Analysis of Three Bayesian Network Inference Algorithms: Variable Elimination, Likelihood Weighting, and Gibbs Sampling

Rose F. Liu, Rusmin Soetjipto
rliu,rusmin@mit.edu

November 4, 2004

1 Introduction

In this paper, we describe and analyze three Bayesian network inference algorithms: variable elimination, likelihood weighting, and Gibbs sampling. Variable Elimination is an exact inference algorithm, while likelihood weighting and gibbs sampling are approximate inference algorithms. For each algorithm we study thier performance under different conditions. In section 2, we analyze variable elimination and study how different elimination orders affect performance. In section 3, we validate and describe our likelihood weghting implementation. In section 4 we analyze Gibbs sampling and experiment with the effect of burn-in (length of the initial prefix that is thrown away) on estimation accuracy. In section 5 we measure the performance of the two approximate inference algorithms in terms of their running times versus the quality of their results. We also draw conclusions on which algorithm is more appropriate for the different bayesian networks. In section 6, we describe another take on the study of how burn-in affects Gibbs sampling performance.

2 Variable Elimination

We implemented the variable elimination algorithm outlined in [1]. Given a Bayesian network, assignments of some variables e (evidence), a query node X, and an elimination order, our variable elimination implementation calculates $P(X|e)$.

One interesting thing about our variable elimination implementation is that when the factors are restricted to the evidence, not only are the entries with incongruent evidence set to 0.0, we also sum over the evidence variable so as to eliminate extra entries in the factors. This optimization in the variable elimination implementation significantly decreases runtime for the Insurance network queries because of the presence of loops in the Insurance network.

2.1 Validation

To test our implementation of variable elimination, we used the following test cases:

Burglary Network:

$$P(Burglary|JohnCalls = true, MaryCalls = true) \tag{1}$$

$$P(Earthquake|JohnCalls = true, Burglary = true) \tag{2}$$

Burglary	Probability
false	0.7158281646356071
true	0.2841718353643929

Table 1: Results to Burglary:(1): Distribution over Burglary given that both John and Mary calls.

Burglary	Probability
false	0.9979800168901211
true	0.002019983109878836

Table 2: Results to Burglary:(2): Distribution over Earthquake given that both Burglary is true and John calls.

Insurance Network:

$$P(PropCost|Age = Adolescent, Antilock = False, Mileage = Fiftythou) \tag{3}$$

Our results for each test case are shown in Tables 1-3 respectively. The results to the Burglary network were compared to the results of the provided Enumeration Solver and are correct. The result of the Insurance network was compared to the answers given in the Project handout and are correct.

2.2 Experiments

In this section we will run some queries on the Insurance and Carpo Bayesian networks. We will show the results of the queries obtained from variable elimination and then give some intuition behind the results.

2.2.1 Insurance Query 1

The results to Insurance Query 1, given by the following equation, are shown in Table 4.

$$P(PropCost|Age = Adolescent, Antilock = False, Mileage = FiftyThou, MakeModel = SportsCar) \tag{4}$$

Property Cost	Probability
Hundred Thousand	0.17297869189641366
Million	0.02709352198178344
Ten Thousand	0.3427002442093676
Thousand	0.4572275419124353

Table 3: Results to Insurance:(3): Distribution over property cost for an adolescent driving a car with 50k miles and no anti-lock brakes.

Property Cost	Probability
Hundred Thousand	0.17179333672003955
Million	0.0309387733436524
Ten Thousand	0.3459303973796924
Thousand	0.45133749255661576

Table 4: Results to Insurance Query 1: Distribution over property cost for an adolescent driving a sports car with 50k miles and no anti-lock brakes.

This probability distribution (i.e. Table 4) is similar to the probability distribution shown in Table 3, except that the former has one additional evidence variable: the make model of the client's car is a Sports car. This is on top of the three original evidence variables: that the client is an adolescent, the car does not have antilock brakes, and the mileage is about fifty thousand.

Intuitively, the additional evidence might affect the property cost of the car in several ways. Below are some of these effects:

- Example 1: The fact that the client is driving a Sports car suggests a high socio-economic background, and hence will live in a higher-class neighborhood with relatively low crime rate. Therefore, it is likely that part of the property cost that is attributed to thefts will decrease. This is shown in the Bayesian network as the following chain: **MakeModel -> SocioEcon -> HomeBase -> Theft -> ThisCarCost -> PropCost**.

- Example 2: Sports cars are relatively more expensive, hence we would expect that the property cost would increase. This is shown in the Bayesian network as the following chain: **MakeModel -> CarValue (-> Thief) -> ThisCarCost -> PropCost**.

Note that we cannot infer the following conclusion:

- Example 3: Most sports cars are equiped with advanced features such as Antilock braking, and are therefore less likely to be involved in an accident. This is shown in the Bayesian network as the following chain: **MakeModel -> Antilock -> Accident -> OtherCarCost -> PropCost**.

This is because *Antilock* is observed: we already know that the car does not have an antilock brake. Therefore, no new information can travel from MakeModel to PropCost through the path **MakeModel -> Antilock -> Accident -> OtherCarCost -> PropCost**.

In the three examples we have presented above, the first two examples show how PropCost might be affected by a new piece of knowledge: the make/model of a car. The first of the two examples suggests that the property cost would go down (since theft becomes unlikely). However, the second of the two examples suggests that property cost would go up. In fact, all the other possible paths between *MakeModel* and *PropCost* cause conflicting effects. Therefore, it is difficult to intuitively infer the net effect the make/model of the car has on the property cost. One possible answer is to take into account the length of the path that gets us from *MakeModel* to *PropCost*. The longer the path between two variables (evidence and query) , the less effect the evidence variable has on the query. This is because a lot of other variables compete to affect the query variable (i.e. *PropCost*). The second of the two examples (i.e. **MakeModel -> CarValue -> ThisCarCost**

-> **PropCost**) seems to be much shorter than any of the other paths. Therefore, we would suspect that its effect will dominate, i.e. the property cost will increase.

In fact, we can only be sure of the net effect of make/model on property cost by doing computation on the Bayesian network, and Variable Elimination does just that. At first glance, it is not clear in Table 3 and 4 that there is a difference between the two probability distributions. Upon closer examination of the results, however, we see that both the probabilities of property cost in Thousands and in Hundred Thousands decreased in Table 4. On the other hand, the probabilities of property cost in Ten Thousands and in Millions increased in Table 4.

One explanation for these results is that there are two types of people who insure their cars. The first type of person incurrs property cost between Thousands and Ten Thousands, while the second type of person incurrs property cost between Hundred Thousands and Millions. By splitting the population into two, it is easier to observe that the property cost will indeed increase in light of the new fact that the car is a Sports car. Now, the first type of person will more likely incur Ten Thousands than before and less likely to incur Thousands. Similarly they second type of persion will now more likely incur Millions than before and less likely to incur Hundred Thousands. In fact, upon calculation of the mean (expected) property cost, we see an increase from $ 241,380 to $ 260,140. These means are calculated by assuming the middle value of thousands, ten thousands, hundred thousands and millions to be $ 5000, $ 50000, $ 500000, and $ 5000000 respectively.

2.2.2 Insurance Query 2

The results to the Insurance Query 2 described below are shown in Table 5.

$$P(PropCost|Age = Adolescent, Antilock = False, Mileage = FiftyThou, GoodStudent = True) \tag{5}$$

The probability distribution shown in Table 5 is similar to the probability distribution shown in Table 3, except that it is has an additional piece of evidence: the client is a good student. This is on top of the three original pieces of evidence: the client is an adolescent, the car does not have antilock brakes, and the mileage is about fifty thousand.

The fact the client is a good student can only affect property cost through the fact that it is more likely that the client is of good socio-economic background. The other way that property cost could have been affected is through the *Age*, but we know it's impossible because we already know that the student is an adolescent.

Since the new fact can only affect property cost through *SocioEcon*, it is not clear that any of the paths from *GoodStudent* to *PropCost* would dominate. We also know that the paths will have conflicting affects:

- The client is more likely to be risk averse and have a better driving skills, since he/she is a good student. Therefore, property cost will decrease since the client is now less likely to get into an accident.
- The client is more likely to have a more expensive car, due to his/her high socio-economic status. Therefore, property cost will go up.

Therefore, we can only find out the final outcome through computation. As shown in Table 3 and 5, we see a slight increase in the property cost. Again, we see an increase in the expected

Property Cost	Probability
Hundred Thousand	0.18374679176160608
Million	0.029748793596801576
Ten Thousand	0.32771416728772235
Thousand	0.45879024735387

Table 5: Results to Insurance Query 2: Distribution over property cost for an adolescent, who is a good student, driving a car with 50k miles and no anti-lock brakes.

N112	Probability
"0"	0.9880400004226929
"1"	0.01195999957730707

Table 6: Results to Carpo Query 1: Distribution over N112 given N64 = "3", N113 = "1", N116 = "0"

property cost from $ 241,380 to $ 259,300. These means are calculated by assuming the middle value of thousands, ten thousands, hundred thousands and millions to be $ 5000, $ 50000, $ 500000, and $ 5000000 respectively.

2.2.3 Carpo Query 1

The results to the Carpo Query 1 shown below are shown in Table 6.

$$P(N112|N64 = \text{“3”}, N113 = \text{“1”}, N116 = \text{“0”}) \tag{6}$$

2.2.4 Carpo Query 2

The results to the Carpo Query 2 are shown in Table 7.

$$P(N143|N146 = \text{“3”}, N116 = \text{“1”}, N121 = \text{“0”}) \tag{7}$$

2.3 Elimination Orderings

N143	Probability
"0"	0.8999999969611722
"1"	0.10000000303882782

Table 7: Results to Carpo Query 1: Distribution over N143 given N146 = "1", N116 = "0", N121 = "1"

To see how different elimination order affects the running time of our *Variable Elimination* algorithm, we randomize the elimination order with some uniform distribution. By uniform distribution, we mean that any *elimination variables (node)* are equally likely to be the first to be eliminated, the remainings are equally likely to be the next to be eliminated, and so on. Recall that *elimination variables* does not include query variable and any of the evidence variables.

