Memory (and Time) Efficient Sequential Monte Carlo

Seong-Hwan Jun SEONG.JUN@STAT.UBC.CA
The University of British Columbia, Vancouver, Canada

Alexandre Bouchard-Côté BOUCHARD@STAT.UBC.CA
The University of British Columbia, Vancouver, Canada

Abstract

Memory efficiency is an important issue in Sequential Monte Carlo (SMC) algorithms, arising for example in inference of high-dimensional latent variables via Rao-Blackwellized SMC algorithms, where the size of individual particles combined with the required number of particles can stress the main memory. Standard SMC methods have a memory requirement that scales linearly in the number of particles present at all stage of the algorithm. Our contribution is a simple scheme that makes the memory cost of SMC methods depends on the number of distinct particles that survive resampling. We show that this difference has a large empirical impact on the quality of the approximation in realistic scenarios, and also—since memory access is generally slow—on the running time.

The method is based on a two pass generation of the particles, which are represented implicitly in the first pass. We parameterize the accuracy of our algorithm with a memory budget rather than with a fixed number of particles. Our algorithm adaptively selects an optimal number of particle to exploit this fixed memory budget. We show that this adaptation does not interfere with the usual consistency guarantees that come with SMC algorithms.

1. Introduction

Sequential Monte Carlo (SMC) methods is a class of randomized algorithms that aim to approximate difficult expectations (Doucet et al., 2001), and that apply in a range of contexts traditionally approached using MCMC methods. SMC algorithms have several attractive properties, in particular they are well suited to online or real-time contexts, and they can be efficiently parallelized (Lee et al., 2010).

However, in contrast to MCMC algorithms, which are often bound by running time only, SMC methods are bound by both memory and time. This is because the storage needs of MCMC algorithms can typically be limited to finite-dimensional summary statistics of the parameters of interest, while SMC is required to keep all the particles of the current generation in memory in preparation for the resampling step.

Memory efficiency can become an important issue in at least two scenarios: in inference of high-dimensional latent variables where the number of particles required to achieve a desired approximation quality can exceed the main memory, and in online inference algorithms in embedded systems, where memory is severely constrained. In this work, we will focus on the first scenario.

More specifically, an important motivation for this paper comes from Rao-Blackwellized SMC algorithms, where variables are imputed in such as way that conditionally on the imputations, a high-dimensional variable can be marginalized. For example, in phylogenetics SMC algorithms (Teh et al., 2008; Görür & Teh, 2009; Bouchard-Côté et al., 2011), conditionally on an imputed tree, the sum-product algorithm can be used to marginalize over an exponential number of evolutionary paths of nucleotides evolving along the tree. In practice, it is important to compute the sum-product dynamic program incrementally, which means that each particle needs to store dynamic programming tables of significant size.

Our contribution is a simple scheme that makes the memory cost of SMC methods depends on the number of distinct particles that survive resampling. Since SMC particle weights can be highly non-uniform, the number of distinct particles after resampling can be substantially smaller than the number of particles before resampling.

At a high-level, we achieve these gains by generating par-

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

ticle populations via the following two-pass method. In a first pass, the particles and individual weights are created and discarded on the fly while finite-dimensional summary statistics on the weights are accumulated. In the second pass, an algorithm efficiently identifies the subset of resampled particles, which are then recreated by reusing the appropriate subset of random seeds. This second pass is inspired by tools used in SMC in a different context—network-efficient parallel SMC methods (Jun et al., 2012). We call our method Implicit Particle SMC (IPSMC) since the particles are represented implicitly in the first pass.[1]

The expected memory efficiency of our method is strictly superior to that of conventional SMC. In any given instance, it is never worse, and, as we demonstrate numerically, can be several orders of magnitude superior.

Under a naive time complexity analysis, the running time of our method is theoretically slower by a factor of two (since particles need to be recreated). Surprisingly, we found that in practice our method can often be substantially faster than a conventional SMC implementation. This is because fewer memory writes—an important time bottleneck in current computer architectures—are used by our method.

We parameterize the accuracy of our algorithm with a memory budget rather than with a fixed number of particles. Our algorithm adaptively selects an optimal number of particle to exploit this fixed memory budget. We show that this adaptation does not interfere with the usual posterior consistency guarantees that come with SMC algorithms. Using a random number of particles have been explored in recent work, but mainly for a different purposes, in particular, in cases where weights have a significant probability of being exactly zero (LeGland & Oudjane, 2006; Jasra et al., 2013).

In online contexts, there has been work on improving the memory efficiency of SMC algorithms (Schröter et al., 2007; Oreshkin et al., 2011; Eichstadt et al., 2012). For example, in the context of simultaneous localization and mapping, where each particle contains a hypothesized map, Schröter et al. (2007) compresses redundant information across the particle maps. However, these contributions are application and model-specific, and can be potentially difficult to implement. Our method in contrast does not make assumptions on the model or proposal distributions. In fact, the details of our memory-efficient algorithm need not be exposed to the user of the SMC library—the only requirement is that the proposal should be deterministic given a random seed.

