AML Lab 1: Graphical Models

Joshua Hudson

5 September 2017

(1)

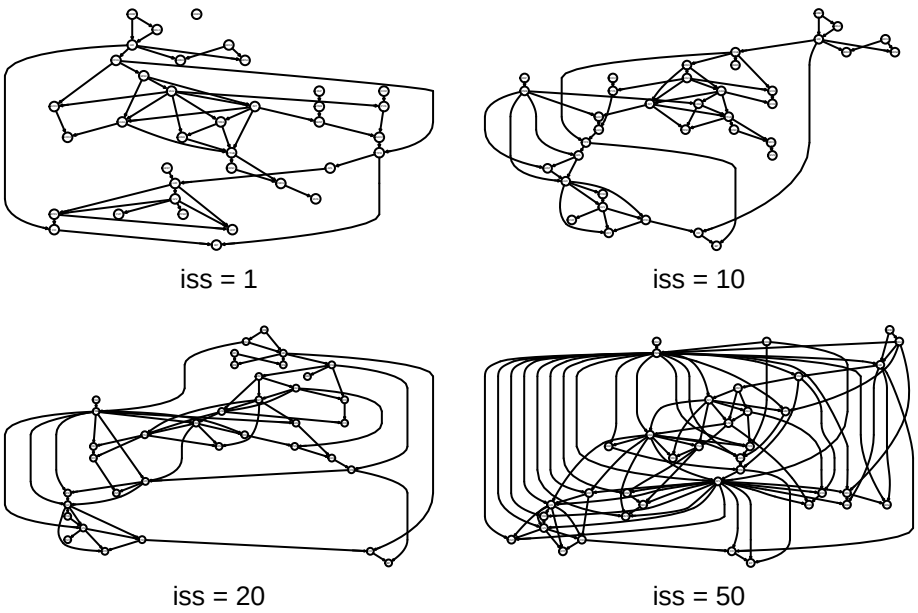
In this assignment, we used the `alarm` data set to show how the hill-climbing algorithm can return non-equivalent DAGs. We ran the algorithm 5 times using the implementation `hc()` from the `bnlearn` package. We then used the `cpdag()` function to obtain the essential graphs for each of the 5 DAGs generated. Finally, we compared the remaining 4 of the essential DAGs in turn with the 1st one (denoted "0th"" graph), usin the `all.equal()` function. The output is shown below, showing a list of the comparison results: comparing graph 0 in turn to graph 1, then graph 2, etc...

```
## [[1]]
## [1] "Different number of directed/undirected arcs"
##
## [[2]]
## [1] "Different number of directed/undirected arcs"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] "Different number of directed/undirected arcs"
```

We see that only the 3rd essential graph is identical to the 0th one, graphs 1, 2 and 4 are different. Therefore, as 2 DAGs are equivalent if and only if they have the same essential graph, we can see that the hill-climbing algorithm can return non-equivalent DAGs. This is because the hill-climbing algorithm is not asymptotically correct as arbitrarily picks an initial node to start from, which can lead to different final graphs.

(2)

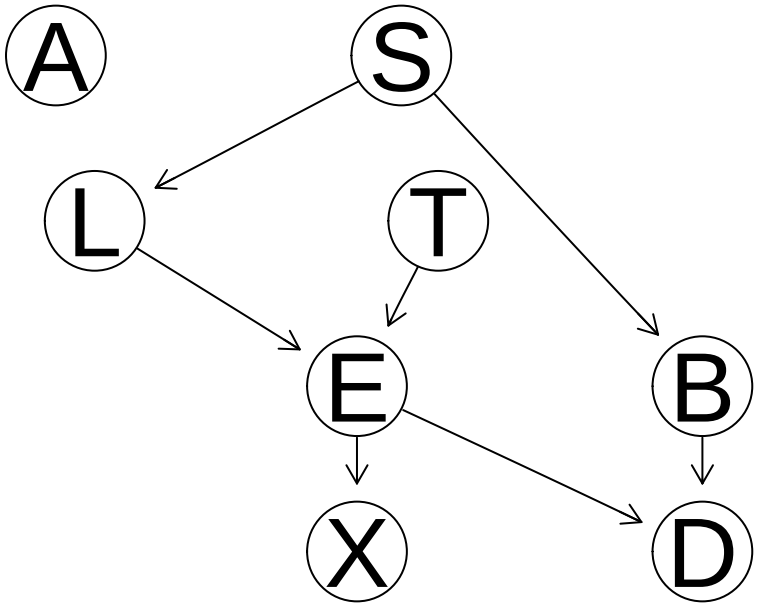
In this assignment, we examined the impact of changing the imaginary sample size (ISS) parameter when using the `hc()` function for structure learning. To do this, we ran the algorithm on the `alarm` data set 4 times, with ISS values 1, 10, 20 and 50 respectively. We then plotted the resulting DAGs, as shown below.



