- **Email**: chrisbrown@utexas.edu
- **EID**: chb595

1 Bayes Nets

The actual network in this homework was small but not trivial. It had nodes without parents, nodes with multiple children, and nodes with multiple parents. What was simple, though, was that each node produced a boolean value. This means that the graph had just 10 total parameters to be learned.

The goal of the homework was to be able to marginalize out any single node using Gibbs sampling. Even though exact inference is easy and fast, the Gibbs sampling approach here would be applicable in other situations where: 1) the parameters of the original distributions are unknown, or 2) the graph is very large and exact inference is computationally intractable (or simply overkill).

On my very first run-through, I forgot to set the Z_i of my Z vector to the new value on each draw. This was confusing, because the algorithm still produced a decent result more than half the time, and the other times, it was exactly the opposite $(1 - exactInf)$ of what the result should have been. That was an obvious error, though; after fixing it, that behavior disappeared.

A naive first implementation worked surprisingly well; at 1,000 iterations and no burn-in, the results are still pretty good:

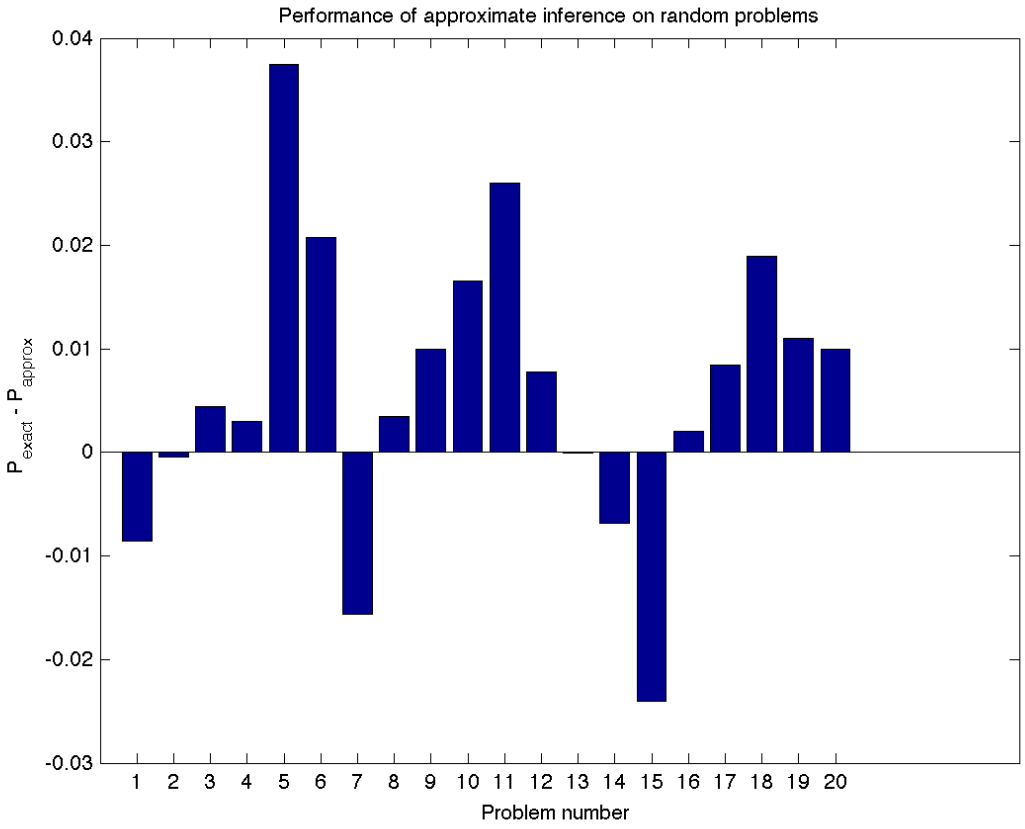

Figure 1: Accuracy (nearness to exact inference result) for 1,000 iterations.

