Statistica Sinica **34** (2024), 1093-1113
doi:https://doi.org/10.5705/ss.202022.0243

A DIVIDE-AND-CONQUER SEQUENTIAL MONTE CARLO APPROACH TO HIGH-DIMENSIONAL FILTERING

Francesca R. Crucinio* and Adam M. Johansen

University of Warwick

Abstract: We propose a divide-and-conquer approach to filtering. The proposed approach decomposes the state variable into low-dimensional components, to which standard particle filtering tools can be successfully applied, and recursively merges them to recover the full filtering distribution. This approach is less dependent on factorizing transition densities and observation likelihoods than are competing approaches, and can be applied to a broader class of models. We compare the performance of the proposed approach with that of state-of-the-art methods on a benchmark problem, and show that the proposed method is broadly comparable in settings in which the other methods are applicable, and that it can be applied in settings in which they cannot.

Key words and phrases: Data assimilation, marginal particle filter, particle filtering, spatio-temporal models, state-space model.

1. Introduction

Particle filters (PFs), an instance of sequential Monte Carlo (SMC) methods, are a popular class of algorithms for performing state estimation for state-space models (SSM), also known as general-state-space hidden Markov models. We consider the class of SSMs with a latent $\mathbb{R}^d$-valued process $(X_t)_{t\geq 1}$ and conditionally independent $\mathbb{R}^p$-valued observations $(Y_t)_{t\geq 1}$. Such an SSM $(X_t, Y_t)_{t\geq 1}$ is defined by the transition density $f_t(x_{t-1}, x_t)$ of the latent process, with the convention that $f_1(x_0, x_1) \equiv f_1(x_1)$, and by the observation likelihood $g_t(y_t|x_t)$. In this work, we are interested in approximating the sequence of filtering distributions, $(p(x_t|y_{1:t}))_{t\geq 1}$, that is, at each time t, the distribution of the latent state at that time given the observations obtained by that time.

Basic PF algorithms are known to suffer from the curse of dimensionality, requiring an exponential increase in computational requirements as the dimension d grows, limiting their applicability to large systems (Rebeschini and Van Handel (2015); Bengtsson, Bickel and Li (2008)). Although the ensemble Kalman filter (Evensen (2009)) can handle high-dimensional problems, it involves approximations that do not disappear, even asymptotically and does not perform well if the model is far from being linear and Gaussian (Lei, Bickel and Snyder (2010)).

To extend the use of PFs to high-dimensional problems, it is natural to

*Corresponding author.

exploit the fact that dependencies in high-dimensional SSMs are often local in space. This enables us to decompose the filtering problem into a collection of local low-dimensional problems that can somehow be combined; examples of this strategy are the block PF (BPF; Rebeschini and Van Handel (2015)), space-time PFs (STPFs; Beskos et al. (2017)), and nested sequential Monte Carlo (NSMC) methods (Næsseth, Lindsten and Schön (2015, 2019)).

We propose a divide-and-conquer approach that divides the state space into smaller subsets, over which we can apply standard particle filtering ideas. These smaller subsets are then recursively merged in a principled manner to obtain approximations over the full state space. Our method is an extension of the divide-and-conquer sequential Monte Carlo (DaC-SMC) algorithm introduced by Lindsten et al. (2017) to the filtering context, where we exploit ideas akin those in Klaas, De Freitas and Doucet (2005) and Lin et al. (2005) to marginalize out the past $x_{1:t-1}$ at a given time t.

In order to apply the DaC-SMC algorithm to the filtering problem, we define a nonstandard sequence of targets, evolving both in space and in time: at a given time t, we define d univariate targets serving as proxies for the marginals of the filtering distribution, $p(x_t(i)|y_{1:t})$, for $i = 1, \dots, d$. Then, we iteratively combine these lower-dimensional targets to obtain approximations of the higher-dimensional marginals (e.g., $p(x_t(i : i+1)|y_{1:t})$), until we recover the full filtering distribution, $p(x_t|y_{1:t}) \equiv p(x_t(1 : d)|y_{1:t})$.

Unlike the NSMC and STPF algorithms, this approach does not require analytic expressions for the marginals of the transition density or observation likelihood. Instead, it requires only point-wise evaluations of f_t and g_t, making it suitable for high-dimensional SSMs in which the observations are correlated in nontrivial ways (cf., the model in Section 4.2), as is common in practice (see, e.g., Chib, Omori and Asai (2009, Sec. 2)).

We review the basic ideas of particle filtering, its marginal variant, and the divide-and-conquer SMC algorithm in Section 2. In Section 3, we extend the ideas underlying the DaC-SMC algorithm to the filtering problem, and we discuss strategies to improve the computational cost and accuracy. In Section 4, we compare the performance of the divide-and-conquer approach with that of the NSMC and STPF algorithms on a simple linear Gaussian SSM, for which the Kalman filter provides the exact filtering distribution. The results of our experiments show that the errors from approximating the true filtering distribution obtained with the divide-and-conquer approach are comparable with those of the NSMC and STPF algorithms. Then, we consider a spatial model whose correlation structure in the observation model makes it impossible to apply the NSMC or STPF algorithm (at least without additional approximations). We show empirically that the proposed approach can recover stable estimates of the filtering distribution which, in small dimensional settings, coincide with those obtained using a bootstrap PF with a large number of particles.

2. Background

2.1. Particle filtering

Here, we describe the basic SMC approach, often referred to as sequential importance resampling (Doucet and Johansen (2011, p.15)); refer to Liu (2001) and Chopin and Papaspiliopoulos (2020) for a more extensive treatment.

Given the sequence of unnormalized target densities $(\gamma_t)_{t\geq 1}$, with

$$\gamma_t(x_{1:t}) = p(x_{1:t}, y_{1:t}) = \prod_{k=1}^{t} f_k(x_{k-1}, x_k) g_k(y_k|x_k), \tag{2.1}$$

defined on $(\mathbb{R}^d)^t$, PFs proceed iteratively, and, at time $t-1$, approximate

$$\pi_{t-1} := \frac{\gamma_{t-1}}{\int \gamma_{t-1}(x_{1:t-1})\mathrm{d}x_{1:t-1}}$$

using a cloud of particles $\{x_{1:t-1}^n\}_{n=1}^N$. The particles are propagated forward in time using a Markov kernel $K_t(x_{1:t-1}, \cdot)$, reweighted using the weight function $w_t := \gamma_t/\gamma_{t-1} \otimes K_t$, and resampled to obtain a new particle population $\{x_{1:t}^n\}_{n=1}^N$ approximating π_t.