An additional example of a situation where memory can become an issue is in certain state-space models where the pointwise computation of the conditional density of the hidden states $Z_t|Z_{t-1}$ is intractable. In other words, where only forward simulation of the hidden states is possible. In this context, the only option for the proposal is the prior over the hidden state dynamics, and it is difficult to exploit proposal distributions informed by the data, as this often requires the evaluation of the density of $Z_t|Z_{t-1}$ in the particle weight calculation (Doucet et al., 2001). In case of a large mismatch between the posterior and prior over the dynamics, a large number of particle may be required to get the desired accuracy.

A final example where the number of particles needed per generation is potentially large is SMC over combinatorial spaces (van Rensburg, 2009; Wang, 2012). In this particular case, it is not uncommon to see a substantial effective sample size collapse in a single SMC generation. Particle MCMC methods (PMCMC) (Andrieu et al., 2010) can in some cases be used to address this issue, but they only apply in offline settings. Even in the offline setting, the performance of particle MCMC methods are known to be sensitive to the trade-off between the number of particles and the number of MCMC iterations (Andrieu et al., 2010; Doucet et al., 2012). Fortunately, our scheme can be combined with particle MCMC when the optimal number of particles in PMCMC is too large for main memory storage.

2. Background

In this section, we setup the notation and provide a brief overview of standard SMC. See Section 1 and 2 of the Supplement for more details on SMC and on the measure-theoretic notation used in the proofs.

Suppose we are given a sequence of distributions of interest, π_r, $r = 1, 2, \ldots$. An important example is when the distribution π_r is the posterior distribution over Markovian latent states given the first r observations in a sequence, a setup known as the State Space Model (SSM).

Our goal is to approximate integrals of the form $\int \phi \, \mathrm{d}\pi_r$, for some function ϕ called a test function. For example, ϕ could be an indicator, in which case the integral become a conditional probability in our SSM example. The need for SMC arises when π_r is only accessible via an unnormalized density γ_r with an intractable normalization Z_r.

We will outline here a simple version of SMC where resampling is done at each step using multinomial resampling (Gordon et al., 1993). Our method applies to more general SMC methods, but we discuss these extensions in Section 5 and focus on the simplest case first.

At each generation $r = 1, 2, \ldots$, SMC approximates π_r via a list of K weighted point masses. We denote the

[1]Note that our scheme is unrelated to the work on implicit sampling for particle filters from the data assimilation literature in a stochastic differential equation context (Chorin & Tu, 2009).

weights of these point masses by $w_{r,k}$, and the normalized weights, by $\bar{w}_{r,k}$. We denote the corresponding locations of the point masses by $x_{r,k}$. Together, the locations and their weights form a discrete distribution $\pi_{r,K}$ which approximates π_r. Each iteration of the particle filter consists in first sampling from the normalized discrete distribution obtained from the previous generation (with replacement), and second, to use a proposal distribution ν_x and weight update formula to transform this old sample into a new sample suitable for the current generation. See the Section 1 of the Supplement for the form of the weight update formula (we will not modify this formula in IPSMC).

The estimator obtained at each step of this algorithm, $\int \phi \, \mathrm{d}\pi_{r,K}$, is a consistent estimator of $\int \phi \, \mathrm{d}\pi_R$ as $K \to \infty$ under regularity conditions (Del Moral, 2004). In practice, the accuracy of the estimation depends heavily on the value of K, which is restricted by the available memory. In large scale inference problems involving complex model structures, the size of the particles can become prohibitive for large value of K thereby impacting the quality of estimation.

3. Implicit Particle SMC

For exposition purpose, we start in Section 3.1 by describing a simplified version of our method. This first version nonetheless illustrate the first key idea used in IPSMC, *implicit particle propagation*, in which the memory requirement is still linear in the number of particles, but with a dramatically lower constant in the applications we are interested in.

We then introduce two other key ideas at the foundation of IPSMC: first, *particle streaming* (Section 3.2) which makes the memory requirement constant instead of linear in the number of implicit particles, and second, a simple adaptation scheme (Section 3.3) that provably preserves both the memory efficiency and the basic asymptotic properties.

3.1. Implicit particle propagation

As in standard SMC, IPSMC constructs a sequence of discrete distributions $\pi^{\mathrm{IP}}_{r,K}$ indexed by r (the generation). As before, each of these distributions is simply a list of weighted particles. We focus on explaining how to create the weighted particles of a single SMC generation assuming that the previous generation has been computed, as SMC and IPSMC are initialized in the same way.

In standard SMC algorithms, the computation of the weighted particles at each generation involves three tasks: resampling, followed by proposals, followed by normalization. In contrast, IPSMC depends on four tasks: expansion, proposals, normalization, contraction (see Figure 1). Moreover, the computations of these four tasks does not follow the same linear order as standard SMC (the proposal task is performed twice for reasons we will explain in detail shortly).

