CS391L Machine Learning: HW4 Approximate Inference in Bayesian Networks

Joel Iventosch UTEID: jwi245 Email: joeliven@gmail.com

April 11, 2015

1 Introduction

The goal of this assignment was to learn about Bayesian Networsk (referred to as Bayes Nets or BNs moving forward) and various methods for computing approximate inferences on BNs. BNs are a very powerful type of graphical model that can be used to model many real world phenomenon. One of the things that makes BNs excpetionally powerful as a modeling tool is that they can encode causal relationships between random variables. Because BNs are tranformed from Directed Acyclic Graphs (DAGs), they naturally encode the depenence relationships between the various variables (i.e. nodes in the DAG).It is due to the directed nature of the links in a DAG that BNs are able to eloquently model these causal relationships. Additionally, BNs are generative models, meaning that we can easily produce samples from the BN's joint (or particular conditional) distributions encoded by the model. This is a key feature of BNs for several reasons: it allows us to learn from incomplete data, it allows us to learn optimal (or near optimal) structure of the DAG representing the BN, and it enables the computation of approximate inferences. While the first two benefits of the generative nature of BNs are very important in and of themselves, the third is the subject of this homework assignment and thus what the remainder of this report will be focused on.

Our specific task in this homework assignment was to create a BN and then perform approximate inference on our BN by implementing rejection sampling and Gibbs sampling methods. Specifically, we were to generate random conditional queries of our BN, and then to compute the corresponding conditional probability of that query using both approximate inference methodologies. (Reference note: several parts of this introductory paragraph are paraphrased from class lecture, notes, and the homework assignment handout, etc.)

2 Methodology

My mindset going into this assignment was that I really wanted my BN to be modelled off a set of real-world data and to represent something concrete, as opposed to simply using randomly generated data. As such, I hand-designed a DAG that attempted to model various aspects of my Dad's life. As a sidenote, I am very interested in personality research and hope to work towards creating improved (and eventually hopefully accurate!) models of human personality in my own research. Accordingly, it seemed like an appropriatie choice for this assignment. The DAG that my BN is modelled on can be seen in the two figures below. One is the original DAG I designed (in power point) showing the nodes and connectivity of the model. The other is a somewhat refined version (although it has the exact same nodes and links) that I generated in a Bayesian Net software program from AIspace.org (as a reference note, one of my colleagues pointed me in the direction of this software). The next step in my methodology, was to fill in the conditoinal probability tables for each node in the BN. As is evident from the figures below, this BN has a fairly high level of connectivity (particulalry to node 9, the "good mood/bad mood" node, which I was especially interested in observing the behavior of), and as such, filling out the Conditional Probability Tables (CPTs) was not an easy task. Although I could have randomly generated the data for these CPTs that would have somewhat nullified the entire point of using a 'real world model'. So the next step was to estimate what I thought to be the true probabilities for each entry in each CPT. Next, I went about creating an object-oriented implementation of a BN in python. I craeted a **BayesianNetwork** class and a **Node** class. Then I converted my DAG into an actual BN in code in Python using my newly created BayesianNetwork and Node classes. I was considering using an exisitng BN toolkit, but wasn't sure if that would (a) be permitted for this assignment and (b) I figured it would be a good experience implementing this type of object-oriented model on my own anyways (and it certainly proved to be informative and a good learning experience, albeit a bit of a time consuming challenge as well).

The next step in my methodology was to implement the two actual inference algorithms. My methodology for implementing the two inference algorithms for this assignment was as follows:

Rejection Sampling

1. Parse the conditional probability query
2. Initialize both the numerator and denominator to 0
3. Generate a sample from the joint probability of the entire BN (using ancestral sampling)
4. If sample agrees with (at least) the evidence being conditioned upon, then keep it and $denominator++$
5. If sample also agrees with the nodes of interest, then $numerator++$
6. Repeat until denominator is equal to or greater than desired number of samples
7. Return $numerator/denominator$ as the conditional probability

Although very straight forward to implement, the downside to this algorithm is that many samples must be rejected, particularly if the evidence being conditioned upon is extensive. The upside, however, is that it produces fairly accurate results and is computationally tolerable if the BN is small, not highly connected, or the evidence being conditioned upon is not extensive.

