FAST PARALLEL SAME GIBBS SAMPLING ON GENERAL DISCRETE BAYESIAN NETWORKS

Daniel Seita, Haoyu Chen & John Canny
Computer Science Division
University of California, Berkeley
Berkeley, CS 94720, USA
{seita,haoyuchen,canny}@berkeley.edu

ABSTRACT

A fundamental task in machine learning and related fields is to perform inference on Bayesian networks. Since exact inference takes exponential time, it is common to use an approximate algorithm such as Gibbs sampling, but this can still be intractable for graphical models with just a few hundred binary random variables. In this paper, we address this issue by presenting our highly optimized Gibbs sampler, which we believe is the fastest one publicly available. Our Gibbs sampler is GPU-accelerated, heavily parallelized, and replicates data via State Augmented Marginal Estimation (SAME) to decrease convergence time while reaching higher quality parameter estimates. Experiments on both synthetic and real data show that our Gibbs sampler is multiple orders of magnitude faster than the state of the art sampler, JAGS, without sacrificing accuracy. Our ultimate objective is to introduce the Gibbs sampler to researchers in many fields to expand their range of feasible inference problems.

1 INTRODUCTION

In many machine learning applications, one has a distribution $P(X, Z \mid \Theta)$ where X is observed data, Z is hidden (latent) data, and Θ represents the model parameters. The goal is generally to find an optimal Θ with respect to X, while marginalizing out Z. To represent these problems, it is common to use graphical models, which combine probability theory and graph theory to present a robust formalism for probabilistic inference. A Bayesian network is a graphical model defined by a directed acyclic graph and a set of conditional probability tables (CPTs). Each CPT represents a local probability distribution $\Pr(X_i \mid X_{\pi_i})$ where X_i is a random variable, and X_{π_i} represents its set of parent nodes in the graph. We denote the full set of CPTs as Θ.

In this paper, our focus is on parameter estimation of Bayesian networks with discrete random variables ("discrete Bayesian networks") based on partially observed data $\mathcal{D} = \{\xi_1, \ldots, \xi_m\}$, where ξ_i is an n-dimensional vector with assignments to the n variables of the graph, or "N/A" to indicate missing data. We assume that the structure of the Bayesian network — its nodes and edges — is known in advance. This type of problem often arises in practice because it is easier to elicit graph structure from human experts than it is to get numerical parameters (**?**). Furthermore, it is often unrealistic to expect our data $\mathcal{D}$ to be completely observed, as data might be missing due to errors (e.g., human oversights) or deliberate omissions.

Well-known strategies for parameter estimation with partially observed data include Expectation-Maximization (**?**) and variations of gradient ascent (**?**). Parameter estimation using these methods requires running probabilistic inference over the missing data, which tends to be the limiting factor since exact inference using the junction tree algorithm takes exponential time. It is therefore common to use approximate inference procedures. One way is to perform Markov Chain Monte Carlo (MCMC) simulation, where one constructs a Markov chain whose states are an assignment to all unobserved variables, such that the stationary distribution of the chain is the posterior probability over these variables.

Gibbs sampling (**?**) is a special case of MCMC simulation, which at iteration t, goes through each unobserved variable and samples from its full conditional based on the samples from the current

or previous iteration: $X_i^{(t)} \sim \Pr(X_i^{(t)} \mid X_1^{(t)}, \ldots, X_{i-1}^{(t)}, X_{i+1}^{(t-1)}, \ldots, X_n^{(t-1)})$. For the parameter estimation problem, one also needs to sample to update Θ from $P(\Theta \mid \mathcal{D})$, the posterior distribution over the complete data data $\mathcal{D}$ ("complete" due to sampling). Since we assume discrete random variables, the samples are multinomial counts, so for Bayesian estimation, one can impose a set of Dirichlet priors on Θ so that the posterior is also a set of Dirichlets.