By repeating many times the *Variable Elimination* algorithm with *random elimination order* explained above, we can learn:

- What would be, on average, the computation time would be
- How large does the computation time varies with different elimination order, in other words, how important it is to choose the "right" elimination order

Again, we use the following 4 probability calculation (used in the previous) subsection as benchmark :

1. Probability distribution shown in equation (4):

   $P(PropCost|Age = Adolescent, Antilock = False, Mileage = FiftyThou, MakeModel = SportsCar)$

   with distribution shown in Table 4.

   After 20 sampling of *Variable Elimination* algorithm with *random elimination order* we see a distribution of running time, shown in Figure 1. From the figure, we can see that Variable Elimination takes less than 10 minutes more than half of the times. (Of course, our confidence will get better with more sampling). Computing mean running time is hard (if not impossible), since all runs that takes more than 10 minutes are lumped together in the last group (10-11 minutes). Even though Variable Elimination will eventually terminates, it will take a very long time on the worst case, since it solves an NP hard problem (if not harder). However, we can bound mean time by setting up the following model:

   > We run the Variable Elimination algorithm with random elimination for N minutes, here we use N to be 10. If program doesn't terminate after 10 minutes, we kill (force terminate) the program and re-sample a new random elimination order. We then run the algorithm for another 10 minutes. We keep repeating the process until a program terminates within 10 minutes.
   >
   > We have essentially set up a Stochastic model, where at each "coin flips" (i.e. runs), we have some probability of winning (finished *VariableElimination* algorithm in time.

   Since, each run lasts 10 minutes, and the probability of succeeding in a run can be estimated as the proportion of our actual test runs that finishes within 10 minutes, we can use expectation (mean) formula of exponential distribution:

   $$E[running\,time] = 10\,minutes \times \frac{1}{P(finishes\,in\,10\,minutes)}$$

   Since, $\frac{12}{20} = 0.6$ of the runs finishes in less than 10 minutes, the mean running time is $10 \times \frac{1}{0.6} =$ 17 minutes.

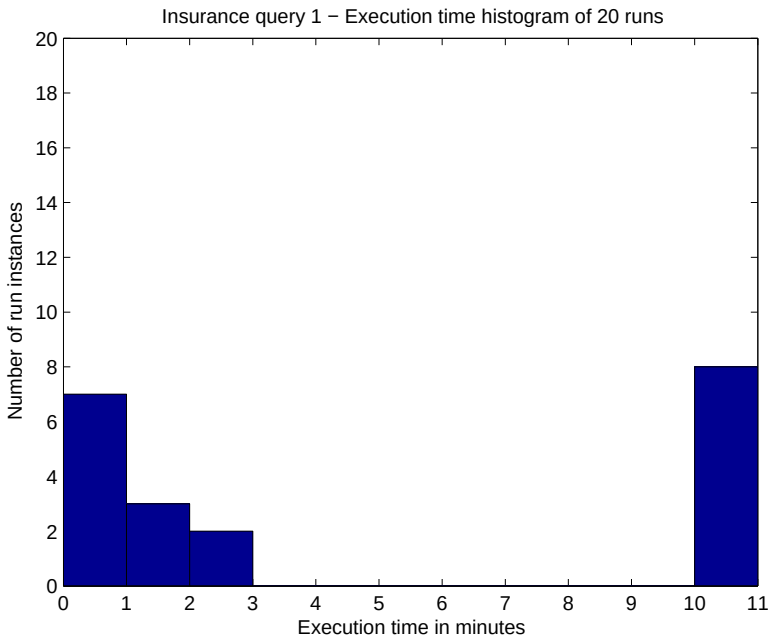

Figure 1: Distribution of running time of 20 independent runs of Variable Elimination algorithm with random elimination order. All runs that takes more than 10 minutes are lumped into the last group (i.e. 10-11 minutes).

2. Probability distribution shown in equation (5):

   $P(PropCost|Age = Adolescent, Antilock = False, Mileage = FiftyThou, GoodStudent = True)$

   with distribution shown in Table 5.

   After 20 sampling of *Variable Elimination* algorithm with *random elimination order* we see a distribution of running time, shown in Figure 2. Unlike the previous testcase, most of our runs now take more than 10 minutes (in which case our VariableElimination times out and terminates with no result).

   We have $\frac{5}{20} = 0.25$ of the runs finishes in less than 10 minutes. Hence, assumming the model that we have set up previously, the mean running time is $10 \times \frac{1}{0.25} = 40$ minutes.

3. Probability distribution shown in equation (6):

   $$P(N112|N64 = \text{“}3'', N113 = \text{“}1'', N116 = \text{“}0'')$$

   with distribution shown in Table 6.

   After 20 sampling of *Variable Elimination* algorithm with *random elimination order* we see a distribution of running time, shown in Figure 3. This particular test case has the largest variance of execution time: it's either very short or very long (the later being the most probable).

   We have $\frac{1}{20} = 0.05$ of the runs finishes in less than 10 minutes. Hence, assumming the model that we have set up previously, the mean running time is $10 \times \frac{1}{0.05} = 200$ minutes.

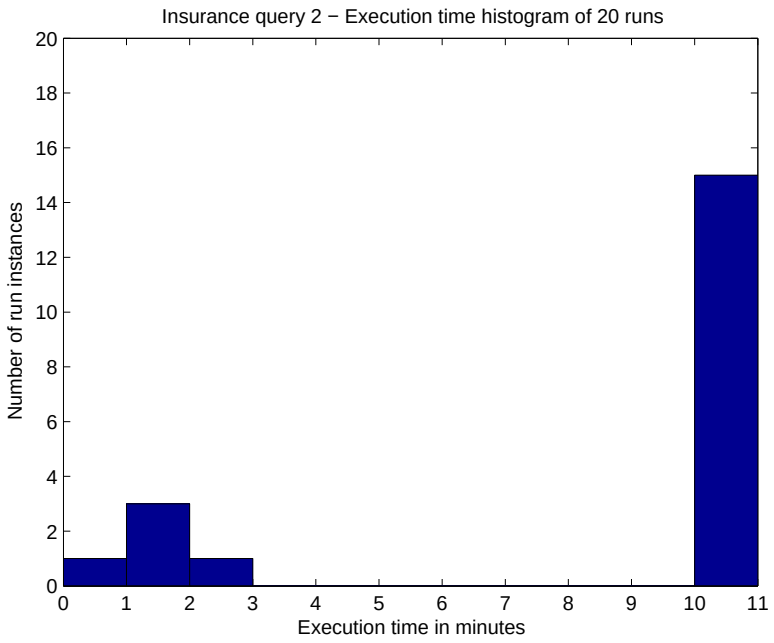

Figure 2: Distribution of running time of 20 independent runs of Variable Elimination algorithm with random elimination order. All runs that takes more than 10 minutes are lumped into the last group (i.e. 10-11 minutes).

4. Probability distribution shown in equation (7):

$$P(N143|N146 = \text{“}3'', N116 = \text{“}1'', N121 = \text{“}0'')$$

with distribution shown in Table 7. After 20 sampling of *Variable Elimination* algorithm with *random elimination order* we see a distribution of running time, shown in Figure 4. This particular test case does not show as large of a variance of execution time as the last one,but it's still relatively large.

We have $\frac{4}{20} = 0.2$ of the runs finishes in less than 10 minutes. Hence, assumming the model that we have set up previously, the mean running time is $10 \times \frac{1}{0.2} = 50$ minutes.

Here, we see that some Bayesian network are more sensitive to elimination order than the other: the Carpo network is more sensitive than the Insurance network. We can see this from the layout diagram of the Carpo network (see Project 2 assignment - 6.825 Fall 04). Each node is connected with a varying number of neighboring (i.e parent and children) nodes. For instance, a good number of nodes are connected with only one or two nodes (so eliminating these nodes first would yield a factor of size one or two). However, some nodes are connected with more than ten neighbors (some even up to 15 neighboring nodes). Eliminating these nodes too early will causes a prohibitively (but unnecessarily) huge intermediate factors. On the other hand, the insurance network, albeit being complicated has relatively uniform degree of connectivity in all of its nodes: most of them are connected to 3-6 neighbors. Therefore, we would always produce factor tables that mostly contain 3-6 variables (and the same time discarding the factors that we have finished multiplying, which

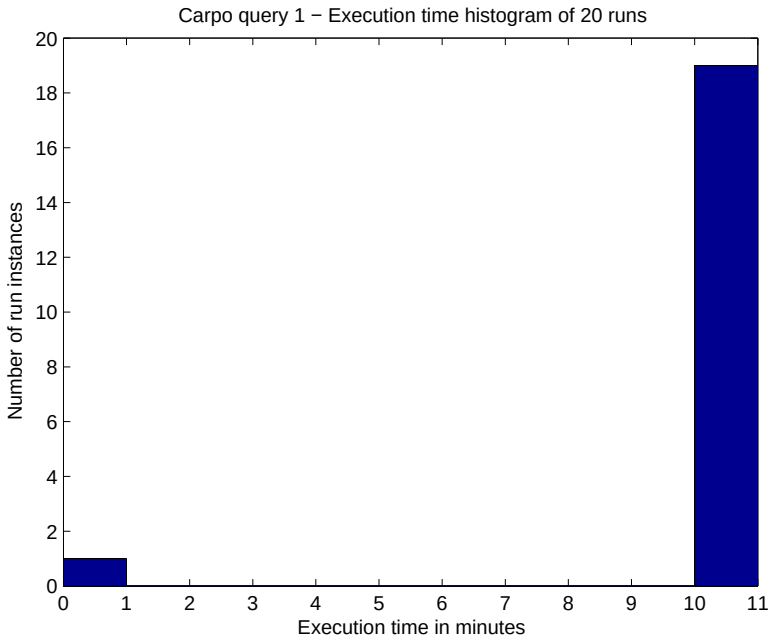

Figure 3: Distribution of running time of 20 independent runs of Variable Elimination algorithm with random elimination order. All runs that takes more than 10 minutes are lumped into the last group (i.e. 10-11 minutes).

mostly also contain 3-6 vairables). Therefore, even though the elimination order affects Variable Elimination of any network (i.e. Some runs takes less than a minute, but some takes more than 1 hour or even causes the computer to run out of memory, for both Insurance and Carpo network), some network are more affected than the others.

