Lee, A., & Whiteley, N. (2016). Forest Resampling for Distributed Sequential Monte Carlo. Statistical Analysis and Data Mining, 9(4), 230-248. DOI: 10.1002/sam.11280

Forest resampling for distributed sequential Monte Carlo

Anthony Lee* Nick Whiteley†

May 31, 2015

Abstract: This paper brings explicit considerations of distributed computing architectures and data structures into the rigorous design of Sequential Monte Carlo (SMC) methods. A theoretical result established recently by the authors shows that adapting interaction between particles to suitably control the Effective Sample Size (ESS) is sufficient to guarantee stability of SMC algorithms. Our objective is to leverage this result and devise algorithms which are thus guaranteed to work well in a distributed setting. We make three main contributions to achieve this. Firstly, we study mathematical properties of the ESS as a function of matrices and graphs that parameterize the interaction amongst particles. Secondly, we show how these graphs can be induced by tree data structures which model the logical network topology of an abstract distributed computing environment. Thirdly, we present efficient distributed algorithms that achieve the desired ESS control, perform resampling and operate on forests associated with these trees.

Keywords: data structures; distributed computing; effective sample size; particle filters

1 Introduction

SMC algorithms are interacting particle methods for approximating sequences of distributions arising in statistics, and are commonly applied to Hidden Markov Models (HMM's) for filtering and marginal likelihood estimation (see, e.g., [14, 13]). We focus here on this HMM setting for simplicity, although our methodology is relevant to other SMC schemes, such as [5], [10] and [6]. It is becoming increasingly important that computationally intensive algorithms are suited to implementation on many-core computing architectures (see, e.g., [30]), and it is well established that standard SMC algorithms naturally have this property (see, e.g., [24]). In particular, the time in which such algorithms run on many-core devices is typically sublinear in the number of particles, N, until N reaches a device- and application-specific critical size, resulting in significant performance improvements for moderate numbers of particles. However, the number of particles required for acceptable accuracy in various settings can be substantially larger than this critical size. In order to provide accurate estimates in these situations in a timely fashion, attention is naturally drawn to distributed implementations of SMC algorithms, in which particles are distributed over multiple devices which can communicate over a network (see, amongst others, [3, 18, 31]). In this environment the interactions between particles, which provide fundamental stability properties of the algorithm, are costly due to relatively slow network speeds in comparison to fast on-device memory accesses.

Motivated by the desire to develop Monte Carlo algorithms whose communication structure is more naturally suited to distributed architectures, Whiteley et al. [34] proposed and studied a generalization of standard SMC algorithms, called αSMC, in which interaction between particles may be modulated in an on-line fashion. The "α" in αSMC refers to certain matrices which are chosen adaptively as the algorithm runs, dictating or constraining this interaction. A special case of αSMC is the popular adaptive resampling strategy originally proposed by Liu and Chen [26]. One of the main results of [34] is a stability theorem which shows that, subject to regularity conditions on the HMM, adapting α so as to enforce an appropriate lower bound on the ESS is sufficient to ensure time-uniform convergence of αSMC filtering estimates, and endow it with other attractive theoretical properties so that the computational cost of the algorithm grows manageably with the length of the data record. This provides a criterion for stabilization of these algorithms when communication constraints influence interaction.

Monitoring and controlling the ESS using α matrices is therefore very important. However, if implemented naively, this monitoring and control itself involves collective operations on the entire particle system, and so

*Department of Statistics, University of Warwick
†School of Mathematics, University of Bristol

remains as an obstacle to parallelization. In this paper, our overall aim is to address this obstacle and formulate approaches to ESS control which are more appropriate for distributed implementation. In order to do so, we consider a logical tree topology which represents an abstract distributed computing environment. This network structure accommodates divide-and-conquer routines and recursive programming, making it suited to distributed computation, and its hierarchical nature lends itself to partitioning and resampling operations. We consider methods of ESS control involving computations which are *local* with respect to the topology of these trees.

After outlining αSMC in Section 2, our first original contribution in Section 3 is a study of the ESS itself, as a functional of the α matrix governing interaction. This study leads us to consider a subset of potential α matrices with a specific associated graphical structure. We then define a partial order on this set of matrices, which makes precise a sense in which they are more or less suited to distributed architectures, and prove that the ESS is (partial) order-preserving. This important relationship connects computational considerations with statistical performance and informs our algorithm design. Section 3 culminates in a lower bound on the ESS phrased in terms of particle sub-populations, and applied recursively this bound leads to an abstract recursive algorithm for enforcing a lower bound on the population-wide ESS. Crucially, each recursive call of this algorithm can require the consideration of only a small number of aggregated weights, and this is what makes it suited to distributed architectures. Section 4 is devoted to practical implementation of this abstract recursive algorithm in a distributed setting using trees, in such a way that all quantities required are available via local computations whose cost is independent of N. An interpretation of the resulting resampling scheme is that it corresponds to a tree sampling procedure involving a number of disjoint trees, and so we term the overall procedure forest resampling. All proofs are given in the appendix.

2 αSMC

2.1 A hidden Markov model

In this section we overview relevant aspects of the general methodology proposed in [34]. An HMM with measurable state space $(\mathsf{X}, \mathcal{X})$ and observation space $(\mathsf{Y}, \mathcal{Y})$ is a process $\{(X_n, Y_n); n \geq 0\}$ where $\{X_n; n \geq 0\}$ is a Markov chain on X, the observations $\{Y_n; n \geq 0\}$, valued in Y, are conditionally independent given $\{X_n; n \geq 0\}$, and the conditional distribution of each Y_n depends on $\{X_n; n \geq 0\}$ only through X_n. Let π_0 and f be respectively a probability distribution and a Markov kernel on $(\mathsf{X}, \mathcal{X})$, and let g be a Markov kernel acting from $(\mathsf{X}, \mathcal{X})$ to $(\mathsf{Y}, \mathcal{Y})$, with $g(x, \cdot)$ admitting a density, denoted similarly by $g(x, y)$, with respect to some dominating σ-finite measure. The HMM specified by π_0, f and g, is

$$\begin{aligned} &X_0 \sim \pi_0, \\ &X_n \mid \{X_{n-1} = x_{n-1}\} \sim f(x_{n-1}, \cdot), \quad n \geq 1, \\ &Y_n \mid \{X_n = x_n\} \sim g(x_n, \cdot), \quad n \geq 0. \end{aligned} \tag{1}$$

Throughout this paper we consider a fixed observation sequence $\{y_n; n \geq 0\}$ and write

$$g_n(x) := g(x, y_n), \quad n \geq 0. \tag{2}$$

We also work under the mild assumption that for each $n \geq 0$, $\sup_{x \in \mathsf{X}} g_n(x) < +\infty$ and $g_n(x) > 0$ for all $x \in \mathsf{X}$.

For $n \geq 1$, let π_n be the conditional distribution of X_n given $Y_{0:n-1} = y_{0:n-1}$, called the *prediction filter*; and let Z_n be the marginal likelihood of the first n observations, evaluated at the point $y_{0:n-1}$. Due to the conditional independence structure of the HMM the following recursions hold:

$$\pi_n(A) = \frac{\int_{\mathsf{X}} \pi_{n-1}(dx)\, g_{n-1}(x) f(x, A)}{\int_{\mathsf{X}} \pi_{n-1}(dx)\, g_{n-1}(x)}, \quad A \in \mathcal{X},\ n \geq 1,$$

and

$$Z_n = Z_{n-1} \int_{\mathsf{X}} \pi_{n-1}(dx)\, g_{n-1}(x), \quad n \geq 1,$$

with the convention $Z_0 := 1$. Our main computational objectives are to approximate $\{\pi_n; n \geq 0\}$ and $\{Z_n; n \geq 0\}$.

2.2 The αSMC algorithm

We write $[M] := \{1, \ldots, M\}$ for a generic $M \in \mathbb{N}$. We denote by N an arbitrary but fixed positive integer representing the number of particles in the algorithm we are about to describe. To simplify presentation, whenever a summation sign appears without the summation set made explicit, the summation set is taken to be $[N]$, for example we write Σ_i to mean $\Sigma_{i=1}^N$. Finally, let $\mathbb{A}_{[N]}$ be the set of doubly stochastic matrices of size $N \times N$ (this is a special case of the setup of [34], corresponding to their assumption $(\mathbf{B}^{++})$).

The αSMC algorithm simulates a sequence $\{\zeta_n; n \geq 0\}$ with each $\zeta_n := (\zeta_n^1, \ldots, \zeta_n^N)$ valued in X^N. When $n \geq 1$,

this involves choosing a matrix α_{n-1} from $\mathbb{A}_{[N]}$ according to some deterministic function of $\{\zeta_0, \ldots, \zeta_{n-1}\}$, and this matrix specifies the type of interaction that occurs at time n. In particular, whenever $\alpha^{ij}_{n-1} = 0$ this implies that both the particle ζ^i_n and its weight W^i_n are sampled/calculated independently of ζ^j_{n-1}, and so zero entries in α_{n-1} encode a lack of interaction.

Algorithm 1 αSMC

For $n = 0$,
  For $i = 1, \ldots, N$,
    Set $W^i_0 = 1$.
    Sample $\zeta^i_0 \sim \pi_0$.
For $n \geq 1$,
($\star$) Select α_{n-1} from $\mathbb{A}_{[N]}$ as a function of $\{\zeta_0, \ldots, \zeta_{n-1}\}$
  For $i = 1, \ldots, N$,
(†) Set $W^i_n = \sum_j \alpha^{ij}_{n-1} W^j_{n-1} g_{n-1}(\zeta^j_{n-1})$.
(‡) Sample

$$\zeta^i_n | \zeta_0, \ldots, \zeta_{n-1} \sim \frac{\sum_j \alpha^{ij}_{n-1} W^j_{n-1} g_{n-1}(\zeta^j_{n-1}) f(\zeta^j_{n-1}, \cdot)}{\sum_j \alpha^{ij}_{n-1} W^j_{n-1} g_{n-1}(\zeta^j_{n-1})}$$

With δ_x denoting the Dirac measure centred on x, the objects

$$\pi^N_n := \frac{\sum_i W^i_n \delta_{\zeta^i_n}}{\sum_i W^i_n}, \quad Z^N_n := \frac{1}{N} \sum_i W^i_n, \quad n \geq 0, \tag{3}$$

are regarded as approximations of π_n and Z_n, respectively. The validity of the weight updates in Algorithm 1 and general theoretical properties of the approximations π^N_n and Z^N_n are the focus of [34].

In general, some algorithm design is involved at line ($\star$) of Algorithm 1; one has to decide on a rule which dictates how α_{n-1} is chosen from $\mathbb{A}_{[N]}$, and in practice one will often select α_{n-1} from $\mathbb{A}_{[N]}$ as some function of $\left(W^1_{n-1}, \ldots, W^N_{n-1}\right)$ and $\left(g_{n-1}(\zeta^1_{n-1}), \ldots, , g_{n-1}(\zeta^N_{n-1})\right)$. Given the lack of interaction encoded by zero entries of α_{n-1}, there is also great practical interest in the situation where α_{n-1} is chosen to be a sparse matrix. This is the primary focus of this paper; a brief discussion of existing choices of ($\star$) follows a key stability result from [34] that ultimately motivates our developments here.

2.3 The ESS and stability of αSMC

Exactly how α_{n-1} is chosen in line ($\star$) of Algorithm 1 is the only degree of freedom in an implementation of αSMC. Consequently, one of the main contributions of [34] is a stability theorem that provides strong guidance for designing ($\star$) in practice. Central to this stability theorem is the effective sample size (ESS), a measure of the Monte Carlo accuracy of a weighted set of samples introduced in [22], see also [25, Section 4]. The ESS associated with the weights $\left\{W^i_n : i \in [N]\right\}$ is

$$N^{\text{eff}}_n := \frac{\left(\sum_i W^i_n\right)^2}{\sum_i \left(W^i_n\right)^2}. \tag{4}$$

The theorem, reproduced below, gives a rigorous theoretical justification for enforcing a lower bound on N^{eff}_n when selecting α_{n-1}. It relies on the following regularity condition on the HMM, which is often used to establish stability results for non-adaptive SMC algorithms (see, e.g., [9, 4, 33], and also [32] for stability under weaker conditions).

Assumption. (**C**) *There exists* $(\delta, \epsilon) \in [1, \infty)^2$ *such that*

$$\sup_{n \geq 0} \sup_{x,y} \frac{g_n(x)}{g_n(y)} \leq \delta, \qquad f(x, \cdot) \leq \epsilon f(y, \cdot), \quad (x, y) \in \mathsf{X}^2.$$

For μ a measure on $(\mathsf{X}, \mathcal{X})$ and φ a real-valued, $\mathcal{X}$-measurable function on X we define $\mu(\varphi) := \int_{\mathsf{X}} \varphi(x) \mu(dx)$, allowing us to compare π^N_n with π_n via the differences $\pi^N_n(\varphi) - \pi_n(\varphi)$, for suitable φ. For example, when $A \in \mathcal{X}$ and $\varphi = \mathbf{1}_A$ then $\pi_n(\varphi)$ is the conditional probability that $X_n \in A$ given $Y_{0:n-1} = y_{0:n-1}$ and $\pi^N_n(\varphi)$ its αSMC estimate.

