6.825 - Techniques of Artificial Intelligence
Project II

Lawrence A. Bush and Daniel M. Roy
{larrybush, droy}@mit.edu

November 4, 2004

Abstract

In this paper, we investigate exact and approximate inference algorithms using three Bayesian graphical models of increasing complexity. Specifically, we calculate the exact values for a set of queries using variable elimination, and then use these values to judge the relative merits of two approximate inference algorithms, likelihood weighting and Gibbs sampling. We make this comparison precise by employing the Kullback-Leibler divergence measure. Finally, we test the efficacy of greedy heuristics for choosing elimination orders, and compare the relative accuracy of random v. systematic resampling in the Gibbs sampling algorithm.

Introduction

We implemented the algorithms in both Java and MATLAB. Beyond the advantage of verifying results, implementing the code in both languages provided insight into the relative merits of the two languages for math intensive programming. Unsurprisingly, the Java implementation was bloated and riddled with excessive amounts of support code, while the MATLAB code required virtually none; The first-class status of matrices in MATLAB simplified implementation, and the instantaneous access to plotting and other graphical and statistical functions made the MATLAB implementation a breeze. We highly recommend MATLAB to future classes.

The Java Implementation: We produced Java versions of the variable elimination, likelihood weighting and Gibbs sampling algorithms presented in chapters 7-9 of *Bayesian Networks and Beyond*, a unpublished draft of a book by Daphne Koller and Nir Friedman. The Java implementation used the provided base code[1].

The primary class in this implementation is *FactorSet*, which manages a set of conditional probability tables, and their respective variables. For example, *FactorSet* includes member functions to store, multiply and marginalize variables from conditional probability tables. The class implementation includes member functions which make query construction easy and error-free. The *Queries* class manages the queries and time trials. The *Output* class manages the output of the code into human and machine readable formats. For example, time trials are output to a file in a format readable by a graphics package. Overall, the Java Implementation provides an object-oriented package, which lends itself to reuse in a larger application. However, Java is not the best tool for experimenting with algorithms.

[1]BN.jar http://courses.csail.mit.edu/6.825/proj2/new/BN.jar

The MATLAB Implementation: We also implemented the aforementioned algorithms in MATLAB. We mimicked the external interface of Kevin Murphy's Bayes Net Toolkit[2] so that we could take advantage of Ken Shan's BIF2BNT converter program to translate the input files from BIF format into MATLAB script drivers[3]. Specifically, we implemented our own versions of `mk_bnet` and `tabular_CPD`, allowing us to use and query any BNT-compatible network. However, because the provided code used a different set of probabilities, we also wrote code to change the probabilities in the network to those in the provided code[4]. Knowing only the meaning of the input parameters to Murphy's BNT, we implemented our own suite of Bayesian inference tools. The same script files for BNT are runnable with our toolkit. The MATLAB implementation proved faster and easier to develop and is much easier to read and understand. With the added benefit of immediate access to plotting functionality, MATLAB was the superior choice and should be recommended to students in the future.

1 Variable Elimination

To test our implementation of variable elimination, we verified the implementation return the correct distributions for known queries. Three of these queries were provided in the problem set handout. The results of these queries appear below[5]:

$$P(B|J=t, M=t) = \langle 0.7158, 0.2842 \rangle$$
$$P(E|J=t, B=t) = \langle 0.9980, 0.0020 \rangle$$

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

PropCost	Probability
1000	0.4572
10000	0.3427
100000	0.1730
1000000	0.0271

These numbers match those in the handout, and both implementations agree on the values. See Appendix B for the variable elimination code and Appendix B(a) for the code used to generate these results.

2 Exact Inference

2(a) Insurance

We next investigated changing the above queries on the Insurance network. In particular, we were interested in the effect of knowing that the insured car was sporty and the effect of knowing that the insured driver was a good student. By common sense, the expected value of PropCost should increase conditioned on the car being sporty and knowing that the driver is a good student should result in a cheaper property cost. However, the following query results show that only the former intuition is consistent with the calculated results:

PropCost	SportsCar	GoodStudent
1000	0.4513	0.4588
10000	0.3459	0.3277
100000	0.1718	0.1837
1000000	0.0309	0.0297

The table above shows the conditional probability of PropCost, over 4 possible domain values ($1,000, $10,000, $100,000, $1,000,000).