While Gibbs sampling is commonly used in machine learning, it is a very broad technique that may not be able to match the performance of special-purpose inference algorithms without extensive fine-tuning or using complicated variations (**?**). **Daniel: I am not sure if I am explaining this last sentence well enough. Murphy's book has some explanation, but not much.** In a recent result, **?** showed that by combining the State Augmented Monte Carlo (SAME) technique (**?**) with Gibbs sampling, one can get fast, high quality parameter estimates for discrete graphical models, but they only applied it to two specific models that do not have mutual dependencies between discrete states. In this paper, we build upon that result by presenting a SAME Gibbs sampler for general discrete Bayesian networks. To be precise, the novelty and aspects of our Gibbs sampler is that

- our sampler uses an adjustable SAME parameter to replicate data, causing the Gibbs sampler to converge faster to higher-quality MAP or ML parameter estimates. We believe this is due to reduction of excess variance from standard Gibbs sampling.
- we run our sampler on state of the art GPUs and parallelize it as much as possible. Our sampler can ultimately scale up to problems with hundreds of random variables.
- the sampler is designed to maximize throughput on only one computer, thus avoiding the need to work though complicated distributed systems.
- it is open source as part of the BIDMach library[1] and comes with various diagnostic tools. Furthermore, BIDMach's mini-batch updating and matrix caching features mean we could even run our Gibbs sampler on data too large to fit in a computer's RAM.

We benchmark our sampler versus the state of the art Gibbs sampler, JAGS (**?**), and show that our Gibbs sampler is multiple orders of magnitude faster. Consequently, in addition to introducing our Gibbs sampler to researchers, we argue in this paper that Gibbs sampling augmented with SAME can be competitive with or superior to the fastest special purpose inference algorithms for Bayesian inference. **Daniel: my main concern is that we are not talking about these "special purpose inference algorithms." We're just presenting a fast SAME Gibbs sampler. Also, I don't know a good way to incorporate factor graphs here; if I did, then we can cite (?).**

2 Related Work

The problem of Bayesian inference for graphical models is old and well-studied, and for reference, we refer the reader to textbooks (e.g., see **?** for a broad background) and survey papers (e.g., see **?** for a discussion on variational inference as an alternative to MCMC methods). **Daniel: I would like to have a survey paper that actually compares Gibbs sampling versus other methods. I think this paragraph should be expanded, while condensing the introductory section.**

Gibbs sampling is also relatively old and well-studied, but it has recently been getting more attention as the research community explores efforts to improve speed and scalability. The Gibbs sampling research most related to our contribution involves efforts to (1) parallelize Gibbs sampling and (2) achieve high throughput. By exploiting the conditional independence assumptions of Bayesian networks, is possible to have an exact, semi-parallel Gibbs sampler that iterates through color groups of nodes, and within each group, sampling each node independently (**?**), a strategy that we employ in our sampler. It is also possible to relax the sequential nature of the algorithm and approximate Gibbs sampling by limiting global communication (**?**). The databases community has also been able to contribute to speeding up Gibbs sampling by showing how improved system design can lead to higher throughput (**?**).

Recently, the use of Graphics Processing Units (GPUs) has become essential for developing high performance software, as exemplified by popular packages such as Theano for evaluating mathematical expressions with multidimensional arrays (**?**) and CAFFE for neural networks (**?**). The

[1] `https://github.com/BIDData/BIDMach`

Augur probabilistic programming language (**?**), which generates inference code for Bayesian networks, demonstrates the importance of combining GPU code with extra parallelism introduced from exploiting conditional independence assumptions.

The result most directly related to our paper, as briefly mentioned in Section 1, is one that shows how the addition of SAME to a GPU-accelerated Gibbs sampler can be very fast for Latent Dirichlet Allocation and the Chinese Restaurant Process (**?**). In that paper, they explored the application of SAME to graphical model inference on modern hardware, and showed that combining SAME with factored sample representation (or approximation) gives throughput competitive with the fastest symbolic methods, but with potentially better quality. We (non-trivially) extend this result by implementing a more general-purpose Gibbs sampler that can be applied to arbitrary discrete graphical models.