2.4 Greedy Elimination

After realizing how important the elimination ordering is towards the performance (and even feasibility) of *Variable Elimination* algorithm in computing conditional probability (and other interesting problems), we will attempt to compute a "good" elimination order. Note that computing the best elimination order is an NP or harder problem. We will therefore use a greedy algorithm to compute a "semi-optimal" or "good" elimination order.

Our greedy algorithm chooses, at each elimination iteration, a node that when eliminated will produces a factor with the smallest variable number. Of course, the elimination order is precomputed before the actual *Variable Elimination* algorithm is run, hence setting a level playing ground for every method that we use, since all of them precomputed elimination order prior to actual query of the Bayesian network. Note however, that it has been observed that greedy algorithm precomputation of semi-optimal elimination order takes less than 1 second for all test cases.

Table 8 shows the performance comparison of Variable Elimination algorithm given different elimination ordering: reverse topological, random, and greedy ordering.

We see that greedy algorithm is almost always faster than the other two methods. An exception is found on testcase 4 (which is a test on the Carpo network), where Reverse Topological ordering

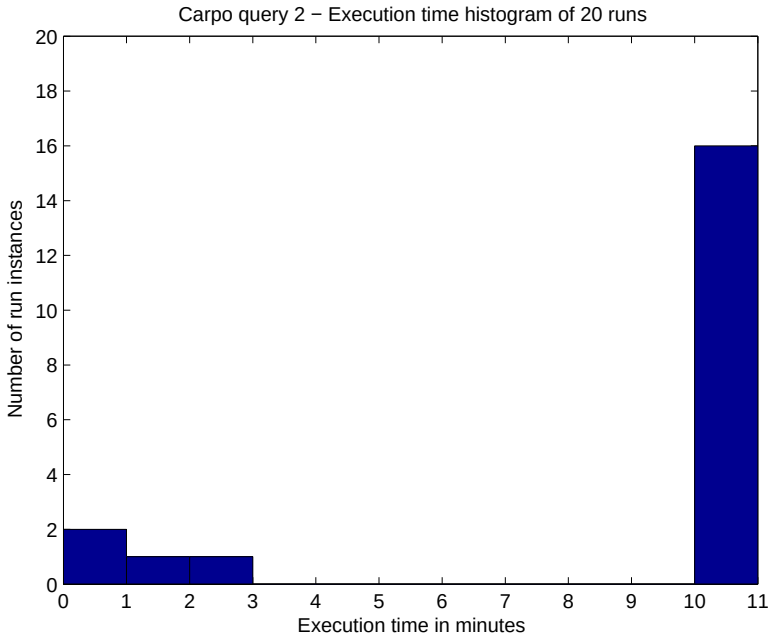

Figure 4: Distribution of running time of 20 independent runs of Variable Elimination algorithm with random elimination order. All runs that takes more than 10 minutes are lumped into the last group (i.e. 10-11 minutes).

Testcase	Reverse Topological	Random (expected)	Greedy Algorithm
1 (Insurance)	7.5 sec	17 min	1 sec
2 (Insurance)	22.5 sec	40 min	5.5 sec
3 (Carpo)	224 msec	200 min	210 msec
4 (Carpo)	183 msec	50 min	223 msec

Table 8: Performance comparison between *reverse topological* ordering, randomized ordering, and ordering obtained from *greedy algorithm*

gives a better performance that that obtained from *greedy algorithm*. This supports our earlier observations that *greedy algorithm* only guarantees "semi-optimal" answer. This is because, it might be stuck in some local minimum of the search space.

Burglary	Probability	KL Divergence
false	0.8071153021860509	0.024194
true	0.19288469781394904	-

Table 9: Results from likelihood weighting to query described by equation (1).

Burglary	Probability	KL Divergence
false	0.9981937989056889	$1.22 * 10^{-}5$
true	0.0018062010943110723	-

Table 10: Results from likelihood weighting to query described by equation (2).

3 Likelihood Weighting

We implemented Likelihood Weighting as described in [2]. Given a Bayesian network, assignments of some variables e (evidence),and a query node X, our likelihood weighting implementation estimates $P(X|e)$.

3.1 Validation

The results of the queries given by equations (1)-(7) are shown in Tables 8-14 respectively. Each query was run for 10,000 samples.

We evaluated the accuracy of the likelihood weighting estimates using the Kullback-Leibler divergence (KL divergence) between the estimate and the true distribution (from variable elimination). The smaller the KL divergence, the better the estimate. KL divergence is given by:

$$\sum_{x \epsilon X} P(x|e) log \left(\frac{P(x|e)}{\hat{P}(x|e)} \right) \tag{8}$$

In Tables 8-14, the first entry in the third column shows the KL divergence of the likelihood weighting results to the true distribution given by variable elimination. As can be seen from the very small KL divergences, our implementation of likelihood weighting is correct.

Property Cost	Probability	KL Divergence
Hundred Thousand	0.1728511874428114	0.000238
Million	0.026064452637473152	-
Ten Thousand	0.3524424295237774	-
Thousand	0.4486419303959379	-

Table 11: Results from likelihood weighting to query described in (3)

Property Cost	Probability	KL Divergence
Hundred Thousand	0.17016701486884045	$1.11 * 10^{-}5$
Million	0.031112169971533002	-
Ten Thousand	0.34577087074979307	-
Thousand	0.45294994440983355	-

Table 12: Results from likelihood weighting to query described in (4)

Property Cost	Probability	KL Divergence
Hundred Thousand	0.182654291932025	$9.27 * 10^{-}5$
Million	0.03191464153881571	-
Ten Thousand	0.32953975403946134	-
Thousand	0.4558913124896979	-

Table 13: Results from likelihood weighting to query described in (5)

N143	Probability	KL Divergence
"0"	0.987323593420549	$2.09 * 10^{-}5$
"1"	0.012676406579450994	-

Table 14: Results from likelihood weighting to query described in equation (6)

N143	Probability	KL Divergence
"0"	0.9020458194381775	$2.35 * 10^{-}5$
"1"	0.09795418056182242	-

Table 15: Results from likelihood weighting to query described in equation (7)

Burglary	Probability	KL Divergence
false	0.7148	$2.59 * 10^-6$
true	0.2852	-

Table 16: Results from Gibbs Sampling to query described by equation (1).

Burglary	Probability	KL Divergence
false	0.9981	$3.72 * 10^-6$
true	0.0019	-

Table 17: Results from Gibbs Sampling to query described by equation (2).

4 Gibbs Sampling

Given a Bayesian network, assignments of some variables e (evidence),and a query node X, our Gibbs sampling implementation estimates $P(X|e)$. We implemented Gibbs sampling slightly differently from the outline given in [2]. As suggested in lecture, instead of updating the count right after assigning each non evidence variable, we wait until all the non evidence variables have been assigned before updating the count. Using our implementation we would generate the exact number of samples specified by the input. The Gibbs sampling described in [2] would generate $N * (numberofnonevidencevariables)$ samples.

4.1 Validation

The results of the queries given by equations (1)-(7) are shown in Tables 15-21 respectively. Each query was run for a total of 11,000 samples, of which 10,000 samples were counted and the first 1000 samples were discarded (burn-in). The first entry of the third column of each table shows the KL divergence of the results. As can be seen from the very small KL divergences, our implementation of Gibbs sampling is correct.

4.2 Experimenting with Burn-in

We studied the effect of burn-in on the accuracy of the estimate given by Gibbs sampling. We define burn-in as the number of initial samples that are thrown away (not counted in the estimate). We expected that accuracy would increase as burn-in is increased because burn-in will erase the initial

Property Cost	Probability	KL Divergence
Hundred Thousand	0.1467	0.004893
Million	0.0188	-
Ten Thousand	0.3456	-
Thousand	0.4889	-

Table 18: Results from Gibbs Sampling to query described in (3).

Property Cost	Probability	KL Divergence
Hundred Thousand	0.1987	0.003494
Million	0.0344	-
Ten Thousand	0.3157	-
Thousand	0.4512	-

Table 19: Results from Gibbs Sampling to query described in (4).

Property Cost	Probability	KL Divergence
Hundred Thousand	0.1905	0.001513
Million	0.0319	-
Ten Thousand	0.3024	-
Thousand	0.4752	-

Table 20: Results from Gibbs Sampling to query described in (5).

N143	Probability	KL Divergence
"0"	0.9867	$7.08 * 10^{-}5$
"1"	0.0133	-

Table 21: Results from likelihood weighting to query described in equation (6).