We now examine in detail how weighted particles are produced by the simplest version of IPSMC:

Expansion: The expansion step is a simple generalization of resampling. But instead of resampling from a list of K weighted particles K times, we resample $N > K$ times. Later, we will make N random (adaptive), but for now, think of N as a deterministic function of K, with values of N much larger than K.

Note that even if N is large, storing N resampled particles is feasible since it only involves storing K particles, each attached with a multiplicity.

Proposal (implicit pass): While the algorithm can use resampling to create a large number N of particles, naively proposing N times from these particles is impractical because the proposed particles will potentially all be distinct, preventing efficient storage via multiplicities and therefore breaking memory constraints.

Instead, IPSMC represents the N proposed particle *implicitly*: as in (Jun et al., 2012), at this stage IPSMC only stores the weight of the particle, a pointer to its parent, as well as the random seed used to produce it from its parent. This still requires linear storage in N, but with a potentially much lower constant. Again, this will be relaxed to constant storage using the more complicated IPSMC algorithm described in the next section.

Note that in our terminology, we incorporate the weight update calculation into the proposal step, so overall this task creates a collection of particles $((x_{r,n}, w_{r,n}) : 1 \leq n \leq N)$.

Normalization and contraction (implicit pass): Once the implicit particles are produced, we can normalize their weights and resample K times from the normalized weights $\bar{w}_{r,k}$. The actual contents $x_{r,n}$ of the particles is not needed for these stages.

Note that we will need in the next section the following decomposition of the normalization tasks into two subtasks: summing over the weights (computing $s_r = \sum_{n=1}^{N} w_{r,n}$, and dividing each weight by that sum to get $\bar{w}_{r,n} = w_{r,n}/s_r$.

Repeat Proposal (concrete pass): Now that the algorithm has determined which particles survived, it can reuse the relevant random seeds to create concrete versions of these particles in preparation for the next step.

3.2. Particle streaming

In this section, we modify the simplified version of IPSMC presented in the previous section to transform the linear memory requirement in N into a constant memory requirement with respect to N. The memory requirement is still linear in K, but we show in our experiments that large values for N can provide significant improvements in posterior quality, even for moderate K.

Note that in the previous version, each task (expansion, proposals, normalization, contraction) was performed in isolation (in other words, each one has its own loop over the particles). In contrast, particle streaming groups tasks (in other words, for each group of tasks, it loops over particles, and for each particle, execute all the tasks in the group of tasks for the current particle).

The first group of tasks contains the expansion task, the implicit proposal task, and the summation subtask. The second group of tasks contains the explicit proposal task, the division subtask, and the contraction task. See Figure 1 and Algorithm 1 in the Supplement. We need to show that each group can be performed in memory constant in N, while producing a result equivalent to the result of the algorithm presented in the previous section.

Task Group 1: streaming expansion, implicit proposal, summation. Expansion can clearly be done one particle at the time, by generating a discrete uniform random variable U on $\{1, \ldots, K\}$ and accessing the corresponding ancestor $x_{r-1,U}$ from the previous generation. Next, the implicit proposal creates a weight $w_{r,n}$, which we simply discard after adding it to an accumulator, $s \leftarrow s + w_{r,n}$. Note that we only need to store the seed of the sequence of proposals instead of individual seeds. See Algorithm 2 in the Supplement for details.

Task Group 2: streaming explicit proposal, division subtask, contraction. Here, we start by generating K sorted uniform number $V_{(1)}, \ldots, V_{(K)}$. Then, we regenerate particles using the same seed, but this time since we know the normalization $s_r = s$ in advance (from the previous task group), we can obtain the stream of normalized weights $\bar{w}_{r,n} = w_{r,n}/s$. We accumulate the sum $\bar{s}$ of normalized weights produced so far, $\bar{s} \leftarrow \bar{s} + \bar{w}_{r,n}$, and if the interval of length given by the current normalized weight $\bar{w}_{r,n}$ and translated by $\bar{s}$ contains one or more of the K sorted uniform numbers, the particle is kept as a concrete particle, otherwise, the particle is simply discarded as it will not be needed in subsequent iteration. See Algorithm 3 in the Supplement for details.

Note that the K sorted uniform numbers can be produced in a streaming fashion as well via Beta random variable (although they are less of a concern since K is small relative to N). See Section 3 of the Supplement.

3.3. Adaptive number of implicit particles

In this section, we show how to modify Task Group 1 to dynamically determine the number of particles needed to achieve a prescribed number of distinct particles after contraction. We also need to show that this criterion can be computed in a streaming fashion.

