Entangled Monte Carlo

Seong-Hwan Jun\textsuperscript{1}

\textsuperscript{1}Department of Statistics  
University of British Columbia

Sept 29th, 2012 / SFU-UBC joint seminar
My supervisor Alexandre Bouchard-Côté for his innovative ideas and collaborator Liangliang Wang for application of Entangled Monte Carlo to Phylogenetics.
This talk is organized as follows,

- Sequential Monte Carlo
- Entangled Monte Carlo

My contribution is explained in the second part of the talk but it cannot be explained without talking about SMC. Instead of trying to explain SMC using my own words (which would be too difficult for me), I decided to base it on the words of the experts found in [1].
Outline

1. Intro to Sequential Monte Carlo (SMC)
2. The SMC algorithm
3. Entangled Monte Carlo
Motivation for using SMC

Sequential Monte Carlo (SMC) methods are a class of Monte Carlo methods. It has wide range of application areas and in many cases, more suitable than MCMC. Also, it is

- Simple to implement
- Easy to parallelize over multi-cores or GPU’s
Let’s fix a problem setting to ease the explanation of SMC,

- Observations $y_t$ arrive sequentially
- There is an unobserved hidden state $x_t$ for each observation $y_t$
- Model the hidden variables with the Markov process
- The observation $y_t$ is conditionally independent of all other variables given the hidden state $x_t$
Notations

- \{y_t\}; y_t \in \mathcal{Y}, t = 1, \ldots, T
- \{x_t\}; x_t \in \mathcal{X}, t = 0, \ldots, T
- p(x_0); p(x_t|x_{1:t-1}) = p(x_t|x_{t-1}) \text{ for } t \geq 1
- p(y_t|x_{1:t}, y_{1:t-1}) = p(y_t|x_t) \text{ for } t \geq 1
- x_{1:t} \text{ is the shorthand for } (x_1, x_2, \ldots, x_t)
Graphical model view
In this problem setting, the posterior distribution is given by,

\[ p(x_{0:t} | y_{1:t}) = \frac{p(y_{1:t} | x_{0:t})p(x_{0:t})}{\int p(y_{1:t} | x_{0:t})p(x_{0:t})} \]

When it’s clear to do so, I will denote the posterior distribution as,

\[ p(x_{0:t} | y_{1:t}) = \frac{\gamma(x_{0:t})}{Z} \]
Update rule

This framework and problem setting allows to update the posterior recursively as new observation $y_{t+1}$ arrives,

$$p(x_{0:t+1} | y_{1:t+1}) = p(x_{0:t} | y_{1:t}) \frac{p(y_{t+1} | x_{t+1})p(x_{t+1} | x_{t})}{p(y_{t+1} | y_{t})}$$

These simple update rules are the reasons why SMC is so attractive.
The main goal is to compute the posterior distribution $p(x_{0:t}|y_{1:t})$
We may also want to compute the marginal distribution $p(x_t|y_{1:t})$
And to compute some expectation of interest w.r.t. the posterior...

$$E_{p(x_{0:t}|y_{1:t})}[f_t(x_{0:t})] = \int f_t(x_{0:t})p(x_{0:t}|y_{1:t})dx_{0:t}$$
Outline

1. Intro to Sequential Monte Carlo (SMC)
2. The SMC algorithm
3. Entangled Monte Carlo
Suppose we want to evaluate,

\[ E_{p(x_0:t|y_1:t)}[f_t(x_0:t)] = \int f_t(x_0:t)p(x_0:t|y_1:t)dx_{0:t} \]

Monte Carlo approximation to the above quantity can be obtained by simulating \( N \) i.i.d random samples \( \{x_0^{(i)}: i = 1, \ldots, N\} \) from the posterior, \( p(x_0:t|y_1:t) \) to obtain the estimate,

\[ \int f_t(x_0:t)p(x_0:t|y_1:t)dx_{0:t} \approx \frac{1}{N} \sum_{i=1}^{N} f_t(x_0^{(i)}) \]
Problem is that we usually cannot sample efficiently from the posterior.
So, what do we do when we cannot sample from the posterior directly? We do **Importance Sampling**. The idea is as follows,

1. Choose an *importance sampling distribution*, $\pi(x_{0:t})$ such that,
   - It is easy to sample from,
   - $\gamma(x_{0:t}) > 0 \Rightarrow \pi(x_{0:t}) > 0$

