10-708: Probabilistic Graphical Models, Spring 2015

28 : Approximate Inference - Distributed MCMC

Lecturer: Avinava Dubey *Scribes: Hakim Sidahmed, Aman Gupta*

1 Introduction

For many interesting problems, drawing samples from a complicated probability distribution is a difficult problem and approximate methods have to be used. A popular class of sampling algorithms is the "Markov Chain Monte Carlo" (MCMC) methods. These methods use an adaptive proposal $Q(x'\|x)$ to sample from the true distribution $P(x)$ using a Markov Chain. If certain conditions are met, samples from the adaptive distribution can be used as surrogate samples from the true distribution. Gibbs sampling is a particular type of MCMC algorithm for which each random variable x_i is sampled given the other variables.

In this note, we briefly review MCMC methods. We then go on to describe strategies to scale these methods to large datasets and models.

2 Review of MCMC methods [6]

We briefly review MCMC methods in this section.

Monte Carlo Methods are used to draw samples from a probability density function that is hard to compute. This situation can occur in Markov Random Fields (MRFs) where the normalization constant can be too hard to compute, or in high-dimensional models. MCMC methods are used when it is impossible to sample directly from the original distribution. They use a proposal distribution $Q(x'\|x)$ which proposes which point to sample next.

2.1 The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm is a popular MCMC method and is a generalization of the Metropolis algorithm. The algorithm is described in Algo 1. In Metropolis-Hastings, the form of the proposal distribution evolves based on the present state. A common choice of family for the proposal distribution is the isotropic Gaussian. The choice of the variance for this distribution is critical for the performance of the sampling: A smaller variance will increase the number of accepted samples, but will reduce the speed of search in of the space. Thus, choosing a proposal distribution forces us to perform a tradeoff between speed and acceptance rate, and requires care.

2.2 Gibbs Sampling - Special case of Metropolis-Hastings

Gibbs sampling is another popular MCMC method. In Gibbs sampling, each random variable x_i is sampled separately, with all the other variables fixed. In this particular case, the proposal distribution Q is chosen

Algorithm 1 The Metropolis Hastings Algorithm [6]

Input: Conditional proposal $Q(x'\|x)$, target $P^*(x)$
Output: Sampled set of N correlated samples drawn from P(x)
Initialize state $x^{(0)}$
Initialize sampled set as Null
for $t = 1...N$ **do**
  Sample proposal $Q(x'\|x^{(t)})$ to obtain candidate x'
  Accept this state with probability $\alpha = \frac{P^*(x')Q(x^{x^{(t)})\|x'}}{P^*(x^{(t)})Q(x'\|x^{(t)})}$
  if s **then**tate is accepted
    Set $x^{(t+1)} = x'$
  else
    $x^{(t+1)} = x^{(t)}$
  end if
  Add $x^{(t+1)}$ to the sampled set
end for
return the sampled set

in the following manner:

$$Q(x'\|x) = P(x'\|x_{-i})$$

This is a special case of the Metropolis-Hastings with an acceptance rate of 1. The details of Gibbs sampling have been described in Algo 2.

Algorithm 2 The Gibbs Sampling Algorithm [6]

Input: Conditional proposal
Output: Sampled set of N correlated samples drawn from P(x)
Initialize state $< x_1^{(0)}, .., x_K^{(0)} >$
Initialize sampled set as Null
for $t = 1...N$ **do**
  for $i = 1..K$ **do**
    Sample $x_i^{(t+1)} \sim P(Z_i\|x_1^{(t+1)}, ..., x_{i-1}^{(t+1)}, x_{i+1}^{(t+1)}, ..., x_K^{(t+1)})$
    Add $x^{(t+1)}$ to the sampled set
  end for
end for
return the sampled set

3 Gibbs sampling

3.1 Taking multiple chains

There is no fixed method which can detect the convergence of a Markov Chain. However, certain heuristics can be used to detect the convergence of MCMC methods. A common method consists of running multiple chains created with different seeds in parallel (one chain per CPU). When all the chains have close log-likelihoods, we can reasonably assume that they have converged. This heuristic is illustrated in figure 1. Each chain is run in parallel on a different core. While the chains on the left plot have not converged, we can reasonably assume the chains on the right plot have converged.