Standard PFs formally target distributions (2.1) with a dimension that increases at every time step t. However, we are often interested only in the final time marginal of the approximated distributions, in this case, the filtering distribution. An alternative to this approach is given by marginal particle filters (MPFs; Klaas, De Freitas and Doucet (2005)), and the closely related ideas of Lin et al. (2005). MPFs target the filtering distribution directly:

$$\gamma_t(x_t) = p(x_t|y_{1:t}) = g_t(y_t|x_t) \int f_t(x_{t-1}, x_t) p(x_{t-1}|y_{1:t-1}) \mathrm{d}x_{t-1}. \tag{2.2}$$

Furthermore, because the integral w.r.t. x_{t-1} is intractable, MPFs replace $p(x_{t-1}|y_{1:t-1})$ with its particle approximation obtained at time $t-1$. Given the new sequence of targets, MPFs proceed as standard PFs. However, whenever we need to compute an integral w.r.t. x_{t-1}, this is approximated using π_{t-1}^N, obtained by normalizing γ_{t-1}^N, the particle approximation of $p(x_{t-1}|y_{1:t-1})$. Basic MPFs incur an $O(N^2)$ cost for each time step, because of the presence of the integral w.r.t. x_{t-1} in the weight computations, although lower cost strategies might be employed in some cases (Lin et al. (2005); see also Klaas et al. (2006)).

2.2. PFs for high-dimensional problems

We briefly summarize three classes of PFs that use space decompositions to tackle the filtering problem, which we believe to be the state-of-the-art in Monte Carlo approximations of high-dimensional filtering distributions.

The BPF (Rebeschini and Van Handel (2015)) algorithm relies on a decomposition of the state space $\mathbb{R}^d$ into lower-dimensional blocks on which, at each t, one step of a standard PF is run. The approximation of the filtering distribution over the whole state space is obtained as the product of the lower-dimensional approximations on each block. BPFs are inherently biased, because of the decomposition into blocks; however, this bias can be eliminated asymptotically by allowing the blocks to grow at an appropriate rate with computational effort.

STPFs (Beskos et al. (2017)) exploit local dependence structures in the observation y_t to gradually introduce the likelihood term by decomposing the space dimension into smaller subsets, and running N independent PFs on each of the subsets (also called islands). These are then combined using an importance resampling step, which guarantees asymptotically consistent approximations, in contrast to BPFs. Crucial to the implementation of an STPF algorithm is that the joint law (2.1) at time t can be factorized so that the marginal of $x_t(i)$, given the observations and the past, depends only on a neighborhood of $x_t(i)$, $\{x_t(j) : j \in \mathcal{A}\}$, for some $\mathcal{A} \subset \{1, \dots, d\}$, for all $i = 1, \dots, d$. STPFs are particularly useful for SSMs which are time discretizations of stochastic differential equations, because in this case, we can build time discretization schemes that guarantee analytical forms for the marginals (Akyildiz, Crisan and Miguez (2022)). A marginal version of STPFs also exists (Beskos et al. (2017); Xu and Jasra (2019)).

A nested sequential Monte Carlo algorithm (NSMC; Næsseth, Lindsten and Schön (2015)) treats the problem of recovering the filtering distribution as a smoothing problem. In this case, the time variable is replaced with the dimension d, and the method approximates the fully adapted proposal of Pitt and Shephard (1999) using an inner SMC iteration, and then uses the result in the outer level, which corresponds to a standard forward-filtering backward-simulation algorithm. An NSMC algorithm is particularly well suited for Markov random fields in which the temporal and the spatial components can be separated, because this makes the backward simulation straightforward.

2.3. Divide and conquer SMC

The DaC-SMC algorithm (Lindsten et al. (2017)) is an extension of the standard SMC algorithm in which a collection of (unnormalized) target distributions $(\gamma_u)_{u \in \mathbb{T}}$ is indexed by the nodes of a rooted tree, $\mathbb{T}$, and particles evolve from the leaves to the root, $\mathfrak{R}$, rather than along a sequence of distributions indexed by (algorithmic) time. This method shares many of the convergence properties of the standard SMC algorithm (Kuntz, Crucinio and Johansen (2023)).

The target distributions are defined on spaces with dimension that grows as we progress up the tree: for each u, $\pi_u \propto \gamma_u$ is a density over $\mathbb{R}^{|\mathbb{T}_u|}$, where $\mathbb{T}_u$ denotes the sub-tree of $\mathbb{T}$ rooted at u (obtained by removing all nodes from $\mathbb{T}$, except for u and its descendants), and $|\mathbb{T}_u|$ denotes its cardinality. We

focus here on the case in which the state space is $\mathbb{R}^d$. However, essentially the same construction allows for more general spaces, including those with discrete components.

As in a standard SMC algorithm, each distribution γ_u is approximated by a particle population $\{x_u^n\}_{n=1}^N$. However, these distributions are merged whenever the corresponding branches of $\mathbb{T}$ merge, rather than evolving "linearly". For simplicity, we describe the case in which $\mathbb{T}$ is a binary tree, and each non-leaf node u has two children, namely, a left child $\ell(u)$ and a right child $r(u)$.

If u is a leaf node, the algorithm performs a simple importance sampling step with the proposal K_u and the importance weight $w_u := \gamma_u/K_u$ to obtain a weighted particle population $\{x_u^n, w_u^n\}_{n=1}^N$ approximating γ_u. Otherwise, to obtain such a particle population, we gather the particle populations associated with each of the children, and compute the (weighted) product form estimator (Kuntz, Crucinio and Johansen (2022))

$$\gamma_{\mathcal{C}_u}^N = \frac{1}{N^2} \sum_{n_1=1}^N \sum_{n_2=1}^N w_{\ell(u)}(x_{\ell(u)}^{n_1}) w_{r(u)}(x_{r(u)}^{n_2}) \delta_{(x_{\ell(u)}^{n_1}, x_{r(u)}^{n_2})} \tag{2.3}$$

to approximate the product of the marginal distributions $\gamma_{\mathcal{C}_u} := \gamma_{\ell(u)} \times \gamma_{r(u)}$. The $O(N^2)$ cost of evaluating $\gamma_{\mathcal{C}_u}^N$ can be prohibitively large; we discuss lower cost alternatives in Section 3.3.