Theorem. *[34, Theorem 2] Assume* (**C**). *Then there exist finite constants* c_1 *and for any* $r \geq 1$, $c_2(r)$, *such that for any* $N \geq 1$ *and* $\tau \in (0, 1]$, *if*

$$\inf_{n \geq 0} N^{\text{eff}}_n \geq N\tau, \tag{5}$$

then

$$\sup_{n \geq 1} \mathbb{E}\left[\left(\frac{Z^N_n}{Z_n}\right)^2\right]^{1/n} \leq 1 + \frac{c_1}{N\tau}, \tag{6}$$

and for any $\varphi : \mathsf{X} \to \mathbb{R}$ *which is* $\mathcal{X}$*-measurable and bounded,*

$$\sup_{n \geq 0} \mathbb{E}\left[\left|\pi^N_n(\varphi) - \pi_n(\varphi)\right|^r\right]^{1/r} \leq \|\varphi\|_\infty \frac{c_2(r)}{\sqrt{N\tau}}. \tag{7}$$

Importantly, the condition (5) can be ensured in practice without additional simulation. Indeed, from line (†) of Algorithm 1 and (4) together, we see that N^{eff}_n clearly depends on α_{n-1}, $\left\{W^i_n : i \in [N]\right\}$ and ζ_{n-1} but *not* on ζ_n. Therefore, α_{n-1} can be selected adaptively in ($\star$) to ensure that N^{eff}_n exceeds some threshold *before* ζ_n is simulated.

2.4 Existing αSMC algorithms

Two members of $\mathbb{A}_{[N]}$ used implicitly in methods predating αSMC are: $\mathbf{1}_{1/N}$, the $N \times N$ matrix which has $1/N$ as every entry; and Id, the identity matrix. If $\alpha_n = \mathbf{1}_{1/N}$ for every n, αSMC reduces to the bootstrap particle filter (BPF), whereas if $\alpha_n = Id$ for every n, αSMC reduces to sequential importance sampling (SIS). The BPF involves every particle interacting with every other particle in line (‡), in the particular sense that the distribution for each ζ_n^i is a mixture of the form $\sum_j s_j f(\zeta_{n-1}^j, \cdot)$ where each $s_j > 0$. Furthermore, the BPF will always satisfy (5) with $\tau = 1$ since $W_n^i = W_n^j$ for any $i, j \in [N]$ when $\alpha_{n-1} = \mathbf{1}_{1/N}$. In contrast, in SIS there is no interaction in this same sense, as $\zeta_n^i \sim f(\zeta_{n-1}^i, \cdot)$ for each $i \in [N]$ and in general (5) will not be satisfied for a given $\tau > 0$. An alternative existing αSMC algorithm, which does enforce (5) is the adaptive resampling particle filter (ARPF) of [26]. In particular, the ARPF chooses, at each time n, α_{n-1} to be either Id if the resulting ESS of $(W_n^i)_{i\in[N]}$ would be above a prespecified threshold and $\mathbf{1}_{1/N}$ otherwise. Hence there is either no interaction or full interaction at each time. We refer the reader to [34, Section 2.2] for further details of these relationships.

Whiteley et al. [34, Section 5.3] suggested some procedures for adaptively selecting α_{n-1} from $\mathbb{A}_{[N]}$ at line (⋆). Empirical results using these procedures for a given threshold τ in (5) indicated that while the communication cost incurred by the ARPF is lower than that of the BPF, it can be substantially reduced by allowing more general choices of α_{n-1} that are between Id and $\mathbf{1}_{1/N}$ in terms of sparsity, and therefore that αSMC is a promising avenue for implementation in a distributed setting. However, a practical issue concerning these adaptive procedures is that guaranteeing (5) involves evaluating the ESS for some candidate α_{n-1}'s, and this task may itself be demanding in terms of communication cost. Indeed, if one wishes to search through a large set of candidates for α_{n-1}, e.g. when attempting to guarantee (5) with as sparse an α_{n-1} as possible, the cost of step (⋆) may dominate the overall cost of Algorithm 1. On the other hand, the ARPF involves only the two candidates Id and $\mathbf{1}_{1/N}$; evaluating the ESS for the candidate Id can be done cheaply, and if $\alpha_{n-1} = \mathbf{1}_{1/N}$, then we always have $N_n^{\text{eff}} = N$, so step (⋆) is inexpensive. However, if $\alpha_{n-1} = Id$ does not achieve $N_n^{\text{eff}} \geq N\tau$ there is no choice but to set $\alpha_{n-1} = \mathbf{1}_{1/N}$, and one then incurs the communication cost associated with the resulting population-wide interaction at step (‡).

2.5 Objective of the paper

In this paper we will investigate practical methods to carry out the adaptive selection in (⋆) as well as steps (†) and (‡), with α_{n-1} chosen from a large family of matrices of varying levels of sparsity. Our main objective is to design instances of αSMC which guarantee (5) and for which (6) and (7) are therefore guaranteed, whilst achieving a desirable balance between the communication costs associated with steps (⋆) and (‡).

We emphasize that Algorithm 1 is not intended to be implemented explicitly, since this would involve manipulation of large $N \times N$ matrices. For example, SIS, the BPF and the ARPF all perform the steps in Algorithm 1 implicitly. We also aim to avoid here a situation in which all particle weights need to be sent to a central processor in order for (⋆) to be implemented, which was not considered in [34].

While the selection algorithms proposed here will be first devised in terms of the α matrices involved, ultimately all procedures will take the form of recursive, divide-and-conquer algorithms without any α matrices being constructed explicitly. This type of algorithm is well-suited to implementation in a distributed computing environment with a tree network structure [29, Chapter 7]. To help us understand how we can achieve (5) using sparse α_{n-1}, but without excessive communication, we proceed with an investigation of the ESS.

3 Properties of the ESS

In Section 2 it was seen that the ESS of the time n weights can be seen as a function of the stochastic matrix α_{n-1}, following (†) and (4). In this section, we study a generalization of the ESS as a function of an input sparse substochastic $N \times N$ matrix a. This generalization allows us to make statements in Section 3.1 about the ESS of $a + a'$ where both a and a' are substochastic matrices whose non-zero entries do not overlap. In Section 3.2 we then consider graphs that induce a subset of such substochastic matrices, namely disjoint unions of complete graphs. These graphs naturally encode an interaction or communication structure for the particles in αSMC and can be used to define a partial order over the substochastic matrices introduced in Section 3.1. Importantly, the generalized ESS is then shown to be order-preserving with respect to this partial order. In Section 3.3 a lower bound on the generalized ESS is obtained whose calculation can be performed recursively with only a small amount of "local" information being communicated, and which directly

leads to a recursive algorithm for specifying step ($\star$) of Algorithm (1) that relies only on "local" information being available in each recursive call.

3.1 Dependence of the ESS on α

Slightly extending our notation, for each non-empty $V \subseteq [N]$ let $\mathbb{A}_V$ be the set of all substochastic $N \times N$ matrices a with the following properties:

1. a leaves the uniform distribution on V invariant,
2. $a^{ij} = 0$ whenever $(i,j) \in [N]^2 \setminus V^2$.

Note that when $V = [N]$, we have $\mathbb{A}_V \equiv \mathbb{A}_{[N]}$ as defined in Section 2. By convention, when $V = \emptyset$, we define $\mathbb{A}_V$ to contain only the zero matrix. It is readily observed that if $a \in \mathbb{A}_V$ and $a' \in \mathbb{A}_{V'}$ with $V \cap V' = \emptyset$ then $(a + a') \in \mathbb{A}_{V \cup V'}$.

Now let $\mathbb{A} := \bigcup_{V \subseteq [N]} \mathbb{A}_V$ and define the function $N^{\text{eff}} : \mathbb{A} \times \mathbb{R}_+^N \to \mathbb{R}_+$,

$$N^{\text{eff}}(a,c) := \begin{cases} 0 & a \in \mathbb{A}_\emptyset, \\ \frac{\left(\sum_i \sum_j a^{ij} c^j\right)^2}{\sum_i \left(\sum_j a^{ij} c^j\right)^2} & \text{otherwise}, \end{cases} \tag{8}$$

where $c = (c^1, \ldots, c^N) \in \mathbb{R}_+^N$ (for simplicity we shall always assume that each c^i is strictly positive). This generalizes the ESS in (4): let c be given by $c^i := W_{n-1}^i g_{n-1}(\zeta_{n-1}^i)$, $i \in [N]$. Then, if $\alpha_{n-1} = a \in \mathbb{A}_{[N]}$, we have $N^{\text{eff}}(\alpha_{n-1}, c) \equiv N_n^{\text{eff}}$. If instead $\alpha_{n-1} = a + a'$, where $a \in \mathbb{A}_V$ for some strict, non-empty subset $V \subset [N]$ and $a' \in \mathbb{A}_{[N]\setminus V}$, then

$$N^{\text{eff}}(a,c) = \frac{\left(\sum_{i \in V} \sum_{j \in V} a^{ij} c^j\right)^2}{\sum_{i \in V} \left(\sum_{j \in V} a^{ij} c^j\right)^2} = \frac{\left(\sum_{i \in V} W_n^i\right)^2}{\sum_{i \in V} \left(W_n^i\right)^2}$$

represents the ESS associated with the sub-population of $|V|$ weights $\left\{W_n^i : i \in V\right\}$, cf. (4).

The following proposition provides useful properties of N^{eff}.

Proposition 1. *Let $V, V' \subseteq [N]$ such that $V \cap V' = \emptyset$. Let $a \in \mathbb{A}_V$, $a', \tilde{a}' \in \mathbb{A}_{V'}$ and $c \in \mathbb{R}_+^N$ be given such that $N^{\text{eff}}(a,c)$, $N^{\text{eff}}(a',c)$ and $N^{\text{eff}}(\tilde{a}',c)$ are all positive. All of the following hold:*

1. Extremes: $1 \leq N^{\text{eff}}(a,c) \leq |V|$ and $N^{\text{eff}}(a,c) = |V|$ whenever $a^{ij} = |V|^{-1} \mathbb{I}(i,j \in V)$.
2. Subadditivity:
$$N^{\text{eff}}(a + a', c) \leq N^{\text{eff}}(a,c) + N^{\text{eff}}(a',c),$$
with equality only when $\frac{\sum_{j \in V} c^j}{N^{\text{eff}}(a,c)} = \frac{\sum_{j \in V'} c^j}{N^{\text{eff}}(a',c)}$.
3. Monotonicity:
$$N^{\text{eff}}(a',c) \leq N^{\text{eff}}(\tilde{a}',c) \implies N^{\text{eff}}(a + a', c) \leq N^{\text{eff}}(a + \tilde{a}', c),$$
with equality on the right hand side of the implication only when $N^{\text{eff}}(a',c) = N^{\text{eff}}(\tilde{a}',c)$.
4. Lower bound:
$$N^{\text{eff}}(a + a', c) \geq \min\left\{N^{\text{eff}}(a,c), N^{\text{eff}}(a',c)\right\}.$$

The first part of this proposition is well known and identifies extremal values of N^{eff}, of which the maximal value can always be realized by a particular choice of a. The other parts concern properties of N^{eff} when considering elements of $\mathbb{A}_V$ and $\mathbb{A}_{V'}$ for some fixed and disjoint $V, V' \subseteq [N]$. The second establishes the subadditivity of N^{eff} and indicates that the effective sample size associated with $a + a'$ is less than the sum of those associated with a and a' separately. The third shows that nevertheless a monotonicity property holds when comparing two substochastic matrices in $\mathbb{A}_V$, and the fourth provides a simply-proved but tight lower bound on the effective sample size associated with $a + a'$.

3.2 Disjoint unions of complete graphs and a partial order

Whiteley et al. [34, Section 5.3] considered a family of candidate α matrices which have the interpretation of being transition matrices of random walks on regular undirected graphs. In this section, we expand upon this duality between α matrices and undirected graphs, and introduce some mathematical machinery which allows us describe how these objects are related to each other, and N^{eff}. In particular, we consider graphs that are disjoint unions of complete graphs (Definition 1 below): these graphs are not necessarily regular but are highly structured nonetheless and are of interest here because we can define a partial order over them, and then establish a partial order preservation result for N^{eff} (see Propositions 2 and 3) that will ultimately guide the efficient exploration of progressively denser stochastic matrices until one is found, a, for which we can guarantee $N^{\text{eff}}(a,c) \geq N\tau$.

To proceed, let us introduce some standard graph-theoretic notions. A graph $G = (V, E)$ is a set of vertices $V \subseteq [N]$ and a set of edges $E \subseteq V^2$, where an edge $(i, j) \in E$ represents a connection between vertices i and j. We adopt the convention that $(i, i) \in E$ whenever $i \in V$. If G is undirected then $(i, j) \in E \iff (j, i) \in E$. If G is a complete graph then $E = V^2$. Since a complete graph is defined solely by its vertex set, and because complete graphs are important building blocks in the sequel, we define $\kappa(V) := (V, V^2)$ to be the complete graph with vertices V.

Let $\cup$ denote the disjoint union of two graphs: if $G = (V, E)$, $G' = (V', E')$ and $V \cap V' = \emptyset$ then $G \cup G' = (V \cup V', E \cup E')$.

Definition 1. (Disjoint union of complete graphs) A graph G is a disjoint union of complete graphs if for some $K \in [N]$ there exists a set of pairwise disjoint subsets of $[N]$, denoted $\{V_k : k \in [K]\}$ such that $G = \bigcup_{k\in[K]} \kappa(V_k)$.