These queries extend the Insurance query from the previous section by conditioning on the evidence *MakeModel* = *SportsCar* and *GoodStudent* = *True*.

See Appendix C for the code that produced these query results.

To interpret these changes, we calculated the expected value of PropCost for each of the three cases. Given that the vehicle is a sports car, the expected value of PropCost increases 7.68% (from $48,284 to $51,990). The changes caused by knowing that the vehicle is a sports car are caused by the effect on CarValue and RiskAversion. We intuitively expect the PropCost to increase because we expect that a sports car is more expensive than the average car, and that a sports car driver is less risk averse than an average driver. The RiskAversion then affects a broad array of nodes, including the quality of driving, all of which contribute to a larger PropCost.

how? Did you take the min of each range, i.e. 1,000 for ~~"Thou~~ "Thousands"? That's a choice to be justified.

Given that the driver is a good student, the expected value increases by 7.29% to $51,806. Our intuitive expectation of the impact of the driver being a good student is incorrect as it ignores some secondary effects of knowing that a student is a good student.

[2] http://www.ai.mit.edu/~murphyk/Software/BNT/bnt.html

[3] http://www.digitas.harvard.edu/~ken/bif2bnt/

[4] Tomas Izo provided the list of assignments necessary to automate this process

[5] We will use angle brackets ⟨, ⟩ to denote binary distributions where the first element is the probability of *false*.

Based on the GoodStudent conditional probability table, an upper-middle or wealthy adolescent is much more likely to be a good student than a middle or proletariat adolescent. Therefore, being a good student increases the probability that one is in the upper-middle or wealthy socio-economic class. GoodStudent is connected to RiskAversion through SocioEcon (Age is in the evidence and is, therefore, unaffected by GoodStudent). Based on the RiskAversion conditional probability table, if an adolescent is upper-middle class or wealthy, that adolescent is less risk averse. Therefore, a GoodStudent is actually more risk prone in this situation. In addition, a wealthier driver is more likely to have a expensive car. Together, these effects contribute to an increase in PropCost.

2(b) Carpo

We next tested our variable elimination algorithm by calculating two queries on the Carpo network. The Carpo network encodes the causal relationships between diseases and symptoms. This is a "cleansed" network, meaning that the actual variable names and domain value names have been renamed to arbitrary symbols. As such, there is little intuitive understanding of the network, over and above topological observations.

The following two queries probably represent a posterior probability query of two diseases given a set of symptoms:

$$P(N112|N64 = 3, N113 = 1, N116 = 0) = \langle 0.9880, 0.0120\rangle$$

$$P(N143|N146 = 1, N116 = 0, N121 = 1) = \langle 0.9000, 0.1000\rangle$$

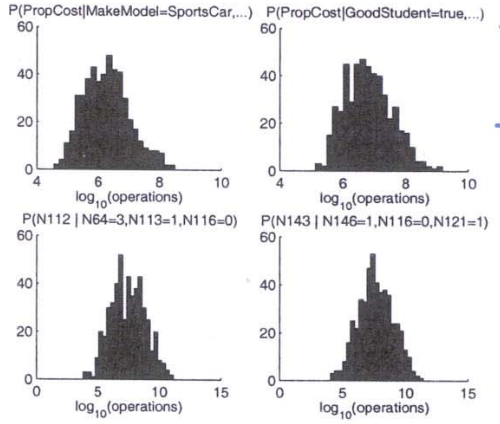
We computed the above results using both implementations. The results from both implementation were identical within the rounding error of double precision.