We reweight the particle approximation $\gamma_{\mathcal{C}_u}^N$ of $\gamma_{\mathcal{C}_u}$ to target γ_u; the resulting *mixture* (importance) weights

$$m_u(x_{\ell(u)}, x_{r(u)}) := \frac{\gamma_u(x_{\ell(u)}, x_{r(u)})}{\gamma_{\ell(u)}(x_{\ell(u)}) \gamma_{r(u)}(x_{r(u)})} \tag{2.4}$$

capture the mismatch between γ_u and $\gamma_{\ell(u)} \times \gamma_{r(u)}$, and are incorporated prior to resampling, similarly to the auxiliary "twisting" function in the auxiliary PF (see, e.g., Chopin and Papaspiliopoulos (2020, Chap. 10)). This leads to weights of the form

$$\tilde{w}_u(x_{\ell(u)}, x_{r(u)}) := w_{\ell(u)}(x_{\ell(u)}) w_{r(u)}(x_{r(u)}) m_u(x_{\ell(u)}, x_{r(u)}). \tag{2.5}$$

Resampling N times from $\tilde{w}_u \gamma_{\mathcal{C}_u}^N$, using any unbiased resampling scheme (cf., Gerber, Chopin and Whiteley (2019)), we obtain an equally weighted particle population $\{\tilde{x}_u^n, w_u^n = 1\}_{n=1}^N$ approximating γ_u. If necessary, we can then apply a π_u-invariant Markov kernel K_u to rejuvenate the particles. This is summarized in Algorithm 1, which is applied to the root node to carry out the sampling process.

The DaC approach in Algorithm 1 is a special case of that considered in Lindsten et al. (2017) and Kuntz, Crucinio and Johansen (2023), in which the target at each non-leaf node is defined on the product of the spaces on which each of its child targets are defined. DaC-SMC is particularly amenable to a distributed implementation (Lindsten et al. (2017, Sec. 5.3)).

Algorithm 1 dac_smc(u) for u in $\mathbb{T}$.

if u is a leaf **then**
  Initialize: draw $x_u^n \sim K_u$ and compute $w_u^n = \gamma_u / K_u$ for all $n \leq N$.
else
  Recurse: set $(\{x_v^n, w_v^n\}_{n=1}^N) :=$ dac_smc(v) for v in $\{\ell(u), r(u)\}$ and obtain $\gamma_{\mathcal{C}_u}^N$ in (2.3).
  Merge: compute $\tilde{w}_u^{(n_1,n_2)}$ in (2.5) for all $n_1, n_2 \leq N$.
  Resample: draw $\{\tilde{x}_u^n\}_{n=1}^N$ using weights $\tilde{w}_u^{(n_1,n_2)}$ and set $w_u^n = 1$ for all $n \leq N$.
  (Optionally): draw $x_u^n \sim K_u(\tilde{x}_u^n, \cdot)$ for all $n \leq N$.
  (Otherwise): set $x_u^n = \tilde{x}_u^n$ for all $n \leq N$.
end if

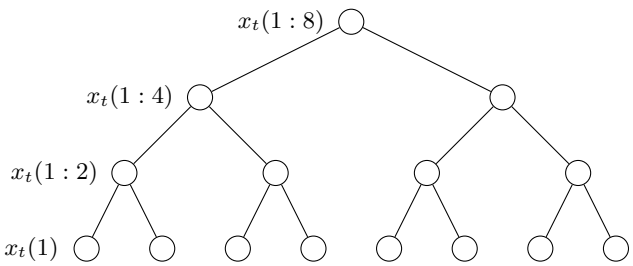

Figure 1. Space decomposition for $d = 8$.

3. Divide and Conquer within Marginal SMC for Filtering

To apply the DaC-SMC algorithm to the filtering problem, we need to identify a suitable collection $(\widetilde{\gamma}_{t,u})_{u\in\mathbb{T}}$ indexed by the nodes of the tree $\mathbb{T}$, for each time t. Graphically, this corresponds to a path graph (corresponding to time), in which each node has associated with it a copy of the tree $\mathbb{T}$ (corresponding to space). In this case, Algorithm 1 takes as input, at the leaves, a particle population approximating the filtering distribution at time $t-1$, and outputs, at the root, a particle population approximating the filtering distribution at time t.

At a given time t, to build the collection $(\widetilde{\gamma}_{t,u})_{u\in\mathbb{T}}$, we consider spatial decompositions of x_t into low-dimensional (often univariate) elements. Here, we consider a simple decomposition obtained by identifying the d components $(x_t(1), \ldots, x_t(d))$ using the leaves of a tree $\mathbb{T}$. As we move up the tree, the components are merged pairwise until $x_t = x_t(1:d)$ is recovered at the root node $\mathfrak{R}$. For simplicity, we assume that $d = 2^D$, for some $D \in \mathbb{N}$, such that $\mathbb{T}$ is a perfect binary tree; essentially the same construction applies to general d.

We denote the set of components associated with node u by $\mathcal{V}_u$, and its cardinality increases from the leaves to the root: at the level of the leaves, $|\mathcal{V}_u| = 1$, whereas $|\mathcal{V}_{\mathfrak{R}}| = d$. Figure 1 shows the space decomposition for $d = 8$.