In analogy with $\mathbb{A}_V$ (and $\mathbb{A}$) we define $\mathbb{G}_V$ to be the set of graphs which have vertices V and which are disjoint unions of complete graphs (and $\mathbb{G} := \bigcup_{V\subseteq[N]} \mathbb{G}_V$). Clearly, if $G \in \mathbb{G}_V$, $G' \in \mathbb{G}_{V'}$ and $V \cap V' = \emptyset$, then $G \cup G' \in \mathbb{G}_{V\cup V'}$. We also define the matrix-valued function ϕ

$$\phi : G = (V, E) \in \mathbb{G} \longmapsto a = (a^{ij}) \in \mathbb{A}$$

where

$$a^{ij} := \begin{cases} \frac{\mathbb{I}\{(i,j)\in E\}}{\sum_{k\in[N]} \mathbb{I}\{(i,k)\in E\}} & (i, j) \in V^2, \\ 0 & (i, j) \in [N]^2 \setminus V^2. \end{cases} \tag{9}$$

One trivial property of elements $G = (V, E) \in \mathbb{G}$ is that $(i, j) \in E$ and $(j, k) \in E$ implies $(i, k) \in E$. It is therefore clear that if $G \in \mathbb{G}_V$ then $\phi(G)$ is a symmetric matrix and leaves the uniform distribution on V invariant, hence, $\phi(G) \in \mathbb{A}_V$. Figure 1 shows an example of a graph $G \in \mathbb{G}$ and the corresponding substochastic matrix $\phi(G)$. Letting $\mathbb{A}_\mathbb{G} := \{\phi(G) : G \in \mathbb{G}\}$ be the image of ϕ, it is straightforward that $\phi : \mathbb{G} \to \mathbb{A}_\mathbb{G}$ is a bijection and so we denote by ϕ^{-1} the inverse of ϕ. In addition, it can be seen that if $G \in \mathbb{G}_V$ and $G' \in \mathbb{G}_{V'}$ with $V \cap V' = \emptyset$, then

$$\phi(G \cup G') = \phi(G) + \phi(G'). \tag{10}$$

We can now introduce a particular relation amongst graphs, and amongst the corresponding substochastic matrices.

Definition 2. (Binary relation $\preceq$) Let $G = (V, E)$ and $\tilde{G} = (\tilde{V}, \tilde{E})$ be members of $\mathbb{G}$. Then we write $G \preceq \tilde{G}$ if and only if $V = \tilde{V}$ and $E \subseteq \tilde{E}$. Since ϕ is a bijection between $\mathbb{G}$ and $\mathbb{A}_\mathbb{G}$ we will also write, for $a, \tilde{a} \in \mathbb{A}_\mathbb{G}$, $a \preceq \tilde{a}$ if and only $\phi^{-1}(a) \preceq \phi^{-1}(\tilde{a})$.

Proposition 2. *(Partial order) $\preceq$ is a partial order over $\mathbb{G}$ and $\mathbb{A}_\mathbb{G}$.*

Definition 2 says that for some $G, \tilde{G} \in \mathbb{G}_V$ we have $G \preceq \tilde{G}$ if $\tilde{G} = G$, or if $\tilde{G}$ can be obtained from G by adding edges in such a way that $\tilde{G} \in \mathbb{G}_V$. Intuitively, one can imagine adding edges by choosing two of the complete graphs comprising G and adding edges between all vertices in these two graphs. Figure 2 shows an example of two graphs $G, G' \in \mathbb{G}$ such that $G \preceq G'$. We note that $\preceq$ is not a total order, because there exist members of $\mathbb{G}_V$, $G = (V, E)$, $G' = (V, E')$ such that $E \not\subseteq E'$ and $E' \not\subseteq E$. Our interest in $\preceq$ is the following order preservation property.

Proposition 3. *(Order preservation) For any $c \in \mathbb{R}^N_+$, $a \preceq \tilde{a} \implies N^{\text{eff}}(a, c) \leq N^{\text{eff}}(\tilde{a}, c)$.*

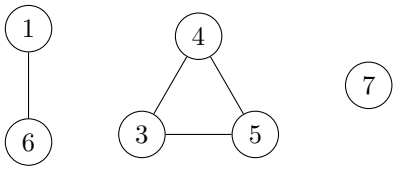
3.3 Local lower bounds on N^{eff}

In this subsection we present Algorithm 2, a recursive method for efficient selection of $a \in \mathbb{A}_{[N]}$; using a corresponding recursive lower bound on N^{eff} (Proposition 4) and the ordering result Proposition 3, we shall validate Algorithm 2 with Proposition 5, which shows that it is guaranteed to achieve $N^{\text{eff}}(a, c) \geq N\tau$.

For purposes of exposition, we first provide an expression for $N^{\text{eff}}(\tilde{a}, c)$ when $\tilde{a}$ is the substochastic matrix associated with a disjoint union of complete graphs. Following (10), let $\tilde{a} = \sum_{k\in[K]} \phi(\kappa(V_k)) \in \mathbb{A}_\mathbb{G} \cap \mathbb{A}_V$ for some $K \in [N]$ and pairwise disjoint $\{V_k : k \in [K]\}$ with $V = \bigcup_{k\in[K]} V_k$. Then, from (8),

$$\begin{aligned} N^{\text{eff}}(\tilde{a}, c) &= \frac{\left(\sum_{k\in[K]} \sum_{j\in V_k} c^j\right)^2}{\sum_{k\in[K]} \sum_{i\in V_k} \left(\sum_{j\in V_k} \phi(\kappa(V_k))^{ij} c^j\right)^2} \\ &= \frac{\left(\sum_{k\in[K]} \sum_{j\in V_k} c^j\right)^2}{\sum_{k\in[K]} |V_k| \left(|V_k|^{-1} \sum_{j\in V_k} c^j\right)^2}. \end{aligned} \tag{11}$$

We note that (11) depends on c only through the values of the sums $\left\{\sum_{j\in V_k} c^j : k \in [K]\right\}$; we can interpret this as saying that $N^{\text{eff}}(\tilde{a}, c)$ is equal to the ESS associated with a collection of $\sum_{k\in[K]} |V_k|$ weights, in which for each $k \in [K]$ there are $|V_k|$ weights all taking the value $|V_k|^{-1} \sum_{j\in V_k} c^j$. This lends interpretation to the lower bound in the following proposition.



$$\begin{bmatrix} 1/2 & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1/3 & 1/3 & 1/3 & \cdot & \cdot \\ \cdot & \cdot & 1/3 & 1/3 & 1/3 & \cdot & \cdot \\ \cdot & \cdot & 1/3 & 1/3 & 1/3 & \cdot & \cdot \\ 1/2 & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{bmatrix}$$

Figure 1: A graph $G \in \mathbb{G}$, with vertex set $\{1, 3, 4, 5, 6, 7\}$, and the corresponding matrix $\phi(G)$. For visual clarity, self-loops are not shown.

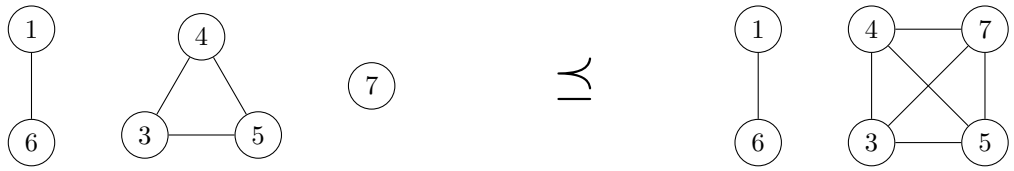


Figure 2: Two graphs $G, G' \in \mathbb{G}$ with $G \preceq G'$.

Proposition 4. *Let $\{V_k : k \in [K]\}$ consist of non-empty and pairwise disjoint subsets of $[N]$ and $\{a_k : k \in [K]\}$ be given such that each $a_k \in \mathbb{A}_{V_k}$. Let $a = \sum_{k\in[K]} a_k$ and $\tilde{a} = \sum_{k\in[K]} \phi(\kappa(V_k))$. Then for any $c \in \mathbb{R}_+^N$,*

$$N^{\text{eff}}(\tilde{a}, c) \geq N^{\text{eff}}(a, c) \geq \min_k \left\{ \frac{N^{\text{eff}}(a_k, c)}{|V_k|} \right\} N^{\text{eff}}(\tilde{a}, c). \tag{12}$$

Importantly, Proposition 4 enables us to calculate a lower bound on $N^{\text{eff}}\left(\sum_{k\in[K]} a_k, c\right)$ without explicit computation of (8). This observation is at the heart of our new algorithms.

A disjoint union of complete graphs with vertices $V \subseteq [N]$ can be succinctly represented by a partition $P = \{V_k : k \in [K]\}$ of V, where $K \in [|V|]$. Overloading our $N^{\text{eff}}(\cdot, c)$ notation so as to conveniently express certain quantities in Algorithm 2, we define for such a partition P,

$$N^{\text{eff}}(P, c) := \frac{\left(\sum_{S\in P} \sum_{j\in S} c^j\right)^2}{\sum_{S\in P} |S| \left(|S|^{-1} \sum_{j\in S} c^j\right)^2}. \tag{13}$$

Since P is a partition of V, we have

$$\frac{N^{\text{eff}}(P, c)}{\sum_{S\in P} \sum_{j\in S} 1} = |V|^{-1} N^{\text{eff}}(P, c) =: \rho(P, c), \tag{14}$$

and this quantity also appears in Algorithm 2.

If P and $\tilde{P}$ are the partitions representing G and $\tilde{G}$ respectively, where $G, \tilde{G} \in \mathbb{G}_V$ for some $V \subseteq [N]$, then $G \preceq \tilde{G}$ if and only if P is a refinement of $\tilde{P}$. This allows us to make the following definition, which will be used extensively in the sequel.

Algorithm 2 Choose an $a \in \mathbb{A}_V$ such that $N^{\text{eff}}\left(\sum_{k\in[K]} a_k, c\right) \geq \tau|V|$

`choose.a`(V, τ)

1. Choose a partition $P = \{V_1, \ldots, V_K\}$ of V such that $\rho(P, c) \geq \tau$.
2. If $P = \{V\}$ then return $\phi(\kappa(V))$.
3. Otherwise, return

$$\sum_{k\in[K]} \texttt{choose.a}(V_k, \tau/\rho(P, c)).$$

Definition 3. (Coarsening) Let P, $\tilde{P}$ be partitions of some subset of $[N]$. Then $\tilde{P}$ is a coarsening of P, written $\tilde{P} \succeq P$, if and only if P is a refinement of $\tilde{P}$.

It follows from Proposition 3 that $\tilde{P} \succeq P \implies N^{\text{eff}}(\tilde{P}, c) \geq N^{\text{eff}}(P, c)$.

Proposition 5. *Algorithm 2 called with (V, τ) satisfying $\emptyset \neq V \subseteq [N]$ and $\tau \in [0, 1]$ returns $a \in \mathbb{A}_V$ such that $N^{\text{eff}}(a, c) \geq \tau|V|$.*

There are a number of ways that step 1 of Algorithm 2 can be implemented. One possibility, motivated by Proposition 3, is to search through a sequence of successively

coarser, candidate partitions until the condition $\rho(P, c) \geq \tau$ is met. In Section 4 we provide a more detailed and practical version of this procedure in Algorithm 6, in which the partitions considered arise from collections of tree data structures.

4 Forest resampling

In this section we introduce tree data structures to represent the logical topology of a distributed computer architecture. Loosely, these trees provide a model for how the operations involved in αSMC can be arranged over a network of communicating devices, each of which has the capacity to store data and to perform basic simulation and arithmetic tasks. In Section 4.1 we explain the connection between the distributed architecture and tree data structures, and in Section 4.2 we explain the connection between trees and forests, and the partitions, graphs and matrices addressed in Section 3. Sections 4.3 and 4.4 describe the role of forests when implementing Algorithm 1, and all these ingredients are brought together in Algorithm 3, which is an implementation of Algorithm 1 using trees and forests. Algorithm 3 is provided here to guide the reader through the developments of these sections, with the definition of a tree and the recursive subroutines `populate`, `sample` and `choose.forest` introduced progressively. Steps 2(a)–(d) correspond to implementing step ($\star$) of Algorithm 1, while step 2(e) implements lines (†) and (‡). The `choose.forest` subroutine itself can be seen as an implementation of Algorithm 2.

A brief description of Algorithm 3 is the following. The graphical structure of T_0, a base tree, will typically be the same at each time n, so at each time n steps 2(a)–(c) fill in the values of the nodes in this base tree, which depend on the weights $\{W_{n-1}^i\}_{i=1}^N$ of the particles at time $n-1$ and the values $\{g(\zeta_{n-1}^i)\}_{i=1}^N$. In step 2(d) a forest, or disjoint union of trees, is chosen such that each time n, particle ζ_n^i is associated with a particular subtree of T_0, subject to the constraint that the effective sample size of the time n weights $\{W_n^i\}_{i=1}^N$ will exceed $N\tau$. In step 2(e) each particle ζ_n^i is then sampled and its weight W_n^i calculated, involving only the communication of particles ζ_{n-1}^j and weights W_{n-1}^j where ζ_n^j is in the same subtree as ζ_n^i. All the procedures are recursive, divide-and-conquer algorithms suitable to distributed implementation, with inputs and outputs of the recursive calls containing only small amounts of local information.