3 Elimination Order

An important question concerning any exact inference algorithm is its computational complexity. Conditional variable elimination is P#-hard, which is believed to be even harder than NP-hard. However, theoretical complexity and actual complexity in the field are sometimes uncorrelated. For example, most random 3SAT problems are "easy" and, therefore, its NP-complete complexity is less of a problem. In this section, we looked at the effect of elimination order on the time complexity of variable elimination. We conducted a series of 500 experimental runs of variable elimination using random elimination orders for each query. As such an experiment would be impossible if the algorithms actually calculated and stored all of the factors, we modified the algorithm to keep only a running tally of the number of addition and multiplications executed during the factor computations. The return value of this variable elimination algorithm is now the total number of operations. It is easy to see that this *fake* implementation can be implemented in polynomial time, making the above experiment tractable. Figure 1 contains four histograms, one for each of the four queries of problem 2. See Appendix H for the code used to collect these results.

strong word.

clever!



This is quite nice - great idea!

Figure 1: Complexity of Random Elimination Orders

Looking at these graphs, we see a very large difference in running time between different runs of each query. Shorter runs require roughly 10^4 to 10^5 operations while longer runs take nearly $10^{12} = 1,000,000,000,000$ operations. Of the two Insurance queries, the SportsCar query seems to be, on the average, easier, with most of the weight of the distribution around 10^6, while the GoodStudent query is centered around 10^7. Reading directly from the graph, a random GoodStudent-conditioned query is roughly 10 times slower than a MakeModel-conditioned (SportsCar) query.

It is also interesting to note that the Carpo queries are much more complex than the Insurance queries given a random order. The Carpo queries range from

why might this be? Offer interpretation.

hundreds of thousands of operations to nearly a trillion operations.

Interesting

Finally, these distributions could be well modelled as Gaussian (or perhaps betas, as the distributions are discrete and bounded). Perhaps a good measure of the complexity of a network would be the mean and variance of the distribution of the number of operations required by variable elimination given random elimination orders.

We also ran timing results for random orders. As can be expected by inspecting Figure 1, many of the timing trials ran out of memory (or if memory was increased, took to long too run). The aggregate results appear in the following table and as a histogram in Appendix I:

	SportsCar	GoodStudent	N112	N143
Time (ms)	769M	903M	865M	903M
Operations	6.5M	6.2M	43M	93M

4 Greedy Elimination Order

Because random elimination orders can result in impossibly long run times, a easy way to generate good elimination orders is essential. One such method is to greedily choose the next node to eliminate such that the resulting factor sizes are minimized. There are several measures of size. One is simply the number of factors, another is the size of the scopes of factors, and yet another is the total size of the domains (which takes into consideration that domain size of each node). n

We entertained a variety of these, including: factor scope size, number of multiplications, factor size. Initially, we tried simply minimizing the scope at each iteration: fewer variables generally results in fewer multiplications. However, this method ignores the sizes of the domains and gave worse results than the other heuristics. We also tried using the number of multiplications required to compute the next factor as a metric. This metric was better on the Carpo network but worse on the Insurance network. The reason may be that, in some cases, choosing fewer multiplications at a particular step resulted in much larger factors being generated down the road.

very nice

We finally settled on choosing the elimination order with the smallest factor size, using the number of multiplications as a tie breaker. This metric proved to be better in all situations. We ran our algorithm for 50 trials each and reported the mean run times[6]. The results of these trials appear as Figure 2. The first column of each set reflects the baseline greedy algorithm based on the resulting factor's scope size. The second column reflects the multiplication metric. The third column reflects the greedy factor size metric with the number of multiplications as a tie breaker.

