Implementation and Evaluation of Exact and Approximate Dynamic Bayesian Network Inference Algorithms

-- CS731 Term Project

Hongfei Guo guo@cs.wisc.edu Wei Luo luo@cs.wisc.edu

Abstract

Particle filtering is currently an area of intensive study. Because it is a sampling algorithm, particle filtering can be used easily with hybrid and continuous Dynamic Baysian Network (DBN)s. However, particle filtering is not a panacea for all dynamic Bayesian networks. In this project, we implemented three alternative inference algorithms for DBNs, namely, unrolling with generic variable elimination, unrolling with customized variable elimination (customized algorithm), and particle filtering. We designed and conducted a series of experiments using those alternatives respectively on different synthetic networks, of which the complexity is adjusted by two parameters: one is the number of state variales, the other is the average number of parents. Based on a thorough analysis of the experiment results, we reached our conclutions, which fit well with a quatitively analysis of each algorithm: (1) The number of samples needed is exponential to the number of state variables, and it is correlated with the belief state distribution skewness; It is not correlated to the average number of parents for the state varaibles. (2) The efficiency of particle filtering decreased sharply as the number of samples increases. (3) The efficiency of the customized algorithm decreases sharply with the increase of the average number of parents of the state variables; It is not sensitive to the number of state varaibles. And hence (4) Particle filtering outperforms the customized algorithm for networks of which state variables have more than 4 parents in average; For networks with large number of state variables yet a small average number of parents, e.g. 2, the customized algorithm is the ideal choice.

1 Introduction

Bayesian network can be used to represent the dependencies among variables and to give a concise specification of the full join distribution. In the context of a static world, in which each random variable has a single fixed value, Bayesian network technique is sufficient for probabilistic reasoning. However, in many real-world domains, a complex situation evolves over time. For example, a patient's vital signs in an internsive care unit, a complex freeway traffic scene with moving vehicles, a robot's location in a complex environment[1], or operons in gene expression that transcribing from time to time[4,5]. This technique can be awkward in representing and reasoning in such dynamic worlds, in which some state variables can have different values at different time slice. Instead, dynamic Bayesian network technique is developed for stochastic dynamic systems that allows us to represent such complex systems in a compact and natural way [2].

Of course, the purpose of building a DBN model is to be able to infer with it. There are four basic inference tasks in a generic temporal model [2], as follows:

- Filtering or monitoring: This is the task of computing the posterior distribution over the current state, given all evidence to date. That is, $P(X_t|E_{1:t})$.
- Prediction: This is the task of computing the posterior distribution over the future state, given all evidence to date. That is, $P(X_{t+k}|E_{1:t})$ for some $k>0$.
- Smoothing or hindsight: This is the task of computing the posterior distribution over a past state, given all evidence up to the present. That is, $P(X_k|E_{1:t})$ for $0 <= k < t$.
- Most likely explanation: Given a sequence of observations, to find the most likely sequence of states that generated those observations. That is $\max X_{1:t} P(X_{1:t}|E_{1:t})$.

Among those, filtering or monitoring is of most common interest. Intuitively, given a series of evidence up to current state, it is very useful to monitor the distribution of current state, or to predict the distribution of a state in a future time point.

How can we solve this problem? At first glance, it may be trivial. Indeed, a dynamic Bayesian network is a Bayesian network. So it seems that all we have to do is unrolling, that is, to construct the full Bayesian network representation of a DBN by replicating slices until the network is large enough to accommodate the observations. The rest of the work can be trivially done by one of those inference techneques in Bayesian networks. Unfortunately, a naïve application of unrolling would not be particularly efficient. The second approach is to customize variable elimination, taking advantage of the special structure of DBNs, which is also called recursive estitimation in [2]. [2] shows that we can achieve constant space and time per filtering update in this way. We are not done yet. It turns out that the "constant" for the per-update time and space complexity is, in almost all cases, exponential in the number of state variables. In most systems, the complexity of the belief state, that is, $p(X_t|E_1...E_t)$, grows unboundedly, preventing any closed-form representation [1]. This leads us to the third approach: approximate inference techniques. A family of algorithms called particle filtering is designed to do just this.