Algorithm 3 αSMC with forest resampling

1. For $i \in [N]$, sample $\zeta_0^i \sim \pi_0$ and set $W_0^i \leftarrow 1$.
2. For $n \geq 1$:
   (a) Create an unpopulated tree T_0 with root ν_0 and leaves $\{\nu_i : i \in [N]\}$.
   (b) For each $i \in [N]$, set the value of leaf node ν_i
   $$\mathcal{V}(\nu_i) \leftarrow \left(1, W_{n-1}^i g_{n-1}(\zeta_{n-1}^i)\right).$$
   (c) Call `populate`(ν_0).
   (d) Set $F \leftarrow$ `choose.forest`(ν_0, τ).
   (e) For each $i \in [N]$:
      i. Set $W_n^i \leftarrow \mathcal{V}_2(\mathcal{R}(t_F(i)))/\mathcal{V}_1(\mathcal{R}(t_F(i)))$,
      ii. Set $j \leftarrow$ `sample`$(\mathcal{R}(t_F(i)))$,
      iii. Sample $\zeta_n^i \sim f(\zeta_{n-1}^j, \cdot)$.

4.1 Distributed computer architecture and trees

For the purposes of this paper, we are interested primarily in a setting where there are a number of possibly heterogeneous computing devices that can communicate by sending data over a network. Qualitatively, the structural assumption will be that communication within a device is far quicker than communication between devices. If there are M devices, we might think each device $i \in [M]$ is capable of handling a particle system with N_i particles. This implies that interactions involving the N_i particles on device i are considerably less costly than interactions involving particles on different devices.

This architecture suggests the use of a particular type of data structure, a tree, to represent possible interactions between computing devices. A tree is a recursive data structure comprising a set of nodes with associated values. Furthermore, networks with a tree structure dictating their communication are naturally suited to recursive, divide-and-conquer algorithms [29, Chapter 7], which are exactly the type of algorithms we will propose.

Definition 4. (Node) A node ν is an object that has a value, $\mathcal{V}(\nu)$, and a (possibly empty) set of child nodes, $\mathcal{C}(\nu)$.

Definition 5. (Finite tree) A (finite) tree T is a finite set of nodes which is either empty, or satisfies the following properties:

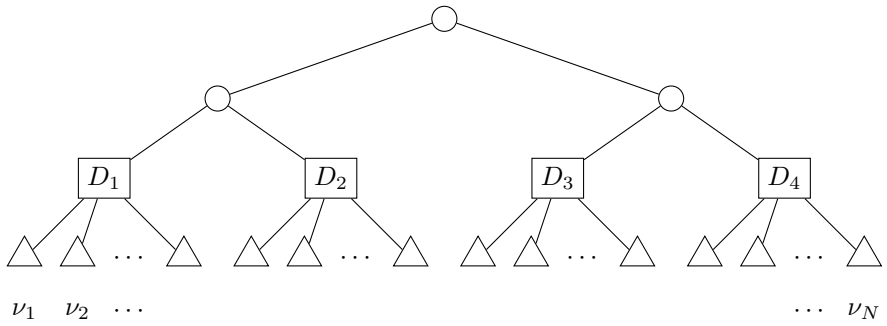


Figure 3: Roles of nodes.

1. $\mathcal{C}(\nu) \subseteq T$ for every $\nu \in T$ (no node has children outside T).
2. $\mathcal{C}(\nu) \cap \mathcal{C}(\nu') = \emptyset$ for any distinct $\nu, \nu' \in T$ (no node is the child of two different nodes in T).
3. There exists a unique element of T called the root and denoted $\mathcal{R}(T)$, such that $\mathcal{R}(T) \notin \bigcup_{\nu \in T} \mathcal{C}(\nu)$ (a unique root node is not a child of any of the other nodes in T).

One can show (e.g., by contradiction) that if T is a tree then every node in T other than $\mathcal{R}(T)$ is a descendant of $\mathcal{R}(T)$, i.e., $T \setminus \{\mathcal{R}(T)\} = \mathcal{D}(\mathcal{R}(T))$ where $\mathcal{D}(\nu)$ denotes the descendants of ν:

$$\mathcal{D}(\nu) := \begin{cases} \emptyset & \mathcal{C}(\nu) = \emptyset, \\ \mathcal{C}(\nu) \cup \left(\bigcup_{\chi \in \mathcal{C}(\nu)} \mathcal{D}(\chi) \right) & \mathcal{C}(\nu) \neq \emptyset. \end{cases}$$

Definition 6. (Subtree) A subtree, of a tree T, consists of a node in T, taken together with all of the descendants of that node. In particular, for some $\nu \in T$ we call $\mathcal{S}(\nu) := \nu \cup \mathcal{D}(\nu)$ the subtree of T with root ν.

The definitions of a tree and its subtrees are equivalent to those found in [21, p. 308], but with an emphasis on their formulation using children. Here, trees serve as data structures in that the value of each node is the data stored there, and data transfer can occur between a node and its children.

It is conventional to call a node of a tree T whose set of children is empty a leaf, and the set of such nodes comprise the leaves of T. Our intention is to have the individual particles, indexed by $j \in [N]$, represented by leaves of a tree and the parents of leaves representing the M devices in the distributed architecture. If each device i is assigned N_i particles then the children of the node associated with device i will be the N_i leaves associated with the particle indices $\left\{1 + \sum_{j \in [i-1]} N_j, \ldots, \sum_{j \in [i]} N_j\right\}$. Beyond these two levels, the structure is purposefully abstract so as to accommodate various choices which could, e.g., be related to more complex architectural considerations such as the geographical location of the devices. It is, however, assumed that each node is physically contained on a single device although more than one node may be physically contained on the same device. The general idea is that a node will both facilitate and modulate interaction between its children. Figure 3 shows a possible tree with 4 devices. In our recursive algorithms, direct communication occurs only between parents and their children. Therefore, we henceforth assume that a base tree T_0 has been defined where each node can communicate with all of its children reasonably quickly.

Let the base tree T_0 have root node ν_0 and exactly N leaves $\{\nu_i : i \in [N]\}$. We now define the set of leaf indices associated with a node ν of T_0 to be the set of indices associated with the leaves of $\mathcal{S}(\nu)$, i.e., we let $\ell(\nu_i) := \{i\}$ for each $i \in [N]$, and for each $\nu \in T_0$ such that $\mathcal{C}(\nu) \neq \emptyset$, $\ell(\nu) := \bigcup_{\chi \in \mathcal{C}(\nu)} \ell(\chi)$. Without ambiguity we also define, for T a subtree of T_0, $\ell(T) := \ell(\mathcal{R}(T))$. For some $c \in \mathbb{R}_+^N$, we define the value of each node ν to be

$$\mathcal{V}(\nu) := (\mathcal{V}_1(\nu), \mathcal{V}_2(\nu)) := \left(|\ell(\nu)|, \sum_{j \in \ell(\nu)} c^j \right),$$

so that the value of leaf node ν_i, e.g., is $\mathcal{V}(\nu_i) = (1, c^i)$. Once the values of the leaves have been set, Algorithm 4 can be invoked on ν_0 to calculate recursively the values of the rest of the nodes in the tree, and is motivated by the fact that, element-wise,

$$\mathcal{V}(\nu) = \sum_{\chi \in \mathcal{C}(\nu)} \mathcal{V}(\chi), \tag{15}$$

when $\mathcal{C}(\nu) \neq \emptyset$. This is an instance of a recursive reduction algorithm suitable for implementation in both parallel and

Algorithm 4 Populate a subtree

`populate`(ν)

1. If $\mathcal{C}(\nu) = \emptyset$, return $\mathcal{V}(\nu)$.
2. Otherwise, set $\mathcal{V}(\nu) \leftarrow \sum_{\chi \in \mathcal{C}(\nu)}$ `populate`(χ), where the summation is component-wise.
3. Return $\mathcal{V}(\nu)$.

distributed settings (see, e.g., [15, 17]) which can be called on the root of the subtree in question. Typically, one will call it on ν_0 to populate the entire tree T_0, as in Algorithm 3. The time complexity associated with each node ν's computation is in $\mathcal{O}(|\mathcal{C}(\nu)|)$.

4.2 Graphs induced by trees and forests

We now take the first step towards connecting our tree data structures with the type of graphs discussed in Section 3. We define the graph induced by a tree T to be

$$G(T) := \kappa(\ell(T)), \tag{16}$$

the complete graph with vertices $\ell(T)$. This allows us to define the substochastic matrix induced by a tree as $\phi(T) := \phi(G(T))$. It is immediately obvious that the only member of $\mathbb{A}_{[N]}$ that can be induced by a single tree is $\phi(T_0) = \mathbf{1}_{1/N}$. The notion of a forest allows a richer subset of $\mathbb{A}_{[N]}$ to be specified using trees.

Definition 7. (Forest) A forest F is a set of pairwise disjoint trees.

It follows from this definition that if $T, T' \in F$ are distinct, then $\ell(T) \cap \ell(T') = \emptyset$. If T is a tree then $\{T\}$ and $\{\mathcal{S}(\nu) : \nu \in \mathcal{C}(\mathcal{R}(T))\}$ are both examples of forests. In what follows, the forests defined will always be comprised of subtrees of T_0. Figure 4 supplements the example from Figure 1 with a possible associated tree data structure and forest of subtrees.

We define the set of leaf indices associated with a forest to be $\ell(F) := \bigcup_{T \in F} \ell(T)$. We also let $\mathbb{F}_V := \{F : \ell(F) = V\}$, where $V \subseteq [N]$, and $\mathbb{F} := \bigcup_{V \subseteq [N]} \mathbb{F}_V$. We can relate any $F \in \mathbb{F}$ to a member of $\mathbb{G}$ by defining

$$G(F) := \bigcup_{T \in F} G(T) = \bigcup_{T \in F} \kappa(\ell(T)).$$

From (10), the substochastic matrix induced by $F \in \mathbb{F}$ is then

$$\phi(F) := \phi(G(F)) = \sum_{T \in F} \phi(\kappa(\ell(T))).$$

One can therefore think of a forest F as being a data structure counterpart to a disjoint union of complete graphs represented by the partition $P = \{\ell(T) : T \in F\}$.

4.3 Forest resampling

We now introduce practical methodology that, given a forest $F \in \mathbb{F}_{[N]}$, enables implementation of step (‡) of Algorithm 1 when $\alpha_{n-1} = \phi(F)$. Let c be given by $c^i := W^i_{n-1} g_{n-1}(\zeta^i_{n-1})$, $i \in [N]$, so that our goal is to sample, for each $i \in [N]$,

$$\zeta^i_n \mid \zeta_0, \ldots, \zeta_{n-1} \sim \frac{\sum_j \alpha^{ij}_{n-1} c^j f(\zeta^j_{n-1}, \cdot)}{\sum_k \alpha^{ik}_{n-1} c^k},$$

which can be implemented in two substeps. First one simulates an ancestor index A^i_{n-1} with

$$\mathsf{P}\left(A^i_{n-1} = j \mid \zeta_0, \ldots, \zeta_{n-1}\right) = \frac{\alpha^{ij}_{n-1} c^j}{\sum_k \alpha^{ik}_{n-1} c^k},$$

and then, secondly, simulates $\zeta^i_n \sim f(\zeta^{A^i_{n-1}}_{n-1}, \cdot)$. Implementation of the second step is a model-specific matter, so we focus on the first step. We define t_F to be the tree-valued map where for any $i \in \ell(F)$, $t_F(i)$ is the unique tree $T \in F$ such that $i \in \ell(T)$. It then follows that $\alpha^{ij}_{n-1} = \mathbb{I}\{j \in \ell(t_F(i))\} / |\ell(t_F(i))|$ and so we can write

$$\mathsf{P}\left(A^i_{n-1} = j \mid \zeta_0, \ldots, \zeta_{n-1}\right) = \frac{\mathbb{I}\{j \in \ell(t_F(i))\} c^j}{\sum_{k \in \ell(t_F(i))} c^k}, \tag{17}$$

which implies that A^i_{n-1} is categorically distributed over $\ell(t_F(i))$ with probabilities proportional to $\{c^k : k \in \ell(t_F(i))\}$.

Following (17), we propose Algorithm 5, which given the root node $\mathcal{R}(T)$ of an arbitrary subtree T of T_0, samples from a distribution over $\ell(T)$ with probability mass function

$$p_T(j) := \frac{\mathbb{I}\{j \in \ell(T)\} c^j}{\sum_{k \in \ell(T)} c^k}, \quad j \in \ell(T).$$

In this algorithm, the time complexity associated with each node ν's computation is in $\mathcal{O}(|\mathcal{C}(\nu)|)$.

Proposition 6. *The probability that Algorithm 5 returns $j \in \ell(\nu)$ is $p_{\mathcal{S}(\nu)}(j)$.*

Sampling according to (17) for each $i \in [N]$ can be accomplished by calling Algorithm 5 N times with potentially different inputs. For example, if $F = \{T_0\}$ then one would call Algorithm 5 N times on $\nu_0 = \mathcal{R}(T_0)$, corresponding to standard multinomial resampling with $\alpha_{n-1} =$

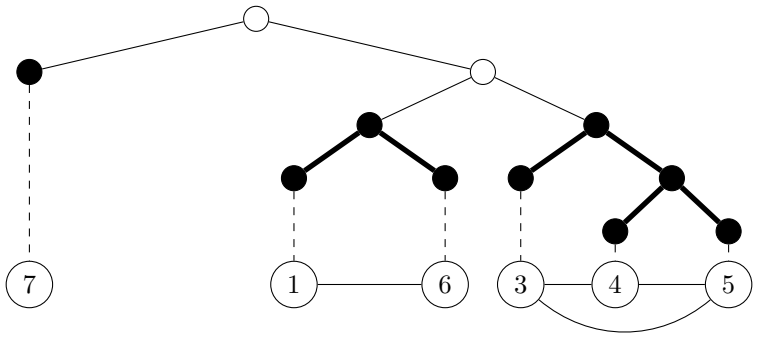


Figure 4: A forest made up of subtrees of a tree, and the complete graphs induced by each tree in the forest.