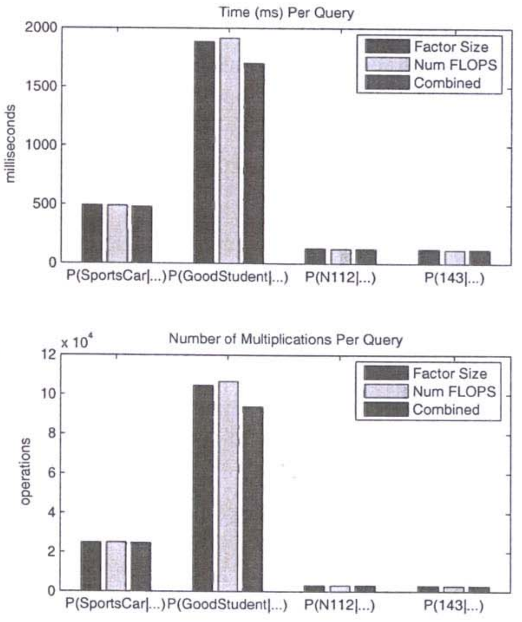


Figure 2: Timing/Complexity Results using a Greedy Heuristic with Variable Elimination in Java

We also implemented the factor-size-based heuristic in MATLAB. We used the *fake* implementation from the previous problem to count the number of operations required (including those necessary to select the minimal node to eliminate). The results, which can be compared to those in Figure 1, appear in the following graph:

What's the overhead of computing the order? Is it included in the # of operations.

[6]Because there are ties when considering factor sizes, the run time can be different each time if we choose a random factor to break the tie.

Ah, I was going to ask.

Why do you get different histograms in Java and Matlab?

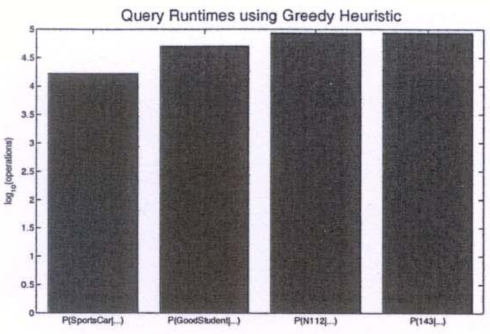


Figure 3: Complexity of Factor-Size Greedy Heuristic

As you can see, these timing results (shown in log base-10) rank amongst the shorter runs represented in the bell curves found in Figure 1. Again, the greedy heuristic works very well, generating efficient elimination orders in these networks. See Appendix H(a) for the driver code that generated these results.

5 Approximate Inference

As can be seen from the previous two sections, exact inference is often expensive, especially when a good elimination order is not known or cannot be cheaply generated. Specifically, it is NP-hard[7]. For the network used in this paper, the complexity was manageable. However, approximate inference is required for densely connected, large networks, due to their intractable complexity[8]. This section and the next discuss approximate inference methods.

5(a) Likelihood Weighting

The first, likelihood weighting, is a variation of direct sampling that better integrates the evidence nodes. Our MATLAB implementation can be found in Appendix E. The following are the result of performing the queries from problems 1 and 2 using this algorithm (see Appendix E(a) for the driver code):

$$P(B|J=t, M=t) = \langle 0.8644, 0.1356\rangle$$
$$P(E|J=t, B=t) = \langle 0.9979, 0.0021\rangle$$

PropCost	Original	SportsCar	GoodStudent
1000	0.4586	0.4409	0.4599
10000	0.3410	0.3518	0.3195
100000	0.1729	0.1779	0.1907
1000000	0.0276	0.0293	0.0299

$$P(N112|N64=3, N113=1, N116=0) = \langle 0.9893, 0.0107\rangle$$
$$P(N143|N146=1, N116=0, N121=1) = \langle 0.9046, 0.0954\rangle$$

The burglary network queries required 3,000 samples. The Insurance queries each required 10,000 samples. Finally, the Carpo queries required 10,000 samples.

define "required"

Overall, the above results match well with the exact values determined by the variable elimination algorithm. Interestingly, the first query on the burglary network is off by a significant amount. However, if we simply increase the samples to 10,000, the query converges through the 2nd decimal point.