Thus, a second question follows: Should we simply open our arms for particle filtering, deserting exact inference algorithms? Surprisingly, the answer is not trivally YES. To motivate, for the umbrala model [2], while particle filtering runs 580ms with 0.009 error measured in KL-distance for an observed evidence chain length 1000, our customized variable elimination runs only 184ms with exact answers! In this project, we implemented three alternative inference approaches for DBNs, in addition, DBNUnrolling, an auxilliary tool which take a DBN as input, and automatically unroll it into a Bayesian network. Then we designed and conducted a series of experiments using synthetic models. Based on that, we did a thorough performance comparison study, trying to answer the following questions:

- For particle filtering , how big should the sample size be?
- When does customized variable elimination outperform particle filtering?
- When does particle filtering outperform customized variable elimination?

Unrolling with generic variable elimination is mainly used as a reference group.

The rest of the paper is structured as follows. Section 2 gives out some related work. In section 3, we give an overview of dynamic Basian networks and models of interest. Section 4 illustrates the implementation of three algorithms: unrolling with generic variable elimination, unrolling with customized variable elimination, and particle filtering. In section 5, we show our experiments methodology. Section 6 presents our experiments results and analysis. We conclude in section 7.

2 Related Work

Particle filtering is currently an area of intensive study. Many variants and improvements have been proposed and the number of applications is growing rapidly. Many researches emphysize the application of particle filtering to certain domains. However, to our knowledge, there is no work done in comparing the performance of particle filtering and customized exact inference algorithm w.r.t. different DBN models. [8] used particle filtering to tracking complex motion patterns in video and [9] applied it to predicting the stock market. In [1], Koller and Lerner studied the number of samples and the error bound of particle filtering with two large DBNs, one is the discrete BAT network, and the other is a hybrid network benchmark. Different from those work, we focused on a thorough comparison study of customized variable elimination and particle filtering, answering questions: (1) Which factors affect the number of samples needed and which factors do not? (2) When should one use customized variable elimination and when should one use particle filtering?

3 Overview of Dynamic Bayesian Networks

A dynamic Bayesian network or DBN is a Bayesian network that represents a temporal probability model. The process of change can be viewed as a series of snapshots, each of which describes the state of the world at a particular time. Each snapshor or time slice contains a set of random variables, some of which are observable and some of which are not. We will use $\mathbf{X_t}$ to denote the set of unobservable state variables at time t and **Et** to denote the set of observable evidence variables. The observation at time t is $\mathbf{E_t} = E_t$ for some set of values Et.
The semantic meaning of the interval between time slices deponds on the problem itself. However, for the model per se, we can conveniently assume fixed, finite interval. This means that times can be labelled by integers. We will assume that everything starts at t = 0, so that times are nongegative integers. In order to bound the number of conditional probability tables, as well as the number of parents of each variable, two assumptions are adopted. First, we assume that changes in the world state are caused by a stationary process – that is, a process of change that is governed by laws that do not themselves change over time: For any evidence variable E, $\mathbf{P}(E_t|\text{Parents}(E_t))$ is the same for all t.

The second problem is solved by making Markov assumption, that is, that the current state depends on only a finite history of previous states, of which the simplest yet most commonly used is the first-order markov process, in which the current state depends only on the previous state and not on any earlier states. In this project, we only consider first-order Markov process, the reason is that it is not trivial to generalize particle filtering to handle higher order Markov process. The corresponding conditional independence assertion states that, for all t,

$$\mathbf{P(Xt|X_{0:t-1}) = P(X_t|X_{t-1})}$$

The laws describing how the state evoles over time are called the transition model. In addition to restrict the parents of the state variables **Xt**, we must also restrict the parents of the evidence variables $\mathbf{E_t}$. Typically, we will assume that the evidence variables at time t depend only on the current state:

$$\mathbf{P(E_t\ |X_{0:t}, E_{0:t-1}) = P(E_t\ |\ X_t)}$$

The conditional distribution $\mathbf{P(E_t\ |\ X_t)}$ is called the sensor model.

In this project, although we restrict our models to satisfy the transition model and the sensor model, our results can be generalized to general models.

4 Inference Algorithms for DBNs

In this section, we illustrate how to apply aforementioned three inference algorithms to dynamic Bayesian networks, namely, unrolling with generic variable elimination, unrolling with customized variable elimination, and particle filtering. They all share the same input and output.