Gibbs Sampling

1. Parse the conditional probability query
2. Initialize both the numerator and denominator to 0
3. Generate an initial sample, whose nodes corresponding to the evidence nodes are assigned to the values of the evidence nodes being conditioned upon; randomly assign True or False values to the remianing nodes
4. At each time step randomly select one of the non-evidence nodes to update (evidence nodes are never updated...although this skews the distribution, which is corrected for in the next step, it also clamps the evidence set, enabling us to consider every sample instead of having to reject most samples)
5. Compute the conditional probability of the update node being true, when conditioned on all other nodes in the BN (note, this step proved to be of significant computational expense - at least in my implementaiton of the algorithm...I attempted to implement the Markov Blanket approach to the calculation, but could not seem to get the probabilities to come out correctly, so I ended up just using the brute force calculation method)
6. randomly generate a number from the uniform distirbution between 0 and 1
7. if this number is less than the conditoinal probability of the update node (conditioned on all other nodes), then update the sample to reflect that this update node is set to True; if the randomly generated number from the uniform is

greater than the conditoinal probability of the update node, set the update node to False in the sample

8. If implementing thinning (which I did in some, but not all of my experiments), then retain this sample in the updating process, but do not factor into the computation of the conditional probability query if the overall count (e.g. iteration number) mod *thinning* variable does not equal zero (note, that this adds significant expense to the Gibbs algorithm, but it also helps improve accuracy significantly, as it decorrelates the data...which is inherently somewhat correlated due to the Markovian nature of the Gibbs sampling algorithm...the added expense of thinning scales linearly with the data, which is typically a good thing - however given the small size of the networks we were dealing with for this assignment, I believe it ended up having a substantial impact)

9. If keeping the sample, $denominator++$

10. If keeping the sample, AND it agrees with the nodes of interests (it will always agree with the nodes being conditioned upon since the are clamped), then $numerator++$

11. Retain this sample as the starting sample in the next iteration of the algorithm

12. Repeat until either the denominator reaches a desired number of samples or a convergence criteria is met (I tried both in my experiments and seemed to have much more accurate results not using convergence, because the algorithm seemed to converge too quickly at times on an inaccurate value)

13. Return $numerator/denominator$ as the conditional probability

I found the Gibbs sampling algorithm to be significantly more challenging to implement than the rejection sampling algorithm, primarily due to the computation required in the update step, in which we must compute the conditional probability of the node being updated, when conditioned upon all other nodes in the BN. I spent a significant amount of time and effort trying to get the Markov Blanket implementation of this step correct, so as to improve the efficiency of the algorithm (see code for implementation details). However, when running experiments, my conditional probability inferences comuted by the Gibbs method were coming out significantly off when using the Markov Blanket implementation of the algorithm. As such, I adjusted to using the brute force calculation, which ultimatley worked and produced fairly accurate resutlts, but at a substantial computational cost.

The final step in my methodology was to figure out a way to compute the true probabilities for each randomly generated conditional probability query (for the sake of measuring the accuracy of the approximate inference results using the two sampling algorithms). This proved to be much more challenging than I'd anticipated. Conceptually the computation is easy enough (albeit expensive - whether doing it by hand or code), however I was unsure how to code it up, given the variable nature of the queries. In other words, each randomly generted query could include as few as 2 of the nodes in the BN, or as many 10 (all of them). As such, when computing the true conditional probability, there will be a variable number of summations involved in order to marginalize out the nodes not involved in the query (and then again when marginalizing out the nodes in the numerator, but note denominator to compute the denominator value). This variability means though that any code to implement it (at least that I know how to write) would involve a variable number of nested for loops to perform the marginalization. I tried to implement this in several different ways (including recursively), but ultimately was unable to arrive at a solution. (As a sidenote, I'm sure that there must be a way to do that - as it seems that there is a substantial amount of existing software that does just that...I would be very curious as to how others have gone about implementing this and solving this type of coding problem in general...). Whatever the case, ultimatley I resorted to using the AIspace.org software that I'd downloaded to compute the true probabilities (although I suppose they might also use approximate inference in there code once you get "under the hood" so to speak...nonetheless I would assume that their implementation is fairly accurate either way, and is thus a resonable enough comparator to use for measure the error of my implmentation). Even with the aid of this software, however, computing the true conditional probabilities was fairly time consuming (they didn't allow for direct computation of the conditional, so I had to compute multiple joints and divide out by hand). As such, I had to limit my experiments to only considering 5 different randomly generated conditional queries.