Because the filtering problem has an inherent (temporal) sequential structure, the collection $(\widetilde{\gamma}_{t,u})_{u\in\mathbb{T}}$ at time t is most easily specified in terms of the filtering distribution at time $t-1$, $\widetilde{\gamma}_{t-1,\mathfrak{R}}$, as shown in (3.1). Similarly to MPFs, we deal with this dependence by approximately marginalizing out the previous time step using the existing sample approximation. We introduce auxiliary functions $f_{t,u} : \mathbb{R}^d \times \mathbb{R}^{|\mathcal{V}_u|} \to \mathbb{R}$ and $g_{t,u} : \mathbb{R}^{|\mathcal{V}_u|} \times \mathbb{R}^{|\mathcal{V}_u|} \to \mathbb{R}$ for $t \geq 1$ and $u \in \mathbb{T}$, such that $f_{t,\mathfrak{R}} = f_t, g_{t,\mathfrak{R}} = g_t$, and for $u \in \mathbb{T} \setminus \mathfrak{R}$, $f_{t,u}$ and $g_{t,u}$ serve as proxies for the marginals of the transition density and observation likelihood, respectively. These auxiliary functions are used to define our collection of target densities $(\widetilde{\gamma}_{t,u})_{t\geq 1,u\in\mathbb{T}}$ over $\mathbb{R}^{|\mathcal{V}_u|}$:

$$\widetilde{\gamma}_{t,u}(z_{t,u}) = g_{t,u}(z_{t,u}, \{y_t(i)\}_{i\in\mathcal{V}_u}) \int f_{t,u}(x_{t-1}, z_{t,u})\widetilde{\gamma}_{t-1,\mathfrak{R}}(x_{t-1})\mathrm{d}x_{t-1}, \tag{3.1}$$

where $z_{t,u} = (x_t(i))_{i\in\mathcal{V}_u}$ are the components of x_t associated with node u, and x_{t-1} denotes the previous state of the system. The requirement that $g_{t,\mathfrak{R}} = g_t$ and $f_{t,\mathfrak{R}} = f_t$ ensures that, at the root, we obtain the distribution in (2.2). The integral w.r.t. $\widetilde{\gamma}_{t-1,\mathfrak{R}}$ in (3.1) cannot be computed analytically. Hence, as in MPFs, we use a sample approximation of this integral, described in the next section.

If the marginals of f_t and g_t are available, one could use them to define $g_{t,u}$ and $f_{t,u}$. However, this is not essential, because these intermediate distributions can be essentially arbitrary up to the absolute continuity required to justify the importance sampling steps, although, of course, the variance of the estimator is influenced by this choice. Specifying these distributions is closely related to choosing the sequence of artificial targets in a standard SMC sampler when one is interested only in the final target distribution (see Del Moral, Doucet and Jasra (2006), who note that optimizing this sequence is a very difficult problem). See Kuntz, Crucinio and Johansen (2023, Sec. 4.2) for a theoretical perspective in a divide-and-conquer context. Their results suggest that the optimal choice of the intermediate target distribution is the appropriate marginal of the root target; hence we suggest using the marginals of f_t and g_t, where these are available. In practice, when doing this exactly is not feasible, we should approximate these distributions using tractable approximations with comparable tail behavior in order to keep all importance weights well controlled. If there is a substantial mismatch between the children of a node and itself, then choosing a parametric path between the two, adaptively specifying a tempering sequence along that path and using SMC techniques to approximate each distribution in turn, may mitigate some difficulties (See Appendix S2).

Although we believe that the best choice of $f_{t,u}$ and $g_{t,u}$ will depend on the model, in some cases, certain choices are preferable. As an example, take f_t to be a Gaussian distribution with mean μ and covariance Σ. Then, a possible choice for $\gamma_{t,u}$ is a Gaussian distribution with mean μ_u equal to the restriction

of μ to the components corresponding to node u, and covariance Σ_u obtained by subsetting Σ, and discarding all components not in u. In this context, we find that setting $\Sigma_u = \Sigma(\mathcal{V}_u)^{-1}$, that is, selecting the components of Σ corresponding to node u and then inverting this matrix, instead of $\Sigma_u = \Sigma^{-1}(\mathcal{V}_u)$ incurs a lower computational cost and leads to more diffuse distributions, and thus better behaved mixture weights (3.5).

3.1. The algorithm

For each time t, having identified the space decomposition over $\mathbb{T}$ and the collection of distributions $(\widetilde{\gamma}_{t,u})_{u\in\mathbb{T}}$, we can apply Algorithm 1 to the root $\mathfrak{R}$ of $\mathbb{T}$. However, because the integral in (3.1) is not analytically tractable, we replace $\widetilde{\gamma}_{t-1,\mathfrak{R}}$ with an approximation provided by the particle population at the root of the tree corresponding to $t-1$, $\{z^n_{t-1,\mathfrak{R}}\}^N_{n=1}$, that is, its particle approximation obtained at the previous time step, as is normally done in MPFs, and define

$$\gamma_{t,u}(z_{t,u}) := g_{t,u}(z_{t,u}, \{y_t(i)\}_{i\in\mathcal{V}_u}) \frac{1}{N} \sum_{n=1}^{N} f_{t,u}(z^n_{t-1,\mathfrak{R}}, z_{t,u}). \tag{3.2}$$

Given $\{z^n_{t-1,\mathfrak{R}}\}^N_{n=1}$, at each leaf node of the tree, we sample one component of x_t per node from $N^{-1}\sum_{n=1}^N K_{t,u}(z^n_{t-1,\mathfrak{R}}, \cdot)$. Then, the importance weights are given by

$$w_{t,u}(z_{t,u}, x_{1:t-1,u}) = \frac{g_{t,u}(z_{t,u}, \{y_t(i)\}_{i\in\mathcal{V}_u}) \sum_{n=1}^N f_{t,u}(z^n_{t-1,\mathfrak{R}}, z_{t,u})}{\sum_{n=1}^N K_{t,u}(z^n_{t-1,\mathfrak{R}}, z_{t,u})}. \tag{3.3}$$

As in the MPF, if we choose $K_{t,u} = f_{t,u}$, (3.3) simplifies dramatically to $w_{t,y}(z_{t,u}, x_{1:t-1,u}) = g_{t,u}(z_{t,u}, \{y_t(i)\}_{i\in\mathcal{V}_u})$, considerably reducing the cost of evaluating the weights at the leaves.