3 Fast Parallel SAME Gibbs Sampling

SAME is a variant of MCMC where one artificially replicates data to create distributions that concentrate themselves on the global modes (**?**). It is an efficient way of performing MAP estimation in high-dimensional spaces when needing to integrate out a large number of variables. Given a distribution $P(X, Z \mid \Theta)$, to estimate the most likely Θ based on the data (X, Z) using SAME, one would define a new joint Q:

$$Q(X, \Theta, Z^{(1)}, \ldots, Z^{(m)}) = \prod_{j=1}^{m} P(X, \Theta, Z^{(j)}) \tag{1}$$

which models m copies of the distribution tied to the same set of parameters Θ, which in our case forms the set of Bayesian network CPTs. This new distribution Q is proportional to a *power* of the original distribution, so $Q(\Theta \mid X) \propto (P(\Theta \mid X))^m$. Thus, it has the same optima, including the global optimum, but its peaks are sharpened (**?**). Note that as m increases, SAME approaches Expectation-Maximization (**?**) since the distribution would peak at the value corresponding to the maximum likelihood estimate.

We argue that SAME is beneficial for Gibbs sampling because it helps to reduce excess variance. It is important, however, not to set the SAME replication factor m too high, because that might result in getting trapped in local maxima in the sharpened distribution. An ideal setup might be to start with a low replication factor and gradually increase it, a technique similar to simulated annealing for gradient descent, because both involve "cooling" the target distribution to sharpen the peaks.

Daniel: I am really confused about what I should include for this section. I feel like this just repeats a lot of the SAME description. It might be necessary to do that, but do we have an idea on what we want to discuss?

4 Implementation of SAME Gibbs Sampling

Our Gibbs sampler is implemented and integrated as part of the open-source BIDMach library (**?**) for machine learning. Figure 1 shows a visualization of how it works on real data. Our sampler expects a (usually sparse) data matrix, with rows representing variables and columns representing cases. BIDMach divides data into same-sized "mini-batches" and iterates through them to update parameters. Going through all mini-batches is one full pass over the data.

Our main contribution is introducing a Gibbs sampler augmented with SAME sampling. Consequently, if m is the SAME parameter, for each mini-batch our sampler forms m copies of the known data. Then, it performs Gibbs sampling to fill in the unknown values in each copy of the mini-batch using the same set of CPTs. These sampled results are combined with an adjustable Dirichlet prior and (if desired) the current CPT[2] to form a set of discrete counts that we then sample from to estimate the updated CPT. To preserve samples across all runs, each time the sampler processes a mini-batch, it stores the sampled results in memory. Then, during the next *pass* over the data (i.e., after having gone through all the batches) it starts the Gibbs sampling process from that stored data.

[2]This would be useful if one wanted to do a moving average update of the CPT. Also, the use of the Dirichlet prior tends to be more important when the data is sparse, since it "smooths" the CPTs.

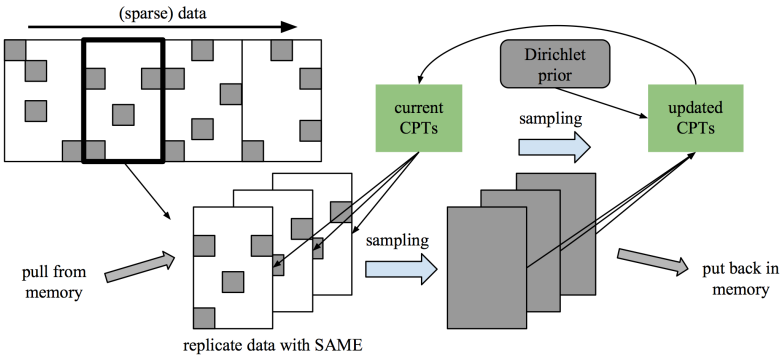

Figure 1: This visualizes our Gibbs sampler at work. The original data is split into four mini-batches of equal sizes (for caching purposes), with each having some known data (shaded gray) and unknown data (white). For each batch, the sampler replicates the data with SAME $m = 3$, samples the unknown values, then uses those to update the CPTs. The resulting samples are stored in memory and are used as the starting point for when we sample this batch again.