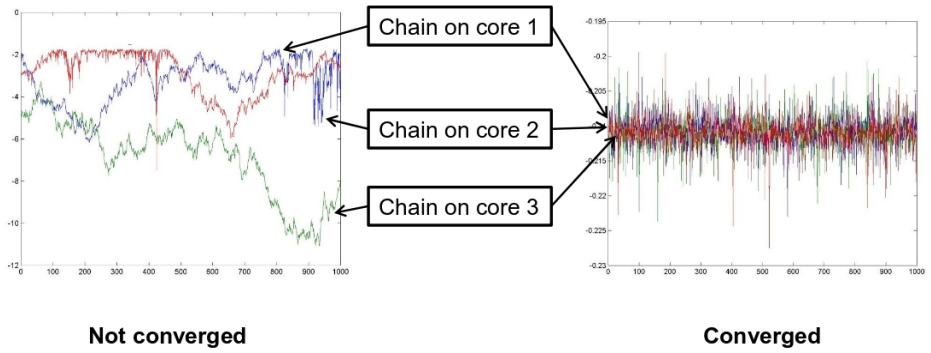

Figure 1: Number of iterations vs log-likelihood for multiple chains - left: No convergence - right: Convergence

However, this strategy of running multiple chains in parallel does not solve all the issues. Indeed, if the chains have a long burn-in period (years for instance), then this strategy will not reduce this duration. Hence, it is important to be able to sample individual chains faster.

4 Naive parallel Gibbs sampling

A naive strategy for speeding up Gibbs sampling is Synchronous Gibbs sampling. This method consists of sampling all the variables of a problem in parallel on a different processor. This method is illustrated in the alarm network example (figure 2):

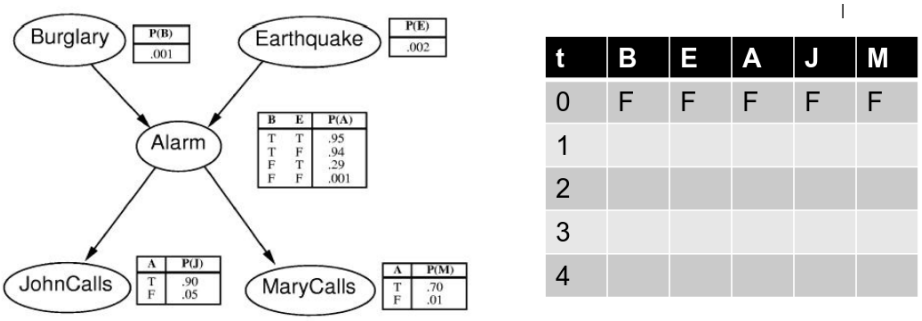

Figure 2: Naive parallel Gibbs sampling for the alarm network

This approach works as follows:

1. Assign each variable to a processor
2. Sample each variable at time t using the results values of the other variables at $t-1$
3. Send all the updated variables to all the processors

The main difference with the Sequential Gibbs Sampler is that each variable is sampled using the *old* values for the other variables, regardless of whether newer values have been sampled for them.

Algorithm 3 The synchronous Gibbs sampler[3]

1: **for all** variables X_j **do in parallel**
2: Execute Gibbs update: $X_j^{(t+1)} \sim \pi(X_j \| x_{N_j}^{(t)})$
3: **barrier end**

In this example, consider that all the variables are set to "false" at $t = 0$. Variable B will be assigned to worker 0, E to worker 1, A to worker 2, J to worker 3 and M to worker 4.

At time $t = 1$, worker 0 will sample variable B using all the other variables at time $t = 0$ (figure 3).

t	B	E	A	J	M
0	F	F	F	F	F
1	F				
2					
3					
4					

Figure 3: Variable B is sampled at time $t = 1$ using the values for the other variables at time $t = 0$

Worker 1 will sample variable E in parallel using all the variables at time $t = 0$ (figure 4).

t	B	E	A	J	M
0	F	F	F	F	F
1	F	T			
2					
3					
4					

Figure 4: Variable E is sampled at time $t = 1$ using the values for the other variables at time $t = 0$

Worker 2 will sample variable A in parallel using all the variables at time $t = 0$ (figure 5).

One possible disadvantage of this method is that it can require a lot of communications (to send the variable updates to all the workers). This problem can be limited by assigning blocks of variables to workers instead of single variables.

Even though this method works well for many models, like collapsed Gibbs sampling for LDA, for many models, the Synchronous Gibbs Sampler is not ergodic and hence does not converge to the correct stationary distribution.