5(b) Gibbs Sampling

Gibbs sampling is one of several sampling methods known as Markov Chain Monte Carlo. We implemented a Gibbs sampler (see Appendix F) and ran the above queries (see Appendix F(a) for the driver code that generated these results):

$$P(B|J=t, M=t) = \langle 0.7150, 0.2850\rangle$$
$$P(E|J=t, B=t) = \langle 0.9983, 0.0017\rangle$$

PropCost	Original	SportsCar	GoodStudent
1000	0.4417	0.4790	0.5017
10000	0.3343	0.3424	0.3361
100000	0.1946	0.1527	0.1388
1000000	0.0294	0.0259	0.0234

$$P(N112|N64=3, N113=1, N116=0) = \langle 0.9889, 0.0111\rangle$$
$$P(N143|N146=1, N116=0, N121=1) = \langle 0.9069, 0.0931\rangle$$

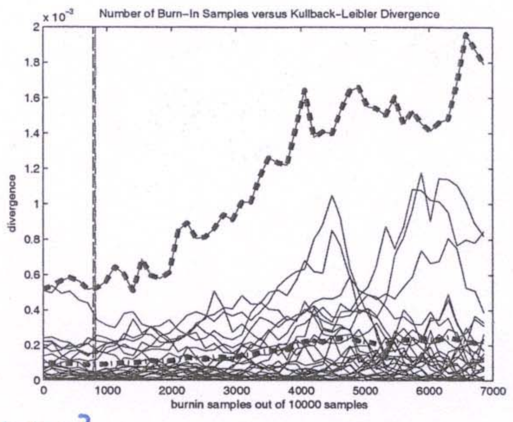
The burglary queries required 3,000 samples. The Insurance and Carpo networks both required 10,000 samples.

The above results match very well with the exact values determined by the variable elimination algorithm. One interesting observation is that Gibbs does better in the Burglary network with the same number of samples, which is to be expected given the limitation of likelihood weighting in the face of "late" evidence. On the larger networks, however, Gibbs does not converge as quickly as likelihood weighting, and therefore, for similar sample sizes, Gibbs results have higher divergence. Section 7 makes this comparison precise by graphing the divergence against the number of samples for both algorithms on all queries.

good.

[7]Cooper, G.F. The computational complexity of probabilistic inference using Bayesian belief networks. Artificial Intelligence 42 (1990), 393-405

[8]Kevin B. Korb, Ann E. Nicholson, Bayesian Artificial Intelligence, 2004



Very nice.

not 7000 ?

Figure 4: *How does burn-in affect Gibbs' accuracy?* These lines represent the KL divergence for every burn-in setting between 0 and 6,000 out of 10,000 samples for a burglary network query. The thick dashed line represents the maximum divergence across all trials, while the dot-dashed line represents the average KL divergence across all trials. The vertical dashed line is the number of samples corresponding to the minimum KL divergence of the maximum, while the dot-dashed vertical line is the number of samples that corresponds with the minimum KL divergence of the average.

6 Burn-in

[good job on the analysis here – best I've seen so far.

This section addresses how the Gibbs sampler burn-in setting affects the accuracy of the estimate as measured by the KL divergence metric. To test the affect of burnin, we conducted the following experiment: We ran the Gibbs sampler on the first Insurance network query 30 times, with 10,000 samples per run. Instead of calculating a single distribution, the implementation returns the estimated distribution *at every sample*. These incremental distributions were then compared with the actual distribution using the Kullback-Leibler divergence measure. This method produced graphs that relate the divergence versus the burn-in at the granularity of single samples.

We plotted the divergence versus the number of burn-in samples (Figure 4). These results suggest that a burn-in of roughly 800 samples is the best for this particular query at 10,000 samples. A question arises as to how these numbers change as the network and number of samples change. The optimal burn-in certainly depends on the number of samples. For example, 800 burn-in out of 1,000 total samples may leave too few samples to get an accurate sample count. The best way to choose the burn-in setting may be to use 10% of the total samples (as suggested by these results). The burn-in should be the amount of time it takes the network to "forget" its initial position. Different networks will require different burn-in settings, therefore this result does not necessarily scale (even to other queries in the same network). We ran an identical experiment on the Insurance network and found that 10% was also a good setting. However, it was only slightly better than having no burn-in at all, which undermines the generalization. Ultimately, burn-in is related to the initial random position, the network, the query, the evidence and the total number of samples. We highly suspect that there is no free lunch when it comes to choosing an optimal burn-in setting across queries. See Appendix F(b) for the code that generated these results.

good.