2. Sample $N$ i.i.d from $\pi(x_{0:t})$

3. For each sample, compute the weight $w^{(i)} = w(x_{0:t}^{(i)}) = \frac{\gamma(x_{0:t}^{(i)})}{\pi(x_{0:t}^{(i)})}$

4. Estimate for the expectation is given by the weighted average.

$$\int f_t(x_{0:t}) p(x_{0:t}|y_{1:t}) dx_{0:t} \approx \frac{1}{\sum_{i=1}^{N} w^{(i)}} \sum_{i=1}^{N} w^{(i)} f(x_{0:t}^{(i)})$$
Note that to shorten the notation, I used the shorthand to omit $y_{1:t}$,

$$\pi(x_{0:t}) = \pi(x_{0:t} | y_{1:t})$$

Also I will denote the normalized weight with a tilde (to avoid writing the sum),

$$\tilde{w}^{(i)} = \frac{w^{(i)}}{\sum_{i=1}^{N} w^{(i)}}$$

The approximation can be written as,

$$\int f_{t}(x_{0:t}) p(x_{0:t} | y_{1:t}) dx_{0:t} \approx \sum_{i=1}^{N} \tilde{w}^{(i)} f(x_{0:t}^{(i)})$$
Problem with IS is that it is not a recursive estimation because it requires all of the $x_{0:t}$ to be sampled, which means all of the observations $y_{1:t}$ must be available for computation.

To see why this is bad, imagine that we get a new observation $y_{t+1}$, then we would have to do the computation all over again. And each time we do computation again, the time complexity is increased due to having an extra observation $y_{t+1}$. 
Sequential Importance Sampling

The solution to the problem is that we constrain the importance distribution to have the following structure,

$$\pi(x_{0:t}) = \pi(x_{0:t-1})\pi(x_t|x_{0:t-1}) = \pi_0(x_0) \prod_{i=1}^{t} \pi_i(x_i|x_{1:i-1})$$
Choose the importance distribution

Generate samples from the importance distribution,

1. For $t = 0$, sample $X_0^{(j)} \sim \pi_0(x_0)$ for $j = 1, 2, \ldots, N$
2. For $t > 0$, sample $X_t^{(j)} \sim \pi_t(x_t|x_1^{(j)}_{:t-1})$ for $j = 1, 2, \ldots, N$

3. Compute the weights and normalize $\tilde{w}^{(j)}$

4. Estimate the posterior or expectation as in the Importance Sampling
Note that for $t > 0$, the weight can be computed recursively

$$
\begin{align*}
    w(x^{(j)}_{0:t}) &= \frac{\gamma(x_{0:t}^{(j)})}{\pi(x_{0:t}^{(j)})} \\
    &= \frac{\gamma(x_{0:t}^{(j)})}{\gamma(x_{0:t-1}^{(j)})} \frac{\gamma(x_{0:t}^{(j)})}{\pi(x_{0:t-1}^{(j)}) \pi(x_{t-1}^{(j)} | x_{0:t-1}^{(j)})} \\
    &= \frac{\gamma(x_{0:t-1}^{(j)})}{\pi(x_{0:t-1}^{(j)})} \frac{\gamma(x_{0:t}^{(j)})}{\gamma(x_{0:t-1}^{(j)}) \pi(x_{t-1}^{(j)} | x_{0:t-1}^{(j)})} \\
    &= w(x_{0:t-1}^{(j)}) \frac{\gamma(x_{0:t}^{(j)})}{\gamma(x_{0:t-1}^{(j)}) \pi(x_{t-1}^{(j)} | x_{0:t-1}^{(j)})}
\end{align*}
$$
We refer to the samples as **particles** because they follow a path like a particle. In this picture, we have 7 particles, generated for 4 generations.
Additional benefit of SIS is that specifying the importance distribution is easier since it just needs to be of one dimension.
The problem with SIS is that it suffers from degeneracy problem in high dimensional spaces. What this means is that as $t$ gets large, it is very likely that only one of the particles have the normalized weight close to 1, whereas all of the other samples have 0 or close to 0 weights. The degeneracy problem leads to poor approximation.
The size of the circles indicate the size of the weights of each of the samples at each generation.

Note that the third particle’s weight keeps getting larger whereas the other particles are decreasing in weight.
The degeneracy problem can be solved with bootstrapping. At each iteration $t$, perform bootstrap on the particles using the weight $\tilde{w}^{(j)}$ as the probability for sampling that particle – survival of the fittest.
Sequential Importance Sampling with bootstrapping,
Generic SMC algorithm

At time $t = 0$
- Sample $X_0^{(j)} \sim \pi(x_0)$
- Compute the weights $w(X_0^{(j)})$ and normalize $\tilde{w}(X_0^{(j)})$
- Resample $\{\tilde{w}_0^{(j)}, X_0^{(j)}\}$ to obtain $N$ particles

At time $t \geq 1$
- Sample $X_t^{(j)} \sim \pi_t(x_t | x_{0:t-1}^{(j)})$
- Set the particle $j$'s path as $X_{0:t}^{(j)} = (X_{0:t-1}^{(j)}, X_t^{(j)})$
- Compute the weight $w^{(j)}$ and normalize $\tilde{w}^{(j)}$
- Resample $\{\tilde{w}_t^{(j)}, X_{0:t}^{(j)}\}$ to obtain $N$ particles
Note that in essence, SMC builds genealogy of particles.
Easy to parallelize

The SMC algorithm is simple to parallelize . . .