Algorithm 5 Obtain a sample according to $p_{\mathcal{S}(\nu)}$

`sample`(ν)

1. If $\mathcal{C}(\nu) = \emptyset$, return the only element in $\ell(\nu)$.
2. Otherwise, let $\chi_1, \ldots, \chi_{|\mathcal{C}(\nu)|}$ be the children of ν.
3. Sample i from a categorical distribution over $[|\mathcal{C}(\nu)|]$ with probabilities proportional to $\{\mathcal{V}_2(\chi_i) : i \in [|\mathcal{C}(\nu)|]\}$.
4. Return `sample`(χ_i).

$\phi(F) = \mathbf{1}_{1/N}$. In contrast, if $F = \{\mathcal{S}(\nu_1), \ldots, \mathcal{S}(\nu_N)\}$ then one would call Algorithm 5 once on each member of $\{\nu_1, \ldots, \nu_N\}$ with the effect that $A^i_{n-1} = i$ for each $i \in [N]$, and this corresponds to $\alpha_{n-1} = \phi(F) = Id$. An intermediate between these two extremes would be if $F = \{\mathcal{S}(\nu^1), \ldots, \mathcal{S}(\nu^M)\}$, where ν^i represents device node i in T_0, cf. Section 4.1. Then, for each $i \in [M]$, one would call Algorithm 5 $\left|\ell(\nu^i)\right|$ times, once to set each ancestor index in $\left\{A^j_{n-1} : j \in \ell(\nu^i)\right\}$. These special cases also exemplify a more general phenomenon: sampling according to (17) using Algorithm 5 does not require the explicit computation of α_{n-1}. In Section 4.4 we address the issue of how a forest can be chosen adaptively.

Finally, we note that step (†) of Algorithm 1 can also be accomplished straightforwardly when $\alpha_{n-1} = \phi(F)$. Indeed, then

$$\begin{aligned} W^i_n &= \sum_j \phi(F)^{ij} c^j = \sum_{k \in \ell(t_F(i))} c^k / \left|\ell(t_F(i))\right| \\ &= \mathcal{V}_2(\mathcal{R}(t_F(i)))/\mathcal{V}_1(\mathcal{R}(t_F(i))). \end{aligned}$$

4.4 Forest selection

Our attention now turns to implementing the ($\star$) step of Algorithm 1. This can be performed by choosing a forest $F \in \mathbb{F}_{[N]}$ such that $N^{\text{eff}}(\phi(F), c) \geq \tau N$. Algorithm 6 is a recursive implementation of such a procedure, and is essentially a practical analogue of Algorithm 2. The ($\star\star$) step in this algorithm is specified only abstractly, with concrete choices the subject of Section 4.5. Like steps (†) and (‡), when implemented according to the procedures of Section 4, step ($\star\star$) also involves only local computations in the following sense. Recalling Definition 3, choosing P' to be a partition of $\mathcal{C}(\nu)$ implies that P is a coarsening of $\{\ell(\chi) : \chi \in \mathcal{C}(\nu)\}$, and so the computation of $\rho(P, c)$ involves only the quantities $|\ell(\chi)|$ and $\sum_{j \in \ell(\chi)} c^j$ for each $\chi \in \mathcal{C}(\nu)$, which are readily available through $\{\mathcal{V}(\chi) : \chi \in \mathcal{C}(\nu)\}$. In the particular case where T_0 is a binary tree then ($\star\star$) involves only choosing between $P' = \{\ell(\chi_1), \ell(\chi_2)\}$, where χ_1 and χ_2 are the two children of ν, and $P' = \{\mathcal{C}(\nu)\}$ so the strategies of Section 4.5 are not relevant.

In Algorithm 6, new nodes can be created. It is assumed that when this happens, the values of the new nodes are set appropriately according to (15).

The recursive nature of the algorithms presented allow them to be fairly straightforwardly translated into architecture specific implementations. In particular, it is imagined that the computations of Algorithms 4, 5 and 6 all take place on the device on which their node argument physically resides, and that the recursive calls then represent messages passed over the network. In addition, all of the algorithms are recursive, divide-and-conquer algorithms naturally suited to parallel implementation. Finally, the forests chosen by Algorithm 6 are intrinsically related to the original choice of T_0, the base tree defined in Section 4.1. This allows, through a judicious choice of T_0 and ($\star\star$), the communication structure involved in Algo-

Algorithm 6 Specify a forest F with $\ell(F) = \ell(\nu)$ and $N^{\text{eff}}(\phi(F), c) \geq \tau|\ell(V)|$

`choose.forest`(ν, τ)

1. If $\mathcal{C}(\nu) = \emptyset$ then return $\{\mathcal{S}(\nu)\}$.
2. ($\star\star$) Choose a partition P' of $\mathcal{C}(\nu)$ such that $\rho(P, c) \geq \tau$, where $P = \left\{\bigcup_{\chi \in S} \ell(\chi) : S \in P'\right\}$.
3. If $P' = \{\mathcal{C}(\nu)\}$ then return $\{\nu\}$. Otherwise, set $R \leftarrow \emptyset$.
4. For each element $S \in P'$
   (a) If $|S| > 1$ then create a node ν' with children $\{\chi : \chi \in S\}$ and set $R \leftarrow R \cup \{\mathcal{S}(\nu')\}$.
   (b) If $S = \{\chi\}$, set $R \leftarrow R \cup$ `choose.forest` $(\chi, \tau/\rho(P, c))$.
5. Return R.

rithm 3 to be related to the specific network architecture available in a beneficial way.

The exact implementation of the algorithms may vary slightly, depending on the architectures involved, without changing in principle. For example, one implementation of Step 2(e) of Algorithm 3 could involve each device sending its list of associated indices "up" the tree until it reaches its root in the forest. From there, the indices may filter "down" the tree in a slight variant of Algorithm 5 until they reach their leaves. If index i reaches leaf ν_j, say, the device housing ν_j can send ζ_{n-1}^j to the device housing ν_i, which can then sample $\zeta_n^i \sim f(\zeta_{n-1}^j, \cdot)$.

4.5 Partitioning strategies

The ($\star\star$) step in Algorithm 6 remains to be specified for general T_0. A simple choice would be to choose the partition $\{\{\chi\} : \chi \in \mathcal{C}(\nu)\}$ if it satisfies the condition in ($\star\star$) and $\{\mathcal{C}(\nu)\}$ otherwise. However, this could lead to more interaction than is necessary.

Before continuing, we note that selecting a partition of child nodes of ν is equivalent to selecting a partition P of $\ell(\nu)$ subject to the constraint that the chosen partition is a coarsening of $P_0 := \{\ell(\chi) : \chi \in \mathcal{C}(\nu)\}$. Therefore, we simplify the presentation by considering partitions of $V \subseteq [N]$ instead of partitions of nodes and our goal is to choose a partition $P \succeq P_0$ of V such that $\rho(P, c) \geq \tau$.

If a specific order over coarsenings of P_0 is defined, one could seek to find the minimal coarsening P^* w.r.t. this order that satisfies $\rho(P^*, c) \geq \tau$. For example, one might wish to find a $P \succeq P_0$ subject to $\rho(P, c) \geq \tau$ with the maximal number of elements, or where the size of the largest element is minimized, both of which could be translated roughly as P being as refined as possible. This can always be achieved by enumerating candidate partitions $P_1, P_2, \ldots$ in the given order and calculating $\rho(P_i, c)$ for each until some $\rho(P_i, c) \geq \tau$, but this can quickly become computationally prohibitive as $|\mathcal{C}(\nu)|$ grows. Indeed, the number of candidate partitions is the $|\mathcal{C}(\nu)|$'th Bell number. This type of integer programming optimization problem is related to the Partition problem (see, e.g., [27]) and is likely to be NP-hard in general. We therefore focus on efficient search strategies for finding a $P \succeq P_0$ subject to $\rho(P, c) \geq \tau$ for which we hope that P is not much coarser than necessary.

Both of the strategies we introduce below consider a sequence of successively coarser partitions $P_1, P_2, \ldots$ which satisfy the constraint that $P_0 \preceq P_1 \preceq P_2 \preceq \cdots$, where P_0 is as above, and returns P_j such that $j = \min\{i : \rho(P_i, c) \geq \tau\}$. This general procedure has the property that $\rho(P_i, c) \geq \rho(P_{i-1}, c)$ and $|P_i| \leq |P_{i-1}| - 1$ for $i \in [|\mathcal{C}(\nu)|]$. The latter, together with the fact that (from part 1 of Proposition 1) $|P| = 1 \implies \rho(P, c) = 1 \geq \tau$, implies that the total number of partitions considered is at most $|\mathcal{C}(\nu)|$. The specific strategies below are therefore defined by the precise way in which the sequence $P_1, P_2 \ldots$ is chosen. Finally, we note that if $\max_\nu |\mathcal{C}(\nu)|$ does not grow with the number of particles N then the computational cost of step ($\star\star$) in Algorithm 6 using either of the following strategies is bounded independently of N. This is one of the advantages of the hierarchical tree structure, in contrast to the flat structure considered in [34, Section 5.4].

Pairing strategy for structured trees

This strategy applies when each node in T_0 has a number of children that is a power of 2 and the number of leaves associated with each child is equal.

Definition 8. (Pairing of a partition) Let P be a partition of $V \subseteq [N]$. A pairing P' of P is a partition of V where each element of P' is the union of two elements of P.

Whiteley et al. [34, Section 5.4] suggested a "greedy" pairing strategy, which we formalize in the following proposition.

Proposition 7. *Let P be a partition of $V \subseteq [N]$ with $P = \{V_i : i \in [2M]\}$ for some $M \in [N]$, $M \leq N/2$. Let*

V_i be ordered such that $0 \leq \sum_{j\in V_1} c^j \leq \cdots \leq \sum_{j\in V_{2M}} c^j$ and assume that $|V_i| = |V_j|$ for any $i, j \in [2M]$. Then a pairing P' of P that maximizes $\rho(P', c)$ is given by $P' = \{\{V_1, V_{2M}\}, \{V_2, V_{2M-1}\}, \ldots, \{V_M, V_{M+1}\}\}$.

In the pairing strategy, then, we define the sequence of partitions $P_1, P_2, \ldots$ by each P_i being the optimal pairing of P_{i-1} provided by Proposition 7. Since the combined computational cost of sorting arrays of length $M, M/2, \ldots, 1$ is $\mathcal{O}(M \log M)$ the computational cost of step $(\star\star)$ is $\mathcal{O}(|\mathcal{C}(\nu)| \log |\mathcal{C}(\nu)|)$ under this strategy.

Matching strategy

This strategy does not rely on any particular structure of T_0 and therefore is applicable more generally than the pairing strategy.

Proposition 8. *For some $K \in [N]$ let $P = \{V_i : i \in [K]\}$ be a partition of V and $P_{k,l} := \{V_i : i \in [K]\} \setminus \{V_k, V_l\} \cup \{V_k \cup V_l\}$ a coarsening of P associated with the indices $k, l \in [K]$. Then the choice of $k, l \in [K]$ that maximizes $\rho(P_{k,l}, c)$ is*

$$\arg \max_{(k,l)\in[K]^2} \frac{|V_k|\,|V_l|}{|V_k| + |V_l|} \left(\frac{\sum_{j\in V_k} c^j}{|V_k|} - \frac{\sum_{j\in V_l} c^j}{|V_l|} \right)^2 .$$

When $[K]$ is large, maximizing this expression by evaluating it for each $(k, l) \in [K]^2$ has a time complexity in $\mathcal{O}(K^2)$, which we wish to avoid. Therefore, we resort to finding the $(k, l) \in [K]^2$ for which only the squared expression is maximized. This happens when k and l correspond to the sets of indices whose associated terms in the squared expression are most different.

The matching strategy therefore defines the successively coarser partitions $P_1, P_2, \ldots$ by letting $S^{\min}_{i-1} = \arg\min_{S\in P_{i-1}} |S|^{-1} \sum_{j\in S} c^j$, $S^{\max}_{i-1} = \arg\max_{S\in P_{i-1}} |S|^{-1} \sum_{j\in S} c^j$, and setting

$$P_i = P_{i-1} \setminus \left\{S^{\min}_{i-1}, S^{\max}_{i-1}\right\} \cup \left\{S^{\min}_{i-1} \cup S^{\max}_{i-1}\right\}.$$

An interpretation of this is that the elements of the partition with whose associated values are most different are successively matched. The computational cost of step $(\star\star)$ is also $\mathcal{O}(|\mathcal{C}(\nu)| \log |\mathcal{C}(\nu)|)$ using this strategy, as it can be implemented using an initial sort of $|\mathcal{C}(\nu)|$ values with $\mathcal{O}(|\mathcal{C}(\nu)|)$ deletions and insertions into this sorted list.