3 Experiments and Results

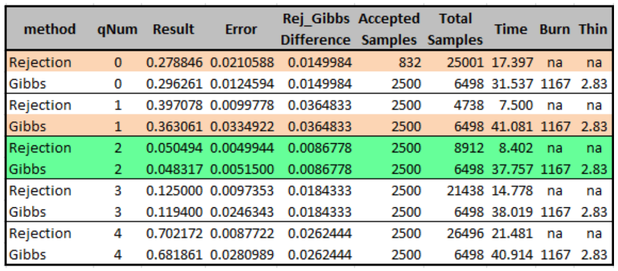
To test the accuracy of my implementation of Rejection sampling and Gibbs sampling as approximate inference methods for BNs, I conducted numerous experiements against many different conitional probability queries. Although I ran many tests and on many different queries throughout the coding and modelling process, I ultimatley ran 5 well-defined experiments, each against 5 different conditional queries (the 5 for whicih I had the true conditional probability values for to compare to). The results of these experiments can be seen in the summarizing chart below. The 5 conditional probability queries that I ran my experiments on can be seen in one of the below tables as well. The full spreadsheet with all of my detailed experimental results information (e.g. burn in rates, minimum sample number, and other parameters) is included as a csv file in the zip file along with my code.

A few intereting observations about my experiments were:

- Overall, Rejection sampling was slightly more accurate: $RejectionAvgErr = 0.0109077$ compared to $GibbsAvgErr = 0.0207669$
- On average Rejection sampling was faster than Gibbs: $RejectionAvgTime = 13.9sec$ compared to $GibbsAvgTime = 37.8sec$. This result was somewhat surprising given that Gibbs sampling is theoretically supposed to be much more efficient than Rejection sampling. I think severals factors account for this apparent discrepancy, including: inclusion of burn in time when measurig Gibbs time; use of brute force algorithm in Gibbs computation step; use of thinning in some Gibbs experiments; and probably somewhat suboptimal implementation of the Gibbs algorithm from an efficiency standpoint (use of data stuructures, variables, etc.). I think my implementation is accurate from an error standpoint, just not optimal from a speed standpoint.
- When we conmpare average sample efficiency (i.e. samples taken versus accepted) Gibbs performs much better, relatively speaking: $RejectionAvgSampleEfficiency = 0.1251$ compared to $GibbsAvgSampleEfficiency = 0.07694$. I think that this result tends to support the notion that my Gibbs implemenation was fairly accurate, but a little slow, although this is mititgated by the fact that the "thinned" samples in my Gibbs implementation were still factored into the sample efficiency calculation (although the burn in samples were not).
- Rejection sampling and Gibbs sampling seem to be somewhat complementary in the sense that one seems to be more effective for certain types of queries while the other seems to excel in the opposite category of queries. Specifically, Rejection sampling really struggles with queries that have a substantial number of conditioning evidence, because it has to reject so many samples just to get a sample that counts in the denominator. Gibbs, on the other hand, excels in this environment from an efficency standpoint, but seemingly at the loss of some accuracy.
- Burning in at least several hundred samples was key for Gibbs to be accurate. Initial samples had very low accuracy (as is evidenced in the charts below, even with a decent burn in period at times) which makes sense due to the random initialization of the first sample and Markovian nature of the samples evolution.
- At time Rejection sampling seemed to gain accuracy with fewer accepted samples than Gibbs

did. In hindsight, this makes sense when analyzing the algorithms, but it was somewhat counter-intuitive at first.

All charts and tables for my experiments can be seen below (and at very bottom of report).



method	qNum	Result	Error	Rej_Gibbs Difference	Accepted Samples	Total Samples	Time	Burn	Thin
Rejection	0	0.278846	0.0210588	0.0149984	832	25001	17.397	na	na
Gibbs	0	0.296261	0.0124594	0.0149984	2500	6498	31.537	1167	2.83
Rejection	1	0.397078	0.0099778	0.0364833	2500	4738	7.500	na	na
Gibbs	1	0.363061	0.0334922	0.0364833	2500	6498	41.081	1167	2.83
Rejection	2	0.050494	0.0049944	0.0086778	2500	8912	8.402	na	na
Gibbs	2	0.048317	0.0051500	0.0086778	2500	6498	37.757	1167	2.83
Rejection	3	0.125000	0.0097353	0.0184333	2500	21438	14.778	na	na
Gibbs	3	0.119400	0.0246343	0.0184333	2500	6498	38.019	1167	2.83
Rejection	4	0.702172	0.0087722	0.0262444	2500	26496	21.481	na	na
Gibbs	4	0.681861	0.0280989	0.0262444	2500	6498	40.914	1167	2.83