Let us see such an example where the synchronous Gibbs sampler does not converge to the stationary distribution. Consider the Bayes Network defined in figure 6. This network essentially encodes a XOR relation between (A,B) and (A,C): If A is true, then with high probability B and C will be false. On the other hand, if A is false, then with high probability, B and C will be true.

t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F		
2					
3					
4					

Figure 5: Variable A is sampled at time $t = 1$ using the values for the other variables at time $t = 0$

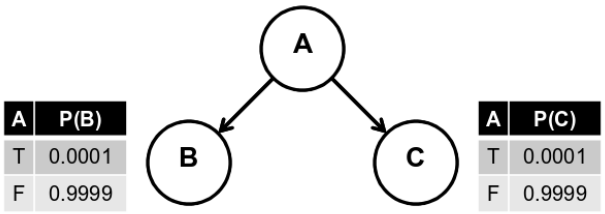

Figure 6: Example of Bayes Network where Naive parallel Gibbs sampling fails

Using the naive parallel Gibbs Sampler strategy, we sample each of the variables A, B, and C using the old values. Starting with a state $t = 0$ where all the variables are set to false, we sample A:

$$P(A\|B, C) \propto P(B\|A)P(C\|A)$$

. Given B and C are false, A will, with high probability, be true:

$$P(A = T\|B = F, C = F) \propto P(B = F\|A = T)P(C = F\|A = T) \propto 0.999x0.999 \approx 1$$

$$P(A = F\|B = F, C = F) \propto P(B = F\|A = F)P(C = F\|A = F) \propto 0.001x0.001 \approx 0$$

Likewise, the naive parallel Gibbs Sampler draws with high probability "true" for variable B:

$$P(B = T\|A = F) \propto 0.999 \approx 1$$

$$P(B = F\|A = F) \propto 0.001 \approx 0$$

and variable C will be, with high probability, drawn to be "true" as well.

At time $t = 1$, all the variables will therefore be true with high probability. At the following iteration, the variables will likely be chosen to be all "false" and so on. In fact, this algorithm will flip to a stationary distribution state with probability only $\frac{1}{10,000^4}$. In sequential Gibbs sampling, in opposite, once A would have flipped, neither B nor C would have flipped.

In general, the naive parallel Gibbs sampler performs poorly on near-discrete distributions.

There are, however, ways to perform good-quality Gibbs sampling in parallel.

t	A	B	C
0	F	F	F
1			
2			
3			
4			

Figure 7: At time $t = 0$, all the variables are chosen false

t	A	B	C
0	F	F	F
1	T		
2			
3			
4			

Figure 8: At time $t = 1$, variable A will be, with high probability, drawn to be true

5 The Chromatic sampler

An interesting implementation of parallel Gibbs sampling has been proposed by Gonzalez, Joseph, et al (2011). They introduced the Chromatic sampler. This method is based on two important concepts: Graph coloring and Markov Blanket. In Markov Random Fields, the Markov Blanket of a node is defined as its immediate neighbors. A node is independent of all the other nodes of the graph given its Markov Blanket. Consider the graph in figure 12. Because each node of a given color is separated from the other nodes of the same color, we can sample it independently of the other nodes of the same color. The Chromatic sampler will hence:

1. Find subsets of nodes, such that all nodes in a given subset are not in each other's Markov Blankets, and the subsets cover the whole graph

2. Sample each colored subset in parallel

Since each colored subgraph can be sampled in parallel, we want to find a graph coloring that creates as large subgraphs as possible. In the two colors graph given as example, the Chromatic sampler samples all the nodes in green in parallel, sends the updates to all the nodes, then samples all the red nodes in parallel and so on.

t	A	B	C
0	F	F	F
1	T	T	
2			
3			
4			

Figure 9: At time $t = 1$, variable B will be, with high probability, drawn to be true

t	A	B	C
0	F	F	F
1	T	T	T
2			
3			
4			

Figure 10: At time $t = 1$, variable C will be, with high probability, drawn to be true

One important particular case is given by trees. These graphs always admit a 2-coloring. More generally, bipartite graphs are always 2-colorable. This strategy does not work for all the graphs. For full cliques for instance, n nodes require n colors. Besides, the problem of determining whether a graph is k-colorable is NP-hard. Heuristics have to be used to find a good coloring.

6 Online parallel MCMC

For many problems, the data is received in a streaming way. Examples include news or twitter feeds. We need to process new data points one at a time. In addition to that, we need to be able to "forget" old data

Algorithm 4 The Chromatic sampler[3]

1: **Input**: k-colored MRF
2: **for** each of the k colors k_i: $i \in \{1, ..., k\}$ **do**
3: **for all** variables $X_j \in \kappa_i$ in the i^{th} color **do in parallel**
4: Execute Gibbs update: $X_j^{(t+1)} \sim \pi(X_j \| x_{N_j \in \kappa < i}^{(t+1)}, x_{N_j \in \kappa > i}^{(t)})$
5: **barrier end**
6: **end for**

t	A	B	C
0	F	F	F
1	T	T	T
2	F	F	F
3			
4			

Figure 11: At time $t = 2$, all the variables will be, with high probability, drawn to be false