5 Discussion

5.1 Numerical illustrations

We consider a simplified HMM whose empirical analysis illustrates the cost of the forest resampling schemes. In particular, we assume that the HMM equations (1) satisfy the additional conditional independence criterion that for any $x \in \mathsf{X}$, $f(x, \cdot) = \pi_0(\cdot)$, and that g_n in (2) is time-homogeneous with $g_n(x) = g(x)$. We further assume that when $X \sim \pi_0$, $g(X)$ is a $\ln\mathcal{N}\left(-\frac{\sigma^2}{2}, \sigma^2\right)$ random variable, with mean 1 and variance $\exp\left(\sigma^2\right) - 1$. This model is not intended to be a realistic, challenging application of SMC. Instead, its greatly simplified structure allows for transparent analysis and easy replication of results; the time-homogeneous nature of the model makes it well-suited for assessing the *computational* cost of resampling for large n, and its conditional independence structure allows us to make some calculations which explicitly show how the ESS is related to the moments of Z^N_n and $\pi^N_n(\varphi)$.

Writing $\mathbb{E}$ and $\mathbb{V}$ for respectively expectation and variance under the SMC algorithm, and for some measure μ and function φ, $\mathrm{var}_\mu(\varphi) := \int_{\mathsf{X}} [\varphi(x) - \mu(\varphi)]^2 \mu(dx)$, one can verify from (†), (4) and (3) that $\mathbb{E}\left(Z^N_n \mid \zeta_0, \ldots \zeta_{n-1}\right) = \pi_0(g) Z^N_{n-1} = Z^N_{n-1}$ with

$$\mathbb{V}\left(\frac{Z^N_n}{Z^N_{n-1}} \mid \zeta_0, \ldots \zeta_{n-2}\right) = \frac{\mathrm{var}_{\pi_0}(g)}{N^{\mathrm{eff}}_{n-1}} = \frac{\exp\left(\sigma^2\right) - 1}{N^{\mathrm{eff}}_{n-1}},$$

and $\mathbb{E}\left(\pi^N_n(\varphi) \mid \zeta_0, \ldots \zeta_{n-1}\right) = \pi_n(\varphi) = \pi_0(\varphi)$ with

$$\mathbb{V}\left(\pi^N_n(\varphi) \mid \zeta_0, \ldots \zeta_{n-1}\right) = \frac{\mathrm{var}_{\pi_0}(\varphi)}{N^{\mathrm{eff}}_{n-1}}.$$

Two natural definitions of the cost of an αSMC iteration are the average and maximum degree of the vertices in the forest corresponding to the α_{n-1} transition matrix chosen in $(\star)$ of Algorithm 1, which we denote d^N_n and $d^{\max}_n$. These measure the communication complexity of the algorithms [2, Chapter 1]. For example, when $\alpha_{n-1} = Id$ both costs are 1 and when $\alpha_{n-1} = \mathbf{1}_{1/N}$ both costs are N, although they are different in general. We ran Algorithm 1 for $n = 200$ iterations with various values of τ and σ and $N = 2^{12} = 4096$ particles. One can think of the value of N reported here as being a large multiple of 4096 since, conceptually, one could imagine that the leaves in this experiment represent devices with a large number of particles. The tree T_0 used at each iteration always consisted of three levels with each node except the leaves having $2^4 = 16$ children, but the leaf/device indices were permuted at each iteration.

Figure 5 shows the behaviour of $\bar{d} = n^{-1}\sum_{p=1}^{n} d_p^N$, $\overline{d_{\max}} = n^{-1}\sum_{p=1}^{n} d_p^{\max}$ and $\overline{N^{\text{eff}}} = n^{-1}\sum_{p=1}^{n} N_p^{\text{eff}}$ as τ and σ vary using the adaptive resampling particle filter (ARPF) of [26], and the two proposed strategies in Section 4.5, all instances of αSMC. We can see that the ARPF is particularly expensive in terms of degree of interaction, and has a higher average ESS than the rest. We note that $d_p^{\max}$ is a suitable proxy for the computational time associated with iteration p of αSMC, when computational time is dominated by the time taken to send $\mathcal{O}(d_p^{\max})$ bits over the network and this communication is performed in parallel. As such, $\overline{d_{\max}}$ then represents a scaled estimate of the expected time per iteration of αSMC. The pairing and matching strategies perform much better than the ARPF and have an ESS much closer to the threshold. In all cases, increases in τ and σ increase the cost of the algorithm, as one would expect. The bootstrap particle filter (not shown) always has an ESS of N and all vertices in its forest have degree N, and it is therefore the most expensive of the stable algorithms considered in this article.

An interesting phenomenon is illustrated by the shape of the curve in Figure 6a, which suggests that increasing τ beyond around 0.5 rapidly becomes expensive. This seems to indicate that, over many iterations, small-scale interactions with more particles can provide roughly the same performance as large-scale interactions with fewer particles. Indeed, the value $\tau = 1$ corresponds to an average degree of 4096 in this example for any of the methods, which is not shown, and is almost 10 times larger than the corresponding cost for $\tau = 224/225 \approx 0.996$, the rightmost point shown. Figure 6b shows further that fixing $N\tau = 2048$ but increasing N has the effect of reducing the average degree to close to 2 and suggests that optimizing, in terms of computational cost, the choice of N and τ with a given target ESS $N\tau$ could involve choosing a large N and a small τ, depending on the relative cost of increasing N compared to the cost of interactions.

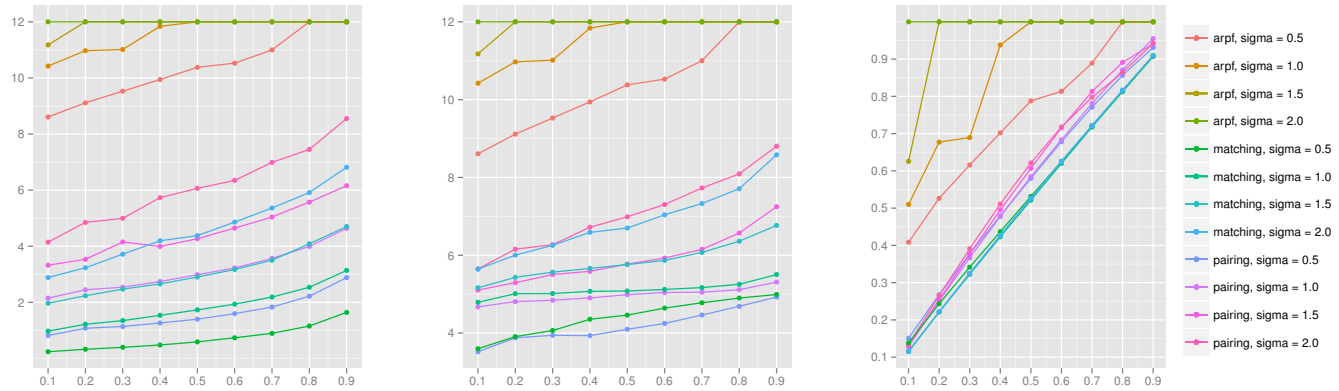
We performed a simulation study using a parallel message passing model in order to determine the impact of algorithm selection in the context of large distributed networks, and to ascertain network communication time associated with the ARPF and forest resampling algorithms. See [28] for an overview of various message passing models in a distributed memory context. Function calls and values returned within Algorithm 3 steps 2a)-e)ii, and the subroutines thereof, involve passing messages along edges of T_0, each message consisting of some number of particles and weights. The size of the messages can be expressed in bytes and we assumed 32 bytes for a particle and that each weight is an 8 byte double-precision floating point number. Since messages are passed along the edges of T_0, network congestion is largely avoided, so the Hockney model [16] for communication time is appropriate. This model relates the communication time of a message to its size and two key quantities: network latency and bandwidth, which we assumed to be respectively 1ms, and either 1Mbps or 1000Mbps. The overall communication time involved at each iteration of step 2 of Algorithm 3 was then calculated as the longest total communication time amongst all sequences of messages with serial dependence. For simplicity, we considered the case where T_0 is a binary tree from the devices up, so that there is no need to consider the partitioning strategies in Section 4.5.

The results are tabulated in Table 1 and Table 2, in which we consider performance in the two bandwidth regimes as the number of devices increases with the number of particles per device, 128, fixed. We take $\tau = 0.5$ and $\sigma = 2$. In both cases, the forest resampling scheme was superior in terms of average time spent communicating over the network per iteration. If the processor time required to sample from f or evaluate g is sufficiently large, both algorithms can provide substantial speedup; the results indicate only that the ARPF is less computationally efficient, particularly when the number of devices is large. For larger bandwidths, the difference is less pronounced and indeed for very large bandwidths the ARPF can even dominate as the depth of the tree in forest resampling becomes a latency bottleneck.

In the case of the ARPF, our implementation involves messages sent between the devices and a single master node. Alternative distributed memory implementations of the ARPF are possible, relying on device-device communication in large quantities and in this case the resulting network congestion indicates even worse performance under an appropriate model in that setting such as a LogP [8] or LogGP [1] model. This is because each device must then communicate with a large number of other devices concurrently, testing the capacity of the network.

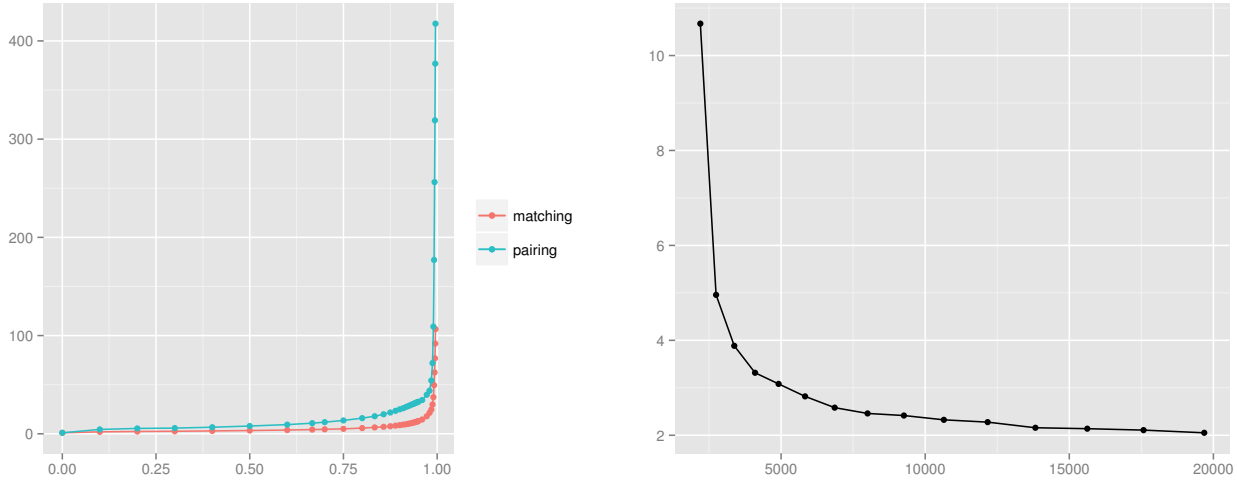
5.2 Connection to existing sampling schemes

Resampling methods other than multinomial can be implemented using trees as well. In order to make this concrete, we assume that the tree is ordered, i.e., the children of each node written in sequence as $\chi_1, \chi_2, \ldots$. This imposes only the constraint that the labelling of children is consistent, and allows the specification of Algorithm 7, which implements Algorithm 5 with a single uniform random variable using the recycling method of [11, Section III.3.7]. Proposition 9 and the Remark that fol-



(a) (b) (c)

Figure 5: Graphs of (a) $\log_2 \bar{d}$ (b) $\log_2 \overline{d_{\max}}$ and (c) $\overline{N^{\text{eff}}}/N$ against τ for various forest selection schemes and choices of σ.



(a) Plot of $\bar{d}$ against τ for $\sigma = 1$.

(b) Plot of $\bar{d}$ against N with $N\tau = 2048$ for $\sigma = 1$ using the matching strategy.

Figure 6: Dependence of $\bar{d}$ on τ

Devices	32	64	128	256	512	1024	2048	4096	8192
ARPF	0.13	0.26	0.52	1.04	2.08	4.16	8.32	16.64	33.27
Forest	0.044	0.063	0.097	0.13	0.16	0.25	0.29	0.41	0.60

Table 1: Average network communication time per iteration (s) for different numbers of devices with a latency of 1ms and bandwidth of 1 Mbps.

Devices	32	64	128	256	512	1024	2048	4096	8192
ARPF	0.0023	0.0036	0.0062	0.011	0.022	0.043	0.084	0.17	0.33
Forest	0.0053	0.0065	0.0078	0.0091	0.010	0.012	0.014	0.016	0.019

Table 2: Average network communication time per iteration (s) for different numbers of devices with a latency of 1ms and bandwidth of 1000 Mbps.

Algorithm 7 Select a value in $\ell(\nu)$ given a $u \in [0,1]$

`select`(ν, u)

1. If $\mathcal{C}(\nu) = \emptyset$, return the only element in $\ell(\nu)$.
2. Otherwise, let $\chi_1, \ldots, \chi_{|\mathcal{C}(\nu)|}$ be the children of ν in order.
3. Set
$$i \leftarrow \min \left\{ k : \sum_{j=[k]} \mathcal{V}_2(\chi_j) \geq u \sum_{j \in [|\mathcal{C}(\nu)|]} \mathcal{V}_2(\chi_j) \right\}.$$
4. Return
$$\texttt{select}\left(\chi_i, \frac{u \sum_{j \in [|\mathcal{C}(\nu)|]} \mathcal{V}_2(\chi_j) - \sum_{j=[i-1]} \mathcal{V}_2(\chi_j)}{\mathcal{V}_2(\chi_i)}\right).$$

lows then imply that we can view this algorithm as a tree-based implementation of the inverse transform method for sampling from a categorical distribution.