It is possible to skip the first few samples (the standard *burn-in* period) and also to only update the parameters every N^{th} iteration to reduce correlated samples.

There are many ways in which we optimize the sampler to maximize throughput on one computer and to avoid excess memory allocation. First, our sampler is GPU accelerated and takes advantage of parallelism in modern hardware by implementing the sampling process using matrix operations. Since memory is scarce on GPUs (3-12 GB is typical), we use a matrix caching strategy to reuse memory for matrices of the same dimensions. This is why BIDMach works with same-sized mini-batches, since the batches themselves, along with all other matrices used as part of the sampling computation, are of fixed size and therefore BIDMach can re-use their memory slots[3]. Even with caching, default mini-batch sizes might be too large, but since BIDMach is highly customizable, one can reduce the number of instances for each batch (i.e., columns in the data matrix) to reduce the memory footprint. In fact, this step is often necessary to use a very high SAME parameter.

As mentioned in Section 2, we further parallelize Gibbs sampling in an exact manner via chromatic sampling. In Bayesian networks, we konw that $u, v \in \mathcal{V}$ are independent conditioning on a set of variables $\mathcal{C}$ if $\mathcal{C}$ includes at least one variable on every path connecting u and v in $\tilde{\mathcal{G}}$, which is the *moralized graph* of the network (i.e., the graph formed by connecting parents and dropping edge orientations). That is, vertex set $\mathcal{C}$ separates the dependency between u and v.

Suppose there is a k-coloring of $\tilde{\mathcal{G}}$ such that each vertex is assigned one of k colors and adjacent vertices have different colors. Denote $\mathcal{V}_c$ the set of variables assigned color c where $1 \leq c \leq k$. One can sample sequentially from $\mathcal{V}_1$ to $\mathcal{V}_k$, where within each color group, it samples all the variables in parallel. This parallel sampler corresponds exactly to the execution of a sequential scan Gibbs sampler for some permutation over the variables and will converge to the desired distribution because variables within one color group are independent to each other given all the other variables. Finding the optimal coloring of a general graph is NP-complete, but efficient heuristics for balanced graph coloring perform well in many problems.

Daniel: TODO Ahhh, I forgot! The stuff in Figure 1 does actually require data to fit in memory. There are ways we can get around this with more efficient storage but we should make that explicit, and say that Figure 1 is only for intuition as most data will fit in memory anyway!

[3]This does not generally apply to the very last mini-batch, but that last batch would not cost too much in extra memory, and one can even ignore it if needed.

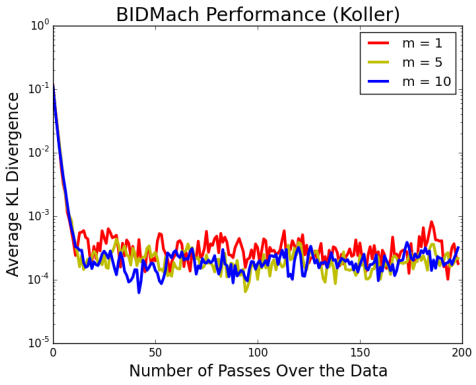

Figure 2: The KL_{avg} from BIDMach.