N143	Probability	KL Divergence
"0"	0.9143	0.001244
"1"	0.0857	-

Table 22: Results from likelihood weighting to query described in equation (7).

bias due to the random initial setting of the variables. In order to isolate the effect of burn-in, we fixed the number of samples for each run while varying the burn-in. Also, we picked a high value for the number of samples so that the variation in the estimates is more likely due to the burn-in than the inaccuracy due to few samples.

For each query described by equations (4)-(7), we fixed the number of samples to 10,000 and experimented with the following values of burnin: 0, 5, 25, 125, 625, 1250, 5000.

We evaluated the accuracy of the estimates using KL divergence. Figures 5 - 8 show plots of KL divergence vs. Burn-in for the queries given by equations (4)-(7). As shown in Figures 5 - 8, the trend is that the larger number of burn-in, the better the estimate. However, after a certain number of burn-in (B*), increasing burn-in will only increase accuracy by smaller and smaller amounts. After throwing away B* samples, the sampler has gotten rid of almost all of the initial bias from the random initialization of variables at the beginning of the run. We define B* as the burn-in number where the curve just starts to flatten.

For each of the queries given by equations (4-7) we identified b* as the closest burn-in from our set of burn in values [0, 5, 25, 125, 625, 1250, 5000] that best fits our definition of B*. These b* values will be used as the burn-in for later analysis with Gibbs Sampling in the next section. For each of the queries described in equations (4-7), the b* values are:

- Equation (4): b* = 1250 samples
- Equation (5): b* = 625 samples
- Equation (6): b* = 1250 samples
- Equation (7): b* = 1250 samples

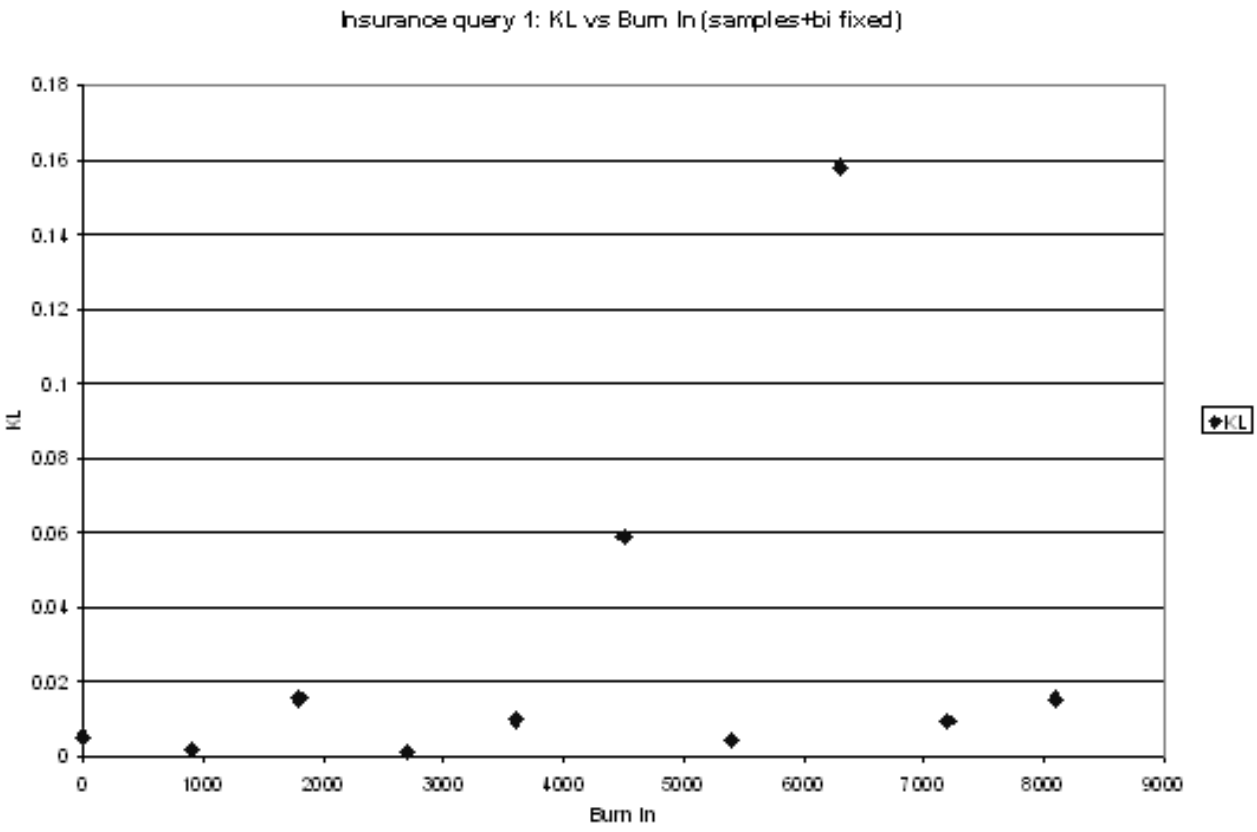

Figure 5: Insurance Query 1: Fixed number of samples, vary burn in.

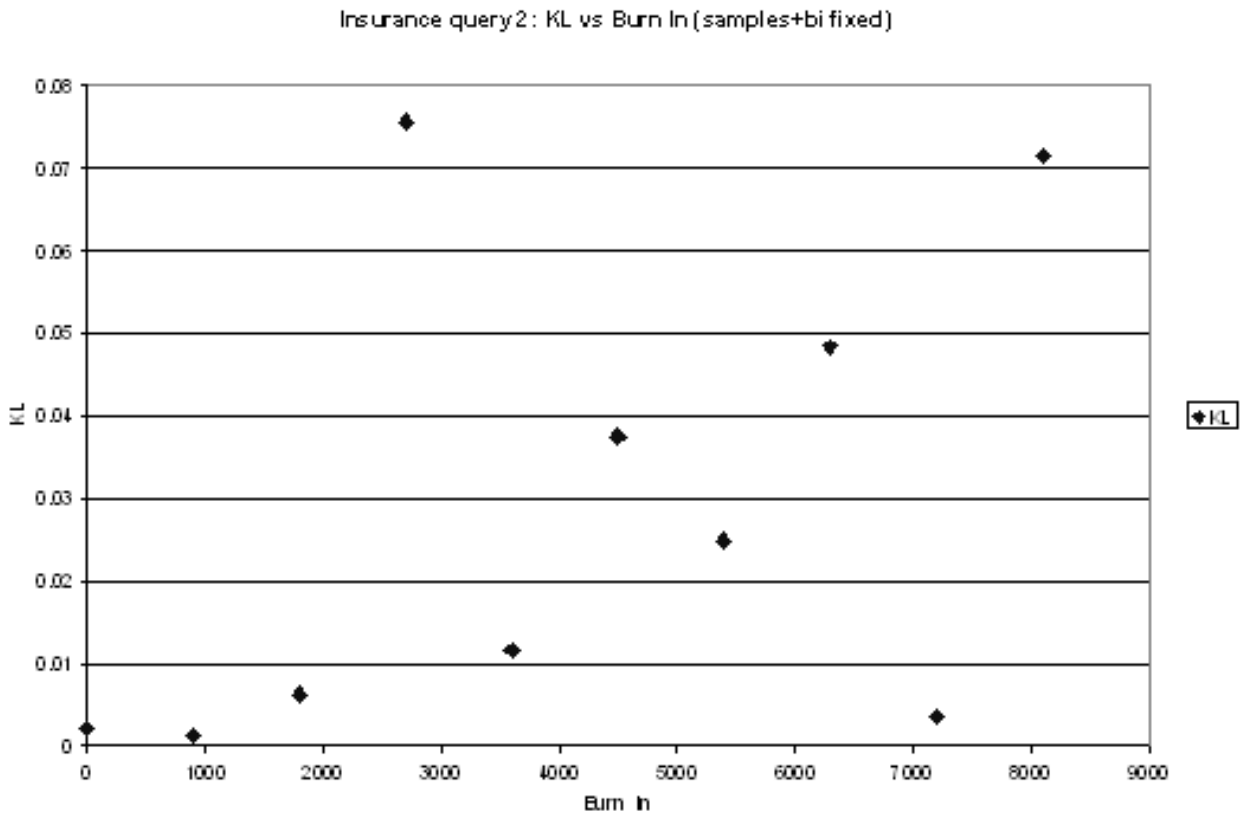

Figure 6: Insurance Query 2: Fixed number of samples, vary burn in.

