# Assignment 5

Due: 14. January 2021, 10:15 CET

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

**Exercise information:** In this assignment you will be implementing functions useful for sampling as well as Gibbs Sampling as one of the most prominent approximate inference methods. You will also find the first exercise about decision making on this assignment. You will again find the reference graph implementation, in the *ccbase* package in the *assignment5.zip* alongside the *assignment5.py* skeleton file. the example-test file and some generated documentation. The ccbase.networks module was updated to now include exact inference algorithms which you can use for exercise 4, or to compare your approximate results to the exact solutions.

Pay attention to the provided docstrings as they further explain what is expected from the different functions. **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 assignment5.py file and submit that. 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:** (2 Points)

A prerequisite for approximate inference methods is the ability to generate *samples* from a single univariate distribution[1]. 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[2]

**Exercise 2:** (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:** (3 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.

[1] In this exercise we will only consider discrete distributions.

[2] 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:** (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:** (2.5 Points) Implement the method *get_markov_distr(node, evidence)* that computes the (local) distribution of the node given the evidence.

**Task 3:** (4.5 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.

You get one 1 point each for correctly handling the *burn_in_period_length* and *thinning* arguments, i.e. the function is worth 2.5 points if you ignore those arguments.

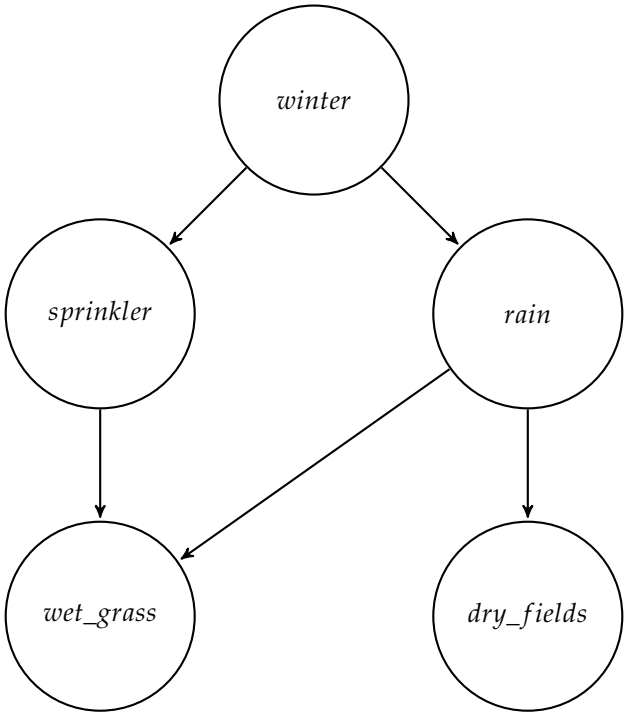
**Exercise 4:** (5 Points)

The next step in the lecture after inference is using the inferred knowledge to act on the world and to determine the likely effects of an agent's actions in order to decide on which action to take.

Consider the following utilities and the corresponding wet_grass, dry_fields network depicted below:

U(wet_grass = True) = 20; U(wet_grass = False) = $-10$

U(dry_fields = True) = $-20$; U(dry_fields = False) = 10




You are able to force the sprinkler to be turned on (*True*) or off (*False*).

**Implement the *expected_utility* function such that it can solve the following tasks:**

**Task 1:** (3 Points) Implement the *expected_utility(net, actions, evidence, utilities, use_do=False)* such that it can compute the expected utility for given actions (which can be considered outcomes of nodes in the network).. The function should be able to handle the *Do*-operator properly if used. When not using the *Do*-operator, interventions should be treated as normal evidence. In both cases, the function should work with additional observational evidence.

**Task 2:** (1 Point) Compute the expected utility for turning the sprinkler on (action 1) or off (action 2) without the *Do*-operator with and without the evidence *winter=False*.

**Task 3:** (1 Point) Computing the expected utility of both actions when applying the *Do*-operator correctly, again with and without the evidence *winter=False*.

You may compute the expected utilities manually or use your function. If you do it manually, show your computation steps, if you use the function, include all required function calls with all parameters in your submission and print out the results in an appropriate way.