Artificial Intelligence

Report for 2nd part of Project – "*Bayesian Networks and Reinforcement Learning*"

Bayesian Networks

A **Bayesian Network** is a probabilistic graphical model that uses a **directed acyclic graph** to represent the probability of variables and their conditional dependencies. It is therefore a very scalable application of the **Bayes' theorem**.

Algorithm used and complexity

The algorithm used to calculate and analyze Bayesian Networks is divided into 3 different methods spread across 2 different classes: "***Node***" and "***BN***" (Bayesian Network). A "***Node***" has information about which nodes are its parents and his **probability value for every single boolean combination of its parents**, representing a **single node** inside a Bayesian Network. A "***BN***" consists of a **group of nodes** that form a Bayesian Network.

The only method of the "*Node*" class is "***computeProb***" which given the information about the boolean values of its parents, **calculates the probability value of the node**. It iterates through all its probability values, as many times as there are parents, in order to find the probability value that corresponds to the parents' boolean combination given. This way, the **time complexity** of this method is ***O(n)***, in which '*n*' represents the **number of nodes** in the network, which is also the most parents a node can have (excluding itself it would be '*n-1*', but it's irrelevant for complexity analysis). The output is given in the format:

$$[P(\overline{N}|Parents), P(N|Parents)]$$

The methods of the "*BN*" class are "***computeJointProb***" and "***computePostProb***".

The "***computeJointProb***" method, given the boolean values of all the nodes, **calculates the joint probability** of the whole network. For every node, it runs its "*computeProb*" method to calculate its probability, and then multiplies all their probabilities. This way, the **time complexity** of this method is ***O(n^2)***, in which '*n*' represents the **number of nodes** in the network. The formula used is the following:

$$\prod_{i=0}^{n-1} P(N_i|N_{i+1,}\dots,N_{n-1})$$

The "***computePostProb***" method, given the boolean values for all known nodes, uses **probabilistic inference** to **calculate the "a-posteriori" probability value** of a given node (referred to as "target node"). It sums the joint probabilities for **every possible combination of boolean values for the unknown nodes** joined with the boolean values of the known ones, for both possible boolean values of the target node, and then returns the quotient between the probability value when the target node is True and the sum of the probability values with both the target node's boolean values. Because the calculation of all possible boolean combinations **relies on the cartesian product** of '*n*' lists with a maximum length of 2 and the joint probability is calculated for each resulting combination, its **time complexity** is ***O(n^2*2^n)***. The formula used to give the output is the following:

$$\frac{P(N_0,\dots,T,\dots,N_{n-1})}{P(N_0,\dots,T,\dots,N_{n-1}) + P\left(N_0,\dots,\overline{T},\dots,N_{n-1}\right)}$$

Advantages and Disadvantages

The main **advantages** are both its **ability to handle incomplete data sets** and the **ease of use of prior knowledge**.

The **ability of handling incomplete data sets** inherits from the exact concept of a Bayesian Network, which is a directed graph representing probabilistic dependencies between nodes. Which means there can be **multiple ways of inferring dependencies and back-tracking values**, and therefore **"information holes"** created by missing data **can be covered** with varying degrees of precision depending on the algorithm used and size of the data set.

The **ease of use of prior knowledge** is based on the dependency setup of Bayesian Networks, in which **a node can have any other node as a parent**, with that dependency having any weight desired, **as long as the graph is kept acyclic**. This makes it so that **a known dependency can be inserted** in the network, with a weight value that represents the importance it is believed to have, and test if the learned results are more satisfactory and/or approximate more closely to real values.

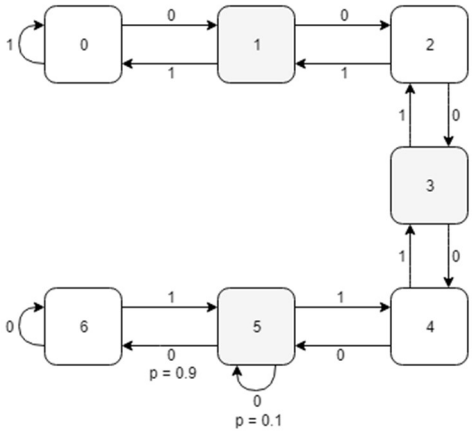
The most glaring **disadvantage** is its **computational inefficiency**, since the calculation of the "a-posteriori" probability values has an **exponential time complexity**.