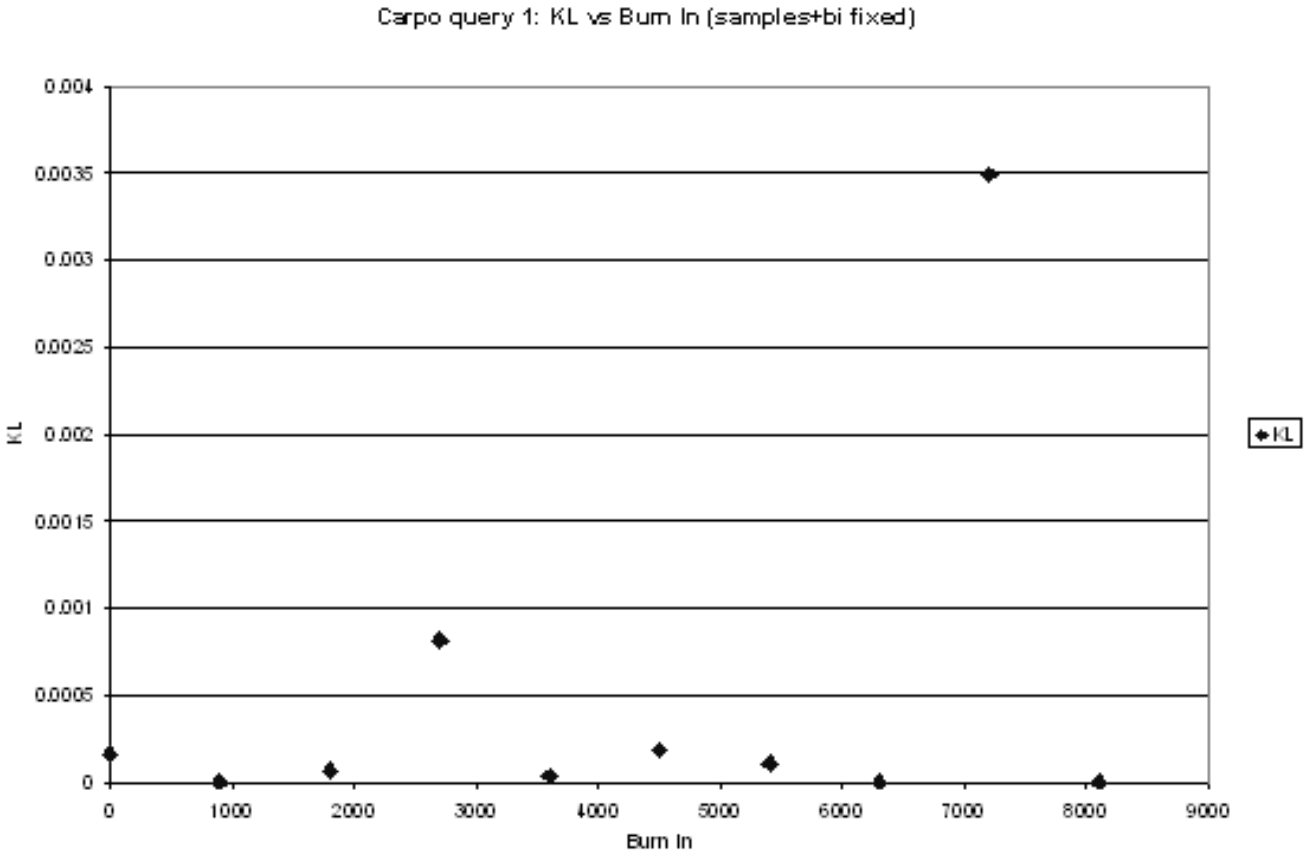

Figure 7: Carpo Query 1: Fixed number of samples, vary burn in.

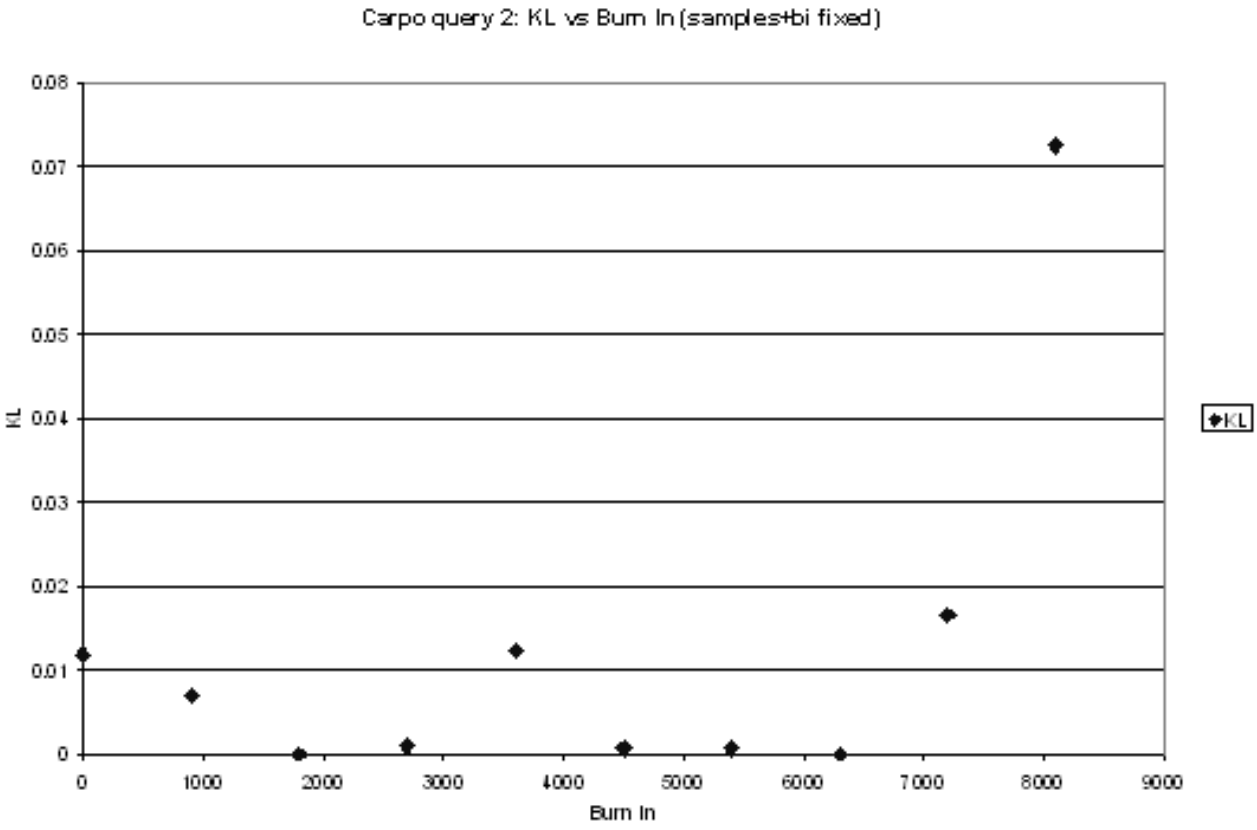

Figure 8: Insurance Query 2: Fixed number of samples, vary burn in.

5 Performance vs. Runtime of Likelihood Weighting and Gibbs Sampling

In order to study the accuracy of the estimates vs. runtime for each of the approximate inference algorithms, we ran 10 experiments for each of the queries described in equations (4)-(7). Each experiment, for a particular query and approximate inference algorithm, consisted of one run of the algorithm with the maximum number of samples to generate set to 10,000. In each run, KL divergence is calculated after the following number of samples have been generated: 0, 10, 30, 100, 300, 1000, 3000. In the experiments using Gibbs sampling, the burn-in is set to the b* values obtained in the previous section. Again, these best burn-in values for each query are:

- Equation (4): b* = 1250 samples
- Equation (5): b* = 625 samples
- Equation (6): b* = 1250 samples
- Equation (7): b* = 1250 samples

Figures 9 - 12 show KL divergence vs. number of samples for the likelihood weighting algorithm. Figures 13 - 16 show the KL divergence vs. number of samples for the Gibbs sampling algorithm. The figures correspond to the queries described by equations (4)-(7).

As can be seen by the Figures 9 - 16, increasing the number of samples decreases KL divergence (increases accuracy) for both likelihood weighting and Gibbs sampling. This is in line with our general intuition because as the number of samples increase, the estimate gets closer to the true value. We have proved in lecture that likelihood weighting and Gibbs sampling give *consistent* estimates. The curves from different runs intersect with each other because of the following two reasons.

1. The error associated with our measurement causes the best fit curve that we have generate to slightly deviate from the true curve. Therefore, as more samplings are obtained, our best fit curve will converge to the true model curves, which might not intersects with each other
2. However, even the true model curves might intersect with each other. This is because, each iteration performs a random sampling. Therefore, a program might make a good progress during some iteration (and pushes KL downward greatly), and might not do so well at some iterations. Therefore, we see a "race" to bring KL towards zero, and it's possible for some instance of runs to "overtake" some other instances during this "race".

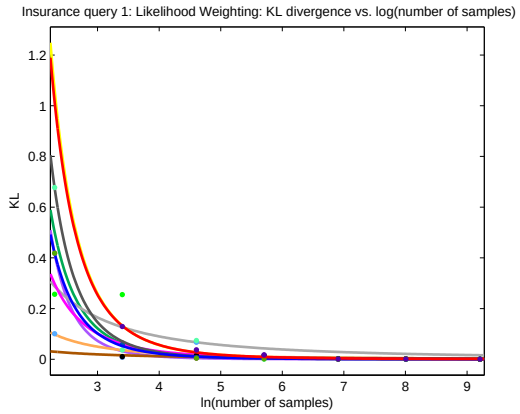

Figure 9: Insurnace Query 1: Ten runs: KL divergence vs. number of samples

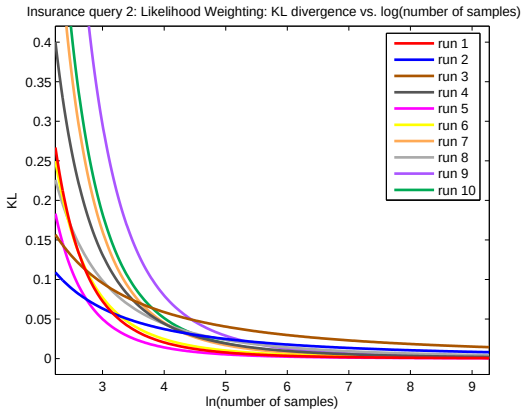

Figure 10: Insurance Query 2: Ten runs: KL divergence vs. number of samples

Figures 17 - 20 show the average KL divergence vs. number of samples for both the likelihood weighting and Gibbs sampling algorithms. As can be see from these figures, after a certain number of samples S*, the divergence does not decrease by much. Therefore there is no point in sampling above S*. ¡?? S* however depends on the query. If the true distribution of the query is heavily biased, then S* is smaller? than if the query distribution is more evenly distributed between the different values of the domain.??¿

