Bachelor thesis

Artificial Intelligence

Radboud University

Exploring possible improvements on likelihood weighting by omitting superfluous information and incorporating backward sampling

Author:
István Horváth
s1011310

First supervisor:
Nils Donselaar, MSc.
Artificial Intelligence
nils.donselaar@donders.ru.nl

Second supervisor:
dr. Max Hinne
Artificial Intelligence
max.hinne@donders.ru.nl

January 30, 2022

Abstract

Since exact inference in Bayesian networks is computationally intractable, approximate inference is the more feasible approach to many problems, in theory. This research focuses on proposed enhancements, integrating backward simulation and exploiting the sufficient information theorem, of stochastic simulation methods. These proposed improvements are added to the likelihood weighting algorithm, and tested on two large benchmark networks from the bnlearn repository. Evidence is shown, that omitting superfluous information improves running time, simply because less sampling has to be performed, regardless of network structure, but does not have an effect on convergence. Results on backward simulation however, are dubious.

1 Introduction

A Bayesian network is a directed acyclic graph that is used for representing and reasoning about uncertain information. The random variables are represented as nodes of this graph, and the dependencies between these variables are represented by the edges.

This thesis is concerned mainly with reasoning within Bayesian networks, in particular with what is known as Bayesian inference. Bayesian inference is the process of computing probabilities of certain nodes in a Bayesian network, given some observation(s). Solutions for Bayesian inference can be divided into exact or approximate inference. During exact inference the exact probabilities are computed, while approximate inference gives us a bounded estimate of the inferred probabilities. The ones that concern this thesis are approximate inference algorithms.

Inference in Bayesian networks, both exact and approximate however, is computationally intractable [1, 2]. This, in many cases, makes the application of exact inference unfeasible, especially if there is a time-constraint. While in general approximate inference is also computationally intractable, many approximate inference algorithms are anytime algorithms, meaning that they can generate an answer up to a certain precision relatively quickly. This gives them an edge over exact inference algorithms in some application domains.

The basis of the experiment is the likelihood weighting algorithm. It is not the state of the art approximation algorithm for Bayesian inference, but it is one of the first stochastic simulation methods, and many newer techniques build on it. Likelihood weighting performs well on Bayesian networks that do not have extreme probabilities, but possibly it could do even better and be of practical use, as it is relatively easy to implement and modify. The low cost of implementation and modification also makes it a great experimental ground for new ideas.

The first possible improvement proposed by [3] is based on the sufficient information for inference theory [7]. The idea behind it is that 'not all pieces

of evidence will bear on all events of interest '. If we take into account all evidence during inference, then it is possible that the noise created by 'unnecessary' evidence will lead to a slower convergence than if only the evidence which gives sufficient information would be considered [3]. This can be generalised to all variables in the network, not only evidence nodes, such that many nodes that do not give any information about the inferred variable can be disregarded, without an effect on accuracy but with an improvement on running time. Computing which variables contain all the sufficient information for the particular inference problem comes with a fixed cost, so even in case this does lead to a better performance rate, it might not be of advantage in practice, or may only lead to a real improvement in certain scenarios.

The second idea for improvement is incorporating backward simulation into the likelihood weighting algorithm. This could help in dealing better with extreme likelihoods. Backward simulation is related to evidential integration [4], which has been used on the likelihood weighting algorithm by [3] but proved to be too costly (although it did achieve a better convergence rate). Backward simulation has a lower cost than evidential integration [4], so it can be useful in dealing with unlikely evidence.

The literature found during the completion of the project suggests that these are yet untested ideas. Research in stochastic simulation focused elsewhere and had resulted in some exciting results, such as importance sampling, in which sampling is done from an approximation of the posterior distribution [5, 8] or algorithms based on likelihood weighting and Stopping-Rule Theorem [5].

2 Preliminaries

2.1 Bayesian Networks

A directed acyclic graph is of the form $G = \langle V, E \rangle$, such that V is a set of N vertices $\{v_1, v_2 ... v_n\}$ and E is a set of tuples (v_x, v_y) where $v_x, v_y \in V$ represents a path from v_x to v_y; in this case v_x is called a parent of v_y which is denoted as $v_x \in Pa(v_y)$. A vertex v_x is a descendant of a vertex v_x iff there is a directed path from v_x to v_y, denoted as $v_x \in Des(v_y) = v_x \in Pa(v_y) \cup Anc(Pa(v_y))$. A vertex v_x is an ancestor of v_y iff $v_y \in Des(v_x)$; denoted as $v_x \in Des(v_y)$. In a directed acyclic graph there always exists at leas one ordering of vertices, such that for every edge $(v_x, v_j) \in V$ v_x comes before v_j, called the topological ordering.

A Bayesian network B is defined as $B = \langle G, Pr \rangle$, where G is a directed acyclic graph, such that the vertices and edges of G represent stochastic variables and the conditional dependencies between them and Pr consists of the probability distributions for each variable $v \in V$, conditioned on its parents.

2.2 Necessary Information

While reading off direct dependencies from a Bayesian network structure is straightforward, there are also indirect dependencies in the network, conditional on other variables.

Let's define two vertices, $v_x, v_z \in V$ and a set of vertices $E \subset V$ such that the values of the vertices in E are known. x and z then, are conditionally independent given Y if all undirected paths between them are d-separated given Y.