We observed that the higher the ISS value, the more densely connected the resulting DAG. In fact, an ISS of 1 gives 55 arcs, while an ISS of 50 gives almost double the amount with 101 arcs. Imaginary Sample Size determines the weight of the prior compared to the sample, which in turn leads to posterior smoothing. Therefore high ISS values give very different networks a similar score so extra arcs can be added unnecessarily.

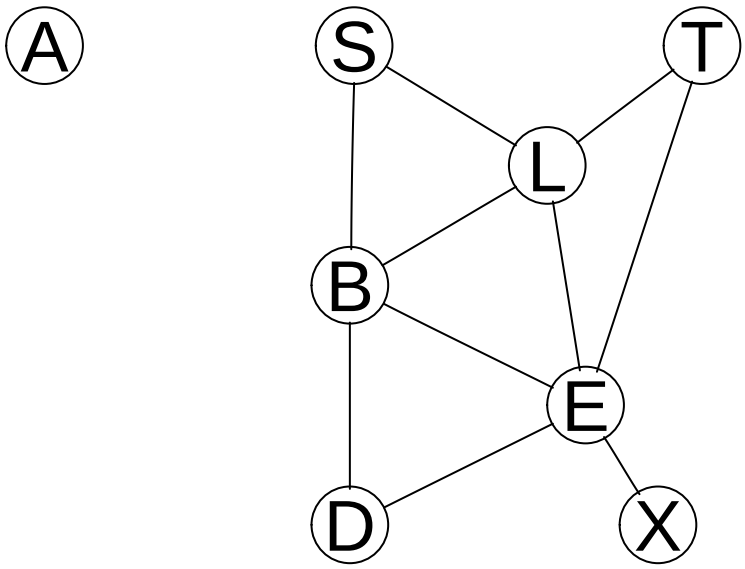
(3)

In this assignment, we switched our focus away from structure learning and onto parameter learning and inference. We started by generating a DAG from the `asia` data set, using the `hc()` function and fitting its parameters using the `bn.fit()` function. The DAG is shown below.



The first algoithm we used was the Lauritzen-Spiegelhalter algorithm for exact inference. We used the implementation in the `gRain` package. The first step of this was to transform our DAG into a junction tree

through moralisation and triangulation; this was done with the `compile()` function. The resulting junction tree is shown below.



Finally we set the evidence, i.e. we set the states of certain nodes to some values: for example we first set E="yes".

(4)

We know that there are 29281 DAGs with 5 nodes. In this assignment, we aimed to find the proportion of non-equivalent DAGs among these DAGs. To do this we generated 5-node DAGs with the `random.graph()` function. We used Melancon and Philippe's Uniform Acyclic Digraphs algorithm as the method. We then used the `cpdag()` function to reduce each to its essential graph and counted the proportion of unique ones with respect to the total number of DAGs we generated.

We generated 10,000 DAGs using the default parameter settings for `random.graph()` and obtained 3422 different independence models, so 0.3422 of the DAGs. Next we increased `every` parameter to 5 and repeated the experiment. This time we obtained a proportion of 0.4791 different independence models. This is because the `every` parameter sets the fraction of graphs that make the final cut: setting it to 5 means the `random.graph()` outputs 1 graphs every 5 steps in the algorithm, leading to an increased diversity in the final graphs. Note that the default setting was 1, so every graph was put in the output. Finally we changed the `burn-in` parameter from the default of 150 (the number of nodes squared, times 6) to 2000.