To start with, we remind the reader of the following elementary result on the expected number of distinct particles produced by multinomial resampling (for completeness, the proof is in Section 5 in the Supplement, where we also establish a concentration bound around this expectation):

Proposition 1 *Let* $S_1, \ldots, S_K \sim \mathrm{Mult}(\bar{w})$ *independently, where* $\bar{w} = (\bar{w}_1, \ldots, \bar{w}_N)$. *Then we have:*

$$\begin{aligned} \psi(w, K) &= \mathbb{E}|\{S_1, \ldots, S_K\}| \\ &= N - \sum_{i=1}^{N} (1 - \bar{w}_i)^K \end{aligned} \tag{1}$$

Let $w_1, w_2, \ldots$ denote a stream of unnormalized weights generated by repeated sampling from Task Group 1. We use the above function ψ to select the largest number of particles n that have the property that if we formed a multinomial with parameters $(\bar{w}_1, \ldots, \bar{w}_n)$ and resampled K times from it, then no more than M distinct particles would be obtained in expectation. Formally, for $K > M$, set:

$$N(K, M) = \sup\left\{n \leq N^* : \psi((w_1, \ldots, w_n), K) \leq M)\right\},$$

where $N^* \gg K$ is a computational ceiling also making sure that this stopping time is finite. Here M is a tuning parameter. The theoretical results of Section 3.5 suggest a value of $M = \alpha K$, where $\alpha = 1 - (1 - 1/K)^K \approx 1 - \exp(-1)$ (see Section 3.5 and Supplement 5 for details). We therefore write $N = N(K) = N(K, \alpha K)$. With this choice, M can be interpreted as the maximum expected number of unique particles stored after contraction (while the maximum realized number is K).

3.4. Binomial Approximation

We now show that the function ψ introduced in the previous section can be approximated in a particle streaming context. In other words, we show that the stopping time can be approximated accurately and efficiently using finite dimensional statistics computed incrementally from the stream of unnormalized weights.

Applying the binomial expansion of $(1 - \bar{w}_i)^m$ into Equation 1 yields:

$$N - \sum_{n=1}^{N} \sum_{k=0}^{K} \binom{K}{k} (-\bar{w}_n)^k \tag{2}$$

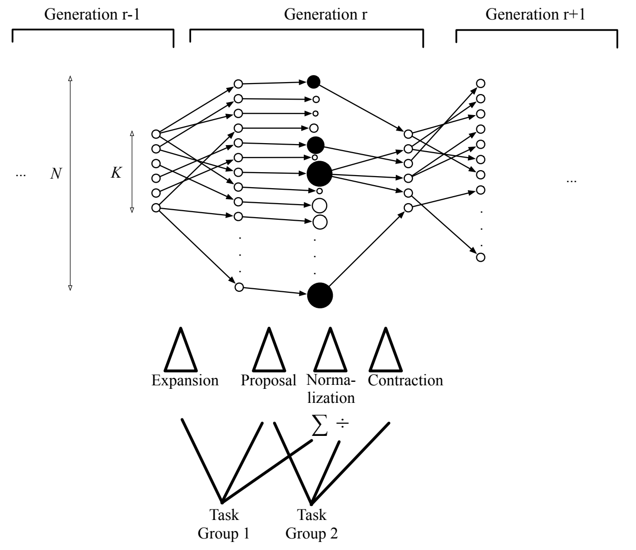


Figure 1: Overview of the implicit particle SMC algorithm. The circles are particles. The black particles show the subset of the implicit particles that need to be reconstructed from their seed in the concrete proposal task. The numbers attached to arrows indicate the number of particles. We also show the grouping of the tasks used by the stream version of the algorithm.

In practice, we observed that the accuracy of the approximation improved by maintaining a priority queue of bounded size, containing the largest weights. If Q denotes the priority queue, we divide the computation of Equation 1 into two parts:

$$N - \left(\sum_{n \in Q} (1 - \bar{w}_n)^K + \sum_{n \notin Q} \sum_{k=0}^{K} \binom{K}{k} (-\bar{w}_n)^k \right)$$

The weights in the priority queue are computed exactly whereas the weights that are not in the priority queue are approximated using binomial expansion.

Using the priority queue, we found that keeping the terms $k \in \{0, 1, 2\}$ gave satisfactory approximation with a priority queue of size 100 (see Section 4.4).

3.5. Consistency of the estimator

In this section, we give sufficient conditions for L_2 convergence of Monte Carlo approximate expectations of a test function ϕ computed from the Implicit Particle SMC algorithms as M goes to infinity.

The proof given in Section 5 of the Supplement is based on arguments used in previous work (Crisan & Doucet, 2002; Wang, 2012), the main point is to show that these arguments can be extended to handle the random number of particles used, and the additional resampling step used to control the number of unique particles.

Let $\pi^{\mathrm{IP}}_{r,K}$ denote the discrete approximation to the target distribution produced by the algorithm described in the previous sections (and formalized in terms of pseudo-code as well as measure-theoretic operators in the supplement).

Proposition 2 *Assume that the test function ϕ and the unnormalized weights w are bounded, and that the proposals ν satisfy $\pi_r \ll \nu_{r-1,x}$. Then:*

$$\int \phi \, \mathrm{d}\pi^{IP}_{r,K} \xrightarrow{\mathrm{L}^2} \int \phi \, \mathrm{d}\pi_r,$$

as $K \to \infty$.