Input:

- A DBN, which is determined by the prior probabilities of the state variables, the transition model CPTs, and the sensor model CPTs,
- A series of observations, $E_{0:t}$
- The query time slice n, where n > = t.

Output:

- Joint distribution of query state n, P(Xn | E0:t)
- Marginalized distribution of each state variable X at the query time slice, P(Xn | E0:t)

Among those three algorithms, the first two both employ inference techniques developed for Bayesian networks. The basic idea is two steps. Step 1, unroll the inputed DBN into a Bayesian network. Technically, the DBN is equivalent to the semi-infinite network obtained by unrolling for ever. However, slices added beyond the query time slice have no effect on inferences within the time periord of interest and can be ommited. Step 2, apply generic or customized variable elimination on the Bayesian network.

5.1 Unrolling with Generic Variable Elimination

The generic variable elimination algorithm works this way. Given the CPTs in BN as initial potential, repeat until only query variables remain:

- Choose the next variable to eliminate
- Multiply all potentials that contain the variable
- If no evidence for the variable then sum the variable out and replace original potential by the new result. Else, remove variable bases on evidence.

After that, normalize remaining potential to get the final distribution over the query variables.

In our implementation, we used two techniques to improve the performance. First, we used a heuristic function to choose a variable to eliminate. Each time we choose the variable that appear in the least number of CPTs. This is a tradeoff between time and space. The other alternative is to choose the variable which least increased the total size of the potentials. This heristic gives us the least size of intermidiate potencials, yet it has worse time complexity. Our least number of potentials function only requires a scan of the variable index, which is O(N) in time complexity, where N is the number of variables. For comparison, for each candidate variable, the latter also need to compare all the variables appearing in involed potentials to be able to figure out the result potencial size. Thus it has $O(N^2)$ time complexity.

Second, we built two hash indexes. One is the potential index. Given a potential ID as a key, the index returns the potencial object in constant time. The other is the variable index. Its key is a variable name, the corresponding entry is a list of potentials in which the variable appears. Figure (?) shows an example with current potentials and according indexes.

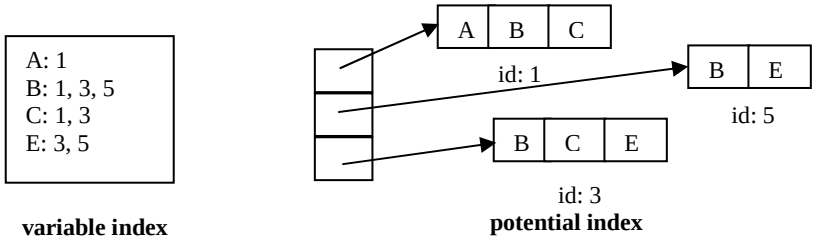

Figure 1: An example of potential index and variable index state

5.2 Unrolling with Customized Variable Elimination

Dynamic Bayesian network is a special kind of Bayesian network, so we can skip the expensive procedure of choosing the variable in the elimination, in other words, we can eliminate the variables with a fixed sequence:
(1) For filtering we can eliminate all the state variables at time 0, then all the evidence variables at time 1, then all the state variables at time 1, and proceed until there are only state variables at time t left.
(2) For prediction, the procedure is almost the same as (1), but the condtition for stopping the variable elimination should be that there are only state variables at time t+k left (k > 0), and we do not have evidences to eliminate between time t and time t + k.

(3) For smoothing, we should elimate from two ends to the middle: from time 0 to time k (forward), and from time t to time k (backward).

5.3 Particle Filtering

Particle filtering is a general purpose Monte Carlo scheme for tracking in dynamic systems. It maintains the belief state at time t as a bag of particles $\{X_t\}$, where X_t is a full instantiation of $\mathbf{X_t}$. Ideally, we want to sample $P(\mathbf{X_{t+1}} \mid X_t, e_t)$. However, this distribution is rarely one that we can compute efficiently. Hence we use importance sampling: We use the samples themselves as an approximate representation of the current state distribution, and use an importance weight to make up for the difference. The algorithm is roughly as follows:

First, a population of N samples is created by sampling from the prior distribution at time 0, $\mathbf{P(X_0)}$. Then the update cycle is repeated for each time step:

- Each sample is propagated forward by sampling the next state value $\mathbf{X_{t+1}}$ given the current value $\mathbf{X_t}$ for the sample, using the transition model $\mathbf{P(X_{t+1} \mid X_t)}$.
- Each sample is weighted by the likelyhood it assigns to the new evidence, $\mathbf{P(}E_{t+1} \mid \mathbf{X_{t+1})}$.
- The population is resamples to generate a new population of N samples. Each new sample is selected from the current population; the probability that a particular sample is selected is proportional to its weight. The new samples are unweighted.