For any non-leaf node u, we gather the particle populations $\{z^n_{t,\ell(u)}\}^N_{n=1}$ and $\{z^n_{t,r(u)}\}^N_{n=1}$ on its left and right child, respectively, and obtain an approximation of the product measure $\gamma_{t,\mathcal{C}_u} := \gamma_{t,\ell(u)} \times \gamma_{t,r(u)}$ using the weighted product form estimator (2.3)

$$\gamma^N_{t,\mathcal{C}_u} = N^{-2} \sum_{n_1=1}^{N} \sum_{n_2=1}^{N} w_{t,\ell(u)}(z^{n_1}_{t,\ell(u)}) w_{t,r(u)}(z^{n_2}_{t,r(u)}) \delta_{(z^{n_1}_{t,\ell(u)}, z^{n_2}_{t,r(u)})}, \tag{3.4}$$

or one of the lower cost alternatives discussed in Section 3.3.

As in the DaC-SMC setting, we reweight the particle approximation of $\gamma_{t,\mathcal{C}_u}$ to target $\gamma_{t,u}$. In this case, the mixture weights are given by

$$m_{t,u}(z_{t,\mathcal{C}_u}) = \frac{\gamma_{t,u}(z_{t,\mathcal{C}_u})}{\gamma_{t,\mathcal{C}_u}(z_{t,\mathcal{C}_u})} \tag{3.5}$$

$$= \frac{g_{t,u}(z_{t,\mathcal{C}_u}, \{y_t(i)\}_{i\in\mathcal{V}_u})}{g_{t,\ell(u)}(z_{t,\ell(u)}, \{y_t(i)\}_{i\in\mathcal{V}_{\ell(u)}}) g_{t,r(u)}(z_{t,r(u)}, \{y_t(i)\}_{i\in\mathcal{V}_{r(u)}})} \times \frac{N^{-1}\sum_{n=1}^{N} f_{t,u}(z_{t-1,\mathfrak{R}}^n, z_{t,\mathcal{C}_u})}{N^{-1}\sum_{n=1}^{N} f_{t,\ell(u)}(z_{t-1,\mathfrak{R}}^n, z_{t,\ell(u)}) N^{-1}\sum_{n=1}^{N} f_{t,r(u)}(z_{t-1,\mathfrak{R}}^n, z_{t,r(u)})},$$

where we define $z_{t,\mathcal{C}_u} := (z_{t,\ell(u)}, z_{t,r(u)})$ as the vector obtained by merging the components of the left and right children of u.

For each pair in (3.4), we obtain the incremental mixture weights $m_{t,u}^{(n_1,n_2)} := m_{t,u}(z_{t,\ell(u)}^{n_1}, z_{t,r(u)}^{n_2})$ in (3.5), and the updated weights

$$\tilde{w}_{t,u}^{(n_1,n_2)} = \tilde{w}_{t,u}(z_{t,\ell(u)}^{n_1}, z_{t,r(u)}^{n_2}) := w_{t,\ell(u)}(z_{t,\ell(u)}^{n_1}) w_{t,r(u)}(z_{t,r(u)}^{n_2}) m_{t,u}(z_{t,\ell(u)}^{n_1}, z_{t,r(u)}^{n_2}),$$

for $n_1, n_2 = 1, \ldots, N$. To avoid unbounded growth in the number of particles, we then use the weights $\tilde{w}_{t,u}$ to resample a population of N particles approximating $\gamma_{t,u}$, $\{\tilde{z}_{t,u}^n, w_{t,u} = 1\}_{n=1}^N$. However, we could also allow the number of particles retained to grow as the simulation approaches the root of the tree to accommodate the growing dimension of the space. Algorithm 2 summarizes the procedure described above with the natural modification at $t = 1$, where it is not necessary to "marginalize" over previous states, and simple importance sampling can be used at the leaf nodes. In contrast to Algorithm 1, we do not include an additional MCMC step in this statement of the algorithm, but one can easily be added, as discussed in Section 3.2. In this case, given a $\pi_{t,u}$-invariant kernel $Q_{t,u}$, with $\pi_{t,u} \propto \gamma_{t,u}$, line 8 should be replaced by: draw $z_{t,u}^n \sim Q_{t,u}(\tilde{z}_{t,u}^n, \cdot)$ for all $n \leq N$.

Algorithm 2 dac_smc(t) for $t \geq 1$. Given $(\{z_{t-1,\mathfrak{R}}^n\}_{n=1}^N) :=$ dac_smc($t-1$).

1: **for** u leaf node **do**
2: *Initialize:* draw $z_{t,u}^n \sim N^{-1}\sum_{n=1}^N K_{t,u}(z_{t-1,\mathfrak{R}}^n, \cdot)$ and compute $w_{t,u}^n$ as in (3.3) for all $n \leq N$.
3: **end for**
4: **for** u non-leaf node **do**
5: *Recurse:* set $(\{z_{t,v}^n, w_{t,v}^n\}_{n=1}^N) :=$ dac_smc(t, v) for v in $\{\ell(u), r(u)\}$ and obtain $\gamma_{\mathcal{C}_u}^N$ in (3.4).
6: *Merge:* compute the mixture weights $m_{t,u}^{(n_1,n_2)}$ in (3.5) and $\tilde{w}_{t,u}^{(n_1,n_2)}$ for all $n_1, n_2 \leq N$.
7: *Resample:* draw $\{\tilde{z}_{t,u}^n\}_{n=1}^N$ using weights $\tilde{w}_{t,u}^{(n_1,n_2)}$ and set $w_u^n = 1$ for all $n \leq N$.
8: *Update:* set $z_{t,u}^n = \tilde{z}_{t,u}^n$ for all $n \leq N$.
9: **end for**
10: *Output* $(\{z_{t,\mathfrak{R}}^n\}_{n=1}^N)$.

The mixture resampling strategy described above requires evaluating the mixture weights (3.5) for each of the N^2 pairs in (3.4). The $O(N^2)$ cost of this operation is often prohibitive for large N; we describe alternatives with smaller costs in Section 3.3, and demonstrate that these approaches perform well in Section 4.

3.2. Choice of proposals

Algorithm 2 describes a general strategy for filtering using the DaC-SMC algorithm; as in the case of the standard SMC algorithm, the performance of the algorithm is heavily influenced by the choice of the proposals at the leaf nodes. Here, we discuss a simple strategy to select $K_{t,u}$.