This proposition could be extended in several ways. First, as previous results assuming similar hypotheses to our proposition, the constant obtained in the L_2 convergence bound depends exponentially on the number of particle generation. Other work has removed this exponential dependency at the cost of stricter mixing assumptions (Del Moral, 2004). Second, results on convergence almost sure and results based on less strict assumptions on ϕ are also available (Crisan & Doucet, 2002). We leave the investigation of possible adaptation of these results to our setup to future work.

The result in the present form nonetheless applies to practical situations, for example, to the bootstrap filter with a bounded observation density, and ϕ equal to an indicator function.

4. Experimental Results

4.1. Demonstration on a simple problem

We begin the experiments section with an illustration of the usefulness of our method on a simple example: the well known 2D Ising model. A state of an Ising model consists of a grid of L^2 nodes $X_i \in \{-1, +1\}$ for $1 \le i \le L^2$. The probability mass function of the target distribution is,

$$\pi_T(x) = \frac{1}{Z} \exp\left\{ \frac{1}{T} \sum_{i \sim j} x_i x_j \right\} \tag{3}$$

where $i \sim j$ indicates that node X_i and X_j are adjacent in the grid and Z denotes the partition function. The model is parameterized by a temperature parameter T, which specifies the strength of interaction between neighboring nodes.

We use the SMC samplers framework of Del Moral et al. (2007) to approach the problem with our algorithm. The proposal is given by a Gibbs move on half of the nodes

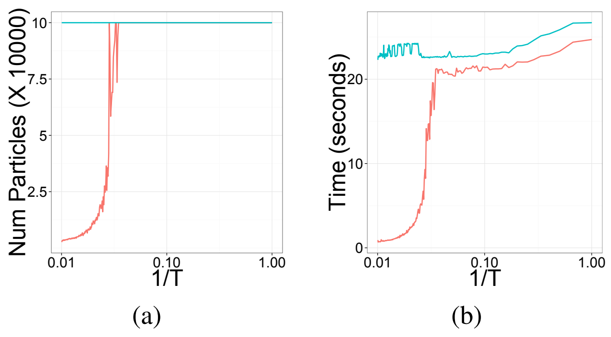


Figure 2: (a) The number of particles used, (b) the time (seconds) for each temperature for IPSMC (red) vs SMC (blue).

(those corresponding to the squares on a chessboard accessible by a bishop). The intermediate distributions π_r are given by annealed versions of the target distribution.

We consider the problem for $L = 32$ at the temperatures $T_{start} = 100$ to $T_{end} = 1$ with the annealing step size of 0.5. Note that at hot temperatures, the distribution over configurations is nearly uniform and it is easier to obtain good samples. At colder temperatures it is more difficult to obtain good samples; in fact, there is a critical temperature $T_0 \approx 2.27$, for which the inference problem becomes difficult as T gets closer to T_0 (Geyer, 1991).

The number of implicit particles used for different values of temperature is indicated by the red line in Figure 2 (a). It can be seen that our adaptive stopping time automatically reacts to the change in difficulty of the problem, and leverages a considerably larger number of particles at colder temperatures. The flat line shows the maximum number of particles that can be used in the same memory constraint (which was 1 gigabytes for this simple example) for the standard SMC algorithm. In Figure 2 (b), we show the actual time. As can be expected from the number of particles used, our algorithm is clearly faster at higher temperatures. We have computed an estimate of $P(X_1 = +1)$ for the two methods, which is shown in Figure 1 of the supplement. It can be seen there that both methods approach to the true value of 0.5. The results shown in this sections are average over 3 different runs initialized with different random seeds.

4.2. Phylogenetic experiments

Given a list of related sequences (taxa), an important problem in Bayesian phylogenetics is to compute a posterior distribution over *phylogenetic trees* describing the shared ancestry of these sequences (Felsenstein, 2003). Phylogenetic trees are composed of a combinatorial part (a discrete tree-shaped graph called a topology) and a continuous part (a length attached to each edge or branch in that tree).

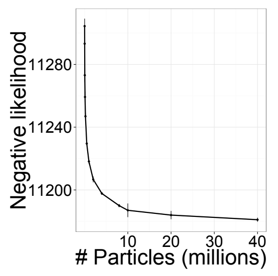
If a tree is fixed, it is possible to efficiently compute the probability of the observations under general evolutionary models such as the GTR model (Felsenstein, 2003). This is done as follows. To simplify exposition, assume first that all the sequences have length one. We are left with a tree-shaped graphical model with the structure given by the topology, and conditional probabilities on each branch of the tree computed via matrix exponentiation of a rate matrix times the length of that branch. It is therefore possible to efficiently marginalize the evolutionary paths on a fixed tree using the sum product algorithm. To handle the fact that sequences have length more than one, we assume they have been broken into one-character sets called sites (a process called multiple sequence alignment), and treat each of these sets independently.

The difficulty comes from the fact that the space of trees topologies grows exponentially in the number of taxa (Semple & Steel, 2003). We therefore use an SMC algorithm that imputes trees while marginalizing the evolutionary paths. We use the proposal algorithm described in (Teh et al., 2008), which starts from a fully disconnected forest over the taxa, picks one pair of trees in the forest at random, and forms a new tree by connecting their roots.