- The most time consuming step is the sample generation
- If we have $N = 100$ samples and $M$ CPUs, we can generate $N$ samples in $N/M$ iterations
- This is why SMC is fast and one of the reasons why SMC is so attractive.
This time... Suppose we have $M$ machines, each with 1 CPUs and again $N$ samples to generate. Theoretically, we need $N/M$ iterations to generate $N$ samples. But there are other difficulties. Let’s trace the SMC algorithm to see why it is not easy to distribute (next slide).
Suppose $M = 2$, $N = 10$. At $t$,

- Each machine is assigned $N/M = 5$ particles to generate
- Each machine computes the weights for its 5 particles
- Each machine needs to do bootstrap – so it sends/receives weights to/from other machines
- Suppose particle $j = 3$ in machine 1 gets resampled 8 times and some particle $j = 7$ in machine 2 gets resampled twice
At $t + 1$,

- Machine 1 needs to generate 8 particles, machine 2 needs to generate 2 particles.
- Clearly, not good as we are overloading one machine.
- So we allocate three particles to be generated by machine 2.
- These three particles need to be extended from the particle $j = 3$.
- Recall that sample generation depends on the path of the particle so machine 1 needs to send the particle genealogy of $j = 3$ to machine 2.

Problem arises if particle sizes are large or if it grows proportionally to $t$. 
Outline

1. Intro to Sequential Monte Carlo (SMC)
2. The SMC algorithm
3. Entangled Monte Carlo
Why did I spend 3 slides on distribution of SMC algorithm?

My contribution to SMC literature is in extending SMC to be distributed over many computing nodes – call it Entangled Monte Carlo (EMC).

In essence, EMC avoids transmission of particles by reconstructing them from the particle genealogy rather than send/receive particle genealogy.
In any simulation, we rely on computers to generate random numbers. Alternative view of random number generation is to view it as a map,

\[ F : \mathcal{X} \rightarrow \mathcal{X} \]
In simulation such as MCMC with $T$ iterations, you can view it as sampling $T$ of these maps $F_1, F_2, \ldots, F_T$ independently and identically. Then, to compute the state of the Markov Chain after $T$ transitions, we would apply the map $T$ times to the initial state $x_0$,

$$F_T(F_{T-1}(\ldots(F_1(x_0))\ldots))$$

Here, you can think of the maps $F_t$ to be generating the random numbers used for computing the Metropolis-Hastings ratio.
We setup the notations first to help in discussing EMC,

- \( M \): the number of machines involved in the computation
- \( K \): the total number of particles involved in the computation
- \( K_m \): number of particles allocated to machine \( m \)
- \( \mathcal{F} \): set of stochastic maps
- \( \rho(j) \): parent of particle \( j \)
- \( s(j) \): sample of particle \( j \)
Note here that we made a distinction between a particle $j$ and its sample $s(j)$. 
Each machine $m$ has a global view of the computation – it knows the existence of all of the other particles $j = 1, 2, \ldots, N$ at every generation $t$.

However, it generates samples for only $K_m$ particles that it owns.

For these particles $s(j) \neq \text{nil}$.
Why did we distinguish the sample from the particle?

- A machine knows of all particles but it does not have samples of all of the particles stored in its memory.
- A machine views a particle $j$ as **concrete** if its sample $s(j)$ is stored in its memory. Otherwise, particle $j$ is viewed by machine $m$ as **compact** as $s(j) = \text{nil}$.
Here, \( M = 3, K = 9, K_m = 3, j = 1, 2, \ldots, K \). Black dots are concrete particles and the blue dots are compact particles from machine 1’s point of view.
The EMC algorithm from the viewpoint of machine $m$

1. Generate $K_m$ particles using the importance distribution
2. Compute the weights for each of $K_m$ particles
3. **Send/receive the weights to other machines - fast**
4. Resample (bootstrap) $N$ particles
5. **Allocate particles** - update the global genealogy
6. **Reconstruct particles** - if necessary
Communication of weights is unavoidable
It is fast however because the weights are at most 4 bytes
We have a way to reduce it down to $M$ weights rather than $N$ weights if communicating $N$ weights really becomes a problem
Each machine performs particle allocation greedily. That is, each machine tries to retain as many particles as it can to the next generation $t+1$ such that $s_t(j) \neq nil$. The surplus of particles can be allocated in many ways. We tried three methods,

- Random
- Most available
- Deterministic
Particle allocation may have resulted in a machine being assigned a compact particle. It needs to reconstruct a genealogy of this compact particle for the sample geneartion stage. We use the concept of stochastic maps for the reconstruction of genealogy.