7 Kullback-Leibler Divergence

Assessing approximate inference performance requires a measure of the quality of the estimated distributions. We used the Kullback-Leibler divergence metric, comparing the estimated distribution to the actual distribution computed by variable elimination.

A sampling method is preferred over an exact method when it computes a sufficiently accurate estimate in a shorter period of time than does the exact method. Sampling methods will be preferred, therefore, as the

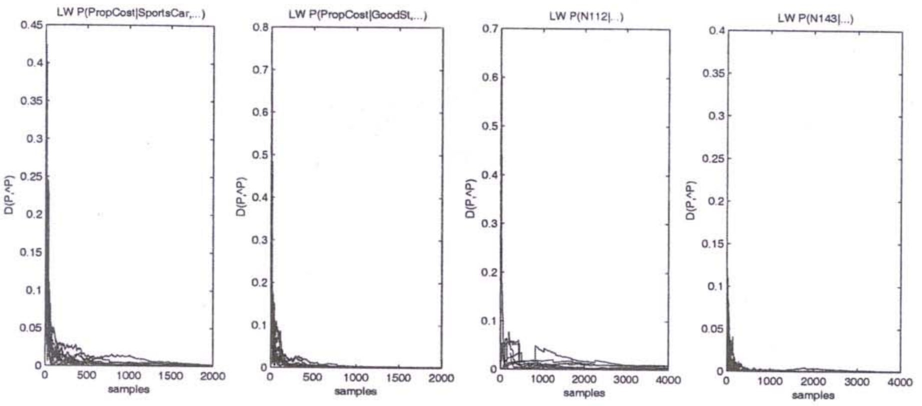


Figure 5: Kullback-Leibler Divergence for Likelihood Weighting

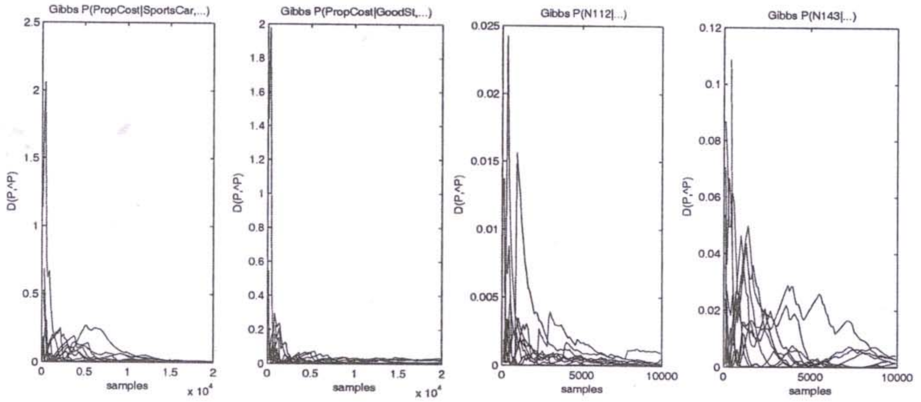


Figure 6: Kullback-Leibler Divergence for Gibbs Sampling

complexity of the graphical model increases. In the case of the networks discussed in this paper, our greedy implementation of variable elimination computes the exact inference extremely fast (between 119 and 1,701 milliseconds). This is equivalent to approximately 900 likelihood weighting samples or 350 Gibbs samples. Neither sampling algorithm converges this quickly. Therefore the exact method is preferred and it is not meaningful to compare the performance of our sampling methods to that of the exact method. However, we will judge the methods against each other in order to get an idea of which method is best for large, dense networks.