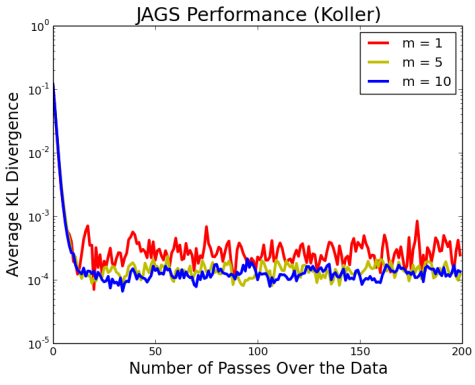

Figure 3: The KL_{avg} from JAGS.

Table 1: BIDMach (CPU) vs. JAGS Runtime on Koller Data

	$m = 1$	$m = 2$	$m = 5$	$m = 10$	$m = 20$
BIDMach Time/Iter	0.055	0.090	0.165	0.311	0.577
JAGS Time/Iter	0.210		1.406	2.675	

5 EXPERIMENTS

We benchmark our Gibbs sampler based on one synthetic and one real dataset. We compare it with JAGS (**?**), which is the most popular and efficient tool for Bayesian inference, and also uses Gibbs sampling as the primary inference algorithm. We could not find another general-purpose software package that substantially out performed JAGS.

For all JAGS experiments, we use a single computer with an Intel Core Xeon processor with eight cores and 2.20 GHz clock speed (E5-2650 Sandy Bridge). The computer also has 64 GB of (CPU) RAM. We run our Gibbs sampler on this same computer when we evaluate the speed of its CPU version, for the fairest comparision with JAGS. These benchmarks would be particularly relevant for those who do not have, or may not wish to use, a GPU.

When we are not directly comparing our sampler's speed with JAGS, or whenever we use its GPU version, we use a second machine. (Note that in terms of pure accuracy, the GPU and CPU versions are roughly equivalent.) This one has an Intel Core i7 processor with four cores and 3.40 GHz clock speed. It has a single NVIDIA GTX Titan X GPU with about 12 GB of RAM. It has 16 GB of CPU RAM. This computer is newer, but its processor is workstation rather than server grade, and has half the cores. We also noticed from our experiments that the CPU-only version of our Gibbs sampler seemed to run at the same speed on both computers.

We emphasize that the use of one computer at a time is deliberate and a strength of our sampler, as it is cheaper and simpler to run as compared to a computing cluster.

5.1 SYNTHETICALLY GENERATED "KOLLER" DATA

Daniel: change all figures to have (Koller) at the end! Also, refer to this as the Koller data! I.e., figures should have the title: XXX (Koller) or XXX (MOOC).

We first use synthetic data generated from a toy Bayesian network to show an example of correctness and the benefits of SAME. The network consists of five variables: $X_0 = \text{Intelligence}$, $X_1 = \text{Difficulty}$, $X_2 = \text{SAT}$, $X_3 = \text{Grade}$, and $X_4 = \text{Letter}$. The directed edges are $\mathcal{E} = \{(X_0, X_2), (X_0, X_3), (X_1, X_3), (X_3, X_4)\}$, where (X_i, X_j) means an arrow points from X_i to X_j. Variable X_3 is ternary, and all others are binary. The goal with this network is to model a student taking a class, considering ability metrics (Intelligence and SAT score), the class difficulty, and the student's resulting grade, which subsequently affects the quality of a letter of recommenda-

Table 2: BIDMach (CPU) vs. JAGS Runtime on (Replicated) Real MOOC Data

	1x	2x	5x	10x	20x	40x
BIDMach Time(sec)/Iter	0.182	0.375	0.931	1.792	3.501	7.181
JAGS Time(sec)/Iter			29.150	94.075		OOM

tion. All this is from Chapter 3 of **?**, which also lists the true set of CPTs. Due to the name of the first author, we label this data "Koller" data to distinguish it from the MOOC data we use in (5.2).

To generate the data for BIDMach, we listed the variables in a topological ordering and directly sampled their values, where X_i got sampled according to the values of its parents (if any) based on the true distributions from **?**. For our initial experiments, we generated 50,000 samples, so the data is formatted in a $(5 \times 50,000)$-dimensional matrix[4]. Then, we randomly hid 50% of the elements of the data matrix. The objective is to use Gibbs sampling to estimate the set of CPTs, and see how close they match the true ones.