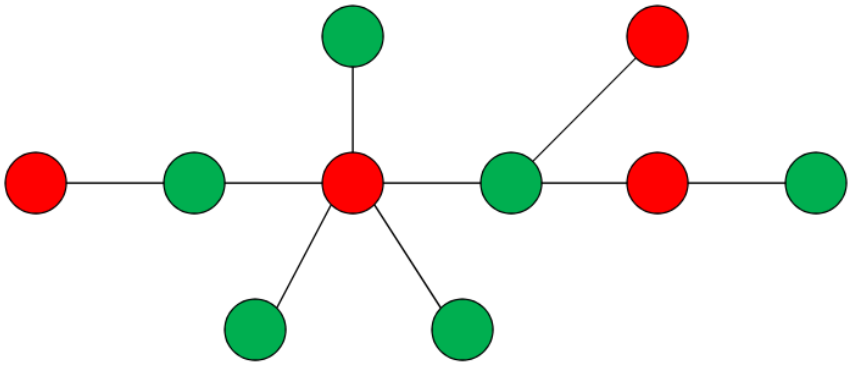

Figure 12: Example of graph coloring

points, because the amount of memory available is limited. Because of the frequency this data is received, we need to process it constant time for each new data point. And because of the amount of data to be processed, it is important to employ methods that can be parallel to scale them up.

6.1 Sequential Monte Carlo

Sequential Monte Carlo is a method that generalizes particle filters. Particle filters incrementally sample latent variables X at time t given observations Y up to t: $P(X_t \| Y_{1:t})$. While particle filters assume the graphical model is a state-space model, SMC does not assume the models has any particular structure at all: Given n random variables x_1,...,x_n, SMC constructs proposal distributions such that, given samples from $P(x_{1:k})$, we can sample $P(x_{1:k+1})$ in constant time. This method does not require the (intractable) computation of the normalization constant as the distribution $P(x_{1:n})$ needs only be evaluated up to a multiplicative constant.

6.2 Sequential Importance Sampling

Sequential importance sampling is the foundation of sequential Monte Carlo. In order to get samples in constant time, we assume the distribution factorizes in the following form:

$$q_n(x_{1:n}) = q_{n-1}(x_{1:n-1} \| x_{1:n-1}) = q_1(x_1) \prod q_k(x_k \| x_{1:k-1})$$

The advantage of this method is that once we have samples from 1 to k, we don't need to redraw these samples. Note that the variables x_i can represent more than single variables. For instance, if the observed variable y_i represents documents, then x_i can represent the distribution over topics.

In normalized importance sampling, the sample weights are defined by:

$$w^i = \frac{r^i}{\sum_j r^j} \tag{1}$$

where

$$r^i = \frac{P'(x^i)}{Q(x^i)}$$

That is, the weights are proportional to the probability of the samples divided by the proposal distribution of that sample. They quantify the importance of that sample in finding the marginal distribution.

We saw previously that the samples can be drawn in constant time. We can also get the weights in constant time.

In Sequential importance sampling, the unnormalized weights r can be written as:

$$r(x_{1:n}) = \frac{P'_n(x_{1:n})}{q_n(x_{1:n})}$$

$$r(x_{1:n}) = \frac{P_{n-1}^{x_{1:n-1}}(x_{1:n-1})}{q_{n-1}(x_{1:n-1})} \frac{P'_n(x_{1:n})}{P_{n-1}\prime(x_{1:n-1}) q_n(x_n \| x_{1:n-1})}$$

$$r(x_{1:n}) = r_{n-1}(x_{1:n-1}) \alpha_n(x_{1:n-1})$$

$$r(x_{1:n}) = r_1(x_1) \prod_{k=2}^{n} \alpha_k(x_{1:k})$$

where

$$\alpha_n(x_{1:n}) = \frac{P'_n(x_{1:n})}{P_n\prime(x_{1:n-1}) q_n(x_n \| x_{1:n-1})}$$

This formula shows that the unnormalized weights can be computed incrementally: We compute α_n and use it to update $r(x_{1:n-1})$ to $r(x_{1:n})$. For this update to be constant time, $P'_n(x_{1:n})$ must be computable from $P'_{n-1}(x_{1:n-1})$ in constant time. We save the unnormalized weights r at each iteration, and compute the normalized weights w as needed from r using 1. Hence, for each new variable x_n, we can sample x and compute the normalized weights w in constant time. And, since the samples do not depend on each other, sequential importance sampling can easily be made parallel.

The algorithm is described in Algo 5. Note that in the algorithm is very fast, even though the computation of the normalized weights is sequential. Although in theory we should synchronize this step, these updates can be done asynchronously [5].