A path between x and z is d-separated by a vertex y, given Y if any of the following rules hold:

- $y \in E$ and $x \in Des(y)$ and $z \in Anc(y)$ (see: Figure 1)
- $y \in E$ and $x \in Anc(y)$ and $z \in Des(y)$ (see: Figure 1)
- $y \in E$ and $y \in Anc(x) \cap Anc(z)$ (see: Figure 2)
- $y \notin E$ and $Des(y) \cap E = \emptyset$ and $y \in Des(x) \cap Des(z)$ (see: Figure 3)

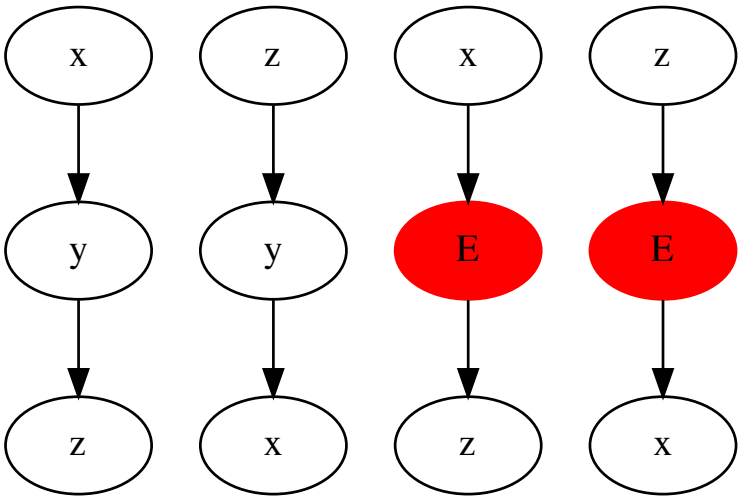

Figure 1: First d-separation rule

In the left subgraphs x and z are connected through y, since $y \notin E$. In the right subgraphs $y \in E$ and therefore blocks the path between x and z.

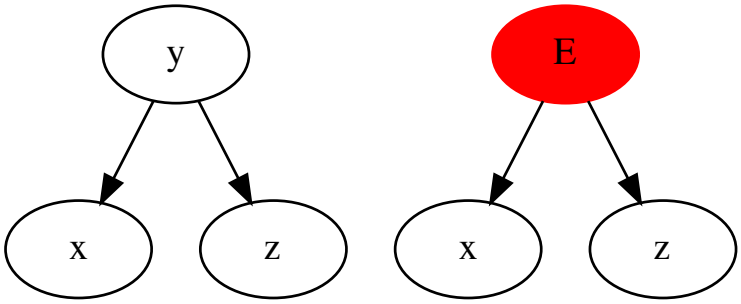

Figure 2: Second d-separation rule

In the left subgraph x and z are connected through y, since $y \notin E$. In the right subgraph $y \in E$ and therefore blocks the path between x and z.

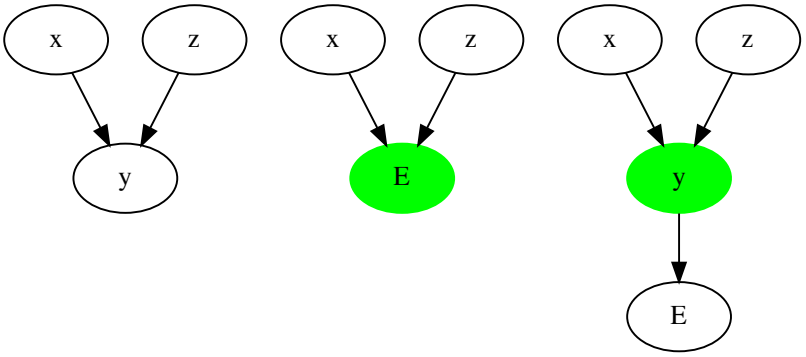

Figure 3: Third d-separation rule

In the left subgraph x and z are blocked by y, since $y \notin E$. In the middle subgraph $y \in E$ and therefore creates a path between x and z. In the right subgraph $Des(y) \cap E \neq \emptyset$ and therefore a path is created between x and z.

2.3 Stochastic sampling

Stochastic sampling methods are used to perform approximate inference. First all variables are sampled, and the probabilities are then estimated

according to the sampled values. In the likelihood weighting algorithm, each sample has a weight that depends on the probabilities of observed variables. The probability of a variable Q with possible values $\{q_0, ..., q_k\}$ given the observed variables can then be estimated according to the following equation:

$$\hat{P}(Q = q_i|E) = \frac{1}{M} \sum_{m=1}^{M} \begin{cases} w[m], & \text{if } Q[m] = q_i \\ 0, & \text{otherwise} \end{cases} \qquad 0 \leqq i \leqq k \qquad (1)$$

Where $w[m]$ is the weight of the sample, $Q[m]$ the sampled value of Q in the m-th sample, M the number of samples and E is the set of observed variables.

2.4 Forward sampling

In forward sampling the sampled nodes are instantiated based on the conditional probabilities given their parents' values. It is therefore necessary that before forward sampling a particular node, all of its predecessors should be instantiated (either through sampling or by being observed). To achieve this the nodes are sampled in topological ordering, forward sampling therefore starts at root nodes. The weight of a sample is simply the product of the probabilities of all the observed variables $E = e_1, ..., e_k$, given their parents.

$$w = \prod_{i=1}^{k} P(e_i|Pa(e_i)) \qquad (2)$$

2.5 Backward sampling

Backward sampling instantiates the uninstantiated parents of the node being backward sampled. It is therefore necessary for a node to be instantiated before performing backward sampling on it. This is achieved by sampling in reverse topological order. Backward sampling therefore starts from observed nodes (or possibly from leaf nodes, by adding a dummy children with uniform distribution to be backward sampled). While in forward sampling the sampling distribution is the node's probability distribution, in order to backward sample from a node, an operation called arc reversal should be performed first. During arc reversal the probability $P(Pa(e)|e)$ of the parents of an observation e is computed by normalizing the probability $P(e|Pa(e))$. After sampling the parents, the normalized probability is used to compute the weights of samples:

$$w = \prod_{i=1}^{k} P(Pa(e_i)|e_i) \qquad (3)$$

3 Methods

The core of the research is a two-by-two design, where the simulation direction and the sampled nodes varied. The simulation direction could either be forward or backward simulation, and the nodes sampled could be all nodes or only those that are d-connected to the queried variable given the observed variables. This resulted in a likelihood weighting algorithm with 4 settings.