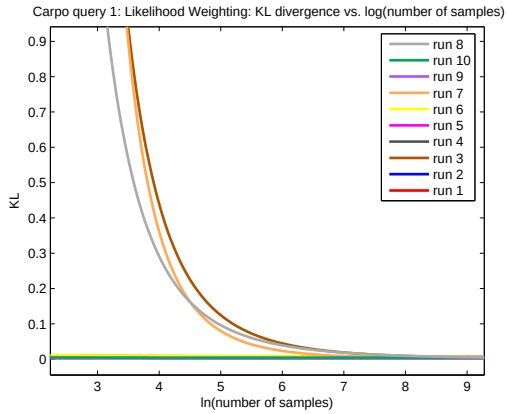

Figure 11: Carpo Query 1: Ten runs: KL divergence vs. number of samples

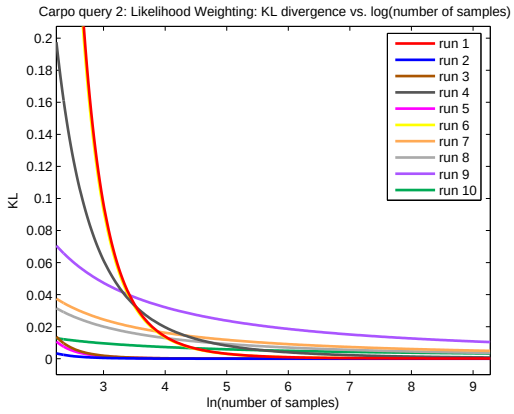

Figure 12: Carpo Query 2: Ten runs: KL divergence vs. number of samples

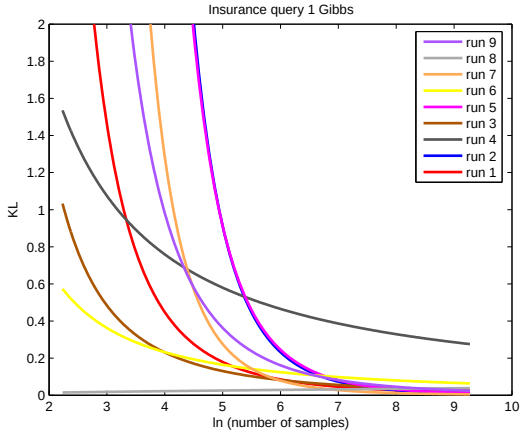

Figure 13: Insurance Query 1: Ten runs: KL divergence vs. number of samples

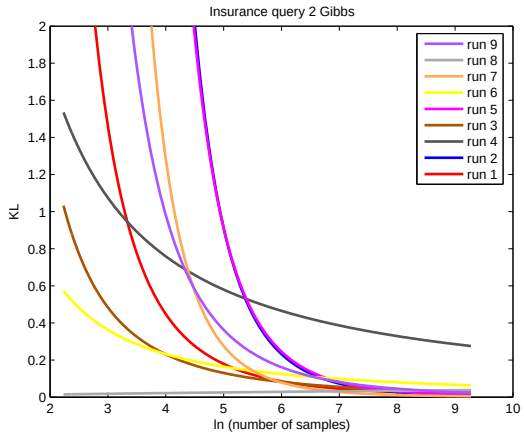

Figure 14: Insurance Query 2: Ten runs: KL divergence vs. number of samples

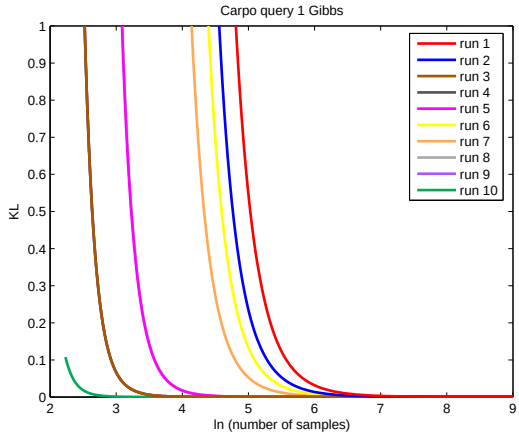

Figure 15: Carpo Query 1: Ten runs: KL divergence vs. number of samples

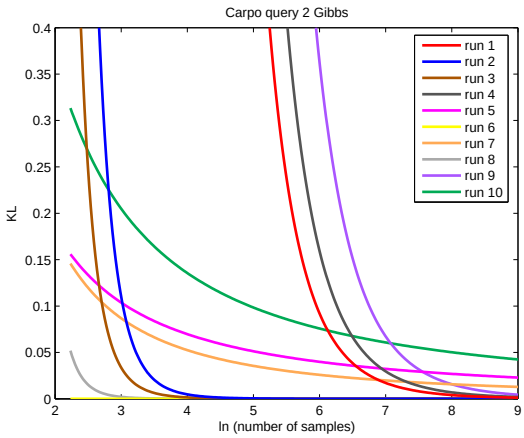

Figure 16: Carpo Query 2: Ten runs: KL divergence vs. number of samples

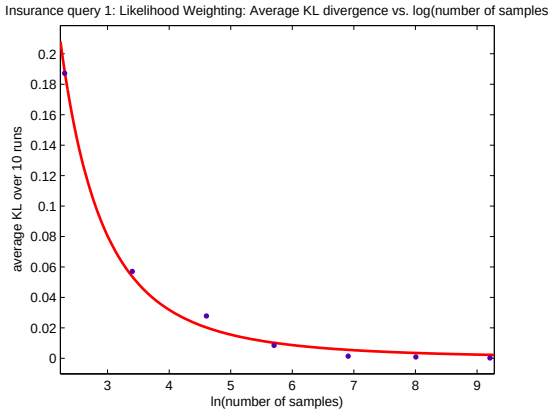

Figure 17: Insurance Query 1: LW: Ten runs: Average KL divergence vs. number of samples

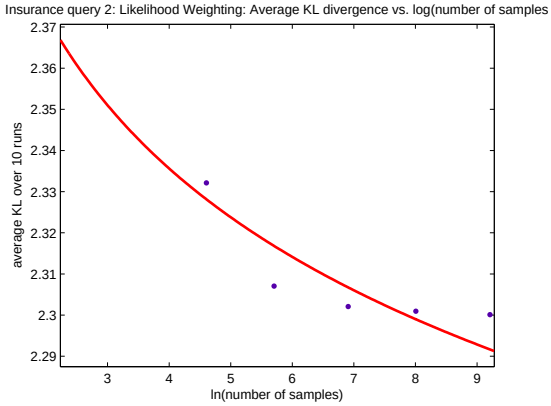

Figure 18: Insurance Query 2: LW:Ten runs: Average KL divergence vs. number of samples

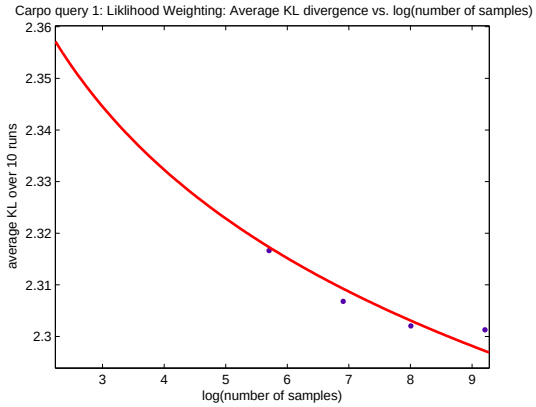

Figure 19: Carpo Query 1: LW: Ten runs: Average KL divergence vs. number of samples

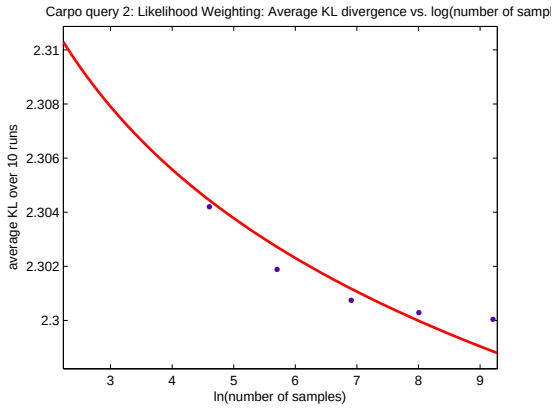

Figure 20: Carpo Query 2: LW: Ten runs: Average KL divergence vs. number of samples

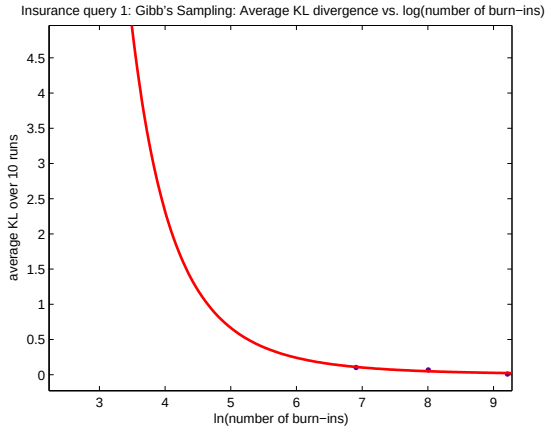

Figure 21: Insurance Query 1: Gibbs: Ten runs: Average KL divergence vs. number of samples

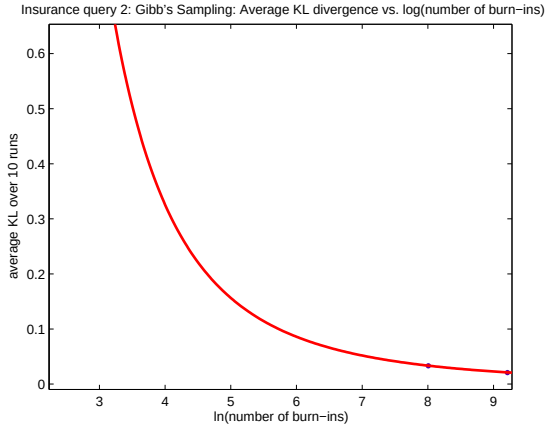

Figure 22: Insurance Query 2: Gibbs: Ten runs: Average KL divergence vs. number of samples

For each query, Table 22 shows the number of samples for which the computation time of each approximate inference algorithm (for each query) equals the computation time of variable elimination (using topological elimination ordering). We realize that the number of samples shown in Table 22 is dependent on the implementation of these algorithms. Therefore these numbers will vary depending on how optimized our implementations are.