Our Gibbs sampler and likelihood weighting implementations (Java and MATLAB) ran at different relative speeds. In other words, the time it takes to do one sample of Gibbs sampling versus one sample of likelihood weighting is implementation dependent. For analysis purposes, we will assume that they are equal and therefore we will treat samples as if they were a fundamental computational unit. Our analysis is based on data collected from our MATLAB implementation.

Preferred Method

The following is an itemization of which sampling method performed the best on each of the four queries:

- Insurance Query #1: Likelihood is Preferred

The Gibbs sampler takes a very long time (approximately 15,000 samples) to converge to a reasonable KL divergence value on this query (Figure 6). One particularly bad run of the first Insurance query maintained a high KL divergence through the 10,000 sample marker. Likelihood weighting, on the other hand, converges quickly on this query (Figure 5). Likelihood weighting achieves a KL divergence value of approximately 0.01, around 1,500 samples.

- Insurance Query #2: Likelihood is Preferred

  Likelihood weighting converges to a respectable KL divergence value in approximately 1,000 samples, on this query (Figure 5). The Gibbs sampler, on the other hand, does a poor job of converging.

- Carpo Query #1: Gibbs Preferred

  The Gibbs sampler converges to a KL divergence of less than 0.025 very quickly (Figure 6). Likelihood weighting converges quickly, on this query, as well, but not as quickly as the Gibbs sampler (Figure 5). This is probably due to the fact that all of the evidence variables are at the lowest level of the network, which hurts the performance of the likelihood weighting method. Late evidence variables whose values are, in general, improbable result in low-weight values and lead to a poor estimation of the probability distribution.

- Carpo Query #2: Likelihood Preferred

  The Gibbs sampler converges on this query to a KL divergence of 0.02 after 7,000 samples(Figure 6). However, likelihood weighting converges to 0.02 with fewer than 1,000 samples (Figure 5). Likelihood sampling is helped by the fact that one of the evidence nodes, N146, has no parents.

Mean Performance

Figure 7, which shows the mean performance of the algorithms across the four queries, is in line with our previous observations. One notable observation from the mean graphs is how the performance of the algorithms vary so greatly. The most significant difference is how well the Gibbs sampler performs on the N112 problem, compared to the other problems.

Additional Observations

Our complexity analysis of these two algorithms reveals that both methods worked well on the Carpo network, but likelihood weighting clearly worked best on the Insurance network. The most apparent reason for Gibbs' relative failure on the Insurance network is the networks *practical* lack of ergodicity despite the last minute attempts to fix this problem by replacing zero probabilities with infinitesimal ones.

- Ergodicity

  While the Insurance network is ergodic, it is only ergodic because we forced any zero probabilities to be a very small number. While this allows the algorithm to converge to a steady state, it may do so *much* more slowly, if it finds itself in certain assignments, particularly those associated with values of 10^{-100}. This would handicap the Gibbs sampling algorithm, since its sampling method depends on the network being ergodic *relative to the number of samples.* With such small values in the network, the number of samples required to converge grows enormously.

- Connectivity

  Another issue is connectivity. The Insurance network is smaller (27 nodes vs. 60 nodes). However, it is also more connected (49 connections for 27 nodes, versus 73 connections for 60 nodes). Consequently, the Carpo network has fewer undirected loops. It is not a poly-tree, which would allow for very efficient inference, but it is less connected and has more singly-connected node pairs. good.

- Likelihood weighting performance is heavily problem dependent

  We can see that the likelihood weighting converged faster for the N143 Carpo query than it did for the N112 Carpo query. As noted above, all of the evidence variables in the N112 query are at the lowest level. This may also be influenced by the fact that N112's only parent (N73) rarely has a value of true, therefore, likelihood weighting continually generates samples with low weight. N143, on the other hand, is a root node and therefore does not have this issue. good.