3.1 Implementation

The likelihood weighting algorithm is based on [6], with the following modifications:

1. The nodes to be sampled from were given as a parameter to the algorithm
2. A backward sampling option was added, and the forward and backward sampling orders computed

The nodes that are backward sampled are the evidence variables in the sampling order and the predecessors of these evidence variables, that are in the sampling order (excluding root nodes). All other nodes in the sampling order are forward sampled.

In conditions 2 and 4, only the nodes that are d-connected to the queried variable given the observed variables are in the sampling order.

3.2 Networks used

The two networks used to conduct the experiments were the Hailfinder and the Hepar2 networks (Appendix A). These are both benchmark networks from the bnlearn repository. These networks differ in some of their attributes, which can influence the behaviour of the inference algorithms investigated. It is well established that stochastic sampling algorithms don't perform well with unlikely evidence, therefore the effect of this will not be investigated in this paper.

Our focus will be on the difference in connectedness of the network, which we can measure with average degree and maximum in-degree. It is theoretized that removing nodes that do not contain necessary information is more efficient in less connected networks, simply because more unnecessary nodes can be removed.

Network	Hailfinder	Hepar2
Number of nodes	56	70
Number of arcs	66	123
Number of parameters	2656	1453
Average degree	2.36	3.51
Maximum in-degree	4	6

Table 1: Network attributes

3.3 Queries

The queries in the experiments were generated randomly, with some fixed parameters. 10 separate queries were generated (for each network) and these were run in all conditions. The inference algorithms were stopped at 3000 iterations, as this number should be enough to converge below an error of 0.05 with 95% probability [6]. The exact values were computed for each query, using variable elimination.

4 Results

The running time and convergence of the algorithms are the main aspects that will be investigated.

4.1 Running time

4.1.1 Hailfinder network

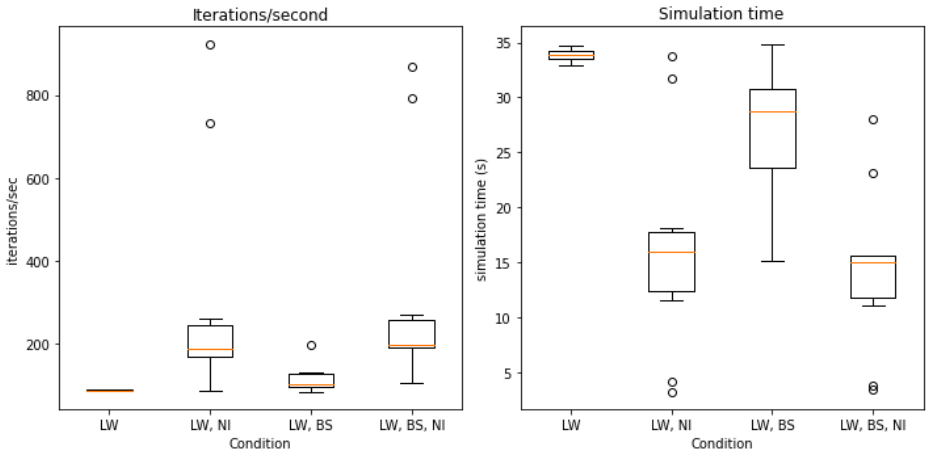

Figure 4: Running time of queries on the Hailfinder network

The queries of the Hailfinder network have achieved a better running time, both in therms of more iterations per second completed and less time it took to complete the simulation, when only the variables containing necessary information were sampled, regardless of sampling direction. Backward simulation was slightly quicker than forward simulation. There are some (high and low) outliers in the backward simulation conditions, they are the results of the same queries.

4.1.2 Hepar2 network

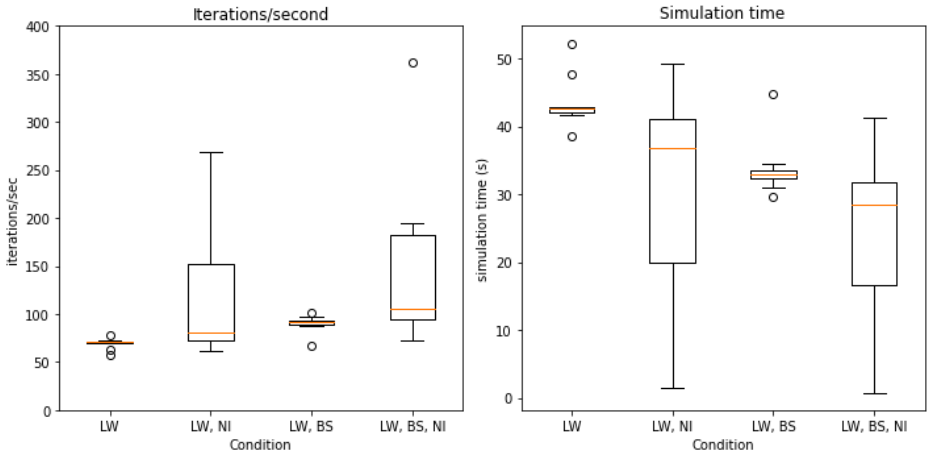

Figure 5: Running time of queries on the Hepar2 network

Much like in the Hailfinder network, improvement is seen in simulation time and number if iterations per second when only the variables containing necessary information were sampled . Backward simulation is also quicker than forward simulation here. Significant variance in running time can be observed when only the variables containing necessary information were sampled. Two outliers are not shown in the plot on the left side. Both data points are the result of backward simulation, one with all nodes sampled, at 2000 iterations/second and another with only variables containing necessary information sampled, at over 3500 iterations/second.