There are several metrics to evaluate our sampler. The one we employ in this paper is the average Kullback-Liebler divergence of all the distributions in the set of CPTs, denoted as KL_{avg}. For two distributions $p(x)$ and $q(x)$, the KL-divergence is $\sum_x p(x) \log(p(x)/q(x))$ where we iterate over x such that $q(x) > 0$. In the "student" data, there are eleven probability distributions that form the set of CPTs. For instance, X_4 "contributes" three distributions: $\Pr(X_4 \mid X_3 = 0), \Pr(X_4 \mid X_3 = 1)$, and $\Pr(X_4 \mid X_3 = 2)$, where X_3 (the parent) is fixed. Considering all the variables means that $KL_{\text{avg}} = \frac{1}{11} \sum_{i=1}^{11} p_i(x) \log(p_i(x)/q_i(x))$ with q_i the distribution our sampler estimates. We do not work with the full joint distribution $P(X_1, X_2, X_3, X_4, X_5)$ since it is more straightforward to work with the "projected" CPTs, and because with higher dimensional data, computing the full joint is computationally intractable and almost all entries would have probabilities essentially at zero.

Figures 2 and 3 plot the KL_{avg} metric for the student data using BIDMach and JAGS, respectively (note the log scale), with three SAME replication factors. Our results indicate that the KL_{avg} for BIDMach and JAGS reach roughly the same values, with a slight advantage to JAGS, though this is in part because of the log scale on the graph. In practice, the difference between BIDMach and JAGS is indistinguishable to humans, and both versions are able to get to the true CPTs to within a tolerance of 0.005. **Daniel: I need to formalize this "tolerance" and explain what this is like "in practice" since that will give more intuition. This is in progress.** Both versions converge to the true distributions quickly after about ten passes. For BIDMach, we used two batches with 25,000 cases each (more batches result in faster convergence), but initialized the CPTs randomly, which may have slowed down the initial convergence.

In addition, we observe that increasing m results in CPT estimates that more closely match the true CPTs. The red curves for Figures 2 and 3 correspond to substantially worse KL_{avg} than the respective yellow and blue curves. The increase from $m = 1$ to $m = 5$ has a much larger relative improvement than the increase from $m = 5$ to $m = 10$, consistent with our observations of diminishing returns.

Next, we evaluate how the SAME parameter m affects the runtime of the sampler, since more samples means the algorithm necessarily runs longer. Figures **??** and **??** plot KL_{avg} versus total runtime (in seconds) for BIDMach and JAGS, respectively. As before, KL_{avg} is on a log scale.

There are several interesting observations. The first is that the curves for higher m values start later due to high initialization costs (this is especially problematic for JAGS). This, along with the relative marginal benefit of increasing m on this data means that SAME may not have a beneficial time-accuracy tradeoff. BIDMach, on the other had, has faster initialization costs and increasing m results in faster convergence.

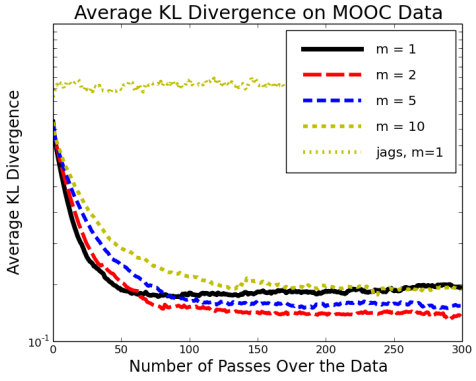

Figure 4: MOOC Data KL_{avg}. **Daniel: Don't worry, I will fix the y-axis labels.**

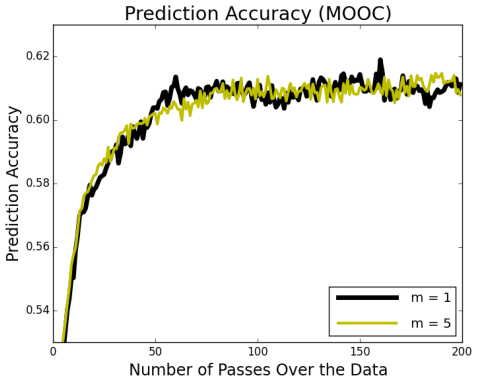

Figure 5: MOOC data prediction accuracy.