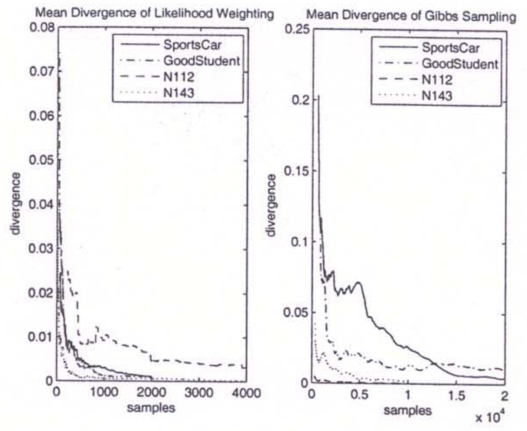


Figure 7: Mean KL Divergence of Likelihood Weighting and Gibbs Sampling

- The "SportsCar" Problem

An interesting observation from this analysis is that the Gibbs sampler is slow to converge on the "SportsCar" query. Specifically, the KL value is quite high until 5,000 samples. Likelihood weighting, on the other hand, does not. This is a very pronounced difference. Gibbs may fail to converge because of the small probabilities in the conditional probability tables.

Interesting

However, there may be something more insidious at work such as strongly correlated variables making the Gibbs sampler take a long time to generate independent samples[9]. All in all, this a very perplexing phenomenon.

8 Non-systematic Gibbs Sampling

There are many different ways to implement Gibbs Sampling. Typically, all the non-evidence variables are resampled systematically on each iteration of the algorithm. An alternative approach is to randomly and uniformly select a non-evidence variable to resample. We implemented this alternative in addition to the systematic approach. The following are results using the random method on queries previously performed using the systematic sampler:

$$P(B|J=t, M=t) = \langle 0.7420, 0.2580\rangle$$
$$P(E|J=t, B=t) = \langle 0.9982, 0.0018\rangle$$

PropCost	Original	SportsCar	GoodStudent
1000	0.4421	0.4879	0.3643
10000	0.3428	0.3874	0.3489
100000	0.1879	0.1124	0.2472
1000000	0.0272	0.0213	0.0395

$$P(N112|N64=3, N113=1, N116=0) = \langle 0.9775, 0.0225\rangle$$

$$P(N143|N146=1, N116=0, N121=1) = \langle 0.8914, 0.1086\rangle$$

The burglary results required 12,000 samples, with 2,000 burn-in samples. This is the same total number of samples as in section 5. The Insurance and Carpo queries required 60,000 samples and 5,000 burn-in samples per trial. This is far less than the 600,000 total resampling iterations required in the systematic trials.

The number of samples, however, must be balanced with the quality of the results. On the small burglary network, the Non-Systematic Gibbs Sampler generated a less accurate estimate, on average, than the Systematic Gibbs Sampler, for the same number of sample iterations.

The accuracy of the Non-Systematic Gibbs Sampler on the larger Insurance and Carpo networks, however, was similar on average to that of the Systematic Gibbs Sampler with far fewer samples. Of the five different queries which we performed on these networks, 3 yielded very similar results, one yielded better results, and one yielded worse results.

[9]David J. C. MacKay, *Information Theory, Inference and Learning Algorithms* (2003), p.371

When we put this into context, it is clear that the random implementation is a much more efficient estimator (for these networks), than the Systematic Gibbs Sampler. Cool.

Conclusion

We have discussed the relative merits of likelihood weighting and Gibbs sampling as approximate inference methods by analyzing their convergence in an empirical setting using the KL divergence metric. In networks with late evidence, likelihood weighting methods required more samples for results as accurate as those obtained by Gibbs sampling with the same number of samples. While approximate methods are usually faster on intractable networks, the networks we investigated were small enough that the exact methods were fastest.

We also tested a variety of heuristics that chose elimination orders for the variable elimination algorithm. We devised a factor size based heuristic using a multiplication-count tie breaker and showed that it outperformed the other heuristics on our examples.

Finally, our dual implementation on Java and MATLAB confirm that MATLAB should be used when experimenting with mathematical algorithms and performing analysis.

Great job!!

Correctness :	30
Experiments :	30
Interpretation :	30
Presentation :	10