Algorithm 5 Sequential Importance Sampling

for $n = 1$ **do**
  Parallel draw samples $x_1^i \sim q_1(x_1)$
  Parallel compute unnormalized weights $r_1^i = \frac{P_1'}{q_1(x_1^i)}$
  Compute normalized weights $w_1^i by normalizing r_1^i$
end for
for $n \geq 2$ **do**
  Parallel draw samples $x_n^i \sim q_n(x_n \| x_{1:n-1} i)$
  compute unnormalized weights $r_n^i = r_{n-1}^i \alpha_n(x_{1:n}^i) = r_{n-1}^i \frac{P_n\prime(x_{1:n}^i)}{P_{n-1}\prime(x_{1:n-1}^i) q_n(x_n^i \| x_{1:n-1}^i)}$
  Compute the normalized weights w_n^i by normalizing r_n^i
end for

One major drawback of sequential importance sampling is that the variance of the samples increases exponentially with n[4], and the accuracy can decrease significantly. One solution to this problem is to resample from $n - T$ to T at every iteration.

Beyond decreasing variance, resampling:

1. removes samples x^k with low weights, which come from low probability regions. However, we want to focus on high probability regions o $P(x)$. Since each sample gets an equal amount of computation, regarless of its weight w_k, resampling makes sure we spend time on the weights computations of high probability samples.

2. prevents a small number of samples x_k from dominating the empirical distribution: Every time we resample, all the weights are reset to $\frac{1}{N}$, thus preventing some sample weights to grow too much.

The sequential Monte Carlo algorithm is then just the sequential importance sampling with resampling at every iteration:

Algorithm 6 Sequential Monte Carlo

for $n = 1$ **do**
  Parallel draw samples $x_1^i \sim q_1(x_1)$
  Parallel compute unnormalized weights $r_1^i = \frac{P_1'}{q_1(x_1^i)}$
  Compute normalized weights $w_1^i by normalizing r_1^i$
  Parallel resample w_1^i, x_1^i into N equally weighted particles x_1^i
end for
for $n \geq 2$ **do**
  Parallel draw samples $x_n^i \sim q_n(x_n \| x_{1:n-1} i)$
  compute unnormalized weights $r_n^i = r_{n-1}^i \alpha_n(x_{1:n}^i) = r_{n-1}^i \frac{P_n\prime(x_{1:n}^i)}{P_{n-1}\prime(x_{1:n-1}^i) q_n(x_n^i \| x_{1:n-1}^i)}$
  Compute the normalized weights w_n^i by normalizing r_n^i
  Parallel resample w_n^i, x_n^i into N equally weighted particles x_n^i
end for

7 Parallel inference for Bayesian Nonparametrics

In this section, we review parallel inference methods for a popular Bayesian nonparametric model, viz. the Dirichlet Process Mixture Model (DPMM). We first recap the model and then discuss normal and parallel inference techniques for DPMMs. Finally, we discuss some comparative results.

7.1 The Dirichlet Process Mixture Model

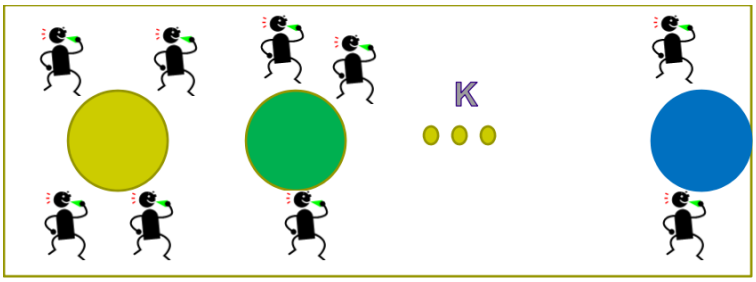

Figure 13: Infinite mixture models : restaurant perspective

Let us begin with a finite representation. Assume a restaurant with K tables. Each table has one and only one dish (represented by a color in figure 13). Specifically, the items in the situation represent the following:

- Table : Cluster
- People : Items to be clustered
- Dish/color on each table : Center of each cluster
- Hidden variables : Assignment of people to each table

The estimation problem is to find out what dish is on each table (parameter computation) and the assignments of people to different tables (cluster assignments). Now, let us take different situations into account.

If people sit on the table with the most preferred dish/color, this algorithm simulates hard k-means with people(items) being assigned to tables(clusters). In another scenario. people sit on the table proportional to appreciation of dish/color. Then, this is a representation of the soft k-means algorithm.