Figure 1: Summary Chart of Results

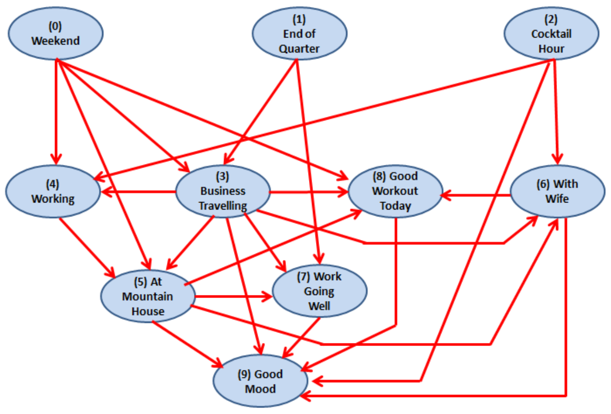
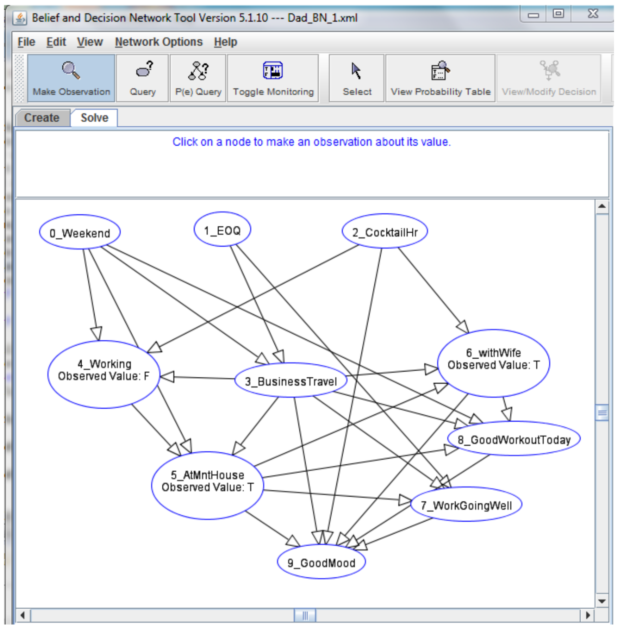


Figure 2: Hand designed Bayesian Network modelling 10 apsects of my Dad's life

4 Discussion

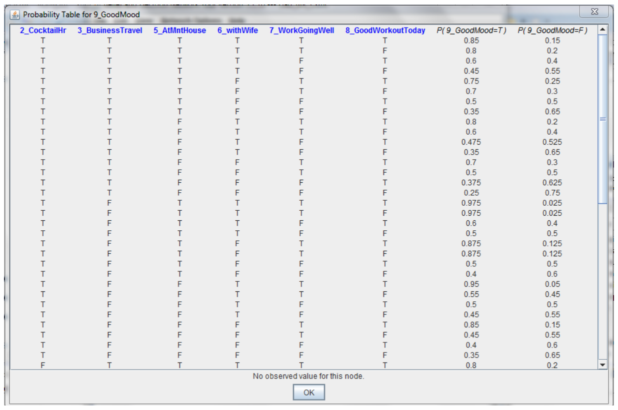
One of the primary goals of this assignment was also to gain a better understanding of why - in principle - Gibbs sampling is much more efficient than rejection sampling. Although my experiemental results were very mixed (likely due to suboptimal tweaking of parameters and/or possible implementation issues), this homework certainly was successful from



Click on a node to make an observation about its value.

Figure 3: Hand designed Bayesian Network modelling 10 apsects of my Dad's life - using AIspace.org software

the standpoint of gaining a better understanding of both Bayesian Networks (in general) and the use of sampling methods as approximate inference algorithms. More specifically, I can now comment with a high degree of conficence on why - in principle - rejection sampling is less efficient than Gibbs sampling. Rejection sampling works by generating many samples from the BNs joint probability distribution, and then computing the ratio of those samples that agree with the full evidence set of the query (both the nodes of interest and the nodes the query is being conditioned upon) over those that agree with the evidence being conditioned upon (and may or may not agree with the rest of the nodes). In other words, rejection sampling computes the frequency with which samples agree with all the evidence compared to when it agrees with the conditioned upon evidence and may or may not agree with the rest of the evidence. However, because the samples accepted in the denominator count must, at a minimum, agree with the evidence being conditioned upon we have to reject all the samples that are generated that don't meet