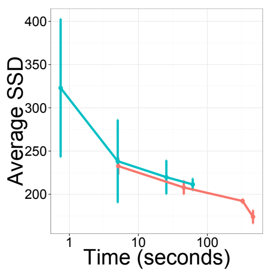
Inference in Bayesian phylogenetics using SMC is a good example where memory is more limiting than time. On one hand, it has been shown that SMC is very competitive when compared to MCMC on small to moderate trees (Bouchard-Côté et al., 2011). On the other hand, scaling to large number of taxa requires considerable memory needs.

In our experiments, we found that because of the storage required by the sum product dynamic programming tables, the memory per particle, for a dataset of 1000 sites is in the order of 100 kilobytes per particle. Note that studies considering multiple genes can involve one or several orders of magnitude more sites. At the same time, we show in Figure 3 (a) cases where tens of millions of particles are required to obtain a reasonable level of accuracy. Concretely, in our largest experiment in Figure 3 (a) the theoretical (extrapolated) memory need if we had used concrete instead of implicit particles would have been around 4 terabytes. In contrast, we were able to run most of these experiments on a laptop using 4 gigabytes of RAM.

To assess the quality of the approximation obtained by large numbers of implicit particles, we first looked at the estimate of the marginal negative log-likelihood as the number of particles is increased. The results over 5 runs are plotted in Figure 3 (a). This experiment was carried out on a simulated dataset of 20 taxa and 1000 sites. We explain data simulation steps in Section 6 of the supplement. It is visible from the sharp decline in the figure that a large

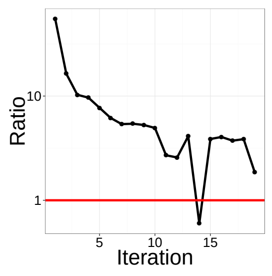


(a)

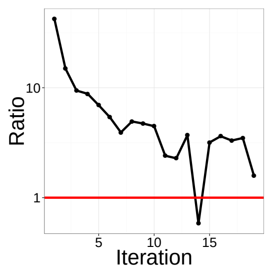


(b)

Figure 3: (a) Decrease in negative marginal likelihood as the number of particles increases. (b) Comparison of SSD versus time for IPSMC (red) and the standard SMC (blue).



(a)



(b)

Figure 4: Speed up result comparing IPSMC with (a) $K = 1000$ and (b) $K = 10000$ versus standard particle filter with $30,000$ particles. The red line indicates the ratio of 1.

number of particles is required for this problem.

In the next experiment, we consider the problem of reconstructing the ancestral relationships between the taxa by inferring the latent tree structure along with the branch lengths. For each pair of taxa, we estimate the pairwise distances; in this case we experimented on a simulated dataset involving 20 taxa and hence, there are $\binom{20}{2}$ such pairwise distances. The estimate of the pairwise distance for i, j is the weighted sum of the branch lengths of the path between i and j, denoted $\hat{d}_{ij}$. The sum of square of difference (SSD) of the estimate to the true value d_{ij} is computed as:

$$\text{SSD} = \sum_{i,j} (d_{ij} - \hat{d}_{ij})^2 \qquad (4)$$

We show in Figure 3 (b) the SSD plotted as time of execution increases. Note that the IPSMC (red) achieves lower error compared to the standard SMC (blue). In fact, the memory limit prevents the standard SMC from running longer (using more particles) to achieve lower error.

Next, we present some further investigations on the running time of our method, which under a naive time complexity analysis would be expected to be higher than that of the standard SMC algorithm. Surprisingly, we found this is not the case, a phenomenon we attribute to the standard particle filter requiring more memory writes. Note that both the implementation of IPSMC and the standard SMC were developed using the same code base and shared as much implementation as possible (in particular, both share the same representation of the particle, and the same proposal). Wall clock times can therefore be fairly compared, so we measured the Effective Sampling Size per second (ESS/s) (Kong et al., 1994).

We compared the speed up factor for IPSMC for $K \in \{1000, 10000\}$ against the standard SMC with 30000 particles (maximum allowed given 4 gigabytes of memory) by computing the ratio ESS/s of IPSMC to ESS/s of the standard SMC. IPSMC outperforms the standard SMC even with a smaller memory budget as shown in Figure 4 (except at iteration 14).