4.1.3 Number of sampled variables

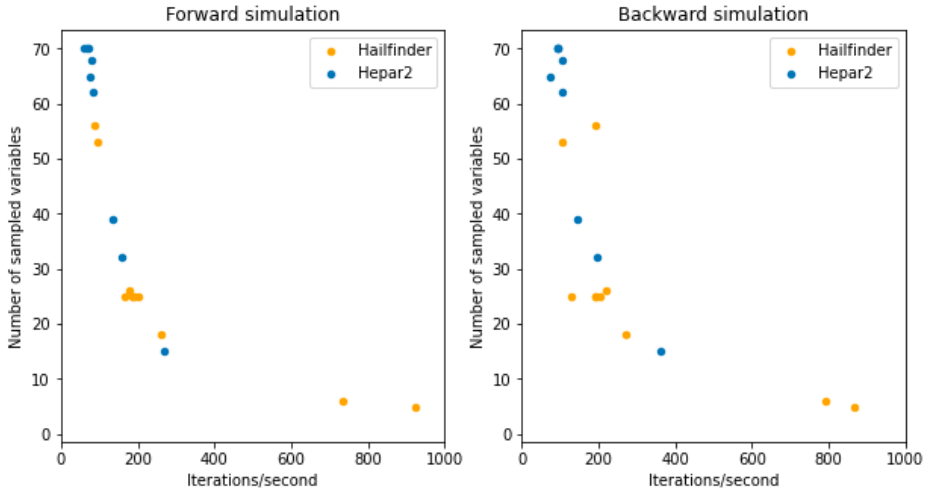

Figure 6: Relation between number of sampled variables and iterations per second

Note, that only results from the necessary information conditions are shown here and the difference in the number of nodes in the two networks used. Two outliers are not shown in the plots. One data point in forward simulation network at 2000 iterations/second and another in Backward simulation at over 3500 iterations/second. Both outliers were Hepar2 queries.

4.1.4 D-separation time

The time it took to find which variables contain relevant information took under the tenth of a second for all queries.

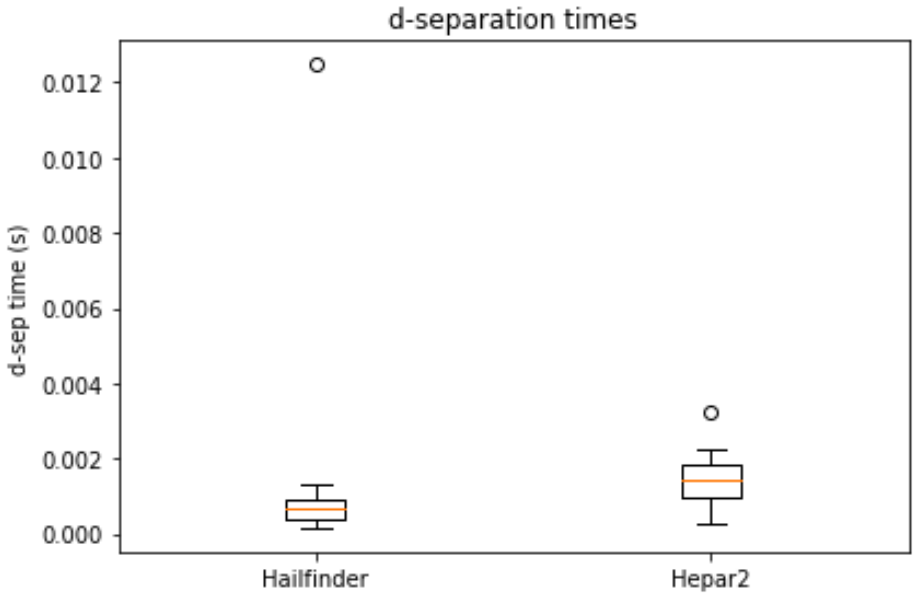

Figure 7: Running time of the d-separation algorithm

4.2 Convergence

Most of the queries have converged towards the right distributions. There are however, some cases where queries seem to converge towards a different distribution.

4.2.1 Forward sampling

In the forward simulation scenarios all queries have converged across both networks. There is no visible difference in the rate of convergence between all variables sampled and only those containing necessary information in any of the queries within the same network. Queries in the Hepar2 network have, in general, reached their lowest error rate in fewer iterations.