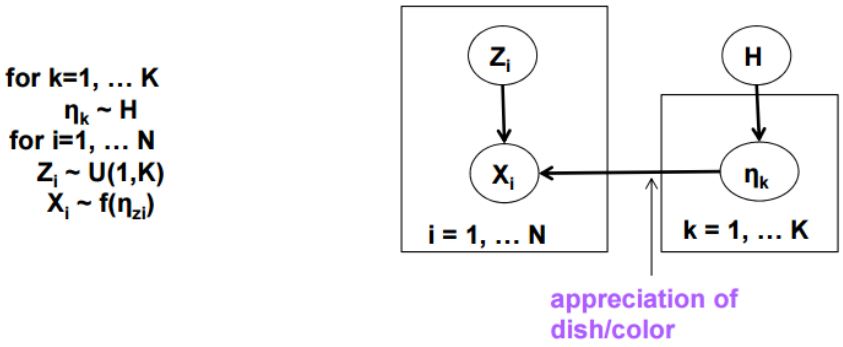

Figure 14: Soft k-means as a a generative process

In general, the soft k-means can be described as a generative process. Each dish (parameter setting) is represented by a distribution, from which items can be drawn. Specifically, we draw K cluster centers from a distribution H. Then, we draw Z_i from a uniform distribution representing K clusters. Using Z_i and cluster information, we then proceed to generate item X_i. This information is depicted as a graphical model in figure 14.

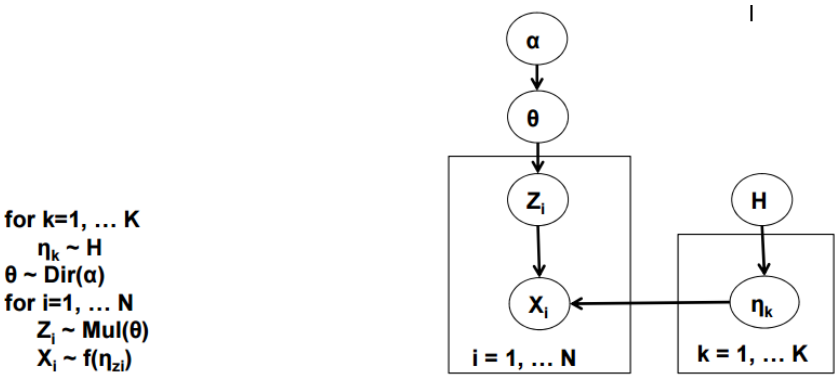

Figure 15: Finite Mixture Model Generative Model

Yet another "flavor" to the above problem is adding the "rich gets richer" property to the above situation. In that case, people sit on the table proportional to appreciation of dish/color and number of people sitting on the table. Thus, tables with more people will attract new people towards them. This is the finite version of the Dirichlet Distribution Mixture Model. The generative process of the Dirichlet Distribution Mixture Model is very similar to the soft k-means' process, except that now cluster numbers are drawn from multinomials instead of the uniform distribution. The graphical model associated with this process is described in figure 15.

7.1.1 Infinite representation of Dirichlet Process Mixture Model

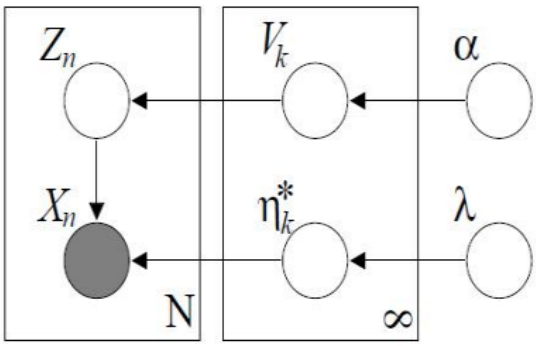

Figure 16: DPMM generative process

1. Draw $V_i \mid \alpha \sim \text{Beta}(1, \alpha), \quad i = \{1, 2, \ldots\}$
2. Draw $\eta_i^* \mid G_0 \sim G_0, \quad i = \{1, 2, \ldots\}$
3. For the nth data point:
   (a) Draw $Z_n \mid \{v_1, v_2, \ldots\} \sim \text{Mult}(\pi(\mathbf{v}))$.
   (b) Draw $X_n \mid z_n \sim p(x_n \mid \eta_{z_n}^*)$.

Figure 17: Description of the generative process

Now, assume that the number of tables is infinite. People either sit on a table proportional to appreciation of dish/color and number of people sitting on the table, or choose a new table and place a new dish on it. They can choose a new table with a certain probability. The posterior distribution of such a setting results in a Dirichlet distribution and hence this process is referred to as a Dirichlet process mixture model.

Samples from a Dirichlet distribution can be created using the stick breaking construction. The probabilities of choosing a table must sum to one. In the stick breaking construction, we break a stick into two parts. The first part is the probability of choosing a table(cluster). We then take the remaining part of the stick and break it into two parts again. Again, we choose the first part as the probability of choosing a table. We keep repeating this process infinite times and essentially get a distribution over an infinite number of tables (clusters).

