Part one: We have implemented the ComputePartitionFunction algorithm. To do so we had to develop two functions:

getCutset(T**,** w)

calculates the cutset of the inputted junction tree until its largest cluster is at most of size w.

We first have found some data needed from the Junction tree we got has an input:

* Variable\_appearences: we computed the appearances of each variable in the Junction tree (we counted for every variable the number of times it appeared in a node from the junction tree – clique nodes):
* Most\_occurent\_variable: the variable that is the most frequent in all the clique nodes.
* Largest\_cluster\_size: the size of the largest clique node.

We then created a while loop that runs as long as the largest\_cluster\_size is bigger than the given threshold (w – input of the function).

In this loop we updated every edge that had a connection to a node that holds the Most\_occurent\_variable to not point on the node but to its new version without the Most\_occurent\_variable.

Then we updated the nodes (clique nodes) that had the Most\_occurent\_variable to no longer have it has part of their variables.

At the end of every iteration of this loop we updated largest\_cluster\_size (part of the stop condition).

When the threshold was reached, we returned the list of variables that were the Most\_occurent\_variable at each iteration in the order we computed them and the new junction tree that was created from all those changes.

generate\_sample(X)

simple function that gets a list of elements and creates a binary permutation of values to each of these elements as a dict. Basically assigns binary values to the elements in the list it gets.

Part two :

computePartitionFunction(MN**,** w**,** N**,** distribution="QRB")

the function we have been asked to implement.

This function gets a Markov network, a threshold as described in getCutset, a number of iterations and an indicator for which type of distribution we are looking for.

uniform distribution:

We calculate the probability of a permutation to be according to the size of X (the removed factors output from getCutset). Then we assign values using the generate\_sample function to these variables. calculated the partition of this assignment using computePartitionFunctionWithEvidence. Using these results we runed this process N times to find the Z value as described in the assignment.

QRB distribution:

The process is theoretically the same as for the uniform distribution. But this time the probabilities of each assignment is different thus, we had to use the belief\_propagation.query(X). using the results of this function, we repeated the process above.

To use our functions:

getCutset(T, w): T – junction tree, w – int.

generate\_sample(X): X – list.

computePartitionFunction(MN, w, N, distribution="QRB"): MN – markov network, w – int, N – number of iterations, distribution – string.

Part 3 results:

Uniform:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 50 | 100 | 1000 | 5000 |
| 1 | {'Time': (0.0907930830602821, 0.07061801160524524), 'Error': (0.45287408386321654, 0.2044511282086539)} | {'Time': (0.10097605371691662, 0.08058217382214589), 'Error': (0.3511577915587769, 0.2370325970422534)} | {'Time': (0.9293993832052427, 0.8218281386911195), 'Error': (0.1678806930543071, 0.06977340506612834)} | {'Time': (2.100712364140398, 1.8837402649489576), 'Error': (0.07170064868444942, 0.004766032007501107)} |
| 2 | {'Time': (0.09672790490811574, 0.0764040378409077), 'Error': (0.44750362546151595, 0.20340893368209695)} | {'Time': (0.10502231928700279, 0.07890173581248452), 'Error': (0.4323123591613141, 0.14762151731324163)} | {'Time': (0.8945669098161178, 0.5872113303877398), 'Error': (0.20368946515576264, 0.04488391228971801)} | {'Time': (2.1546589113596055, 1.8086990617391492), 'Error': (0.12255883825018335, 0.03719175387545182)} |
| 3 | {'Time': (0.1255603869487205, 0.06404499215589378), 'Error': (0.4025865155949947, 0.17442875473343267)} | {'Time': (0.09483553815124915, 0.07780244898559166), 'Error': (0.3108588597057443, 0.11342828317363084)} | {'Time': (0.758905623358191, 0.498395802575647), 'Error': (0.18428334282716524, 0.06002472034965993)} | {'Time': (5.002700072041039, 2.9042655174822762), 'Error': (0.11832802993464443, 0.022110756469564716)} |
| 4 | {'Time': (0.10120546293145005, 0.06690681505316909), 'Error': (0.46848662538007685, 0.22022293679095378)} | {'Time': (0.10347432992079289, 0.08052806952378719), 'Error': (0.3898627990186499, 0.22190443196772308)} | {'Time': (0.48304137192307156, 0.40840660132827117), 'Error': (0.18334147576320325, 0.10368188341602007)} | {'Time': (4.560940472217767, 3.8212394556081604), 'Error': (0.09757917674571699, 0.040607316077523894)} |
| 5 | {'Time': (0.08667840280839313, 0.06708923062971721), 'Error': (0.4718248589763796, 0.251081246265074)} | {'Time': (0.10850557327804844, 0.07754193305434903), 'Error': (0.3833147937443602, 0.07846246619313557)} | {'Time': (0.5723624065533601, 0.3584991142138518), 'Error': (0.2393246123980725, 0.09288682315412723)} | {'Time': (4.169242008052765, 3.3675565644875176), 'Error': (0.11393250082001663, 0.021988778249175944)} |