The algorithm is shown in detail in Figure 2. [2] shows that this algorithm is consistent, but is it efficient? Recall for each time slice, we need to generate N samples each with M data point, where N is the number of samples and M is the number of state variables, the time complexity is O(MN). In our implementation, we used binary search in resampling step. Hence the complecity of resampling is O(N log N). Therefore, when N >> M, the total time complexity for particle filtering is O(N log N). Is this good or bad? We shall interpret it by experiment result analysis in section 6.

```
Function ParticleFilering(E, N, dbn) returns a set of samples for the next time step
 Inputs: E, the new incoming evidence
         N, the number of samples to be maintained
         dbn, a DBN with slice 0 variables X0 and slice variable X1 and E1
  static: S, a vector of weight of size N
  local variables: W, a vector of weights of size N

IF E is empty THEN
   FOR i=1 to N DO
        S[i] <-sample from P(X0)
ELSE DO
   FOR i=1 to N DO
        S[i] <-sample from P(X1 | X0=S[i])
        W[i] <-P(E|X1=S[I])
   S <- WeightedSampleWithReplacement(N, S, W)
Return S
```

Figure 2: Particle filtering algorithm

5 Experiments Methodology

5.1 Test Bed

We run all experiments on CS department machine nova 33, Sun OS, 384M memory, 300 MHz CPU's clock rate, 100 MHz memory's clock rate.

5.2 Synthetic Models

In this project, we used systhetic models instead of benchmarks. There are mainly two reasons. First, the purpose of our experiments is to study for what kind of models particle filtering works better, and for what kind of models unrolling with customized variable elimination works better. Therefore, we need the flexibility to adjust the model structures, which can not be provided by any benchmarks. Second, our imprementations only supports DBNs that satisfy transit model and sensor model, since our results can be easily generalized to general models. However, we could not find any benchmarks that satisfy such assumptions.

There are two parameters of a model. First, the number of state variables. Second, the average number of parents of a state variable. By adjust these two parameters, we can adjust the complexity of our DBN models. Figure 3 shows a DBN model example we used in our experiments.

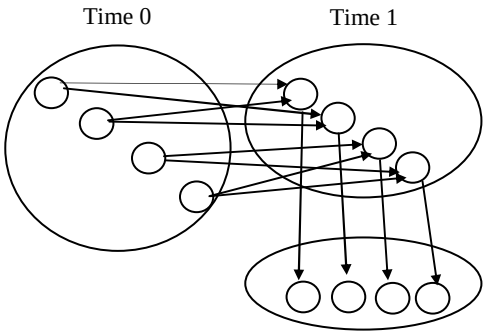

Figure 3: A sample DBN network with 4 state variables and 2 average parents each

5.3 Error Metrics

We used Kullback-Leibler distance as error metrics [7]. KL-distance for two distributions p(X) and q(X) is defined as follows:

$$D(p \parallel q) = \sum_{X_i} p(x_i) \mid \log \frac{p(x_i)}{q(x_i)} \mid$$

Intuitively, KL-distance measures how similar those two distributions are to each other. The logarithm is used to assign the same scale of weight to point pairs with the same degree of divergence, regardless of the order of the pair in the division. For example, the degree of divergence between 10 and 1 should be the same between 1 and 10. With the logarithm, we have log (1/10)=-1, log(10/1) = 1. They get the same scale of weight. Absolute values of each divergence are used to avoid the mutual cancelling of the positive divergnce with the negative ones.

