# Fast Parallel SAME Gibbs Sampling on General Discrete Bayesian Networks

Daniel Seita, Haoyu Chen & John Canny
seita@berkeley.edu, haoyuchen@berkeley.edu, jfc@cs.berkeley.edu

## Introduction

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 project, we:

- Build a highly optimized Gibbs sampler
- Apply SAME strategy to reduce variance
- Use our Gibbs sampler on real dynamic learning map "MOOC" data to reveal our benefits.

## Fast Parallel SAME Sampling

SAME (State Augmentation for Marginal Estimation) [1, 2, 3] can be viewed as cooling the posterior parameter distribution and allows annealed search for the MAP parameters, often yielding very high quality (lower loss) estimates. Given a distribution $P(X, Z \mid \Theta)$, to estimate the most likely $\Theta$ based on the data $(X, Z)$ using SAME, one would define a new joint $Q$:

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

which models $m$ copies of the distribution tied to the same set of parameters $\Theta$.

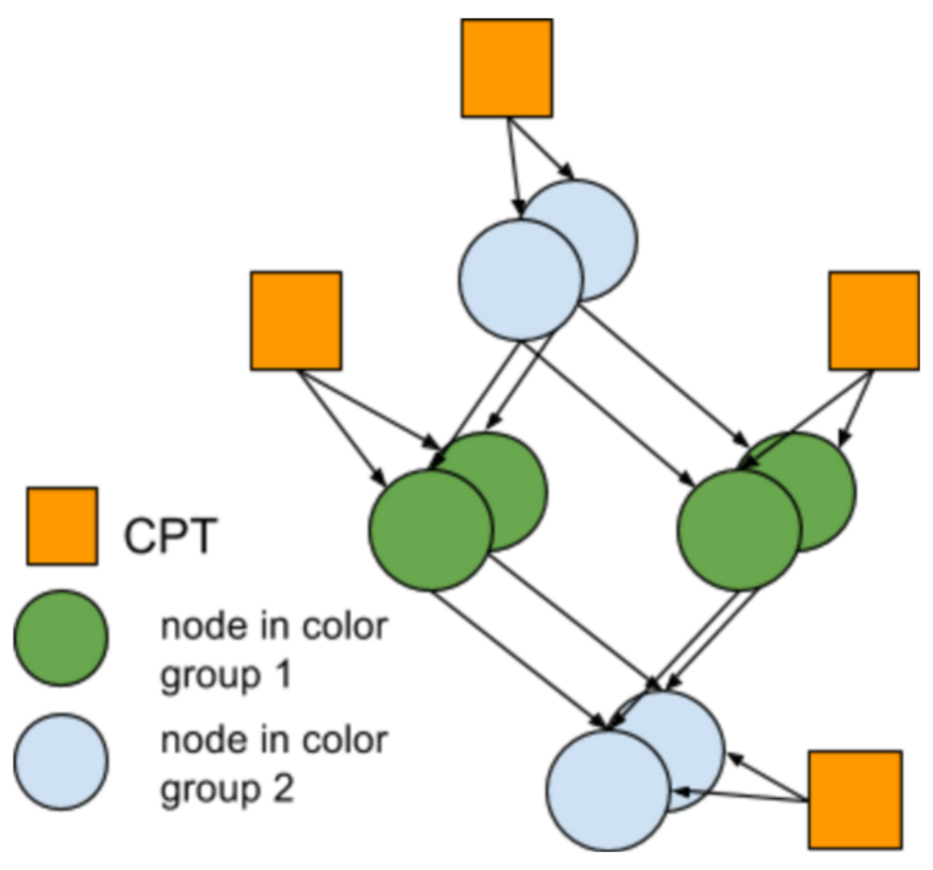

**Figure 1:** Gibbs sampler framework (m=2)

In order to parallel the sampling, we apply graph coloring to the moralized graph of the original network, and within each color group, sample all the variables in parallel. Left figure shows a simple example of three color groups with m=2.

## Conclusion

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.