The generative process of the DPMM is described in figure 16 and 17. This process now involves using a multinomial distribution over an infinite number of clusters. Each component V_k of the multinomial is drawn from a *Beta* distribution to simulate the stick breaking process. Each Z_n is drawn from an infinite multinomial. Using Z_n and cluster information, samples can be drawn from the distribution.

7.2 Inference of Dirichlet Process Mixture Model

A simple Gibbs sampling algorithm would involve sampling each of the variables given the rest of the variables having fixed values. Variables to sample are table proportion V_k , table assignment to each customer (Z) and dish at each table η. This is easily parallelizable and many different parallelization schemes are described subsequently.

7.3 Parallel Inference of Dirichlet Process Mixture Model

An easy way to parallelize Gibbs sampling for DPMMs is to use the observation that given V_k and η_k, all Zs are independent of each other. Thus, we can parallelize the inference of all the Zs, but are forced to sample all V_k and η_k together. Empirical results show that this process leads to poor mixing, and thus is not of great practical utility.

An interesting way to improve the above-mentioned process was introduced recently [7] and involves sampling all Zs in different processors and maintaining a local copy of all V_k and η_k by taking noisy samples for each processor.

Collapsed Gibbs sampling involves integrating out all V_k and η_k . This leads to good mixing, but couples all the Zs and makes parallel inference hard. Thus, collapsed Gibbs sampling suffers from large computational cost.

Yet another inference technique is variational inference. In variational inference, we approximate the posterior with a distribution belonging to a more manageable family of distribution. This typically involves making simplifying assumptions about the original model and relaxing many dependencies. Parallel inference is easy in variational methods, but the search space for "good" models becomes limited and these methods are typically less accurate than MCMC methods.

Sequential Monte Carlo (SMC) Methods keep a pool of particles, approximate the distribution using weighted combination of the pool. Parallel inference is easy, but SMC methods suffer from high variance. To alleviate this, it is sometimes necessary to resample via MCMC methods.

7.3.1 Parallel MCMC for DPMMs

A recent idea for parallel inference of DPMMs uses a very simple trick. It involves running local collapsed Gibbs samplers on different cores/processes and then simple/approximate recombination of the results produced by each core. This generally does not work very well, because two newly discovered clustered in two different processors may not be the same. Also, re-combining results is a problem. However, this model has shown to give good results empirically for some models.

Another parallel inference uses the idea that Dirichlet Mixture of Dirichlet processes are Dirichlet processes. This can be understood using a nice analogy. Each restaurant can be reached via a key π_i (drawn from a multinomial distribution) earmarked for that restaurant. Thus, choosing the restaurant can also be thought of as a Dirichlet process. Given the keys of all items, each of the processors is independent of other processors. Conditioned on the Restaurant allocation data are distributed according to P independent Dirichlet process. We can then perform local collapsed Gibbs sampling on the independent DPs on each processor, and combine results. We can propose to move a cluster c to a processor p. The acceptance ratio depends on the cluster size. This can be done asynchronously without affecting algorithm performance.

The generative process is described in figure 18. The figure shows the equivalence between the auxiliary method and the original method. By introducing some auxiliary variables, we were able to add the power

$$D_j \sim \mathrm{DP}\left(\frac{\alpha}{P}, H\right), \quad j = 1, \ldots, P$$
$$\phi \sim \mathrm{Dirichlet}\left(\frac{\alpha}{P}, \ldots, \frac{\alpha}{P}\right)$$
$$\pi_i \sim \phi$$
$$\theta_i \sim D_{\pi_i}$$
$$x_i \sim f(\theta_i), \quad i = 1, \ldots, N.$$

$$\Longleftrightarrow$$

$$D \sim \mathrm{DP}(\alpha, H),$$
$$\theta_i \sim D,$$
$$x_i \sim f(\theta_i)$$

Figure 18: Comparison of many inference methods for DPMMs

of parallelism to the process.

7.4 Results

Figure 19 compares the performance of the above described algorithms on the same dataset. Gibbs sampling suffers from computational complexity. Variational methods lead to poor accuracy. SMC methods improve in terms of accuracy when we increase the number of particles, but suffer from high variance. It is clear from the figure that the auxiliary variable method outperforms the naive parallel MCMC method and leads to the best performance in terms of time and accuracy.

Figure 20 compares the perplexity of different models for Hierarchical Dirichlet Processes. It is clear that the AVparallel method outperforms the naive parallel MCMC method by achieving the lowest perplexity.