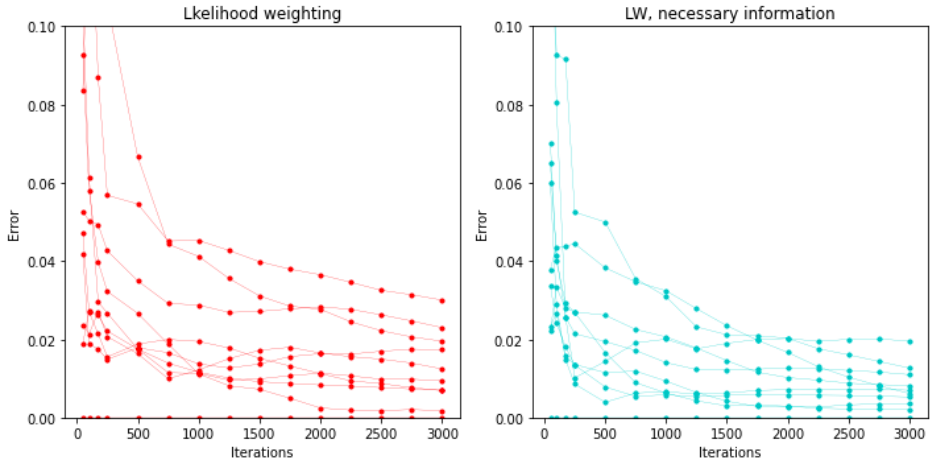

Figure 8: Convergence of queries on the Hailfinder network

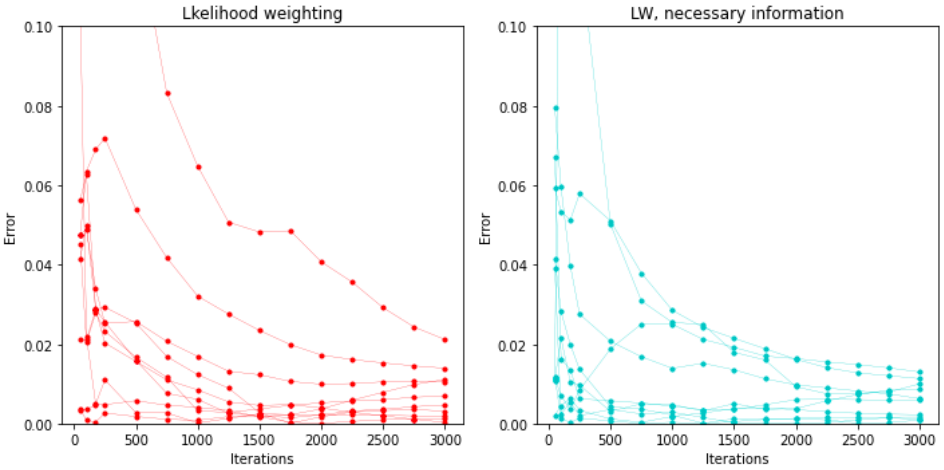

Figure 9: Convergence of queries on the Hepar2 network

4.2.2 Backward sampling

Most of the queries have converged in the backward simulation conditions in both networks.[1]. Some queries have converged to a very low error rate (below 0.02). A clear distinction can be seen between these and the less converged queries, especially in the right side plot of Figure 10. It can clearly be seen that the queries that have achieved a better result have converged more smoothly, getting closer in each iteration. Some queries have stopped improving quite early on. Of the queries that have reached a higher accuracy, Hepar2 queries have done it earlier on, just as when using

[1] 7 in Hailfinder and 5 in Hepar2. More about the rest in subsection 4.3

forward simulation. Note, that a few queries have reached a very low error at the first checkpoint, after 50 iterations.

Some queries, especially in the Hepar2 network have started converging away from the correct distribution at some point.