5.2 Real Dynamic Learning Maps "MOOC" Data

We now benchmark our code on a Bayesian network with a nation-wide examination dataset from the Dynamic Learning Maps (DLM) project. The data contains the assessment (correct or not) of student responses to questions from the DLM Alternate Assessment System. After our preprocessing, there were 4367 students and 319 questions. From the DLM data source, each of the questions is considered to be derived from a set of 15 basic concepts. We encode questions and concepts as variables and describe their relationships with a Bayesian network from the DLM data. Each question is considered an "observed" node in the Bayesian network, but with a very high missing value rate, as students only got to answer a fraction of the questions. Our data matrix, which has dimensions (334×4367), ultimately ended up with a 2.2% sparsity level. Furthermore, the first 15 rows in the data matrix are completely missing across all 4367 columns, since the 15 concepts are latent variables. The inference task is to learn the CPTs of the Bayesian network from this extremely sparse data. **Daniel: I need a reference (maybe we can ask Huasha?) to help me describe this data better.**

We evaluate our sampler on this MOOC data using two methods. The first involves running our sampler on the original data until convergence. Then, we use the estimated set of CPTs and sample from that via direct sampling to generate synthetic data, also of dimension (334×4367). We modify that synthetic data matrix so that it also has a 2.2% sparsity level and has missing data at the same elements as the original data matrix. Then we re-run BIDMach on that data and test using the same KL_{avg} metric from (5.1), since we know the CPTs that "generated" the data.

Figure 4 plots the KL_{avg} for BIDMach with varying m values. We ran the sampler for several thousands of iterations but only show the first 300 for the sake of readability. We see that, indeed, using $m = 1$ as a reference, the Gibbs sampler appears to converge quickly to the true distribution. It seems like the curve gets worse after the first 100 iterations, but the long term trend (not shown here) is that it settles down at the same KL_{avg} from the 100 iteration point. It is also apparent from Figure 4 (and the long-term trend) that $m = 2$ and $m = 5$ are better, showing that there is indeed a benefit to SAME with extremely sparse data. What is interesting is that the performance of $m = 10$ is *worse* than $m = 1$. Even after thousands of iterations, the $m = 10$ curve was essentially neck-to-neck with the $m = 1$ curve, but corresponds to a much slower runtime.

Daniel: We are planning on including one JAGS curve ($m = 1$) in Figure 4. We will need to explicitly state the runtimes for BIDMach/JAGS, probably just here in the text and not in a figure/table. Also, I should explain the "long term" trend stuff better, perhaps even extending the plot a bit. Finally, like before, I want to give an intuitive feel of the accuracy, i.e., how many decimal points of accuracy?

[4]It should be noted that this is only for the sake of comparison with JAGS. BIDMach can handle millions of samples, but for JAGS, we limited the number of cases to a reasonable number to facilitate comparisons.