We computed the error on both the entire joint distribution and the average error on the marginals of all the state variables. Let **q(X)** be the exact joint distribution, q(X) the approximate joint distribution, then

$$jo\,\text{int}\ error\ (p) = \sum_{X_i} q(x_i) \mid \log \frac{q(x_i)}{p(x_i)} \mid$$

We define the average error on the marginalized distritribution as follows: (N is the number of variable states)

$$maginal\ \ error\ (p) = \frac{\sum_{x_i} jo\,\text{int}\ error\ (p(x_i))}{N}$$

6 Experiment Results and Analysis

First we compared the performance of the generic variable elimination with the customized variable elimination. In this experiment, the average number of parents for the state variables was fixed, and the number of state variables varied. Figure 4 shows the result. The customized algorithm dominates as the number of state variables increases. For our network with 8 state variables (the average number of their parents is 2), the customized algorithm needs 304 ms, but the generic algorithm needs 10130 ms! Clearly, the customized elimination order for DBNs is a big win. Two factors slow down the generic variable elimination: (1) It needs additional $O(N^2)$ time in deciding the elimination order, and (2) The heuristic method cannot always find the best variable for elimination.

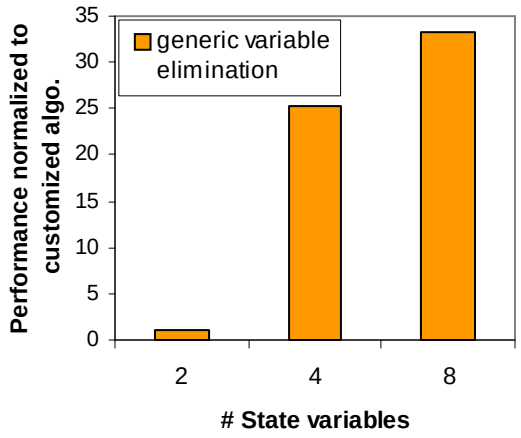

Figure 4: Performance of generic algorithm normalized to customized algorithm, #parents=2

Customized variable elimination is an exact algorithm, while particle filtering is a sampling algorithm. Quite often, there is a trade off between an exact algorithm and an approximate algorithm: the former is always accurate but might be slow; the latter is faster but might be less accurate. Is this the case for those two algorithms in question?

Figure 5 shows the relationship between the error and the number of samples used. The curve above shows the KL-distance of joint distribution, while the curve below shows the KL-distance of marginalized distribution. Both curves have similar shape: the error drops sharply initially and then the improvements become smaller and smaller. So we have to increase the number of samples if we want to decrease the KL-distance, but this will increase the running time. Thus there is a trade off. Obviously, estimating the marginals is much easier than estimating the full joint distribution, therefore the error of marginalized distribution is always smaller than that of joint distribution for the same data set. We only consider the joint distributions in our other experiments below.

Figure 6 shows the effects of the number of state variables on the number of samples needed. For each number of state (2, 4, and 8), we fixed the average number of parents of state variables while varying the number of state variables. The result shows that the curve moves upward in an exponential order of the number of state variables. This is easy to understand: The number of samples needed is propotional to the number of the states in the full joint table, which in turn is exponential to the number of state variables.

Surprisingly yet reasonably, the skew of the belief state distributions moves the curve downward (Figure 7). The reason is as follows: For a uniform distribution of the belief state, it is very possible to get zero samples for

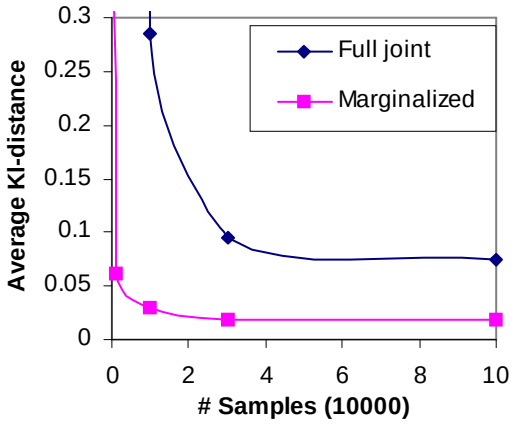

Figure 5: Error of particle filtering, #state varible = 8, #parents =1

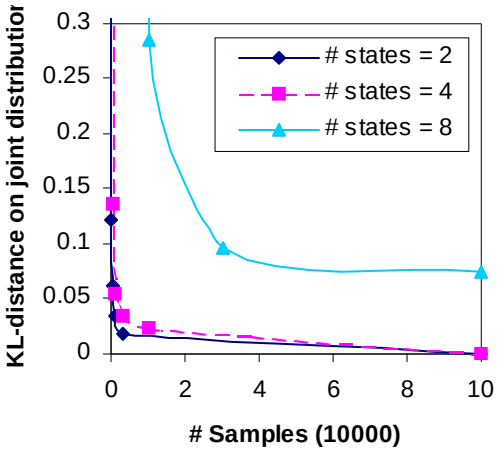

Figure 6: Error of particle filtering, #parents =1

some joint states even when the number of samples is already large. And such error is magnified by the KL-distance calculation.