Proposition 9. *Assume that the tree is ordered such that for each node its children* (χ_i) *have* $j \in \ell(\chi_i), k \in \ell(\chi_{i+1}) \implies j < k$. *Calling Algorithm 7 with* (ν, u) *returns* $\min\left\{k : \sum_{j \in \ell(\nu) \cap [k]} c^j \geq u \sum_{j \in \ell(\nu)} c^j\right\}$.

Remark. The ordering specified above is w.r.t. the indices of particles and imposes no real constraint on how the tree is actually constructed, as long as a specific order is used in step 2 of Algorithm 7. If an alternative ordering is assumed in Proposition 9 the resulting returned value will still be deterministic and of the form given with a slight modification to account for this alternative ordering.

Multinomial resampling corresponds to sampling N i.i.d. uniform random variables $u^1, \ldots, u^N$ and calling `select`(ν, u^i) for each $i \in [N]$, thereby providing N i.i.d. draws from a categorical distribution. One can view other resampling methods as making dependent draws from a categorical distribution by the inverse transform method by using random variables $u^1, \ldots, u^N$ that are not i.i.d. but for which the distribution of u^K, where K is chosen uniformly at random from $[N]$, is uniform on $[0,1]$ [see, e.g., 12]. Therefore, to implement alternative resampling schemes, one again calls `select`(ν, u^i) for each $i \in [N]$, but with $u^1, \ldots, u^N$ distributed in a dependent fashion as in [12]. The dependent u^i can be interpreted as "trickling" down a tree whose leaves represent ancestor indices in a manner reminiscent of the approach in [7], which most closely resembles the systematic resampling scheme in [20].

5.3 Concluding Remarks

For ease of presentation, we have chosen to work with a particularly simple version of αSMC, in which new samples are proposed using the HMM Markov kernel f. As noted in [34], the algorithm is easily generalized to accommodate other proposal kernels.

This paper, and the methodology of [34] more generally, naturally complements the contribution of [18]. In particular, the methods in the latter allow particles to be "reconstructed" on a device on the basis of only a small amount of communicated information, and could be used in tandem with the algorithms here in appropriate applications.

Both the approaches in Section 4.5 resemble in some ways greedy strategies for solving the classical Partition problem. It would be of interest to consider analogues of more sophisticated solutions to this problem such as those in [19] and [23]. More generally, it would be of interest to have quantitative theoretical results enabling the comparison of particular tree structures and partition selection schemes.

In practice, it may often be the case that devices are homogeneous, with the network connections between any two devices being of similar latency and bandwidth. In such situations, one will often create the structure of the tree at levels above the device and particle layers in a highly structured way. The use of a randomly generated tree may be beneficial, as suggested by the Random adaptation rule of [34, Section 5.4], which had however no hierarchy. A random permutation of the device nodes, in an otherwise constant tree was used in Section 5.1 for this reason.

Finally, this paper is concerned primarily with α matrices that are induced by disjoint unions of complete graphs, and hence have a particular structure. It would be of interest to explore similar results and methodology for more general α matrices.

Acknowledgements

We thank Dr. Kari Heine for assistance with the figures, and three referees for helpful comments. The second author is supported in part by EPSRC grant EP/K023330/1.

A Proofs

Proof of Proposition 1. It is straightforward to show that if $a \in \mathbb{A}_V$ then $\sum_i a^{ij} = \sum_{i \in V} a^{ij} = \mathbb{I}(j \in V)$. Therefore,

$$\begin{aligned} N^{\text{eff}}(a,c) &= \frac{\left(\sum_i \sum_j a^{ij} c^j\right)^2}{\sum_i \left(\sum_j a^{ij} c^j\right)^2} = \frac{\left(\sum_{i \in V} \sum_{j \in V} a^{ij} c^j\right)^2}{\sum_{i \in V} \left(\sum_{j \in V} a^{ij} c^j\right)^2} \\ &= \frac{\left(\sum_{j \in V} c^j\right)^2}{\sum_{i \in V} \left(\sum_{j \in V} a^{ij} c^j\right)^2}. \end{aligned}$$

We define $x_1 := \sum_{j \in V} c^j$, $x_2 := \sum_{j \in V'} c^j$, $y_1 := \sum_{i \in V} \left(\sum_{j \in V} a^{ij} c^j\right)^2$, $y_2 := \sum_{i \in V'} \left(\sum_{j \in V'} (a')^{ij} c^j\right)^2$ and $\tilde{y}_2 := \sum_{i \in V'} \left(\sum_{j \in V'} (\tilde{a}')^{ij} c^j\right)^2$. This allows us to write $N^{\text{eff}}(a,c) = x_1^2/y_1$, $N^{\text{eff}}(a',c) = x_2^2/y_2$ and $N^{\text{eff}}(\tilde{a}',c) = x_2^2/\tilde{y}_2$, and since $V \cap V' = \emptyset$, $N^{\text{eff}}(a+a',c) = (x_1+x_2)^2/(y_1+y_2)$ and $N^{\text{eff}}(a+\tilde{a}',c) = (x_1+x_2)^2/(y_1+\tilde{y}_2)$.

1. The lower bound holds because

$$\begin{aligned} \left(\sum_i \sum_j a^{ij} c^j\right)^2 &= \sum_{i,k} \left(\sum_j a^{ij} c^j\right)\left(\sum_j a^{kj} c^j\right) \\ &\geq \sum_i \left(\sum_j a^{ij} c^j\right)^2, \end{aligned}$$

so $N^{\text{eff}}(a,c) \geq 1$. The upper bound holds because, using Jensen's inequality,

$$\begin{aligned} \left(\sum_i \sum_j a^{ij} c^j\right)^2 &= |V|^2 \left(\sum_{i \in V} |V|^{-1} \sum_{j \in V} a^{ij} c^j\right)^2 \\ &\leq |V|^2 \sum_{i \in V} |V|^{-1} \left(\sum_{j \in V} a^{ij} c^j\right)^2 \\ &= |V| \sum_{i \in V} \left(\sum_{j \in V} a^{ij} c^j\right)^2, \end{aligned}$$

so $N^{\text{eff}}(a,c) \leq |V|$. The upper bound is attained when $a^{ij} = |V|^{-1} \mathbb{I}(i,j \in V)$ since then

$$\begin{aligned} \sum_{i \in V} \left(\sum_{j \in V} a^{ij} c^j\right)^2 &= \sum_{i \in V} \left(\sum_{j \in V} |V|^{-1} c^j\right)^2 \\ &= |V|^{-1} \left(\sum_{j \in V} c^j\right)^2. \end{aligned}$$

2. The result follows from

$$\begin{aligned} & N^{\text{eff}}(a,c) + N^{\text{eff}}(a',c) - N^{\text{eff}}(a+a',c) \\ = \; & \frac{x_1^2}{y_1} + \frac{x_2^2}{y_2} - \frac{(x_1+x_2)^2}{y_1+y_2} \\ = \; & x_1^2 \left\{\frac{y_2}{y_1(y_1+y_2)}\right\} + x_2^2 \left\{\frac{y_1}{y_2(y_1+y_2)}\right\} - \frac{2x_1x_2}{y_1+y_2} \\ = \; & \frac{y_1 y_2}{y_1+y_2} \left(\frac{x_1}{y_1} - \frac{x_2}{y_2}\right)^2 \geq 0, \end{aligned}$$

with equality only when $\frac{x_1}{y_1} = \frac{x_2}{y_2}$, corresponding to $\frac{\sum_{j \in V} c^j}{N^{\text{eff}}(a,c)} = \frac{\sum_{j \in V'} c^j}{N^{\text{eff}}(a',c)}$.

3. Since $N^{\text{eff}}(a',c) \leq N^{\text{eff}}(\tilde{a}',c) \implies y_2 \geq \tilde{y}_2$,

$$\begin{aligned} & N^{\text{eff}}(a+\tilde{a}',c) - N^{\text{eff}}(a+a',c) \\ = \; & \frac{(x_1+x_2)^2}{y_1+\tilde{y}_2} - \frac{(x_1+x_2)^2}{y_1+y_2} \geq 0, \end{aligned}$$

with equality only when $y_2 = \tilde{y}_2$, corresponding to $N^{\text{eff}}(a',c) \leq N^{\text{eff}}(\tilde{a}',c)$.

4. We have

$$\begin{aligned} & N^{\text{eff}}(a_1+a_2,c) \\ = \; & \frac{(x_1+x_2)^2}{y_1+y_2} \\ = \; & \frac{x_1^2}{y_1} \cdot \frac{y_1}{y_1+y_2} + \frac{2x_1x_2}{y_1+y_2} + \frac{x_2^2}{y_2} \cdot \frac{y_2}{y_1+y_2} \\ \geq \; & \min_i \left\{\frac{x_i^2}{y_i}\right\} = \min\left\{N^{\text{eff}}(a,c), N^{\text{eff}}(a',c)\right\}. \end{aligned}$$

□

Proof of Proposition 2. We prove the result for $\mathbb{G}$ since the result for $\mathbb{A}_{\mathbb{G}}$ then follows. Let $V \subseteq [N]$ and consider $G_1 = (V, E_1)$, $G_2 = (V, E_2)$ and $G_3 = (V, E_3)$. It suffices to check that $\preceq$ is reflexive ($G_1 \preceq G_1$), antisymmetric ($G_1 \preceq G_2$ and $G_2 \preceq G_1$ implies $G_1 = G_2$) and transitive ($G_1 \preceq G_2$ and $G_2 \preceq G_3$ implies $G_1 \preceq G_3$). Since $E_1 \subseteq E_1$, it follows that $G_1 \preceq G_1$. When $G_1 \preceq G_2$ and $G_2 \preceq G_1$, this implies $E_1 \subseteq E_2$ and $E_2 \subseteq E_1$ and it follows that $E_1 = E_2$ and so $G_1 = G_2$. Finally, $G_1 \preceq G_2$ and $G_2 \preceq G_3$ implies that $E_1 \subseteq E_2$ and $E_2 \subseteq E_3$ and so $E_1 \subseteq E_3$ and therefore $G_1 \preceq G_3$. □

Proof of Proposition 3. Since $a \preceq \tilde{a}$ we have that $a, \tilde{a} \in \mathbb{A}_{\mathbb{G}} \cap \mathbb{A}_V$ for some $V \subseteq [N]$. Therefore, for some $K, \tilde{K} \in [N]$ we can write $a = \sum_{k \in [K]} \phi(\kappa(V_k))$ and $\tilde{a} = \sum_{\tilde{k} \in [\tilde{K}]} \phi(\kappa(\tilde{V}_{\tilde{k}}))$ where each V_k and $\tilde{V}_{\tilde{k}}$ are subsets of

V. Since $a \preceq \tilde{a}$, for each $k \in [K]$ there exists $\tilde{k} \in [\tilde{K}]$ such that $V_k \subseteq \tilde{V}_{\tilde{k}}$. We now define a sequence, with $a_0 = a$, and for $i \in [\tilde{K}]$

$$a_i := a_{i-1} + \phi(\kappa(\tilde{V}_i)) - \sum_{k \in [K], V_k \cap \tilde{V}_i \neq \emptyset} \phi(\kappa(V_k)),$$

and note that $a_{\tilde{K}} = \tilde{a}$. Now for each $i \in [\tilde{K}]$, $a_i \in \mathbb{A}_{\mathbb{G}} \cap \mathbb{A}_V$ and letting $\check{V}_i := \bigcup_{j=1}^{i} \tilde{V}_j$ and

$$b_i := \sum_{k \in [K], V_k \cap \check{V}_i = \emptyset} \phi(\kappa(V_k)) + \sum_{j=1}^{i-1} \phi(\kappa(\tilde{V}_j)),$$

we can write $a_{i-1} = b_i + \sum_{k \in [K], V_k \cap \tilde{V}_i \neq \emptyset} \phi(\kappa(V_k))$ and $a_i = b_i + \phi(\kappa(\tilde{V}_i))$. From the first part of Proposition 1 we have that $N^{\text{eff}}\left(\sum_{k \in [K], V_k \cap \tilde{V}_i \neq \emptyset} \phi(\kappa(V_k)), c\right) \leq \left|\tilde{V}_i\right| = \phi(\kappa(\tilde{V}_i))$ and so by the monotonicity property in Proposition 1 we have $N^{\text{eff}}(a_{i-1}, c) \leq N^{\text{eff}}(a_i, c)$ for each $i \in [\tilde{K}]$. It follows that $N^{\text{eff}}(a, c) \leq N^{\text{eff}}(\tilde{a}, c)$. □

Proof of Proposition 4. The first inequality follows from Proposition 3 since $a \preceq \tilde{a}$. For the second inequality, assume that $\min_k |V_k|^{-1} N^{\text{eff}}(a_k, c) \geq D$. This implies that for any $k \in [K]$,

$$\sum_{i \in V_k} \left(\sum_{j \in V_k} a_k^{ij} c^j \right)^2 \leq \frac{1}{D|V_k|} \left(\sum_{j \in V_k} c^j \right)^2.$$

Therefore

$$\begin{aligned} N^{\text{eff}}(a, c) &= \frac{\left(\sum_{k \in [K]} \sum_{j \in V_k} c^j\right)^2}{\sum_{k \in [K]} \sum_{i \in V_k} \left(\sum_{j \in V_k} a_k^{ij} c^j\right)^2} \\ &\geq D \frac{\left(\sum_{k \in [K]} \sum_{j \in V_k} c^j\right)^2}{\sum_{k \in [K]} |V_k|^{-1} \left(\sum_{j \in V_k} c^j\right)^2} \\ &= D N^{\text{eff}}(\tilde{a}, c). \end{aligned}$$