Using our current implementations and the values in Table 22, we would use variable elimination for the carpo network, sampling method for insurance network, and variable elimination for Burglary. In insurance network, both sampling method (i.e. Likelihood Weighting and Gibbs Sampling) give a good performance (i.e. low KL). Likelihood Weighting seems to be better for the first query (i.e. insurance Q1). For the second query (i.e. insurance Q2), both methods give a similar KL measurement. Therefore, we pick Likelihood Weighting, because it achieved similar KL error with less iteration (i.e. longer computation time per iteration). This means that if we optimize our code for Likelihood, we can generate the same amount of samples in less time. Therefore Liklihood may take less time.

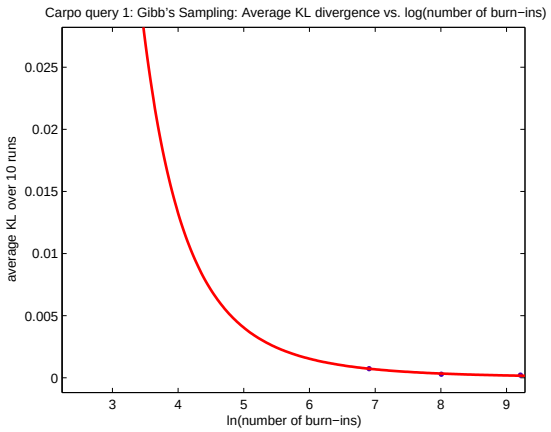

Figure 23: Carpo Query 1: Gibbs: Ten runs: Average KL divergence vs. number of samples

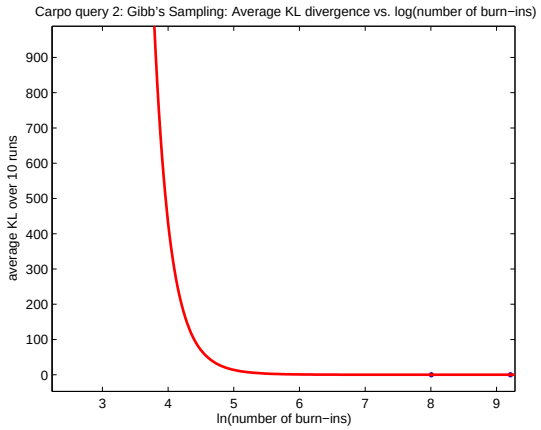

Figure 24: Carpo Query 2: Gibbs: Ten runs: Average KL divergence vs. number of samples

From Table 22, we see that Likelihood weighting is very slow compared to variable elimination. For the Carpo network, variable elimination completes in the time it takes to generate 6-8 samples. The slow speed of Likelihood weighting relative to variable elimination is partially due to the fact that our implementation is not optimized, and variable elimination on the carpo network using topological ordering is very fast.

Query	Num. Samples	Variable Elim Runtime (ms)	Approx KL Div.
Insurance Q1 (LW)	142	7627	less than 0.05
Insurance Q1 (Gibbs)	2709	7627	less than 0.1
Insurance Q2 (LW)	405	22650	less than 0.05
Insurance Q2 (Gibbs)	7249	22650	less than 0.05
Carpo Q1 (LW)	8	224	very large
Carpo Q1 (Gibbs)	59	224	very large
Carpo Q2 (LW)	6	183	large
Carpo Q2 (Gibbs)	49	183	large

Table 23: Variable elimination running time expressed in terms of number of samples that can be generated by each approximate inference algorithm.

6 Work In Progress/Future Work

An alternative way of analyzing the affect of burn-in on Gibbs sampling accuracy is to fix number of samples + burn-in. When this value is fixed, as burn-in increases, the number of samples counted decreases. This analysis of burn-in is interesting because it attempts to answer the question of what is the best burn-in value given that your running time is limited. In our previous analysis of burn-in, we investigated the question of what is the best burn-in for a given query without any constraint on running time.

Since the running time is fixed, we expect that as burn in increases, the accuracy will initially increase (KL divergence decreases). Then after some burn-in value, the accuracy will decrease because the number of counted samples will be too low.

We ran some experiments using the following values of burnin and samples. Their sum is fixed at 10,000.

Figures 25 - 28 show our initial results for each problem. The results did not turn out as we expected. Instead of a U shape, the points are very scattered. We hypothesis that this is due to the variation in initial assignments. Some experiments got lucky and picked a good initialization, therefore although the burn-in is small, accuracy is still high. Other experiments were unlucky and therefore even with higher burn-in, accuracy is low.

To eliminate the effect of the initialization, we can either re-run our experiments with a fixed initialization of variables. Or, we could run a set of experiments with our current setting and then take an average of the results. Due to time constraints, we were not able to explore these possibilities.