2_CocktailHr	3_BusinessTravel	5_AtMntHouse	6_withWife	7_WorkGoingWell	8_GoodWorkoutToday	P(9_GoodMood=T)	P(9_GoodMood=F)
T	T	T	T	T	T	0.85	0.15
T	T	T	T	T	F	0.8	0.2
T	T	T	T	F	T	0.6	0.4
T	T	T	T	F	F	0.45	0.55
T	T	T	F	T	T	0.75	0.25
T	T	T	F	T	F	0.7	0.3
T	T	T	F	F	T	0.5	0.5
T	T	T	F	F	F	0.35	0.65
T	T	F	T	T	T	0.8	0.2
T	T	F	T	T	F	0.6	0.4
T	T	F	T	F	T	0.475	0.525
T	T	F	T	F	F	0.35	0.65
T	T	F	F	T	T	0.7	0.3
T	T	F	F	T	F	0.5	0.5
T	T	F	F	F	T	0.375	0.625
T	T	F	F	F	F	0.25	0.75
T	F	T	T	T	T	0.975	0.025
T	F	T	T	T	F	0.975	0.025
T	F	T	T	F	T	0.6	0.4
T	F	T	T	F	F	0.5	0.5
T	F	T	F	T	T	0.875	0.125
T	F	T	F	T	F	0.875	0.125
T	F	T	F	F	T	0.5	0.5
T	F	T	F	F	F	0.4	0.6
T	F	F	T	T	T	0.95	0.05
T	F	F	T	T	F	0.55	0.45
T	F	F	T	F	T	0.5	0.5
T	F	F	T	F	F	0.45	0.55
T	F	F	F	T	T	0.85	0.15
T	F	F	F	T	F	0.45	0.55
T	F	F	F	F	T	0.4	0.6
T	F	F	F	F	F	0.35	0.65
F	T	T	T	T	T	0.8	0.2

No observed value for this node.

Figure 4: Conditional Probability Table from Node 9 of my BN

this minimum crieteria - and since we have no way of knowing whether a given sample will agree with the evidence or not until we've produced it, the result is a significant amount of wasted computational time and resources. Gibbs sampling, on the other hand, utilizes (in prinicple) all of the samples that are generated (due to its design) and thus does not have the wasted cost of computing samples that will be rejected and not included in the calculation of the conditional query anyways. In theory this makes perfect sense. However, in practice I believe there are several mitigating factors that I would argue limit the extent to which Gibbs sampling is actually more efficient than rejection sampling:

- The computational complexity of each sample can be costly
- To get accurate results a significant number of initial samples must be "burned off"
- Thinning the data is helpful for decorrelation, but adds inefficiency

With regard to the first point, the cost of computing $p(x_i|x_{/i})$ can be costly, parituclarly in a BN that has a high level of connectivity. Although this can be mitigated by only computing the $p(x_i|MB(x_i))$ where $MB(x_i)$ is the Markov Blanket of x, the computational cost is not trivial. With regard to the second point, my experiments showed very poor results for Gibbs sampling when I did not use a large enough burn in period. This makes sense since Gibbs begins with a random initialization of each node in the joint distribution, hence you would want to burn off a significant number of samples before beginning to tally the frequency with which a sample agrees with the evidence set, so that the samples are no longer random, but are actually coming from the appropriate conditional distribution. Finally, with regard to the third point, since Gibbs is used in a Markov Chain framework, it seems that it would be beneficial to "thin" the set of samples that Gibbs sampling produces in order to decorrelate the data. However, this adds significant cost to the efficiency of Gibbs sampling as well - particularly because inefficiency associated with thinning is not just constant time overhead, but scales with the number of samples taken. The burn in period, although costly, is constant overhead, but let's say that we thin the data by only considering every 10th (or 100th) Gibbs sample that is taken...the result is obviously inefficiency that scales with our number of samples. In light of all that, at the end of the day I think Gibbs maintains a significant advantage in efficiency over rejction sampling (at least in theory), simply because you don't have to reject samples. If the conditioning set of the query is extensive then this is particularly true, since most samples will be rejected by rejection sampling, and thus not even factored into the denominator count of the conditional query. Although my experiments might not have perfectly demonstrated the above points, the process of working through this assignment has led to a very solid conceptual understanding of the efficiency analysis of Gibbs and rejection sampling.