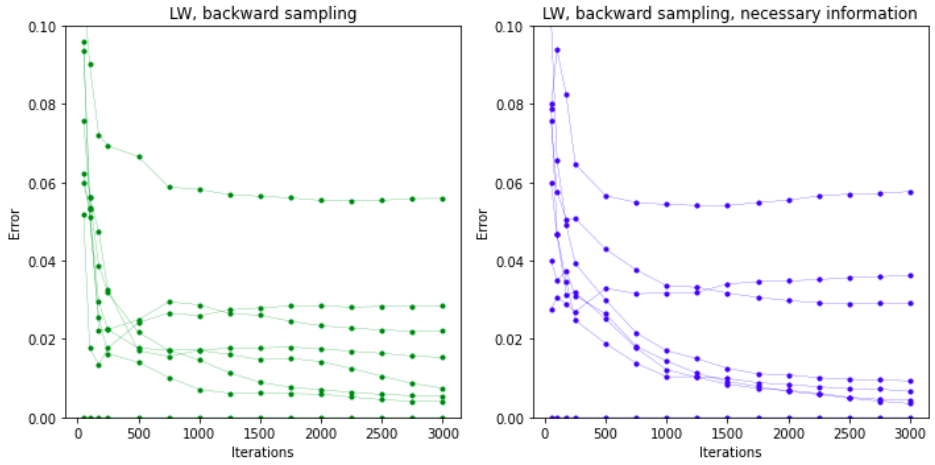

Figure 10: Convergence of queries on the Hailfinder network

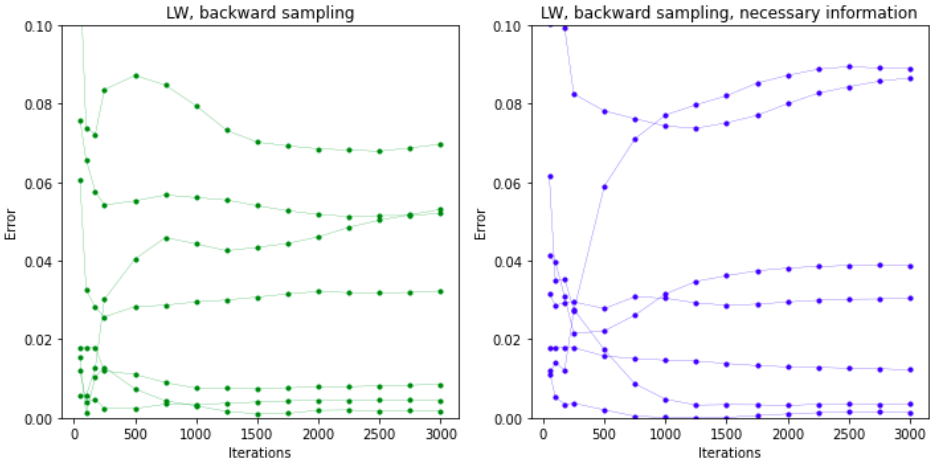

Figure 11: Convergence of queries on the Hailfinder network

4.3 Non-converging queries

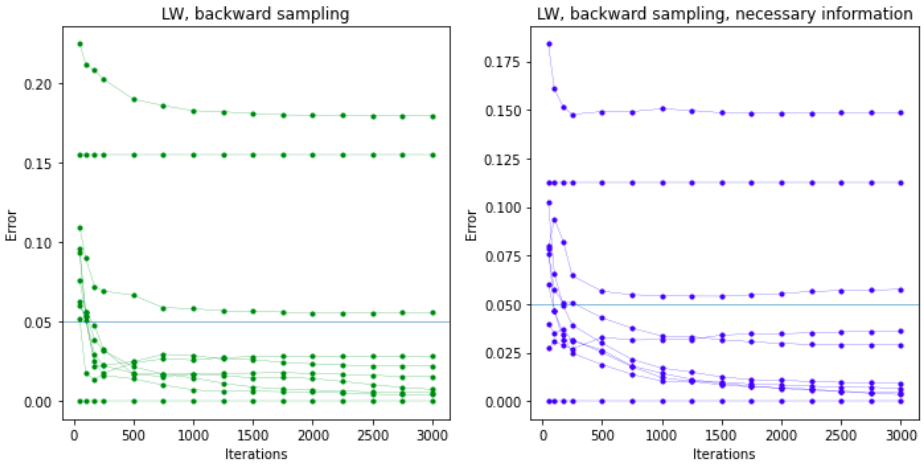

Figure 12: Hailfinder queries

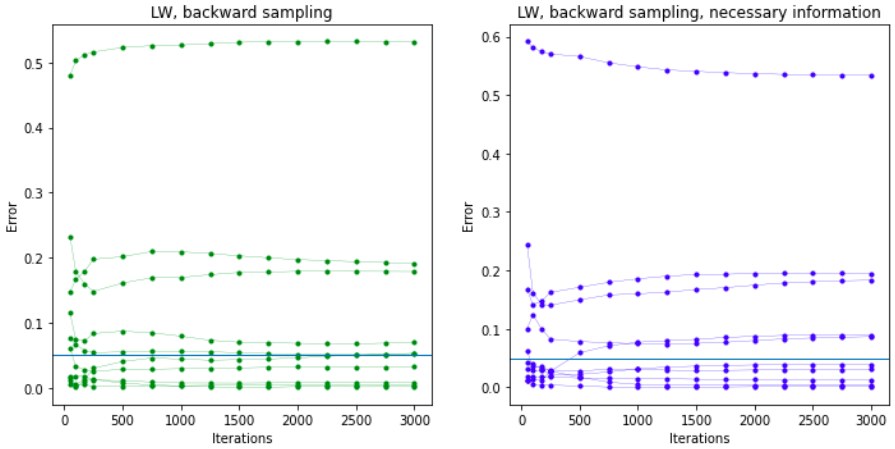

Figure 13: Hepar2 queries

While most queries converge towards the correct distribution, some are quite far away from it. All queries however, seem to stay constant after around 1500 iterations (Figure 14 [2]). To get a better idea of what is behind the non-converging queries, all of them were run a further 20 times using the algorithm with backward simulation and all nodes being sampled.

[2]The errors shown in this figure are the errors of each individual value of a variable. In the rest of the paper, the average error of a variable is used.

