Mini-project report

Martin Pettersson

Project: Construction of Bayesian Networks from data. It can be tested with a Python 3.0 interpreter.

Implementation:
I implemented a BayesNet class in Python with the following functionality:
It's able to

- Randomize a valid structure given the number of variables and causal links.
- Randomize all the parameters of the structure.
- Generate data from the structure + the parameters.
- Calculate log(P(Data|BN)) given a data-set.
- Calculate the Bayesian Information Criterion (used as the utility function), given a data-set.
- Estimate the parameters given a data-set.
- Do a hill-climbing local search for the best structure given a data-set.

Generally, since I didn't have any test data, I had to generate data from a given structure to have something to test it with. This process looked like this:

- Create an instance of the BayesNet class, with a specified number of variables and causal links. This will generate a random valid BN without cycles.
- Randomize the parameters.
- Generate some data from this structure (taking all the conditional probabilities into consideration).
- Create a new BayesNet with the same number of variables and do a local search for the best structure.

Specifics about the techniques I used:

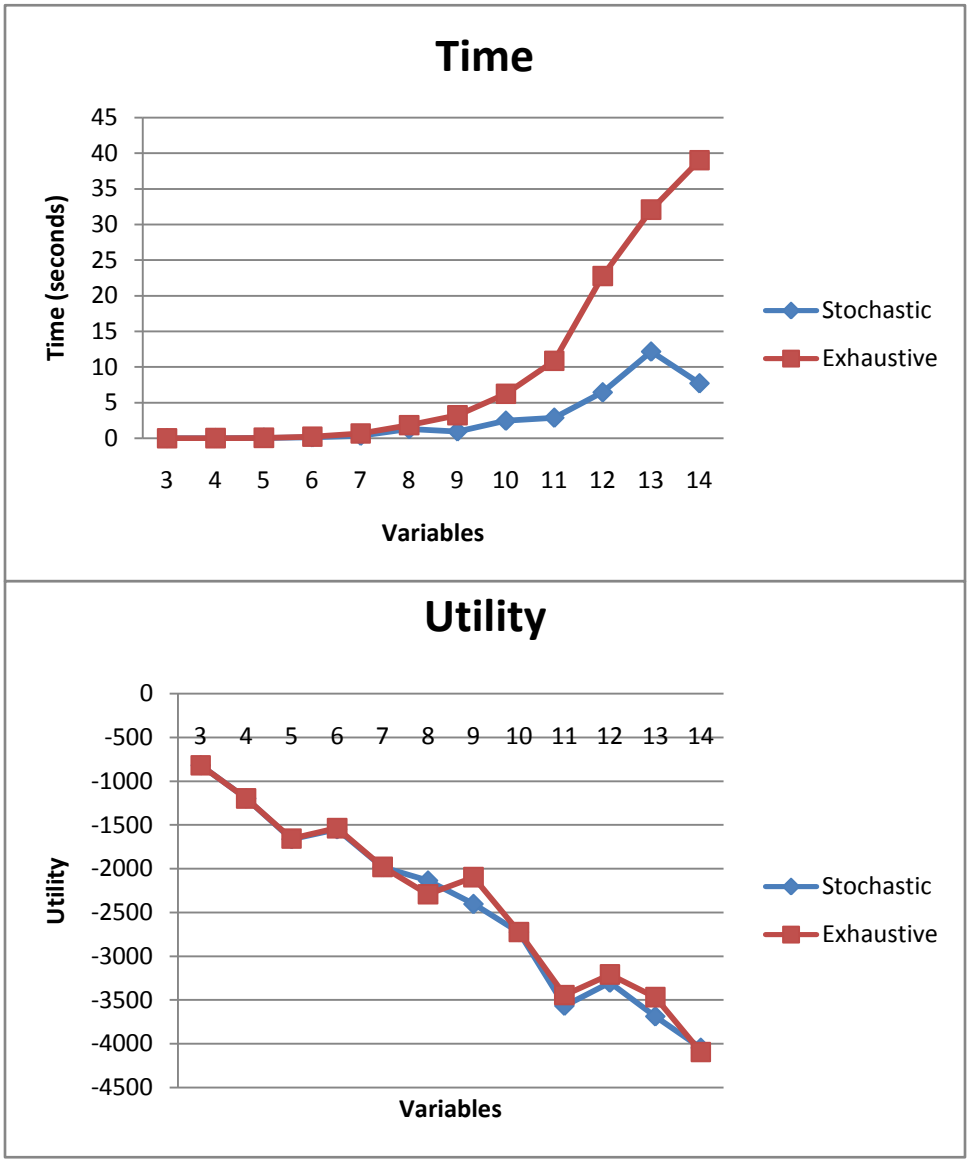
- For representing the BN-graph I used two types of adjacency-lists in parallel, one for getting the variables you have a causal link to, and one for getting your parents. This because I needed to get the parents for calculating the parameters etc and the other one for searching etc.
- For storing the adjacency-lists and the parameters, I used the built in Dictionary in Python.

What I learned:

- When dealing with lots of data, I gained a huge speed-up by aggregating it, e.g. summing all equal data points to one. This gives an upper bound for the slow-down by the amount of data one has, if you aggregate it before searching.
- I was distressed at first when my local search didn't find the same structure as the original one, but I realized that when randomizing the parameters, many of them will be close to 0.5 and thus not give any useful information to the network. When I looked at the actual utility of the new structure, it was often higher than the original one.

- I had to deal with log(0) in some way. I chose to simply add/subtract a small number to the probability when necessary.
- When it comes to searching, I used three different operations, add, remove or reverse a link.
- When searching, I came up with two different methods for choosing the best operation, either an exhaustive search of all possibilities or randomly check a specified number of different operations and choose the best one. When choosing randomly, I also had to add an "iteration buffer" so that it doesn't stop prematurely. 5 iterations seemed like a good number.
- I tried starting the search with either an empty network or one with randomized links, but the difference seemed negligible.

Here is a performance experiment for the two methods of choosing an operation; I was using 3-15 variables and 500 data-points and measured the utility of the final structure produced by both methods, together with the time it took to produce it. In the stochastic method I chose how many random operations to choose from to be the same as the number of variables.



From the graphs we can see that searching with the exhaustive method may not pay off as you have more variables. However, the parameters I chose for the BNs, the amount of data etc were a bit arbitrary. Because of the data aggregation, the slow-down with more data-points levels off after a while when the number of unique data-points goes towards 2 to the power of #Variables. The aggregation itself of course takes time, but it's not a part of the actual search, only a kind of preprocessing.