- The stochastic maps are available globally before simulation begins because these maps are propagated from only one or two “seeds”
- Sharing of the initial seeds is easy as each machine needs to retrieve this information once at the beginning
Reconstruction algorithm

1: \( F \leftarrow I \)
2: \textbf{while} \( (s(i) = \text{nil}) \) \textbf{do}
3: \( F \leftarrow F \circ F_i \)
4: \( i \leftarrow \rho(i) \)
5: \textbf{end while}
6: \textbf{return} \( F(s(i)) \)

Note that this while loop terminates because the genalogies coalesce eventually.
Components decreases by one at every step. More precisely, we will build each rooted $X$-tree $t$ by proposing a sequence of $X$-forests $s_1, s_2, ..., s_R = t$, where an $X$-forest $s_r = \{(t_i, X_i)\}$ is a collection of rooted $X_i$-trees $t_i$ such that the disjoint union of leaves of the trees in the forest is equal to the original set of leaves, $\bigcup X_i = X$. Note that with this specific construction, a forest of rank $r$ has $|X| - r$ trees.

The sets of partial states considered in this Section are assumed to satisfy the following three conditions:

1. The sets of partial states of different ranks should be disjoint, i.e. $S_r \cap S_s$ for all $r \neq s$ (in phylogenetics, this holds since a forest with $r$ trees cannot be a forest with $s$ trees when $r \neq s$).

2. The set of partial state of smallest rank should have a single element denoted by $\bot$, $S_1 = \{\bot\}$ (in phylogenetics, $\bot$ is the disconnected graph on $X$).

3. The set of partial states of rank $R$ should coincide with the target space, $S_R = X$ (in phylogenetics, at rank $R = |X| - 1$, forests have a single tree and are members of the target space $X$).

These conditions will be subsumed by the more general framework of Section 4.5, but the more concrete conditions above help understanding the poset framework.

In order to grow particles from one rank to the next, the user needs to specify a proposal probability kernel $\nu^+$. Given an initial partial state $s$ and a set of destination partial states $A$, we denote the probability of proposing an element in $A$ from $s$ by $\nu^+ s (A)$. In the discrete case, we abuse the notation $14$.
Why it works I: CPU cycle is faster than network communication

Timing result comparing EMC and particle transmission for Phylogenetics experiment is shown below. The blue line is EMC and the red line is for SMC implemented with particle transmission.
Why it works II: Kingman’s coalescent

- The speed of the reconstruction depends on how deep the genealogy needs to be traced back.
- The Kingman’s coalescent theory gives approximation on expected number of waiting times for the coalescent of genealogies.
- Viewing the bootstrap stage as the Wright-Fisher model, the coalescence of genealogies is well-approximated when the $K$ is large [2].
Kingman’s coalescent example

Kingman’s coalescent theory approximates the expected time spent waiting for the last $k$ particles to coalesce as $(1 - 1/k)/(1 - 1/K)$.

For $k = 2$ and $K = 10,000$, $\frac{0.5}{0.9999} \approx 0.5$. So, it is expected that 9,998 particles will have finished coalescence before next half of the time is spent trying to coalesce the two remaining genealogies.
Phylogenetics inference using the kernel from [3] yielded great results,
Sample generation time via EMC and particle transmission,

Run time elapsed per generation

Time (milliseconds)

Generation
Applicable settings

- BOINC - Berkeley Open Infrastructure for Network Computing (http://boinc.berkeley.edu/)
- High performance computing cluster - consisting of multiple nodes each with hundreds of GPUs or CPU cores
- Amazon Web Services EC2 - each node with one CPU, can be combined to perform large scale simulation
Computational statistics - “aiming at the design of algorithm for implementing statistical methods on computers…” – taken from Wikipedia entry on “Computational Statistics”

So my contribution is to the field of computational statistics. My contribution is in allowing SMC to be implemented on multiple machines useful for big data and high dimensional problems.
If you are interested in finding out the details, this paper is to appear in the proceedings of NIPS 2012 conference.

Thank you!
References