Table 3: BIDMach (GPU) Runtime vs. GigaFlops on Large Data

	$m = 1$	$m = 5$	$m = 10$	$m = 20$	$m = 30$	$m = 40$	$m = 50$
GigaFlops (K)	2.38	6.98	9.08	11.00	11.70	12.27	12.35
Time/Iter (K)	0.307	0.523	0.804	1.328	1.872	2.380	2.957
GigaFlops (RM)	1.65	4.58	7.08	9.69	10.92	11.62	12.02
Time/Iter (RM)	0.666	1.192	1.541	2.252	2.998	3.753	4.536

The second way we evaluate our sampler is with prediction accuracy. We divide the original data into a training and testing batch so that 80% of the known data is in training. The training set is for updating the CPTs. During the testing batch, we sample for 500 iterations using the current CPTs (but *without updating* them). For each of the known test points, we compute the number of 0s and 1s sampled (from the 500 iterations), pick the majority to be our prediction, and then compare it with the true value. Since only 2.2% of the data is known, this is a challenging prediction problem. Moreover, we divide the training and testing data into batches, so for each student (i.e., column) in the test data matrix, we have no evidence. Random guessing is equivalent to 50% accuracy since this is a binary prediction task.

We ran the prediction accuracy algorithm with SAME parameters $m = 1$ and $m = 5$ using three different random seeds for each, to reduce the chances of one trial skewing the results. Figure 5 plots the mean prediction accuracy (among three trials) for both SAME parameters. Our sampler indeed learns to predict reasonably well and reaches roughly 61% accuracy. SAME is not beneficial in terms of accuracy or even the standard deviation of the accuracies (not plotted here). This is likely because sampling 500 times to determine our prediction may cause convergence to a result invariant to minor differences in CPTs. Sampling only one time to determine prediction accuracy resulted in about 56% accuracy for $m = 1$, and beyond 500 samples, we did not observe improvements.

5.3 Runtime vs. Throughput Tradeoff

We now discuss the impact of the SAME parameter in terms of runtime and throughput. We measure the throughput and speed of our Gibbs sampler in terms of GigaFlops (gflops), a billion floating point operations per second. It is possible to compute reasonable estimates of the maximum gflops attainable for a computer. Therefore, in order for us to claim that the Gibbs sampler is as optimized as it could be, we should argue that it attains gflops values that are close to the theoretical limit, indicating that further improvement is unlikely beyond simply upgrading the hardware.

Daniel: We should compute the theoretical gflops limit.

As the SAME parameter increases, it increases both the throughput (good) and runtime (bad) by increasing the number of data points in our computations. The results from (**?**) suggest that increasing m for small values will increase throughput while not costing too much in runtime. As one increases m beyond a certain data-dependent value, then SAME "saturates" the algorithm and results in stagnant throughput while significantly increasing runtime. It follows that a reasonable objective for an experimenter is to run the Gibbs sampler on data with varying values of m, and identify the point where further increases in m start having an unfavorable runtime and throughput tradeoff.

As an example, we used the Koller data from (5.1), with 50% of the data known versus missing, but with *one million* cases, beyond the feasible datasets of standard Gibbs samplers. We ran the Gibbs sampler for 200 iterations on this data with a batch size of 50,000. Table 3 shows the time and gflops of the Gibbs sampler using different values of m. One can see that increasing m for $m < 30$ results in steady gflops increases but *without* a substantial increase in time. Then SAME saturates the algorithm and the runtime increases while the gflops stalls, reinforcing the conclusions from (**?**) and argues for the importance of testing with a variety of m values.

Notice that the runtimes listed are in seconds, and even with $m = 50$, 200 iterations takes less than 10 minutes to complete — we have barely scratched the limit of our sampler. In fact, one of the main things that limits our sampler is the memory of GPUs, since we need to shrink the batch size with large m. As memory on GPUs becomes cheaper, we will be able to run Gibbs sampling and perform graphical model inference on larger datasets.

6 CONCLUSIONS

We conclude that our Gibbs sampler is much faster than the state of the art (JAGS) in Gibbs sampling and can be applied to data with several hundreds of variables. We also argue that SAME is beneficial for Gibbs sampling, and that it should be the go-to method for researchers who wish to perform inference on (discrete) Bayesian networks. Future work will explore the application of our sampler to a wider class of real-world datasets.

ACKNOWLEDGMENTS

We thank Yang Gao, Biye Jiang, and Huasha Zhao for helpful discussions.