## References

[1] A. Doucet, S. Godsill, and C. Robert. Marginal maximum a posteriori estimation using markov chain monte carlo. Statistics and Computing, 12:77-84, 2002.
[2] C. Robert, A. Doucet, and S. Godsill. Marginal MAP estimation using markov chain monte-carlo. In IEEE Int. Conf. on Acoustics, Speech and Signal Processing, volume 3, pages 1753-1756. IEEE, 1999.
[3] Z. Huasha, J. Biye, and C. John. Same but different: Fast and high quality gibbs parameter estimation. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, pages 1495-1502, NY, USA, 2015. ACM

## Implementation

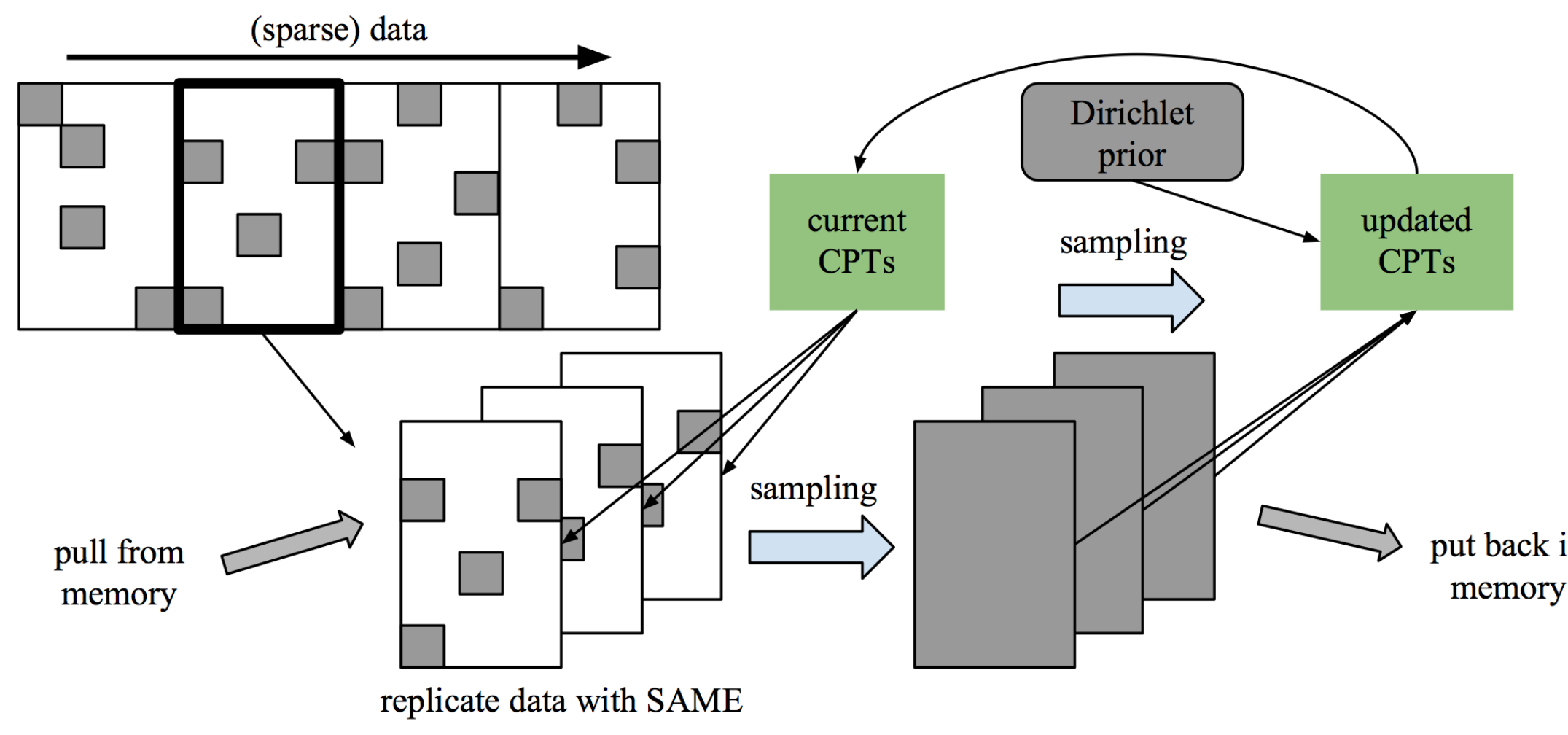

**Figure 2:** Gibbs sampler framework

Our Gibbs sampler is implemented and integrated as part of the open-source BIDMach library for machine learning. Figure 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.