The results we got were calculated relatively quickly, we see that the time of computation rises with the number of iteration N, something that is logical as repeating an action several times is longer than doing it only once.

We additionally constate that the standard deviation seems to lower as N grows. We do not seem to find a correlation between the parameter w and the error.

QRB:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 50 | 100 | 1000 | 5000 |
| 1 | {'Time': (2.2057193992410125, 0.7749968292440951), 'Error': (0.0, 0.0)} | {'Time': (2.8296003140408574, 1.8123415671389516), 'Error': (1.69620910604371e-16, 5.253464969929325e-17)} | {'Time': (3.235196564881405, 2.7570564048604167), 'Error': (2.776944503795803e-16, 2.776944503795803e-16)} | {'Time': (3.469420087022029, 2.67157136233405), 'Error': (1.1107778015183213e-15, 1.1107778015183213e-15)} |
| 2 | {'Time': (1.5012370315028536, 1.0832192692326201), 'Error': (0.0, 0.0)} | {'Time': (1.679246273702557, 0.9605034916935611), 'Error': (1.2723928885287183e-16, -1.6161508701039693e-17)} | {'Time': (2.995167172517959, 1.4511964673135842), 'Error': (2.776944503795803e-16, 2.776944503795803e-16)} | {'Time': (3.657477278016185, 2.6193334157246597), 'Error': (1.1107778015183213e-15, 1.1107778015183213e-15)} |
| 3 | {'Time': (1.6322636853400134, 1.012785505181227), 'Error': (0.0, 0.0)} | {'Time': (1.4997053699712426, 1.0335079593439431), 'Error': (1.6886985051021531e-16, 8.105515483140698e-17)} | {'Time': (3.714327582665095, 3.0654603392404227), 'Error': (2.776944503795803e-16, 2.776944503795803e-16)} | {'Time': (3.3863830770264842, 2.1592678343046927), 'Error': (1.1107778015183213e-15, 1.1107778015183213e-15)} |
| 4 | {'Time': (1.494854337478153, 1.1456100637669642), 'Error': (0.0, 0.0)} | {'Time': (1.252194793052461, 1.0108930041124558), 'Error': (1.2723928885287183e-16, -1.6161508701039693e-17)} | {'Time': (3.731563577421995, 2.8696542169963397), 'Error': (2.776944503795803e-16, 2.776944503795803e-16)} | {'Time': (6.781833082327585, 5.999097389092702), 'Error': (1.1107778015183213e-15, 1.1107778015183213e-15)} |
| 5 | {'Time': (1.8895530083095384, 1.2795986793125318), 'Error': (0.0, 0.0)} | {'Time': (1.743564258841554, 1.0259891640863024), 'Error': (1.4260252566056868e-16, -3.7553004707785376e-18)} | {'Time': (3.4421799506089337, 2.8778069172957292), 'Error': (2.776944503795803e-16, 2.776944503795803e-16)} | {'Time': (6.68073324586501, 4.293319803222515), 'Error': (1.1107778015183213e-15, 1.1107778015183213e-15)} |

Using this distribution, we notice that a low number of iterations N directly affects the Error rate, the lower the N the smaller the error.

Here too the number of iterations affects the time run time similarly to the above distribution.