Simply by taking 10x samples, still without a burn-in period, we get much better results:

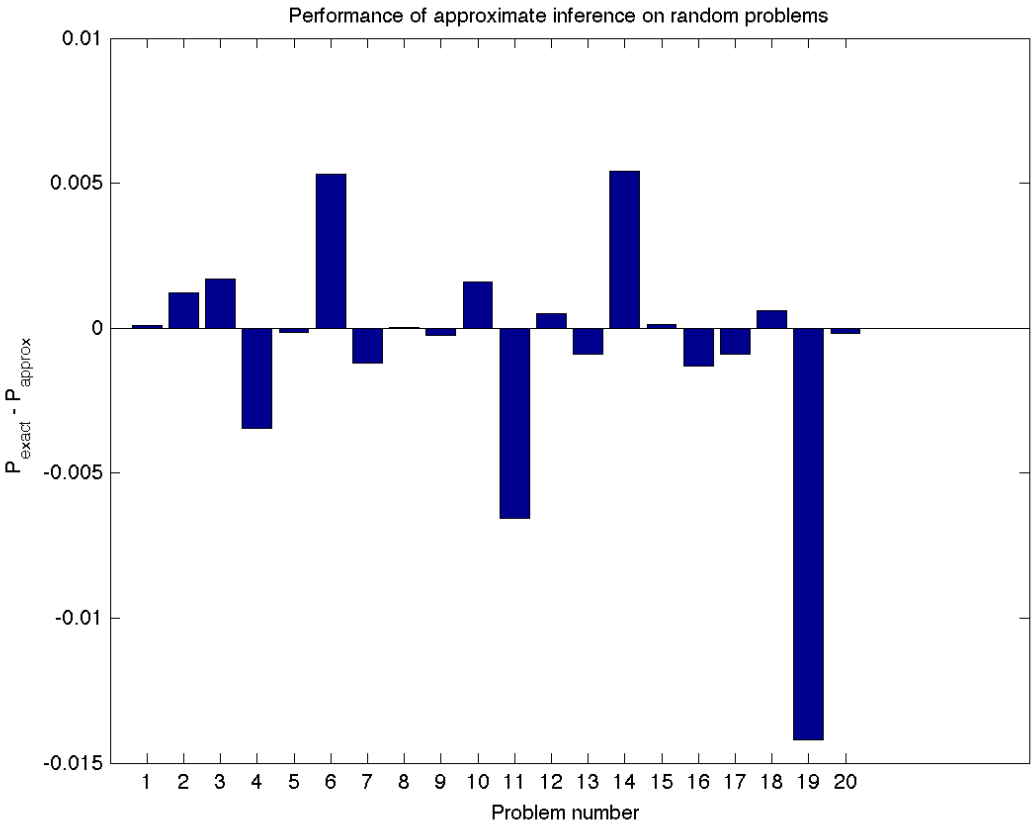

Figure 2: Accuracy (nearness to exact inference result) for 10,000 iterations.

In Figure 2, Sample 19 is a lot further off than we might like. Allowing a burn-in period of 1/10 the total number of iterations gets us to the next plateau of accuracy:

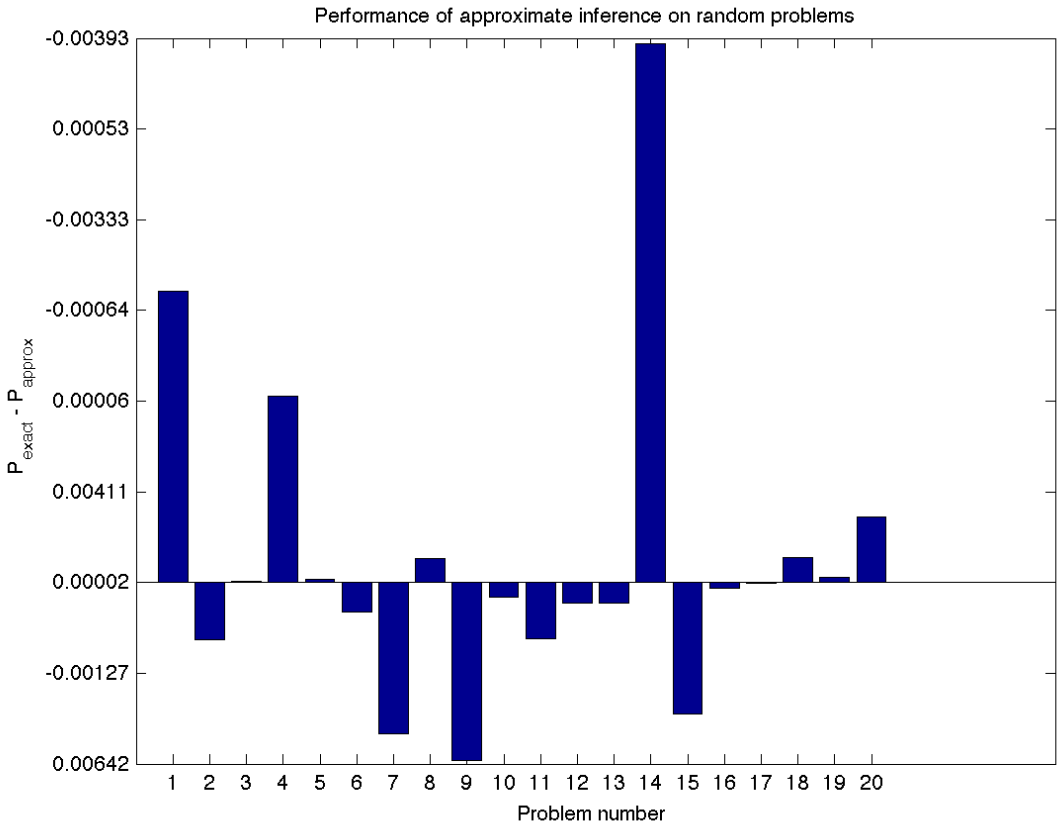

Figure 3: Accuracy (nearness to exact inference result) for 10,000 iterations, allowing for 1,000 iterations of burn-in.

2 Comments

I'm still curious why I didn't have to (or get to) use the intermediate results from samples as part of the sampling process. Just to restate what I asked in my email, and provide a little elaboration: since we're drawing from an artificial distribution, for which we know the exact parameters at every node, there's no apparent reason how we could possibly do better. However, in the MCMC and Gibbs Sampling tutorial (EEB 581 lecture notes[1]), their example implementation of the algorithm shows them updating the priors upon each new draw/sample (p. 17, in Ex. 4, step (ii) and elsewhere), which is straightforward in that case because they are using Binomial and Beta distributions, which have very simple updates.

To do this for this homework, I would have had to extract the functionality of the `computeJoint(...)` function, digging down into the actual CPD's for each draw, and adding my current number of successes and observations to the parameters/priors as needed.

I think that the reason why they do this in the tutorial is in case you're drawing from a real distribution that you don't know the parameters. If you only have a list of observations, if you incrementally build up your version of the parameters at each step, as you're taking on more and more samples, it's sort of like proposing a slightly more specific hypothesis at each step, and then

[1] http://web.mit.edu/~wingated/www/introductions/mcmc-gibbs-intro.pdf

evaluating it against the next swath of data that you process. My intuition is that it helps with preventing overfitting. If the current proposed parameters were not added back into the sampling process as you go along, you may end up with a set of parameters that very accurately describe all the data you've seen so far, post-hoc, but may not be true to the actual distributions. (However, now that I think through it, it seems like the algorithm could get carried away with the proposed parameters, on the other side of the spectrum, so there is probably a completely different and statistically sound explanation somewhere.)