We have two control experiments. First, the number of state variables was fixed and the number of their parents was varied, our result shows that it is not correlated to the number of sample needed. Second, the length of the observed evidence chain was varied when the number of state variables and the number of their parents were both fixed. Again, the chain length is not correlated to the number of sample needed. See Figure 8.

Next, we conducted two groups of experiments to compare the performance of particle filtering and the customized algorithm. In our experiments, we chose the number of samples for particle filtering such that the error is bounded to 0.05±0.03.

In our first group of experiments, we fixed the average number of parents to 2 and varied the number of state variables. When the number of state variables increased from 2 to 8, the customized algorithm increased roughly 3 times, while the other increased about 200 times. For every number of state variables, particle filtering algorithm always needs more time than customized variable elimination, and the difference increases sharply when the number of state variable increases (Figure 9). The reason is that more samples (exponentially

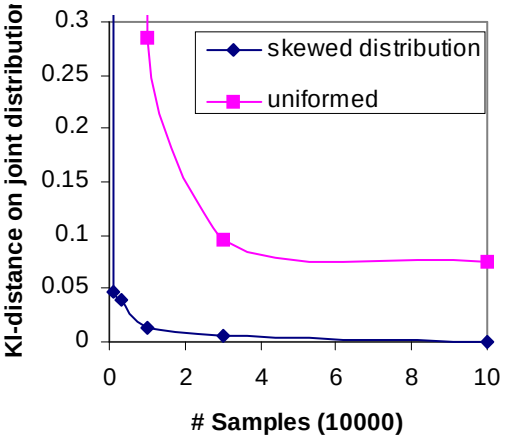

Figure 7: Error of particle filtering on skewed I and uniformed distribution, #state varible = 8, #parents =1

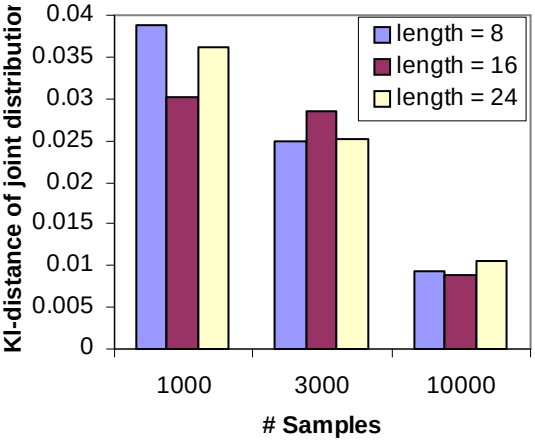

Figure 8: Error of particle filtering on different observed evidence chain length, #state varible = 4, #parents =2

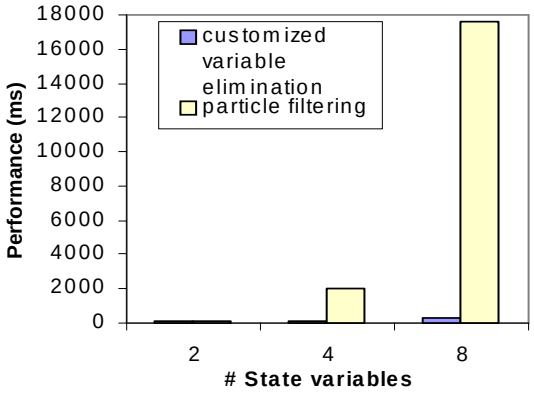

Figure 9: Performance comparison between customized algo. and particle filtering, #parents = 2