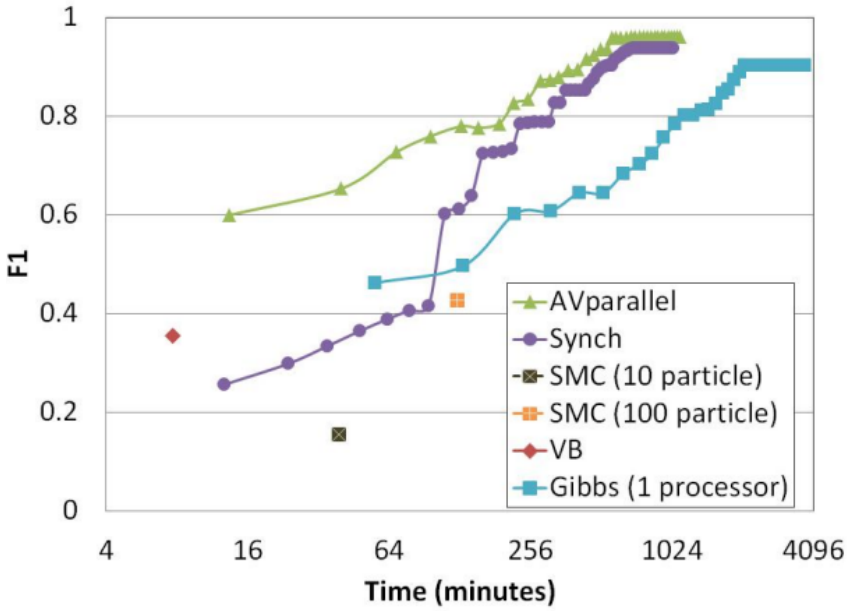

Figure 19: Comparison of many inference methods for DPMMs

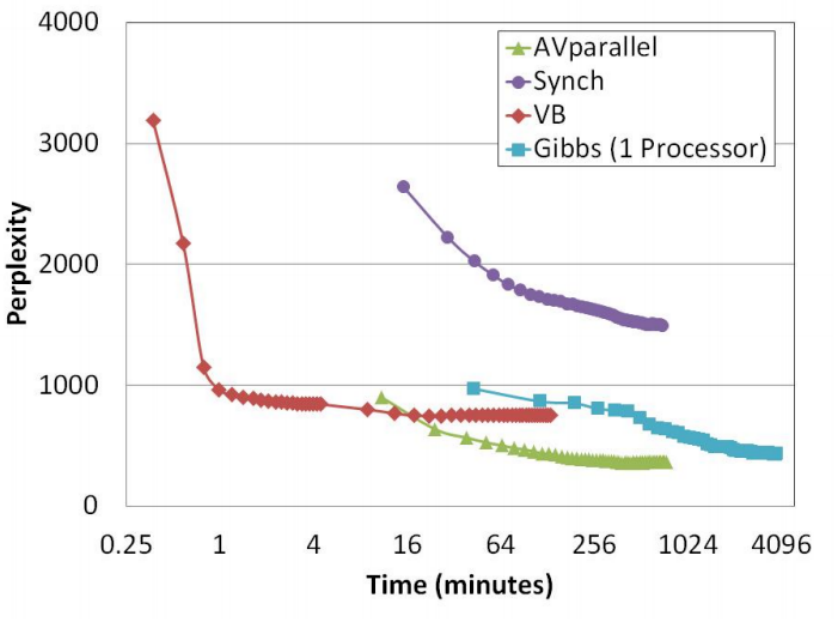

Figure 20: Comparison of inference methods for Hierarchical DPs

8 Conclusion

We discussed some parallel inference schemes for graphical models. Some conclusions are that naive parallel inference scheme does not always work. It is a good idea to utilize the structure of the problem to achieve conditional independence to increase chances of parallelizing. Also, it is important to peform either exact parallel inference or achieve a bound on the error of such methods to ensure good quality of inference results.

References

[1] Eric P.Xing *Probabilistic Graphical Models, lectures notes.*
`http://www.cs.cmu.edu/ epxing/Class/10708/`

[2] MacKay, David JC. *Introduction to monte carlo methods* [*Learning in graphical models. Springer Netherlands, 1998. 175-204.*]

[3] Gonzalez, Joseph and Low, Yucheng and Gretton, Arthur and Guestrin, Carlos *Parallel Gibbs sampling: From colored fields to thin junction trees* [*International Conference on Artificial Intelligence and Statistics*]. P 324–332, 2011.

[4] Doucet, Arnaud, Nando De Freitas, and Neil Gordon. *introduction to sequential Monte Carlo methods.* [*Springer New York, 2001*]

[5] Paige, Brooks, et al. *Asynchronous Anytime Sequential Monte Carlo* [*Advances in Neural Information Processing Systems. 2014*]

[6] Xie, Luu *Scribe notes* 2014

[7] Newman, David and Asuncion, Arthur and Smyth, Padhraic and Welling, Max *Distributed algorithms for topic models* 2014