4.3. Nonlinear State Space Model

Another case where IPSMC can be useful is when the proposal is far from the posterior, for example as a part of an automatic Bayesian inference software package (Todeschini & Caron, 2012), where the main option available to improve approximation quality is to increase the number of particles. We demonstrate this use case on a simple nonlinear state space model where the prior distribution is used as the proposal (Kitagawa, 1987):

$$X_1 \sim \mathcal{N}(0, 5)$$

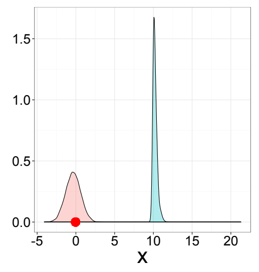
$$X_r = \frac{X_{r-1}}{2} + 25\frac{X_{r-1}}{1 + X_{r-1}^2} + 8\cos(1.2r) + V_r$$

$$Y_r = \frac{X_r^2}{20} + W_r$$

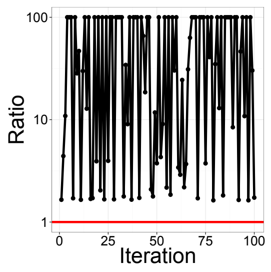
where $V_r \sim \mathcal{N}(0, \sigma_V^2)$ and $W_r \sim \mathcal{N}(0, \sigma_W^2)$. We carried out a simple experiment using $K = 10,000$ as the memory budget for both IPSMC and SMC, and $N^* = 1$ million as the computation ceiling for IPSMC. We examined the density estimated by the particles for $R = 100$ using $\sigma_V^2 = \sigma_W^2 = 1$. The result is shown in Figure 5 (a). The red dot indicates the true value x_{100}, which shows that even with poor choice of proposal the density estimate is accurate for IPSMC compared to SMC under the same memory budget. In Figure 5 (b), we show the ratio of the number of implicit particles to K. The red line indicates the number of particles that have been used by the standard SMC.

4.4. Binomial Approximation

The runtime of IPSMC depends on determining the stopping time efficiently. In Section 3.4, we presented a bi-



(a)



(b)

Figure 5: (a) The density estimate of IPSMC (red) versus SMC (blue) where the red dot indicates the true value and (b) the ratio of the # of implicit particles to K.

nomial approximation that requires only finite-dimensional summary statistics. In this section, we demonstrate the accuracy of this approximation. At the end of each proposal step, we computed the exact value of the expectation and compared it to the binomial approximation using the first 2, 4, and 8 moments (in particular, using two terms corresponds to recording only the sum and the sum of the squares). In each entry of the table, we computed the average of the absolute deviation from the exact value and divided by K. We observed that as K increases, more terms are needed to obtain an accurate approximation (see Table 1), but at a relatively slow rate. Increasing the number of terms did not hinder the speed of execution. The size of the priority queue was set at 100 in all of the experiments.

# of terms	2	4	8
Nonlinear SSM ($K = 1000$)	0.11450	0.01450	0.00004
Nonlinear SSM ($K = 5000$)	0.13022	0.02340	0.00120
Phylogenetics ($K = 1000$)	0.12181	0.01595	0.00003

Table 1: Accuracy of the binomial approximation.

5. Discussion and extensions

In this work, we have introduced an algorithm that performs SMC simulation in a space-efficient manner, and we analyzed its theoretical and empirical behavior. We have explained the algorithm in the context of a simple SMC setup, but we now discuss potential extensions to more complex scenarios.

The first such extension consists in avoiding to do resampling at each stage. If the interval between resampling steps is a constant c, this can be achieved by replacing the (one-step) proposal by a c-steps proposals. Note that the c-steps proposal is no harder to re-generate than the one-step proposal, again, the user simply has to reuse the seed of this sequence of c proposals. If the interval between resampling steps is adaptive (for example, via effective sample size monitoring (Doucet & Johansen, 2009)), extending our method is not as direct, but still possible. One could for example use the above idea sequentially with c ranging from $2^0, 2^1, 2^2, 2^3, \ldots$ until the adaptive resampling criterion is met (as long as this criterion can be checked from a stream of weights). Pragmatically, one could also simply use a small pilot run to determine the value of c.

Second, while we have described the algorithm in a simple context of the bootstrap sampler motivated by SSM, there are several important extensions to consider in practice. For one, nothing prevents using IPSMC as the implementation of general SMC samplers (Del Moral et al., 2006). The only difference comes in the weight update, which potentially requires a correction coming from a backward proposal. See (Del Moral et al., 2006) for details. This means that SMC extensions that can be interpreted as auxiliary variables augmentations can be combined with our method. This is the case for example (Johansen & Doucet, 2008) with Auxiliary Particle Filters (Pitt & Shephard, 1999).

Third, when N^* is moderate, the resampling step can also be replaced by alternative schemes (Douc & Cappé, 2005), by using the first version of IPSMC presented in the previous section. We leave the extension to other resampling schemes in the streaming version of IPSMC for future work.

Finally, note that we have designed our particle generation stopping times to terminate as soon as memory is used optimally. Empirically, this gives a good combination of optimized memory usage while preserving fast running time. If running time is less of an issue however, it is possible that the approximation quality can be further improved by using higher stopping values.

This leads to a related issue, regarding the practicality of SMC asymptotic results taking a memory limit K to infinity. Since memory is bounded in practice, this type of asymptotics is arguably less practical than MCMC asymptotic results, where the variable going to infinity represents time. Another type of asymptotics, more natural in the IPSMC case, would consist in letting the number of implicit generated particles N going to infinity, while still resampling a finite number of them to fit into memory. It would be interesting to translate the related results of Douc & Moulines (2008) that are closer in spirit, but did not consider the memory constrained setup.