5 References

1. class lecutures, notes, handouts, etc.
2. AIspace.org

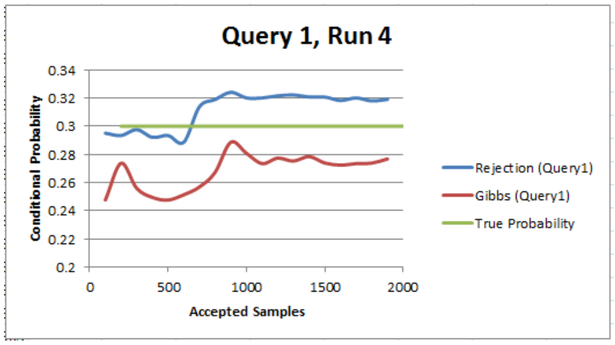


Figure 5: Results from Query 1 of experimental Run 4

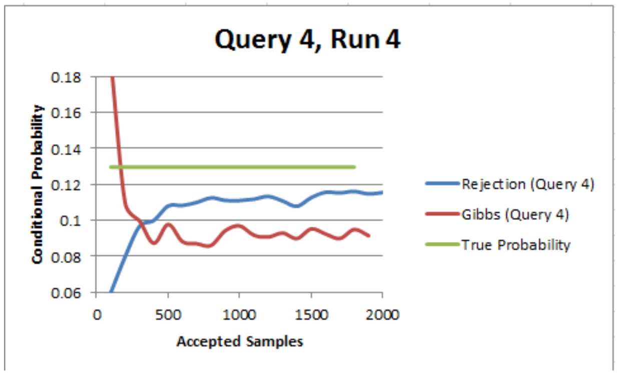


Figure 8: Results from Query 4 of experimental Run 4

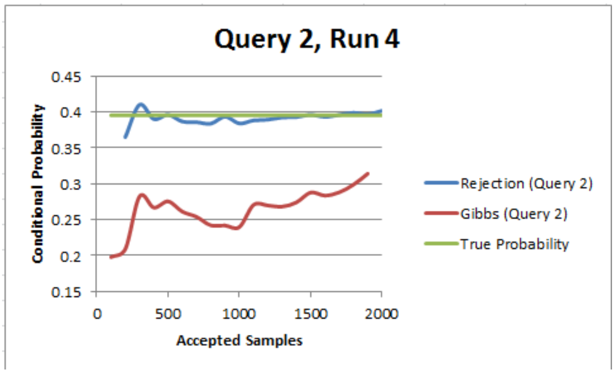


Figure 6: Results from Query 2 of experimental Run 4

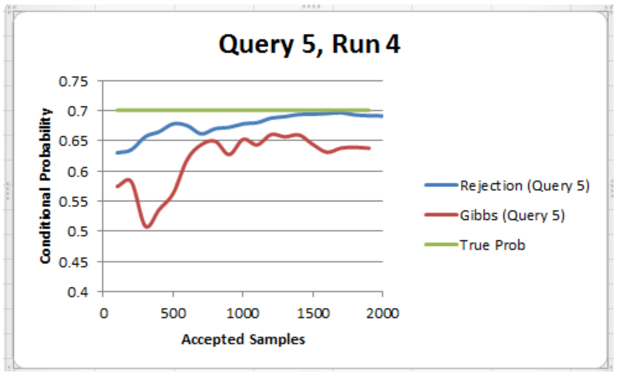


Figure 9: Results from Query 5 of experimental Run 4

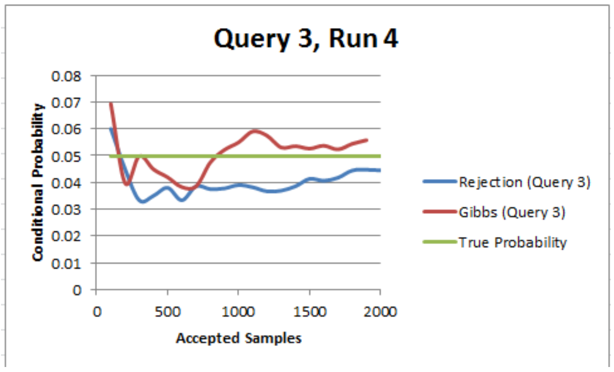


Figure 7: Results from Query 3 of experimental Run 4