We assume that we can sample from $f_{t,u}$ in (3.1), and set $K_{t,u}(z^n_{t-1,\mathfrak{R}}, \cdot) = f_{t,u}(z^n_{t-1,\mathfrak{R}}, \cdot)$ so that the importance weights (3.3) reduce to $w_{t,u} = g_{t,u}$. This choice corresponds to the proposal used in the bootstrap PF of Gordon, Salmond and Smith (1993). However, (locally) optimal proposals also exist (see, e.g., Chopin and Papaspiliopoulos (2020, Chap. 10)), and are expected to lead to better performance, but incur a higher computational cost.

Although picking $K_{t,u}(z^n_{t-1,\mathfrak{R}}, \cdot) = f_{t,u}(z^n_{t-1,\mathfrak{R}}, \cdot)$ causes standard marginal PFs to reduce to the bootstrap PF (as described in Klaas, De Freitas and Doucet (2005, Sec. 3.3)), this is not true for our marginal DaC-SMC, because the integral w.r.t. $z_{t-1,\mathfrak{R}}$ still appears in the mixture weights (3.5).

If needed, to avoid particle impoverishment, one might consider applying a Markov kernel $Q_{t,u}$ that leaves $\pi_{t,u} \propto \gamma_{t,u}$ invariant after the resampling step in line 7 of Algorithm 2. These $\pi_{t,u}$-invariant kernels can be selected based on the vast literature on sequential MCMC methods (e.g., Gilks and Berzuini (2001); Septier et al. (2009); Carmi, Septier and Godsill (2012); Septier and Peters (2016); Pal and Coates (2018); Han and Nakamura (2021)) to employ proposals with a cost less than $O(N)$, as it would be for some naïve choices.

3.3. Adaptive lightweight mixture resampling

The mixture resampling in lines 6–7 of Algorithm 2 becomes computationally impractical for large N, because it requires evaluating the mixture weights for N^2 particles (Lindsten et al. (2017); Kuntz, Crucinio and Johansen (2023); Corneflos, Chopin and Särkkä (2022)). Several strategies have been proposed to alleviate this cost by constructing only a subset of the N^2 combinations in (3.4), including the multiple matching strategy of Lin et al. (2005), which gives rise to the lightweight mixture resampling of Lindsten et al. (2017) in this context, strategies borrowed from the literature on incomplete U-statistics (Kuntz, Crucinio and Johansen (2023)), and lazy resampling schemes (Corneflos, Chopin and Särkkä (2022)).

We consider the lightweight version of the mixture resampling proposed in Lindsten et al. (2017), which considers only a subset θN, with $\theta \ll N$, of the N^2 possible pairs. However, instead of setting θ to some prespecified value (e.g., $\theta = \lceil \sqrt{N} \rceil$), we propose a simple strategy for selecting θ adaptively based on the effective sample size (ESS; Kong, Liu and Wong (1994)),

$$\mathrm{ESS} := \frac{\left(\sum_n \tilde{w}_{t,u}^n\right)^2}{\sum_n (\tilde{w}_{t,u}^n)^2}, \tag{3.6}$$

where the sum is over all pairs $n = (n_1, n_2)$ obtained from (3.4). This is similar in spirit to the adaptive tempering strategies commonly encountered in the SMC literature (see, e.g., Jasra et al. (2010); Johansen (2015)), and aims to do just enough computation to obtain a good N-sample approximation.

The merge step in lines 6–7 of Algorithm 2 is replaced by Algorithm 3: after building all the N pairs obtained by concatenating the particles associated with each of the two children, further permutations are added until the ESS achieves a prespecified value $\mathrm{ESS}^\star$ (e.g., $\mathrm{ESS}^\star = N$). To avoid θ getting too large, we stop adding permutations when $\theta = \lceil \sqrt{N} \rceil$, thereby allowing us to bound the worst-case computational cost by $O(N^{3/2})$. We empirically compare several mixture resampling approaches in Appendix S1.

Algorithm 3 Adaptive lightweight mixture resampling.

1: *Correct:* compute the mixture weights $m_{t,u}(x^n_{\ell(u)}, x^n_{r(u)})$ as in (3.5) and $\tilde{w}_{t,u}(x^n_{\ell(u)}, x^n_{r(u)})$ for all $n \le N$ and the ESS (3.6).
2: *Set:* $\theta \leftarrow 1$ and $\tilde{x}^n_u = (x^n_{\ell(u)}, x^n_{r(u)})$ for $n \le N$.
3: **while** $\mathrm{ESS}((\tilde{x}^n)_{n=1}^{\theta N}) < \mathrm{ESS}^\star$ and $\theta < \lceil \sqrt{N} \rceil$ **do**
4: *Set:* $\theta \leftarrow \theta + 1$.
5: *Permute:* draw one permutation of N, $\pi(N)$, set $\tilde{x}_u^{N(\theta-1)+n} = (x^n_{\ell(u)}, x^{\pi(n)}_{r(u)})$, compute the mixture weights $m_{t,u}(\tilde{x}_u^{N(\theta-1)+n})$ in (3.5) and the updated weights $\tilde{w}_{t,u}(\tilde{x}_u^{N(\theta-1)+n})$ for $n \le N$ and update the ESS.
6: **end while**
7: *Resample:* draw $\{z^n_{t,u}\}_{n=1}^N$ from $\tilde{x}_u^{1:N\theta}$ with weights $\tilde{w}_{t,u}(\tilde{x}_u^{1:N\theta})$ and set $w^n_{t,u} = 1$ for all $n \le N$.

Algorithm 3 can be implemented in a space-efficient manner by storing the permutations $\pi(N)$ corresponding to each value of θ, rather than building the θN pairs $\tilde{x}_u^{1:N\theta}$.

