Assignment 3

Due: 3. 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 assignment3.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 inference methods by using the previously developed network classes as well as a factor class that we provide. You will find a reference graph implementation in the *ccbase* package in the *assignment3.zip* alongside the skeleton file. The graph implementation can be found in *ccbase/networks.py*, while the used node implementation is located in *ccbase/nodes.py*. The *assignment3.py* file contains the skeletons for the functions that you should implement for this assignment. For any function that can take either a Node object or a Node's name as string as an argument, you only need to ensure that your solution works with one or the other, although you can also implement it in such a way that works with both. You are free to use your own graph class from the first assignment instead, however, incorrect behavior of the assignment's tasks due to bugs in your graph class may still lead to point reductions.

Exercise 1: (5 Points)

Implement the functions required to perform a (conditional) independence check using *d-separation*:

Task 1: (1 Point) Implement *is_collider(dg, node, path)* which returns *True* only if the given node is a collider with respect to the given (undirected) path within the directed graph.

Task 2: (2 Points) Implement the function *is_path_open(dg, path, nodes_z)* that returns *True* only if the given path is *open* (according to d-separation) when conditioned on a (potentially empty) set of variables **Z**.

Task 3: (1 Point) Implement the function *unblocked_path_exists(dg, node_x, node_y, nodes_z)* which checks (and returns *True*) if there exists an unblocked undirected path between two variables X and Y, given a (potentially empty) set of variables **Z**.

Task 4: (1 Point) Implement *check_independence(dg, nodes_x, nodes_y, nodes_z)* which tests if two *sets* of variables **X** and **Y** are conditionally independent given a (potentially empty) variable set **Z**.

Exercise 2: (6 Points)

Now implement the following functions for the general test. You may want to re-use some of the functions you implemented for the last exercise.

Task 1: (2 Points) Implement the *make_ancestral_graph(graph, nodes)* function which takes a (directed or undirected) *graph* and sets of nodes and returns the corresponding ancestral graph for these nodes.

Task 2: (1 Point) Implement the *make_moral_graph(graph)* function which takes a directed (ancestral) graph and returns its (undirected) moral graph.

Task 3: (1 Point) Implement the *separation(graph, nodes_z)* function which seperates all links from nodes in the set *nodes_z* within the (undirected) graph g.

Task 4: (2 Points) Implement *check_independence_general(graph, nodes_x, nodes_y, nodes_z)* which again tests if two *sets* of variables **X** and **Y** are conditionally independent given a (potentially empty) variable set **Z**. Contrary to the d-separation version above, this should work for both directed and un-directed graphs using the general graphical test.

Exercise 3: (3 Points)

The order in which variables are being eliminated has a significant effect on the performance (runtime) of variable elimination algorithms. Finding an optimal elimination order is NP-hard, but different heuristics exist that can yield mostly good orders, such as the *MinFillOrder* heuristic:

When removing a variable, we need to update the interaction graph by connecting the variable's neighbors that were not connected before. In order to keep factors small, we want to eliminate the variable that causes the least amounts of edges to be added between neighbors when removing that variable.

Task 1: (3 Points) Implement the *get_elimination_ordering(bn)* function, which computes the elimination order according to the *MinFillOrder* heuristic.

Exercise 4: (6 Points)

Implement exact inference in Bayesian networks following the *variable elimination* algorithm. Make use of the provided factor class.

To this end you should implement:

Task 1: (2 Points) The *initialize_factors(bn, evidence)* function, which creates a list of factors from the given Bayesian network and applies the given evidence to them.

Task 2: (2 Points) The *sum_product_elim_var(factors, variable)* function, which takes a set of factors and removes the given variable from this set of factors.

Task 3: (2 Points) The *calculate_probabilities(bn, variables, evidence=None)* function, that actually computes the (joint) probability of the given variables, considering the provided evidence.

Your algorithm should handle the following cases:

- query and calculate prior marginals for every variable in the network
- query and calculate posteriors given evidence observed for one or more other nodes (evidence can be empty)

Hint: You may want to use the functions of earlier tasks when solving later ones. You could hard code solutions for some of these functions for simple examples, if you want to test later ones.