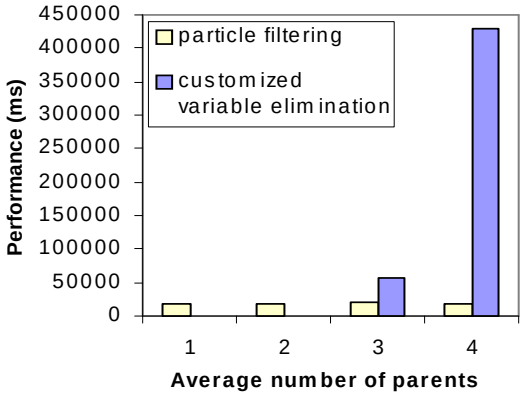

Figure 10: Performance comparison between customized algo. and particle filtering, #state = 12

increased) are needed to keep the error within the same bound for particle filtering, while the customized variable elimination does not suffer from this problem. For a network with 8 state variables and average 2 parents, we need about 50000 samples to get an average error of about 0.075, which takes more than 18000ms, while customized algorithm only need about 300ms.

In our second group of experiments, we fixed the number of state variables to 8 and varied the average number of parents. When the average number of parents increased from 1 to 4, the performance of particle filtering remained almost the same, while the other increased 2264 times. The reason is that the cost of multiplying potentials increases sharply for variable elimination algorithm if some nodes in the network have many parents, while particle filtering algorithm is not affected (Figure 10). For a network with 8 state variables and average 4 parents, we need about 50000 samples to get an average error of about 0.075, which takes only 18s, while customized algorithm needs about 430s.

7 Conclutions

As a sampling algorithm, particle filtering can be used easily with hybrid and continuous DBNs. In the literature, much research has put emphysis on the importance of particle filtering. However, particle filtering should not be blindly applied to any DBN without taking into account its characteristcs. In this project, we implemented three alternative inference algorithms for DBNs, namely, unrolling with generic variable elimination, unrolling with customized variable elimination (customized algorithm), and particle filtering. We designed and conducted a series of experiments using those alternatives respectively on synthetic networks, of which the complexity is adjusted by two parameters: one is the number of state variales, the other is the average number of parents. Our experiment results show that (1) The number of samples needed is exponential to the number of state variables, and it is correlated with the belief state distribution skewness; It is not correlated to the average number of parents for the state varaibles. (2)The efficiency of particle filtering decreased sharply as the number of samples increases. For a network with 8 state variables and average 2 parents, we need about 50000 samples to get an average error of about 0.075, which takes more than 18000ms, while customized algorithm only need about 300ms. (3) The efficiency of the customized algorithm decreases sharply with the increase of the average number of parents of the state variables; It is not sensitive to the number of state varaibles. And Therefore (4) Particle filtering outperforms the customized algorithm for networks of which states variables have more than 4 parents in average; For networks with large number of state variables yet a small average number of parents, e.g. 2, the customized algorithm is the ideal choice.

References:

[1] Daphne Koller and Uri Lerner, "Sampling in Factored Dynamic Systems", A book chapter in *Sequential Monte Carlo Methods in Practice*, A.Doucet, J.F.G. De Freitas, and N. Gordon, Eds., Sprinter-Verlag 2000.

[2] S. Russell and P. Norvig, "Probabilistic Reasoning over Time", A book chapter draft.

[3] S. Russell and P. Norvig, "Probabilistic Reasoning System", A book chapter draft.

[4] Ireng M. Ong, David Page, "Inferring Regulatory Pathways in E. Coli using Dynamic Bayesian Networks", *Computer Sciences Technical Report No. 1426*, University of Wisconsin-Maison, May 2001

[5] Ireng M. Ong, Jeremy D. Glasner and David Page, "Modeling Regulatory Pathways in E. coli form Time Series Expression Profiles", *Proceedings of the 10th International Conference on Intelligent Systems for Molecular Biology (ISMB 2002)*, edmonton, CA, Auguest 2002

[6] Dean, T. and Kanazawa, K. "A model for Reasoning about persistence and causation", *Computational Intelligence 5(3)*: 142-150, 1989.

[7] Cover, T. and Thomas, J. Elements of Information Theory, Wiley, 1991.

[8] Michael Isard, Andrew Blake: Contour Tracking by Stochastic Propagation of Conditional Density. *ECCV (1)* 1996: 243-356

[9] N. de Freitas, A. Doucet, M. Niranjan and A. Gee. Global optimization of neural network models via sequential sampling. *Advances in Neural Information Processing Systems (NIPS11)*. Pages 410-416. MIT Press, 1999.