Reinforcement Learning

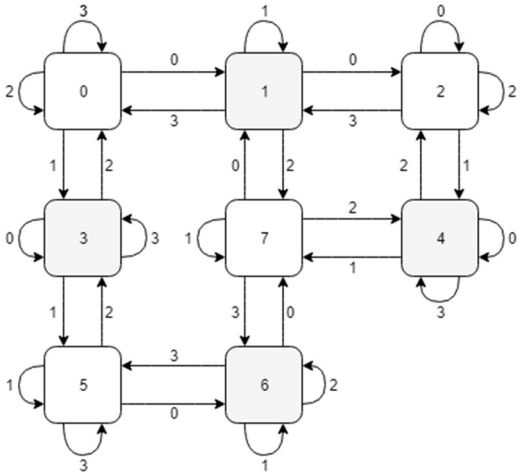
Reinforcement Learning is a learning technique in which an agent **learns the world by interacting with it** in a **reward-based system**.

Environments

There are two different environments being used. Their **graphical representations** are illustrated by **graphs 1 and 2**, as well as their **reward functions** (R) and **optimal policy** (π^*).



Graph 1: Graphical representation of Environment 1



Graph 2: Graphical representation of Environment 2

$$R(s,a) = \begin{cases} 1, & if\ s = 0\ \vee s = 6 \\ 0, & otherwise \end{cases}$$

$$\pi^*(a,s) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$R(s,a) = \begin{cases} 0, & if\ s = 7 \\ -1, & otherwise \end{cases}$$

$$\pi^*(a,s) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Environment 1 is **not deterministic**, because the transaction from state 5 to state 6 **relies on a probabilistic value,** while Environment 2 is **deterministic**.

Algorithm used and complexity

To apply the **Reinforcement Learning** technique, we are using the **Q-Learning** algorithm, that for **any finite Markelov decision process** finds a policy that **maximizes the total reward** over all its steps and given **infinite exploration time** and a **partly-random policy**, can achieve an optimal solution.

The algorithm used consists of 5 different methods: "***Q2pol***", "***VI***", "***policy***", "***runPolicy***" and "***traces2Q***", which are all part of the "***finiteMDP***" class that encapsulates the algorithm.

The "***Q2pol***" method, given the "*Q*" function, generates a pseudo-random policy using the "*softmax*" function. This means that its **time complexity** is ***O(s*a)***, in which '*s*' represents the number of states and '*a*' represents the number of actions possible. The "*softmax"* function is given by the formula:

$$\sigma(s)_i = \frac{e^{s_i}}{\sum_{i=1}^{a} e^{s_i}} \; for \; i = 1, \dots, a$$

The "***VI***" method **continuously approximates the "*Q*" function** using the environment-defining values, only stopping when the **difference between iterations is small enough**. This way its **time complexity** is ***O(a*s*i)***, in which '*s*' represents the number of states, '*a*' represents the number of actions possible and '*i*' represents the number of approximations made. The "*Q*" function is being approximated by the formula:

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

The "***policy***" method, given the type of policy to be followed and the current state, **decides the action to be taken next**. It resorts to calling the "*VI*" to reapproximate the "*Q*" function and calculate the new **"*softmax*" function to be used in case a "exploration"** type of policy is to be followed. If a **"exploitation"** type of policy is to be followed, it just **returns the best possible action** with the current data available. This way, its time complexity is ***O(a*s*i)***, in which '*s*' represents the number of states, '*a*' represents the number of actions possible and '*i*' represents the number of approximations made.

The "***runPolicy***" method, given the type of policy to be followed and the initial state, **generates a given number of trajectories** for the agent to follow. It resorts to calling the "*policy*" method for every trajectory generated, which means its time complexity is ***O(t*a*s^2)***, in which '*s*' represents the number of states, '*a*' represents the number of actions possible and '*t*' represents the number of trajectories generated.

The "***traces2Q***" method, given a set of trajectories (usually generated by the "*runPolicy*" method), **continuously approximates the "*Q*" function** using the trajectories' information along with its corresponding rewards, only stopping when the **difference between iterations is small enough**. This way its time complexity is ***O(t*a*s)***, in which '*s*' represents the number of states, '*a*' represents the number of actions possible and '*t*' represents the number of trajectories generated. The "*Q*" function formula used is identical to the one used in the "*VI*" method.

Advantages, Disadvantages and Limitations

The **main advantage** is the fact that it **does not need a model of the environment** in order to operate and learn, while the **disadvantage** is that, **if only by exploration**, the model **will never achieve the exact real solution**, only an approximation of it.

The most glaring **limitation** is that, **if using exploration**, the **minimum number of trajectories** required to be able to **ensure that the algorithm learns correctly** can be relatively high.

Teacher: *Manuel Cabido Lopes*

Group 27: *João Galinho* (87667) and *Filipe Henriques* (87653)