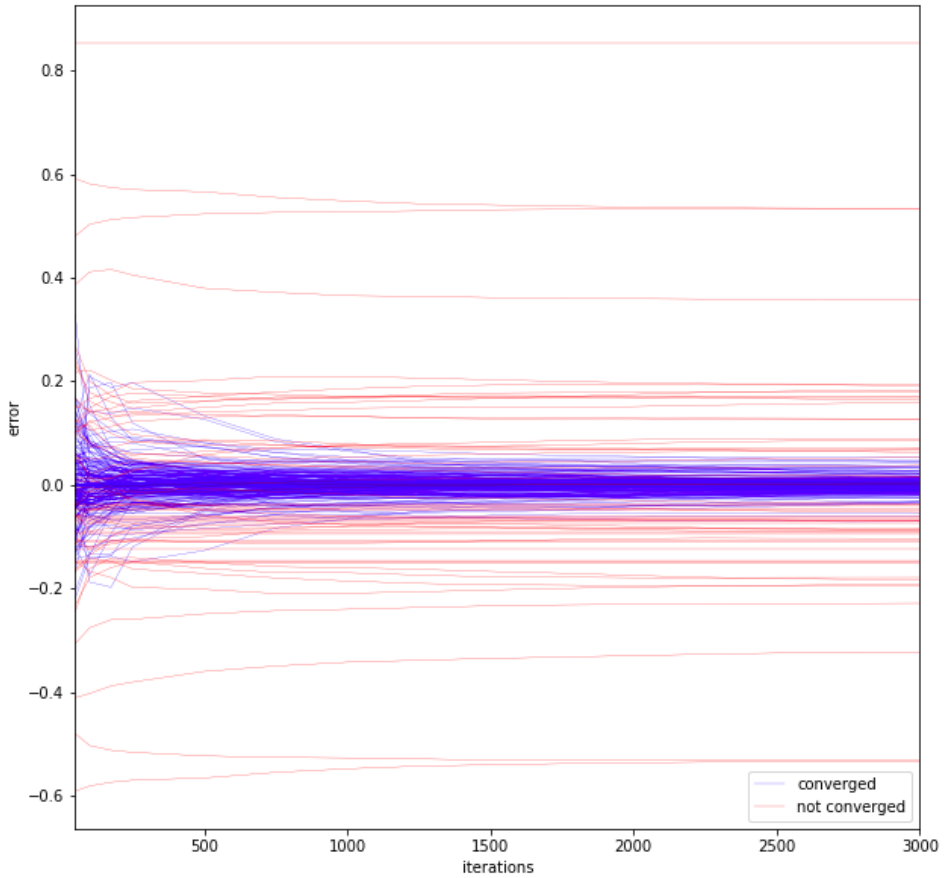

Figure 14: Convergence of queries

The results of this is is shown in Figure 15 . The rerun queries converge towards some distribution, although not the correct one. After further inspecting these queries, no mistake in their sampling was found, and no notable shared attributes between the queries was observed.

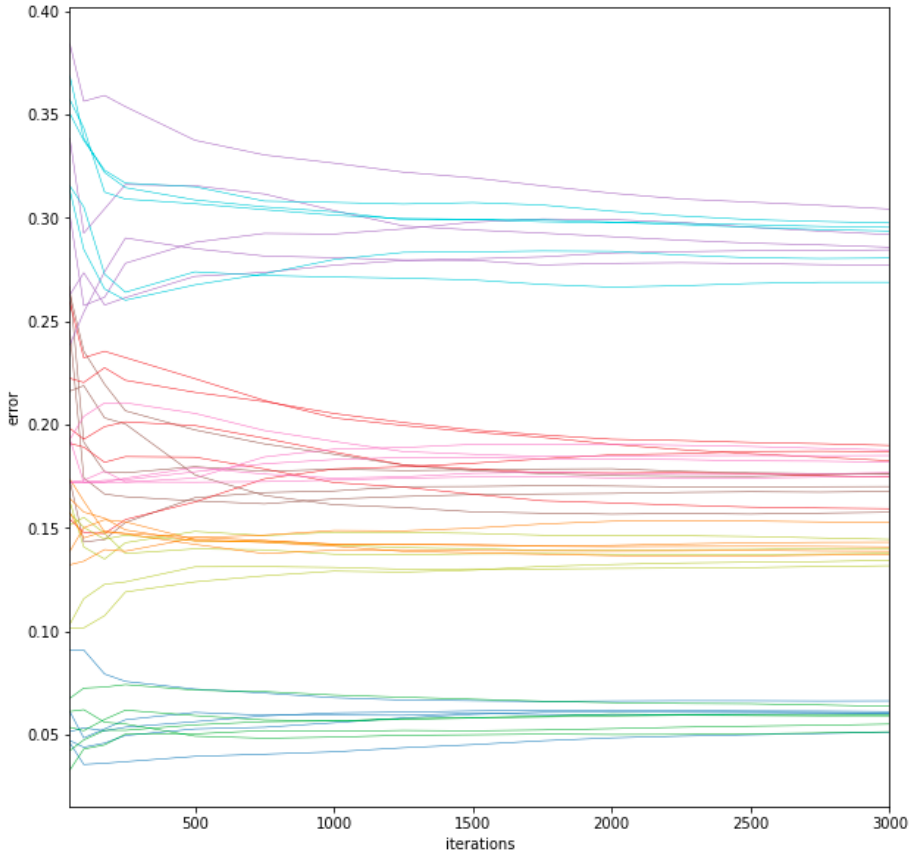

Figure 15: Re-run of non-converging queries

5 Conclusions

In conclusion only a fraction of the hypothetised improvements were achieved. Omitting superfluous information does lead to an improvement on running time, without having a negative effect on convergence, via making an iteration of the likelihood weighting algorithm quicker. This was expected to happen, as simply less computations have to be completed when only sampling a subset of variables, than when all the variables need to be sampled. However, it only leads to an improvement, because the cost of identifying the variables that contain the neccessary information to solve a query is negligible, compared to the running time of the whole simulation. Neither omitting superfluous information, nor incorporating backward sampling led to the expected improvement in convergence. The results of backward simulation however, need further inspection, as they are not consistent with the hypothetical error bounds and the convergence toward different distributions is not explainable with the literature used to complete this experiment.

As a result of the lack of computational power available, the data collected is minimal. In similar experiments many more runs were completed on stochastic simulation algorithms [3, 4]. Repeating a similar experiment with more data to analyse could lead to a more in-depth understanding of the findings presented here.

Due to time constraint and the unexpected results in the backward simulation conditions, only a limited group of queries were investigated. Exploring the effect of different number and likelihood of observations could yield interesting findings.

The factors that determine the varying performance of omitting superfluous information were only investigated superficially. It is clear that the number of sampled variables is what determines the reduction in running time and therefore the number of nodes in a network has an effect on it. Investigating what graph and query properties determine how much information can be excluded would be a logical follow-up to this research.

6 Acknowledgements

I would like to thank Nils Donselaar for his helpful guidance and thorough feedback throughout the project.

References

[1] Gregory F Cooper, *The computational complexity of probabilistic inference using bayesian belief networks*, Artificial intelligence **42** (1990), no. 2-3, 393–405.