Acknowledgments

We thank the reviewers for the detailed feedback. This work was partially funded by NSERC Discovery Grant and a Google Faculty Award. Computing was supported by WestGrid.

References

Andrieu, C., Doucet, A., and Holenstein, R. Particle Markov chain Monte Carlo methods. *J. R. Statist. Soc. B*, 72(3):269–342, 2010.

Bouchard-Côté, A., Sankararaman, S., and Jordan, M. I. Phylogenetic inference via Sequential Monte Carlo. *Systematic Biology*, 2011.

Chorin, Alexandre J and Tu, Xuemin. Implicit sampling for particle filters. *Proceedings of the National Academy of Sciences*, 106(41):17249–17254, 2009.

Crisan, D. and Doucet, A. A survey of convergence results on particle filtering for practitioners. *IEEE Trans. Signal Processing*, 50(3):736–746, 2002.

Del Moral, P. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Springer, New York, 2004.

Del Moral, P., Doucet, A., and Jasra, A. Sequential Monte Carlo samplers. *Journal of The Royal Statistical Society Series B-statistical Methodology*, 68(3):411–436, 2006.

Del Moral, P., Doucet, A., and Jasra, A. Sequential Monte Carlo for Bayesian computation. *Bayesian Statistics*, 8:1–34, 2007.

Douc, R. and Cappé, O. Comparison of resampling schemes for particle filtering. In *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis.*, pp. 64–69. IEEE, 2005.

Douc, R. and Moulines, E. Limit theorems for weighted samples with applications to sequential Monte Carlo. *Ann. Statist.*, 36 (5):2344–2376, 2008.

Doucet, A. and Johansen, A. M. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12:656–704, 2009.

Doucet, A., de Freitas, N., and Gordon, N. *Sequential Monte Carlo methods in practice*. Springer, 2001.

Doucet, A, Pitt, M, and Kohn, R. Efficient implementation of Markov chain Monte Carlo when using an unbiased likelihood estimator. *arXiv preprint arXiv:1210.1871*, 2012.

Eichstadt, S., Link, A., Harris, P., and Elster, C. Efficient implementation of a Monte Carlo method for uncertainty evaluation in dynamic measurements. *Metrologia*, 49:401–410, 2012.

Felsenstein, J. *Inferring phylogenies*. Sinauer Associates, 2003.

Geyer, C. J. Markov chain Monte Carlo maximum likelihood. In *Proceedings of 23rd Symposium on the Interface of Computing Science and Statistics*, pp. 156–163, 1991.

Gordon, N., Salmond, D., and Smith, A. F. M. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2):107–113, Apr 1993.

Görür, D. and Teh, Y. W. An efficient sequential Monte Carlo algorithm for coalescent clustering. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

Jasra, A., Lee, A., Yau, C., and Zhang, X. The alive particle filter. *arXiv*, 2013.

Johansen, A.M. and Doucet, A. A note on auxiliary particle filters. *Statistics and Probability Letters*, 78:1498–1504, 2008.

Jun, SH., Wang, L., and Bouchard-Côté, A. Entangled Monte Carlo. In *Advances in Neural Information Processing Systems 25 (NIPS)*, 2012.

Kitagawa, G. Non-Gaussian statespace modeling of nonstationary time series. *Journal of the American Statistical Association*, 82 (400):1032–1041, 1987.

Kong, A., Liu, J. S., and Wong, W. H. Sequential imputations and Bayesian missing data problems. *Journal of the American Statistical Association*, 89(425):278–288, 1994.

Lee, A., Yau, C., Giles, M. B., Doucet, A., and Holmes, C. C. On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. *Journal of Computational and Graphical Statistics*, 19(4):769–789, 2010.

LeGland, F. and Oudjane, N. A sequential particle algorithm that keeps the particle system alive. *Lecture Notes in Control and Information Science*, 337:351–389, 2006.

Oreshkin, B.N., Liu, X., and Coates, M.J. Efficient delay-tolerant particle filtering. *Signal Processing*, 59:3369–3381, 2011.

Pitt, M.K. and Shephard, N. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–591, 1999.

Schröter, C., Böhme, H.-J., and Gross, H.-M. Memory-efficient gridmaps in Rao-Blackwellized particle filters for SLAM using sonar range sensors. In *European Conference on Mobile Robots*, volume 3, pp. 138–143, 2007.

Semple, C. and Steel, M. *Phylogenetics*. Oxford, 2003.

Teh, Y. W., Daumé III, H., and Roy, D. M. Bayesian agglomerative clustering with coalescents. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

Todeschini, A. and Caron, F. Bayesian inference with interacting particle systems. `https://alea.bordeaux.inria.fr/biips/doku.php`, 2012.

van Rensburg, E J Janse. Monte carlo methods for the self-avoiding walk. *J. Phys. A: Math. Theor.*, 42:323001, 2009.

Wang, L. *Bayesian phylogenetic inference via Monte Carlo methods*. PhD thesis, University of British Columbia, 2012.