We report in Figure 2 the distribution of the number of permutations θ selected by Algorithm 3 for the linear Gaussian model in Section 4. We observe that the highest values of θ are chosen at the level above the leaves (panel 1). Indeed, the observation y_t is incorporated at the leaf level, causing a larger adjustment to the distribution at the first mixture resampling step than that needed as we move up the tree. The spike at $\theta = \lceil \sqrt{N} \rceil$ at level 1 shows that the target ESS is not always reached, which suggests that the product of the proposal distributions over the children might be a poor proposal for $\gamma_{t,u}$. This is likely, in part, to be because we use a "bootstrap proposal" and can be mitigated in the same way as in standard PFs by designing (marginal) proposal distributions that incorporate the influence of observations. It can also be mitigated by using (adaptive) tempering, as shown in, for example, Jasra et al. (2010), Johansen

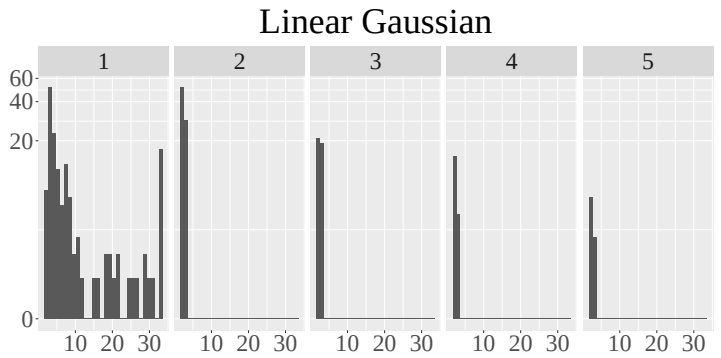

Figure 2. Distribution of the number of permutations θ selected by Algorithm 3 for the simple linear Gaussian model in Section 4.1 with $d = 32$, $N = 10^3$, and 10 time steps; the panels correspond to the levels of the tree from the level above the leaves (level 1) to the root (level 5).

(2015), Wang, Wang and Bouchard-Côté (2020), and Zhou, Johansen and Aston (2016) for the standard SMC algorithm and Lindsten et al. (2017, Sec. 4.2) for the DaC-SMC algorithm. However, a naïve implementation of this approach incurs a substantial computational cost in the marginal context; thus, designing MCMC kernels that are efficient in this context has been explored in the sequential MCMC context (e.g., Septier and Peters (2016, Sec. III-B) and in marginal STPFs (Xu and Jasra (2019))). We give details of a tempering strategy for Algorithm 3 in Appendix S2.

3.4. Computational cost

At a given node, u, the runtime cost of Algorithm 2 with lightweight mixture resampling, given the particle approximations for its children, is $O(h(N, d)\theta N)$, where $h(N, d)$ denotes the cost of obtaining the mixture weights (3.5).

In general, the dependence of $h(N, d)$ on the number of particles N is $O(N)$, as for MPFs. Similarly to MPFs, we can try to reduce the cost of computing (3.5) by adapting N-body learning techniques (e.g., Gray and Moore (2000); Lang, Klaas and de Freitas (2005)), as shown in Klaas, De Freitas and Doucet (2005). Alternatively, one could consider efficient implementations using GPUs (Charlier et al. (2021)), as shown in (Clarté, Diez and Feydy (2022, Sec. 4)), for sums of the form of those in (3.5).

For some models, it might be possible to pick the auxiliary functions $f_{t,u}$ so that the dependence on the past (i.e., $z_{t-1,\mathfrak{R}}$ in (3.2)) vanishes, and thus obtain an $O(1)$ cost w.r.t. N. However, we expect that this type of decomposition will require larger corrections at the root, where $f_{t,\mathfrak{R}} = f_t$, which might offset the cost savings.

In the adaptive case, worst-case costs are given by $O(h(N, d)\theta N)$, with θ replaced with the upper bound imposed on the number of permutations

considered. For instance, for the examples in Section 4, we incur an $h(N, d) = O(N)$ cost to obtain the mixture weights, and set the upper bound for θ to $N^{1/2}$, leading to a cost $O(N^{5/2})$ w.r.t. the number of particles.

Denoting $C_u(d, \theta_u, N)$ as the cost of running Algorithm 2 at node u, we can then bound the total cost of serial implementations of Algorithm 2 applied at the root node $\mathfrak{R}$ as $O(dt \sup_u C_u)$, where the supremum is taken over all nodes in $\mathbb{T}$; a lower running cost $O(t \sup_u C_u \log_2 d)$ can be achieved by parallelizing the computations over each level of the tree (Lindsten et al. (2017, Sec. 5.3)).

In the adaptive case, this upper bound is far from being tight, because, as shown in the histogram in Figure 2, θ tends to be high when the observation is incorporated (level 1), but stabilizes as we move up the tree. Additionally, for large N, one expects the number of permutations required to obtain a good N-particle approximation to converge to some fixed integer and, hence, the cost for sufficiently large N will, with high probability, be of smaller order than these bounds. Furthermore, the constants multiplying the $N^{5/2}$ contribution arising from the level above the leaves are sufficiently small that this is not the dominant cost in our experiments. This is likely to be typical in high-dimensional settings in which it is rarely feasible to employ very large numbers of particles, and the objective is to obtain a good approximation within acceptable time *and space* costs.

4. Experiments

We compare the results obtained using the DaC algorithm with those of the NSMC and STPF algorithms. We do not include the simpler strategies, because both the standard PF algorithm and the BPF algorithm have been shown to perform worse than the NSMC and STPF algorithms for the model considered here (Næsseth, Lindsten and Schön (2015, 2019); Beskos et al. (2017)), nor do we include the marginal version of the STPF algorithm because of the higher cost for large d.

The functions $g_{t,u}$ and $f_{t,u}$ in (3.1) are obtained from g_t and f_t, respectively, by discarding all the terms in those functions involving components $i \notin \mathcal{V}_u$; further details are given in Appendix S3. For Algorithm 2, we use the proposals discussed in Section 3.2 and the lightweight mixture resampling strategies described in Section 3.3. All resampling steps are performed using stratified resampling (Kitagawa (1996)).

First, we consider a simple linear Gaussian SSM, and compare the results obtained by the three algorithms, with the exact filtering distribution given by the Kalman filter. Then, we consider a spatial model with simple latent dynamics, but nontrivial spatial correlations between observations, moving away from the assumption of independent and identically distributed (i.i.d.) observations, which is convenient from a computational perspective, but rarely satisfied in practice

(Chib, Omori and Asai (2009)).

All experiments are executed in serial using a single core of an Intel(R) Xeon(R) CPU E5-2440 0 @ 2.40GHz using R 4.1.0.

