Assignment 4

Due: 17. December 2023, 23:59 CET

General information: This assignment should be completed in groups of **three to four people**. Submit your assignment via the corresponding Moodle activity. You can only upload one .zip file, therefore make sure to place all your files required to run/evaluate your solution into **one archive** and just upload that archive.

Python exercise information: You are only allowed to use the external libraries that we provide or explicitly mention on the assignment sheet (or that are already within the provided skeleton files). Your solutions' functionality will all be tested against automated tests before being reviewed further. **You need to ensure** that your solution can directly be **imported as a Python3 module** for automatic tests and at least contains the name of the skeleton file. The easiest way to achieve this, is to directly develop your solution in the assignment4.py file and submit that. The Moodle contains information about the test environment. You will also find an example test-file alongside the skeleton file that you can use to ensure that your solution works in the evaluation environment.

Exercise information: In this assignment you will be implementing your first sampling methods by using the previously developed network classes as well as a factor class that we provide. You will again find the provided reference implementations (for the graph and the factor class) in the *ccbase* package in the *assignment4.zip* alongside the *assignment4.py* skeleton file and the example test file. As with the last assignments, feel free to use your own graph class. You are free to use **numpy** as a third party library.

Exercise 1: (8 Points)

Apart from probability queries there are also value queries, i.e. queries for the instantiations of all or a subset of variables that maximize the joint probability.

Task 1: (2 Points) The *maximize_out(factor, variable)* function, which removes the given variable from the factor via maximization.

Task 2: (2 Points) The *max_product_elim_var(factors, variable)* function, which takes a set of factors and removes the given variable from this set of factors by maximizing the resulting factor with respect to that variable.

Task 3: (2 Points) The *traceback(factors, order)* function, which will return the instantiations for all variables that maximize the joint probability based on the factors from which variables were removed in *max_product_elim_var* and the given elimination order.

Task 4: (2 Points) The *calculate_MAP(bn, evidence=None)* function, that actually computes probability of the *most probable explanation* (MPE) as well as the instantiations of all variables that make up this MPE.

Hint: Maximizing out a variable from a factor is defined as [1]:

$$(max_x f)(\mathbf{y}) = max_x f(x, \mathbf{y})$$

Hint 2: You can check your result for the MAP function by comparing the probabilities of the joint distribution with your MPE probability.

General hint: Keep in mind how factors change with evidence and that valid probability distributions need to sum up to 1. Also keep in mind that you can still use a function from a previous exercise/task, even if you did not solve it yourself. As long as you use the functions correctly you will still get full points for the subsequent exercises.

Exercise 2: (2 Points)

A prerequisite for approximate inference methods is the ability to generate *samples* from a single univariate distribution. Consider for example the variable *Color* with values *{red, green, blue}* and distribution $P(Color)$:

$$P(Color) = \begin{cases} 0.1 & red \\ 0.6 & green \\ 0.3 & blue \end{cases}$$

Implement the method *sample(distribution)* that generates a sample from a discrete distribution such as the one above[1]

[1]Hint: Make use of python's inbuilt random-lib, e.g. using random.random(), to obtain random numbers in the interval [0,1], or numpy's equivalent.

Exercise 3: (5 Points)

There a many different approximate inference algorithms. Arguably the easiest one is called *Forward Sampling*:

Task 1: (2 Points) Implement the function *get_ancestral_ordering(graph)*, that returns the an ancestral ordering of the nodes in the graph, i.e. parent nodes always come before their children in the list.

Task 2: (2 Points) Implement the function *do_forward_sampling(bn, var_name, num_samples=1000)* for approximate inference in Bayesian networks using *Forward Sampling*.
Your algorithm should be able to calculate marginals for variables **without** given evidence.

Exercise 4: (8 Points)

One problem with basic *Forward Sampling* is that it cannot cope well with evidence for nodes that have parents. A possible solution is to implement *Rejection Sampling* or *Likelihood Weighting*. With *Rejection Sampling*, one discards samples that are not in line with given evidence. However, to obtain a decent approximation for networks with a lot of variables, *Rejection Sampling* often needs too many samples and the computation therefore becomes inefficient. An alternative to costly forward sampling under evidence is a method called *Gibbs Sampling*. Unlike *Forward Sampling*, *Gibbs Sampling* does not start all over again for every sample, but generates a new sample based on the previous sample by creating a MarkovChain with the desired stationary distribution.

Task 1: (1 Point) Implement the function *create_initial_sample(net, evidence)* that creates an initial sample useable for Gibbs sampling.

Task 2: (3 Points) Implement the method *get_markov_distr(node, evidence)* that computes the (local) distribution of the node given the evidence.

Task 3: (4 Points) Implement the method *do_gibbs_sampling(bayesnet, variable, evidence, num_samples=1000, burn_in_period_length=100, thinning=1)* for approximate inference in Bayesian networks using Gibbs Sampling.
Your algorithm should be able to calculate marginals for any single variable in the network with and without given evidence.

References

[1] Darwiche, Adnan. *Modeling and reasoning with Bayesian networks.* Cambridge University Press, 2009.