□

Proof of Proposition 5. The proof is by induction. Note that $\tau \in [0, 1]$. If $|V| = 1$ then $|P| = 1$ and $N_c^{\text{eff}}(P) = 1$, so the claim is true. Now assume that the claim holds true for all V with $|V| \in [s-1]$, $s \in \mathbb{N}$, and consider the case where $|V| = s$. First, note that if $P = \{V\}$ then $N^{\text{eff}}(P, c) = |V|$ by Proposition 1 and so if $|P| = 1$ the claim is true. It remains to check that if $|P| > 1$ then $a = \sum_{k \in [K]} a_k$ satisfies the claim, where $a_k =$ `choose.a`$(V_k, \tau/\rho(P, c))$. By the induction hypothesis, for each $k \in [K]$, $N^{\text{eff}}(a_k, c)/|V_k| \geq \tau/\rho(P, c)$ since $|V_k| \in [s-1]$ and $\tau/\rho(P, c) \in [0, 1]$. Then by Proposition 4, with $\tilde{a} = \sum_{k \in [K]} \phi(\kappa(V_k))$,

$$\begin{aligned} N^{\text{eff}}(a, c) &\geq \min_k \left\{ \frac{N^{\text{eff}}(a_k, c)}{|V_k|} \right\} N^{\text{eff}}(\tilde{a}, c) \\ &\geq \frac{\tau}{\rho(P, c)} \rho(P, c) |V| = \tau |V|, \end{aligned}$$

and we conclude. □

Proof of Proposition 6. Let $s_\nu(j)$ denote the probability that Algorithm 5 returns $j \in \ell(\nu)$. Given $j \in \ell(\nu)$, let ν^1 be the parent of ν_j, ν^2 be the parent of ν^1, etc., until $\nu^m = \nu$ is the parent of ν^{m-1}. Then

$$\begin{aligned} s_\nu(j) &= \frac{c^j}{\sum_{k \in \ell(\nu^1)} c^k} \prod_{i \in [m-1]} \frac{\sum_{k \in \ell(\nu^i)} c^k}{\sum_{k \in \ell(\nu^{i+1})} c^k}, \\ &= \frac{c^j}{\sum_{k \in \ell(\nu)} c^k} = p_{\mathcal{S}(\nu)}(j). \end{aligned}$$

□

Proof of Proposition 7. We define $x_i := \sum_{j \in V_i} c^j$ for $i \in [2M]$ and it suffices to show that P' minimizes the denominator of $\rho(\cdot, c)$,

$$r(P', c) := (2|V_1|)^{-1} \sum_{S \in P'} \left(\sum_{i \in S} x_i \right)^2,$$

since each element of any pairing of P is of size $2|V_1|$. We first prove that $V_1 \cup V_{2M}$ is a member of at least one pairing of P that minimizes $r(\cdot, c)$. Indeed, assume that a pairing $\check{P}$ that minimizes $r(\cdot, c)$ is given. We will show that a pairing $\check{P}'$ containing $V_1 \cup V_{2M}$ exists for which $r(\check{P}', c) \leq r(\check{P}, c)$. Let $V_1 \cup V_j$ and $V_k \cup V_{2M}$ be elements of $\check{P}$. We define $\check{P}' = \check{P} \setminus \{V_1 \cup V_j, V_k \cup V_{2M}\} \cup \{V_1 \cup V_{2M}, V_k \cup V_j\}$. Then

$$\begin{aligned} & 2|V_1| \left(r(\check{P}, c) - r(\check{P}', c) \right) \\ &= \sum_{S \in \check{P}} \left(\sum_{i \in S} x_i \right)^2 - \sum_{S \in \check{P}'} \left(\sum_{i \in S} x_i \right)^2 \\ &= (x_1 + x_j)^2 + (x_k + x_{2M})^2 \\ &\quad - (x_1 + x_{2M})^2 - (x_k + x_j)^2 \\ &= 2x_1 x_j + 2x_k x_{2M} - 2x_1 x_{2M} - 2x_k x_j \\ &= 2x_k(x_{2M} - x_j) - 2x_1(x_{2M} - x_j) \\ &= 2(x_k - x_1)(x_{2M} - x_j) \geq 0, \end{aligned}$$

since x_1 and x_{2M} are the minimal and maximal values of $\{x_i : i \in [2M]\}$, respectively.

Now, let $\check{P}'$ be a pairing of P and $S = V_1 \cup V_{2M} \in \check{P}'$. Then $r(\check{P}', c) = r(\{S\}, c) + r(\check{P}' \setminus \{S\}, c)$ and so it follows that if $\check{P}'$ minimizes $r(\cdot, c)$ then $\check{P}' \setminus \{S\}$ is a pairing of $P \setminus \{V_1, V_{2M}\}$ that minimizes $r(\cdot, c)$. It then follows that at least one pairing $\check{P}'$ of P that minimizes $r(\cdot, c)$ is the union of $V_1 \cup V_{2M}$ and a pairing of $P \setminus \{V_{2M}, V_1\}$ that minimizes $r(\cdot, c)$. But then the argument above shows that $V_2 \cup V_{2M-1}$ is a valid element of such a minimizing pairing. Continuing, we obtain that P' is a pairing of P that minimizes $r(\cdot, c)$ and we conclude. □

Proof of Proposition 8. From (14) we can write $\rho(P_{k,l}, c) = \rho(P, c)\frac{r(P,c)}{r(P_{k,l},c)}$ where $r(P, c) := \sum_{S \in P} |S| \left(|S|^{-1} \sum_{j \in S} c^j\right)^2$. It suffices therefore to find $k, l \in [K]$ minimizing $r(P_{k,l}, c)$. Letting $m_i = |V_i|$ and $x_i = \sum_{j \in V_i} c^j$, we can write

$$\begin{aligned} r(P_{k,l}, c) &= r(P, c) - \frac{x_k^2}{m_k} - \frac{x_l^2}{m_l} + \frac{(x_k + x_l)^2}{m_k + m_l} \\ &= r(P, c) - \frac{m_k m_l}{m_k + m_l}\left(\frac{x_k}{m_k} - \frac{x_l}{m_l}\right)^2, \end{aligned}$$

the equality following along the same lines as the proof of the second part of Proposition 1, and we conclude. □

Proof of Proposition 9. Let $\chi_\nu := (\chi_{\nu,1}, \ldots, \chi_{\nu,K})$ be the ordered children of ν, where $|\mathcal{C}(\nu)| = K$. To alleviate notation, we define $c(\chi_{\nu,k}) := \sum_{j \in \ell(\chi_{\nu,k})} c^j$, $k_\nu(u) := \min\left\{k : \sum_{j \in [k]} c(\chi_{\nu,j}) \geq u \sum_{j \in \ell(\nu)} c^j\right\}$ and $s_\nu(k) := \sum_{j \in [k]} c(\chi_{\nu,j})$. Algorithm 7 with input (ν, u) returns $f_\nu(u)$, where

$$f_\nu(u) := \begin{cases} \min \ell(\nu) & |\ell(\nu)| = 1, \\ f_{\chi_{\nu,k_\nu(u)}}\left(\frac{u \sum_{j \in \ell(\nu)} c^j - s_\nu(k_\nu(u)-1)}{c(\chi_{\nu,k_\nu(u)})}\right) & \text{otherwise.} \end{cases}$$

Now we prove by induction that $f_\nu(u) = \min\left\{k : \sum_{j \in \ell(\nu) \cap [k]} c^j \geq u \sum_{j \in \ell(\nu)} c^j\right\}$. If $|\ell(\nu)| = 1$, the claim is trivially true. Now assume the claim is true for $\{\nu : |\ell(\nu)| \in [p-1]\}$ and consider ν with $|\ell(\nu)| = p > 1$. We have

$$f_\nu(u) = f_{\chi_{\nu,k_\nu(u)}}\left(\frac{u \sum_{j \in \ell(\nu)} c^j - s_\nu(k_\nu(u) - 1)}{c(\chi_{\nu,k_\nu(u)})}\right)$$

and we can apply the induction hypothesis since $|\ell(\chi_{\nu,k_\nu(u)})| \in [p-1]$. Therefore, letting $y := u \sum_{j \in \ell(\nu)} c^j - s_\nu(k_\nu(u)-1)$, and $x := \chi_{\nu,k_\nu(u)}$ we can write $f_\nu(u)$ as

$$\begin{aligned} &\min\left\{k : \sum_{j \in \ell(x) \cap [k]} c^j \geq \frac{y}{c(x)} \sum_{j \in \ell(x)} c^j\right\} \\ = &\min\left\{k : \sum_{j \in \ell(x) \cap [k]} c^j \geq u \sum_{j \in \ell(\nu)} c^j - s_\nu(k_\nu(u) - 1)\right\} \\ = &\min\left\{k : \sum_{j \in \ell(\nu) \cap [k]} c^j \geq u \sum_{j \in \ell(\nu)} c^j\right\}, \end{aligned}$$

and we conclude. □

References

[1] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating long messages into the LogP model – one step closer towards a realistic model for parallel computation. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 95–105, 1995.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: numerical methods.* Prentice-Hall, Inc., 1989.

[3] M. Bolić, P. M. Djurić, and S. Hong. Resampling algorithms and architectures for distributed particle filters. *IEEE Trans. Signal Process.*, 53(7):2442–2450, 2005.

[4] F. Cérou, P. Del Moral, and A. Guyader. A nonasymptotic variance theorem for unnormalized Feynman Kac particle models. *Ann. Inst. Henri Poincaré Probab. Stat.*, 47(3):629–649, 2011.

[5] N. Chopin. A sequential particle filter method for static models. *Biometrika*, 89(3):539–552, 2002.

[6] N. Chopin, P. E. Jacob, and O. Papaspiliopoulos. SMC^2: an efficient algorithm for sequential analysis of state space models. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 75(3):397–426, 2013.

[7] D. Crisan and T. Lyons. Minimal entropy approximations and optimal algorithms for the filtering problem. *Monte Carlo methods and applications*, 8(4): 343–356, 2002.

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von

Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, 1993.

[9] P. Del Moral and A. Guionnet. On the stability of interacting processes with applications to filtering and genetic algorithms. *Ann. Inst. Henri Poincaré Probab. Stat.*, 37(2):155–194, 2001.

[10] P. Del Moral, A. Doucet, and A. Jasra. Sequential Monte Carlo samplers. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 68(3):411–436, 2006.

[11] L. Devroye. *Non-uniform random variate generation*. Springer Verlag, 1986.

[12] R. Douc, O. Cappé, and E. Moulines. Comparison of resampling schemes for particle filtering. In *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis*, pages 64–69, 2005.

[13] A. Doucet and A. M. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. In D. Crisan and B. Rozovsky, editors, *Handbook of Nonlinear Filtering*. Oxford University Press, 2008.

[14] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Stat. Comput.*, 10(3):197–208, 2000.

[15] W. D. Hillis and G. L. Steele Jr. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, 1986.

[16] R. W. Hockney. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel computing*, 20(3):389–398, 1994.

[17] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, 41(3):59–72, 2007.

[18] S. H. Jun, L. Wang, and A. Bouchard-Côté. Entangled Monte Carlo. In *Advances in Neural Information Processing Systems*, pages 2726–2734, 2012.

[19] N. Karmarkar and R. M. Karp. The differencing method of set partitioning. Technical report, University of California, Berkeley, 1982.

[20] G. Kitagawa. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *J. Comput. Graph. Statist.*, 5(1):1–25, 1996.

[21] D. E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wes, 3rd edition, 1997.

[22] A. Kong, J. S. Liu, and W. H. Wong. Sequential imputations and bayesian missing data problems. *Journal of the American statistical association*, 89(425): 278–288, 1994.

[23] R. E. Korf. From approximate to optimal solutions: A case study of number partitioning. In *Proceedings of the 14th international joint conference on Artificial intelligence*, pages 266–272, 1995.

[24] A. Lee, C. Yau, M. B. Giles, A. Doucet, and C. C. Holmes. On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. *J. Comput. Graph. Statist.*, 19(4): 769–789, 2010.

[25] J. S. Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and Computing*, 6(2):113–119, 1996.

[26] J. S. Liu and R. Chen. Blind deconvolution via sequential imputations. *J. Amer. Statist. Assoc.*, 90 (430):567–576, 1995.

[27] S. Mertens. The easiest hard problem: number partitioning. In A. Percus, G. Istrate, and C. Moore, editors, *Computational Complexity and Statistical Physics*, pages 125–139. Oxford University Press, 2006.

[28] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra. Performance analysis of MPI collective operations. *Cluster Computing*, 10(2):127–143, 2007.

[29] J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison–Wesley, 1998.

[30] M. A. Suchard and A. Rambaut. Many-core algorithms for statistical phylogenetics. *Bioinformatics*, 25(11):1370–1376, 2009.

[31] C. Vergé, C. Dubarry, P. Del Moral, and E. Moulines. On parallel implementation of sequential Monte Carlo methods: the island particle model. *Stat. and Comput.* To appear.

[32] N. Whiteley. Stability properties of some particle filters. *Ann. Appl. Probab.*, 23(6):2500–2537, 2013.

[33] N. Whiteley and A. Lee. Twisted particle filters. *Ann. Statist.*, 42(1):115–141, 2014.

[34] N. Whiteley, A. Lee, and K. Heine. On the role of interaction in sequential Monte Carlo algorithms. *arXiv preprint 1309.2918*, 2013. To appear in *Bernoulli*.