## Experiments

we benchmark our code on a Bayesian network with a nation-wide examination dataset, which contains the assessment (correct or not) of student responses to questions. There were 4367 students and 319 questions. Each question is considered an "observed" node in the Bayesian network. We only know 2.2% nodes' value. We call the student responses data "MOOC" data. We compare our sampler with JAGS.

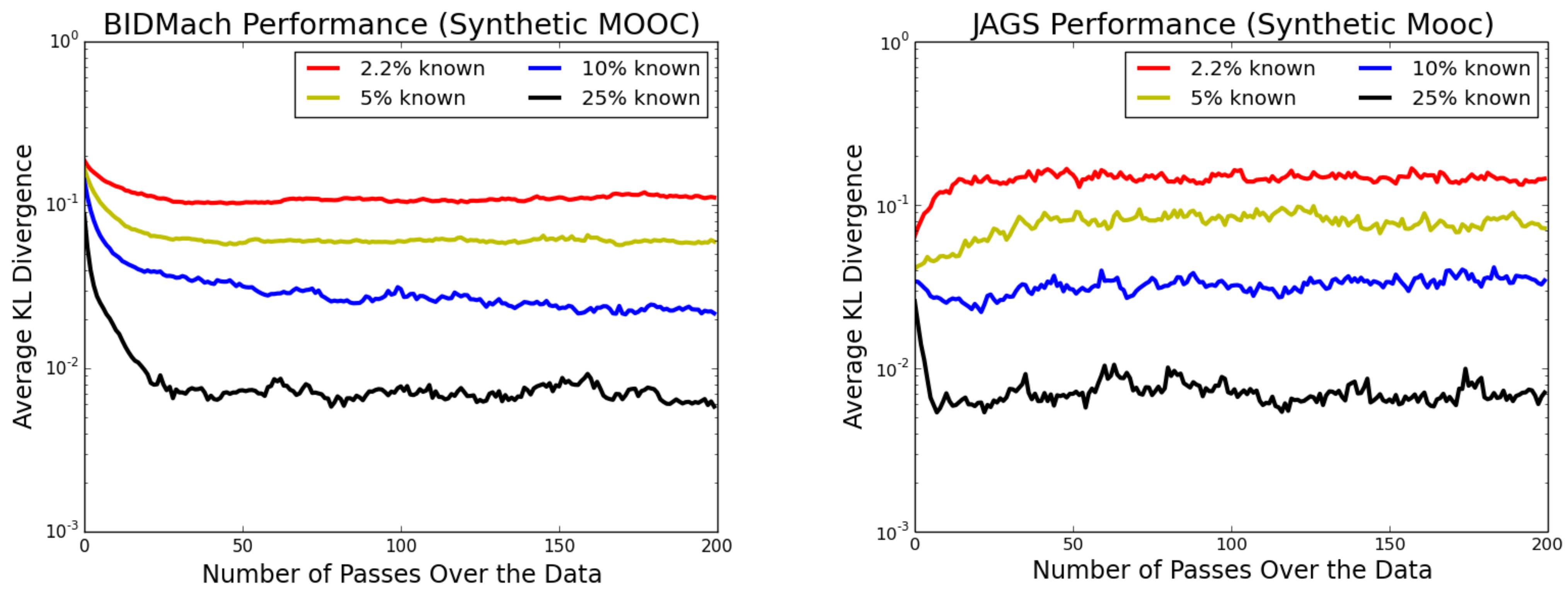

**Figure 3:** The $KL_{\text{avg}}$ from BIDMach. (left); The $KL_{\text{avg}}$ from JAGS.(right)

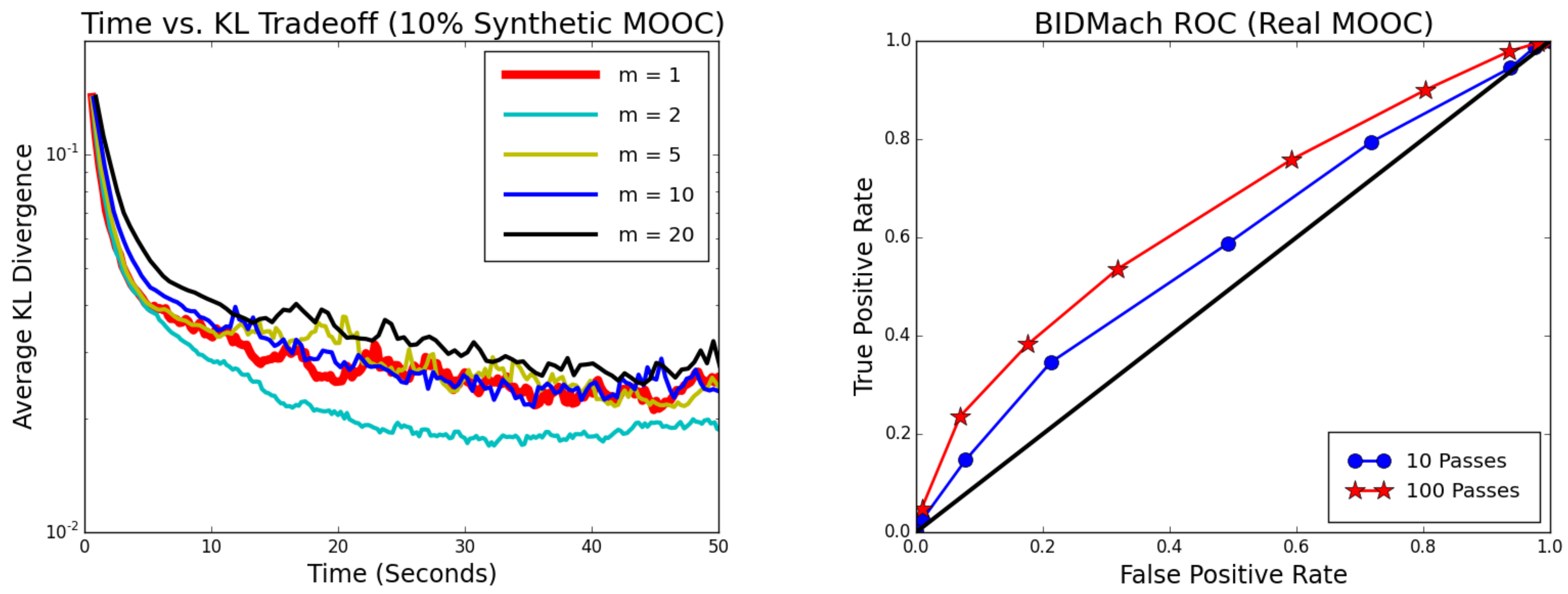

**Figure 4:** The $KL_{\text{avg}}$ vs. runtime for BIDMach. (left); Prediction accuracy of BIDMach(right)

In order to show the performance of our sampler, we replicate the data to increase its size. Table 1 records the runtime (CPU) per iteration for both BIDMach and JAGS, which reveals that our sampler has much shorter runtime than JAGS even without GPU.

| | 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 | 4.876 | 13.745 | 29.150 | 94.075 | | OOM |

**Table 1:** BIDMach (CPU) vs. JAGS Runtime on (Replicated) Real Data

Table 2 records the BIDMach runtime (GPU) and GigaFlops, which indicates that our sampler is as optimized as it could be by showing its gflops values are close to the theoretical limit for 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 |

**Table 2:** BIDMach (GPU) Runtime vs. GigaFlops on Large Data