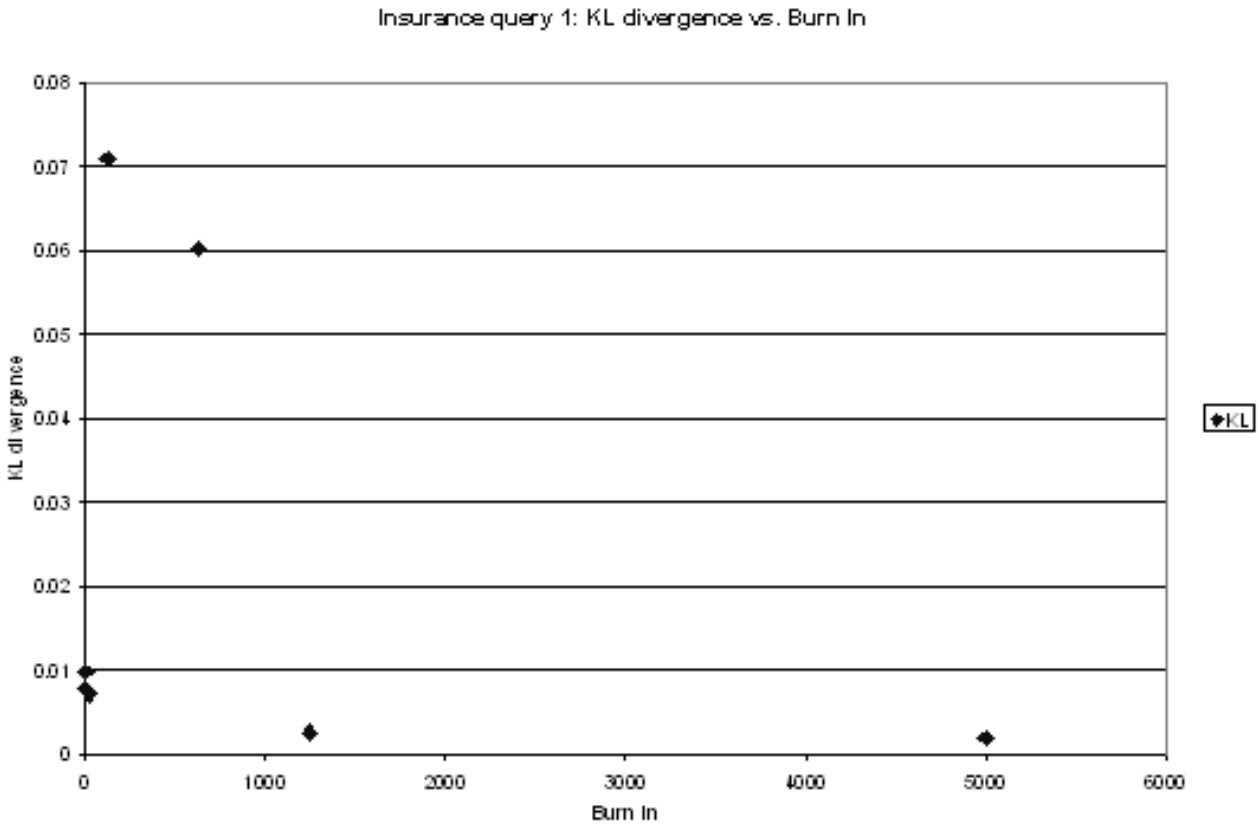

Figure 25: Insurance Query 1: Fixed (burnin in + samples)=10,000

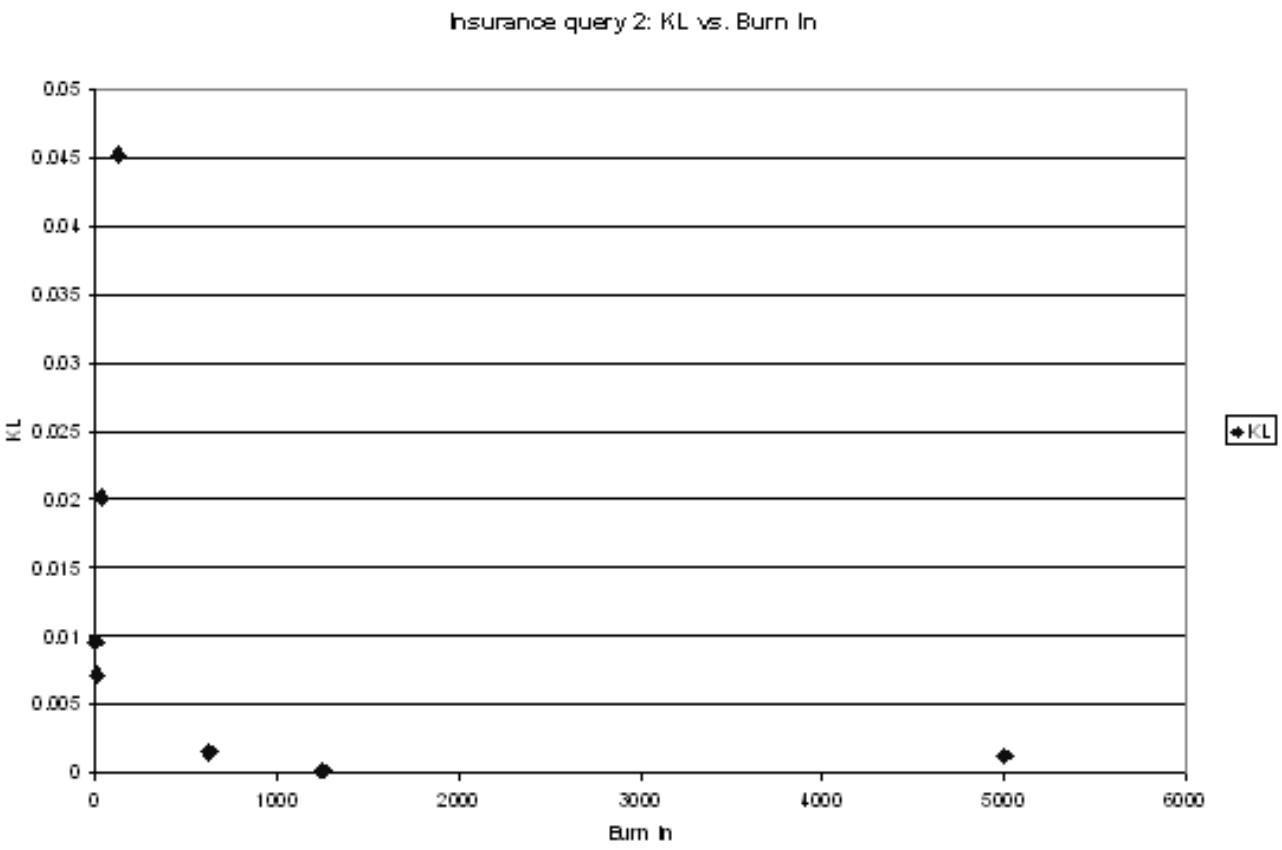

Figure 26: Insurance Query 2: Fixed (burnin in + samples)=10,000

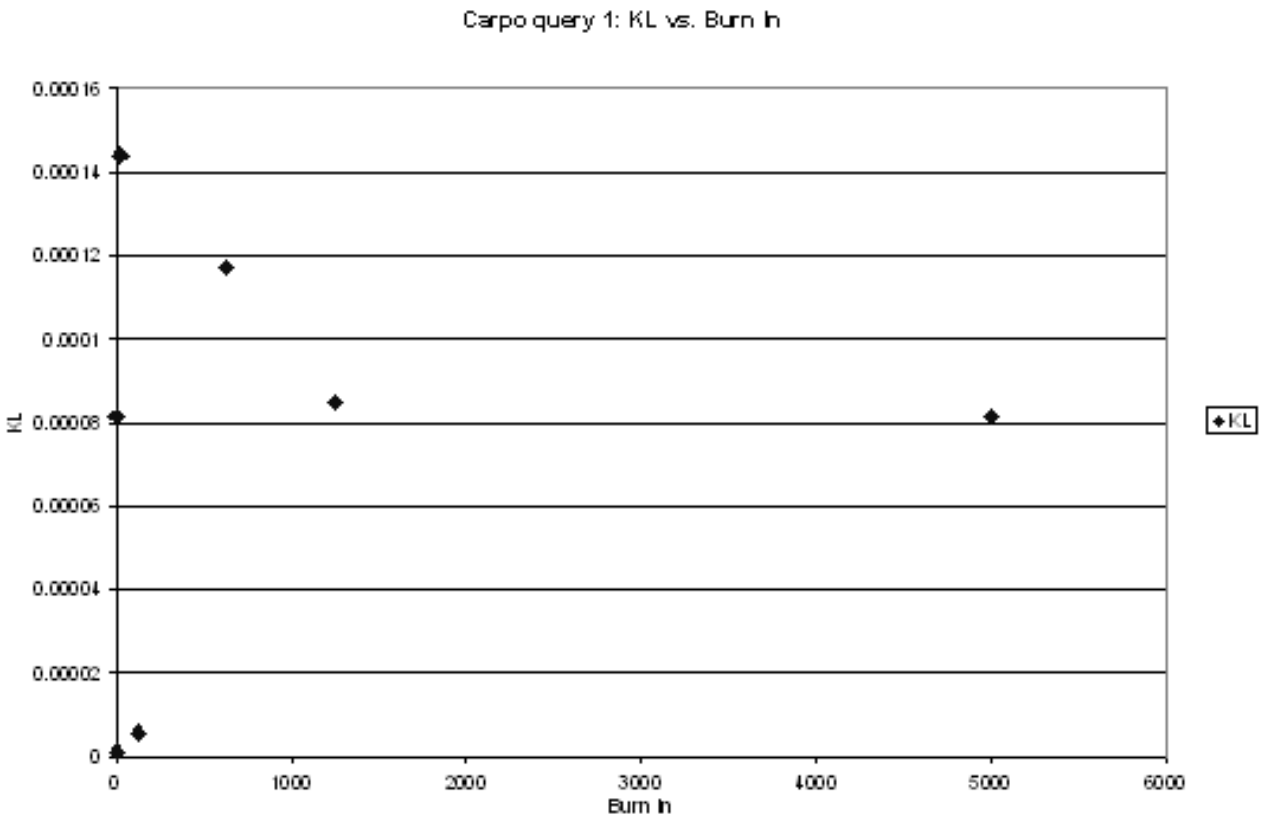

Figure 27: Carpo Query 1: Fixed (burnin in + samples)=10,000

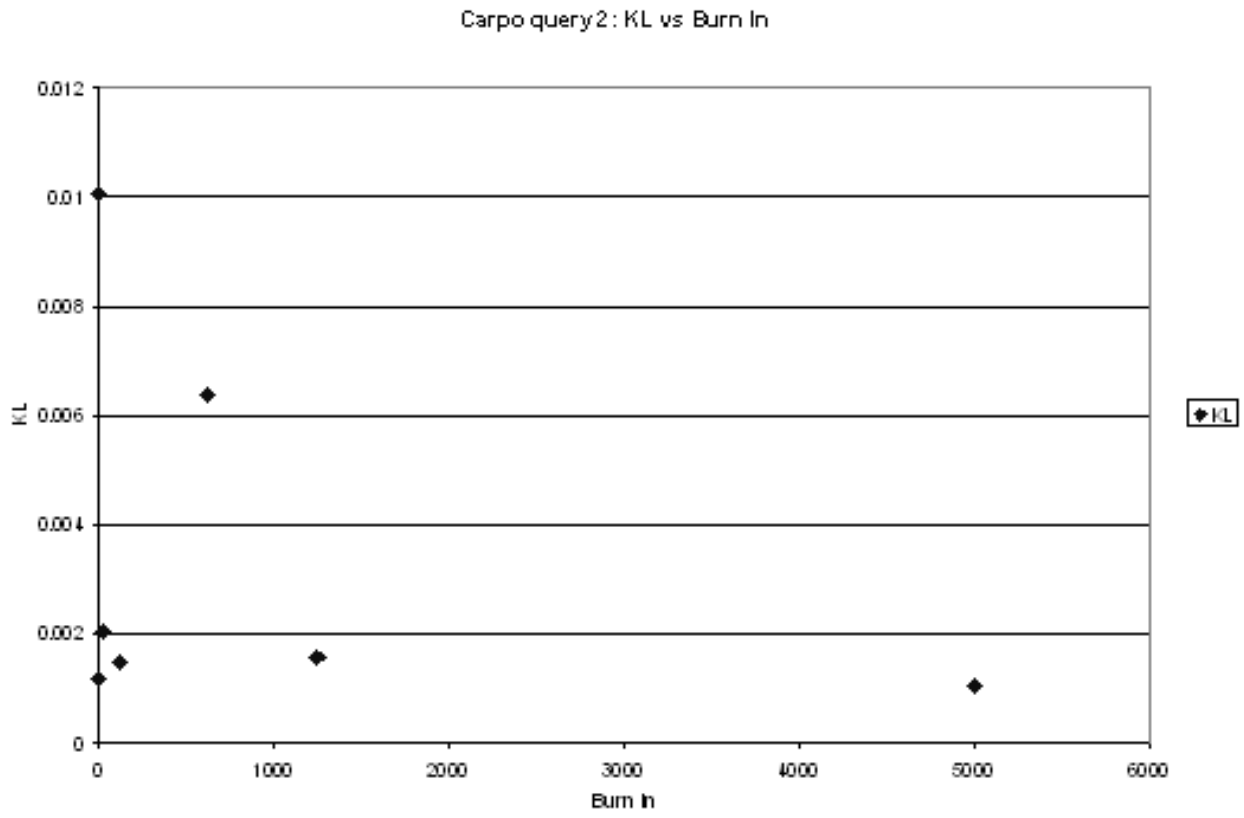

Figure 28: Carpo Query 2: Fixed (burnin in + samples)=10,000

References

[1] Koller and Friedman. Draft: Bayesian Networks and Beyond.

[2] Russel and Norvig. Artificial Intelligence: A Modern Approach.