[2] Paul Dagum and Michael Luby, *Approximating probabilistic inference in bayesian belief networks is np-hard*, Artificial intelligence **60** (1993), no. 1, 141–153.

[3] Robert Fung and Kuo-Chu Chang, *Weighing and integrating evidence for stochastic simulation in bayesian networks*, Machine Intelligence and Pattern Recognition, vol. 10, Elsevier, 1990, pp. 209–219.

[4] Robert Fung and Brendan Del Favero, *Backward simulation in bayesian networks*, Uncertainty Proceedings 1994, Elsevier, 1994, pp. 227–234.

[5] Haipeng Guo and William Hsu, *A survey of algorithms for real-time bayesian network inference*, Join Workshop on Real Time Decision Support and Diagnosis Systems, 2002.

[6] Daphne Koller and Nir Friedman, *Probabilistic graphical models: principles and techniques*, MIT press, 2009.

[7] Ross D Shachter, *Intelligent probabilistic inference*, Machine Intelligence and Pattern Recognition, vol. 4, Elsevier, 1986, pp. 371–382.

[8] Changhe Yuan and Marek J Druzdzel, *Importance sampling algorithms for bayesian networks: Principles and performance*, Mathematical and Computer Modelling **43** (2006), no. 9-10, 1189–1207.

A Appendix

The gitlab repository containing all the code and the raw data can be found here.

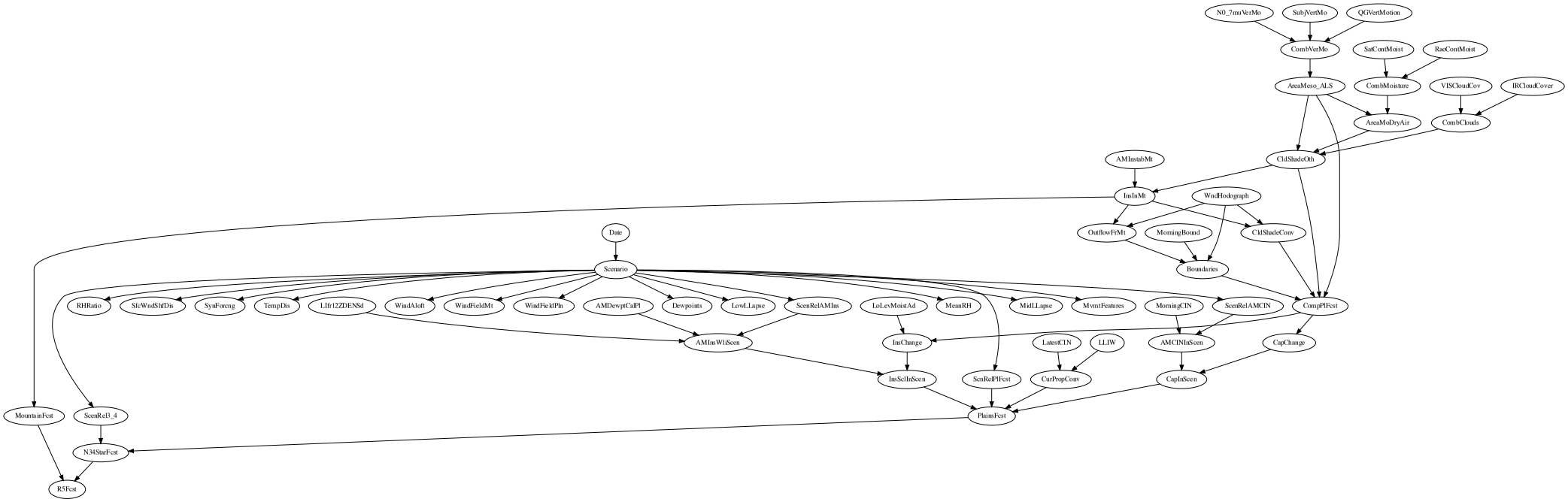

Figure 16: Hailfinder network

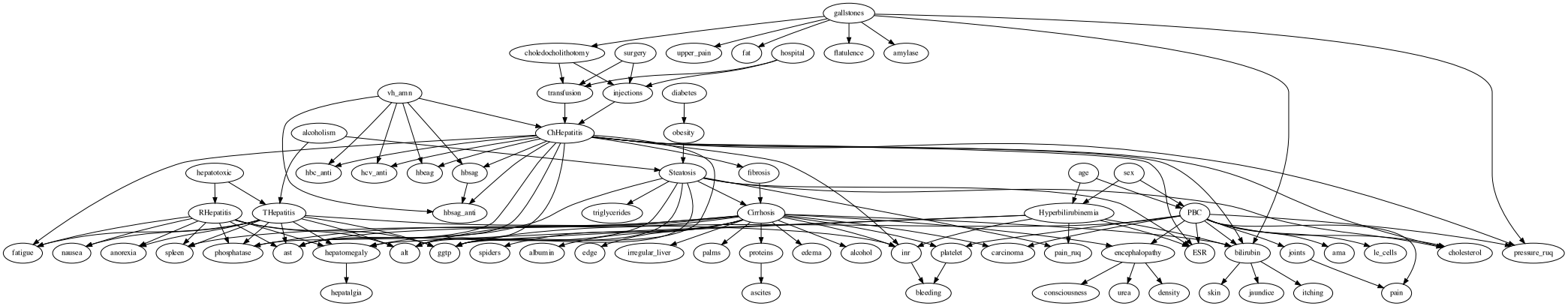

Figure 17: Hepar2 network