizagem por reforço com base em valores numéricos. O algoritmo de Q-Learning baseia-se na ideia de "recompensa", ou seja, que para cada ação que o agente pode efetuar, ele recebe uma recompensa que tanto pode ser positiva ou negativa. Estes valores da recompensa são mantidos numa matriz com os vários movimentos possíveis – tendo cada posição da matriz 4 possíveis estados finais (esquerda, baixo, direita, cima) – e tem como objetivo calcular o valor máximo de recompensa esperado.
Existem duas formas de explorar esta matriz e tomar decisões: o agente pode estar em modo "exploitation" – o agente escolhe sempre a decisão com maior nível de recompensa – ou em modo "exploration" – o agente escolhe uma decisão aleatória, que nem sempre pode ter o maior nível de recompensa.
Para calcular a recompensa para cada ação, a função "traces2Q" percorre os vários caminhos possíveis dentro da matriz, e calcula a recompensa para cada ação dado o caminho usando a seguinte fórmula, denominada equação de Bellman.

$$NewQ(s,a) = \underbrace{Q(s,a)}_{\text{New Q value for that state and that action}} + \underbrace{\alpha}_{\text{Current Q value}}[\underbrace{R(s,a)}_{\text{Reward for taking that action at that state}} + \underbrace{\gamma}_{\text{Learning Rate}} \underbrace{max\, Q'(s',a')}_{\text{Discount rate}} - Q(s,a)]$$

Maximum expected future reward **given the new s' and all possible actions at that new state**

A aplicação desta fórmula implica a atualização de Q. Eu suma, o algoritmo de QLearning baseia-se no seguinte conjunto de ações: Inicializar a matriz Q → Copiar os valores de Q para uma matriz auxiliar que, posteriormente serve para comparar valores → Escolher uma ação a → Realizar ação → Calcular recompensa → Atualizar Q.
As vantagens do uso deste algoritmo vêm do seu uso de movimentos parcialmente aleatórios que, aliados ao cálculo do valor da função recompensa para cada entrada da matriz Q, facilita a diminuição da "aleatoriadade" dos movimentos do agente, atingindido uma chamada "ótima" matriz Q. Apesar disto, como o algoritmo de QLearning se baseia numa de função de aproximação, pode tornar-se instável em fases mais avançadas da sua execução. Além disso, o algoritmo de QLearning não jé o algoritmo mais eficiente em termos de tempo de execução.
A complexidade do algoritmo implementado depende do espaço

ocupado pelo número de estados presentes. Sendo s o número de estados, a complexidade do algoritmo é **O(sn)**. Este pode ser melhorado, evitando que existam ações que são efetuadas múltiplas vezes.