4.1. Simple linear Gaussian model

We start by considering a simple linear Gaussian SSM, taken from Næsseth, Lindsten and Schön (2015), for which the filtering distributions can be computed exactly using the Kalman filter. The model is given by $f_t(x_{t-1}, x_t) = \mathcal{N}(x_t; Ax_{t-1}, \Sigma)$ and $g_t(x_t, y_t) = \mathcal{N}(y_t; x_t, \sigma_y^2 \mathrm{Id}_d)$, with $A \in \mathbb{R}^{d\times d}$, $\sigma_y^2 > 0$, $\Sigma \in \mathbb{R}^{d\times d}$ a tridiagonal covariance matrix, and Id_d the d-dimensional identity matrix (see Appendix S3.1 for full details and the computation of the mixture weights (3.5)).

We compare the DaC algorithm with both non-adaptive and adaptive lightweight mixture resampling using a two-level NSMC algorithm with a fully adapted outer level and an STPF algorithm on data simulated from the model for $d = 2^5, 2^8, 2^{11}$ for $t = 100$ time steps. We use different numbers of particles $N = 100, 500, 1000$ for Algorithm 2 and the outer levels of the NSMC and STPF algorithms. We fix the number of particles for the inner level of the NSMC algorithm and the number of particles for each island of the STPF algorithm to $M = 100$, as suggested in Næsseth, Lindsten and Schön (2015) and Beskos et al. (2017).

To evaluate the results, we consider two global measures of accuracy for each of the d marginals, namely, the Wasserstein-1 distance (see, e.g., Vallender (1974)), and the Kolmogorov–Smirnov distance

$$W_{1,i} := \int \left| F_{t,i}(x) - \widehat{F}_{t,i}(x) \right| \mathrm{d}x, \qquad \mathrm{KS}_i := \max_x \left| F_{t,i}(x) - \widehat{F}_{t,i}(x) \right|,$$

where $F_{t,i}$ denotes the one-dimensional cumulative distribution function of marginal i at time t, and $\widehat{F}_{t,i}$ is its particle approximation. Appendix S4.1 contains further comparisons that show that the mean squared error (MSE) of the filtering mean behaves similarly.

For lower-dimensional problems (e.g., $d = 32$), the STPF algorithm achieves the best results in terms of both the Wasserstein-1 distance and the KS distance (Figure 3), and the relative MSE of the reconstructions is considerably smaller (almost one order of magnitude smaller; see Appendix S4.1). The results provided by the STPF algorithm deteriorate quickly as d grows; for $d = 256$, the W_1 and KS distance estimates are significantly worse than those provided by the NSMC algorithm or the DaC algorithm without adaptation.

The cost of the STPF algorithm grows quadratically with d, and becomes unmanageable for large d; it is therefore not included in the bottom panels of Figure 3. The cost of the STPF algorithm is also higher for lower dimensions (Figure 3, top panel), but in this case, provides the best results. The STPF

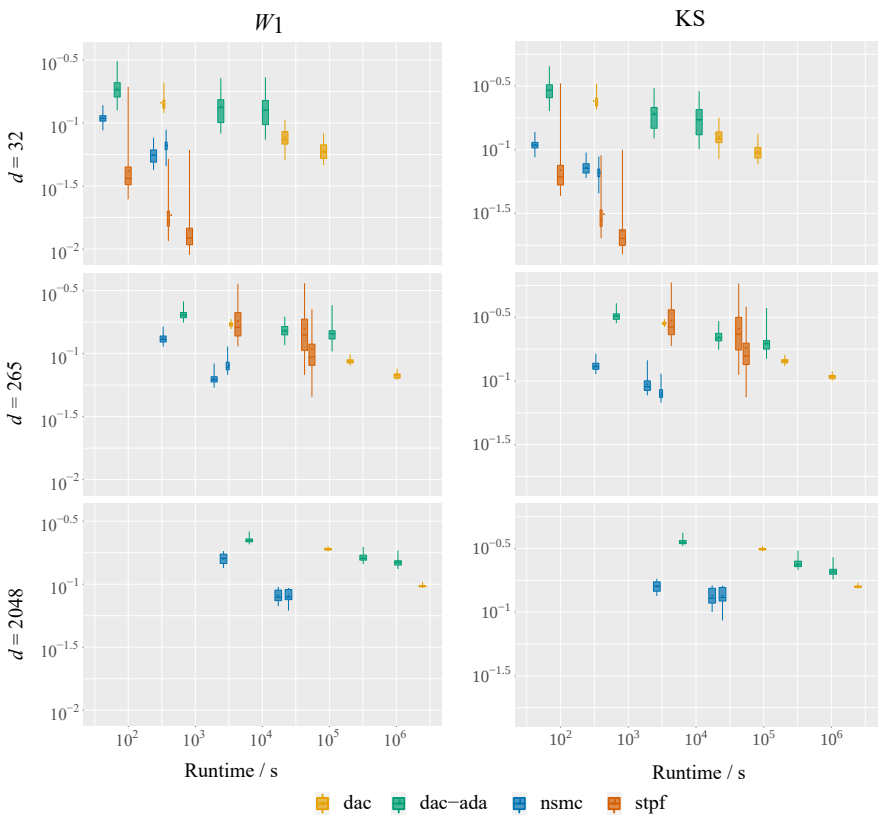

Figure 3. Comparison of DaC, NSMC, and STPF for $d = 2^5, 2^8, 2^{11}$. Distribution of the average (over dimension) W_1 and KS distance at the last time step $t = 100$ for 50 runs; the boxes, from left to right, correspond to increasing number of particles ($N = 100, 500, 1000$). Owing to their excessive cost, we do not include the results for STPF with $d = 2048$ and those of the non-adaptive version of DaC $d = 2048, N = 1000$.

algorithm has higher variability than the other methods, and even when the average results are better than those of the DaC and NSMC algorithms (e.g. $d = 32$), W_1 and KS can take considerably high values.

The results in terms of KS are, in general, more variable, probably because KS is a measure of the worst case mismatch between $F_{t,i}(x)$ and $\widehat{F}_{t,i}(x)$, whereas for W_1, the mismatch is averaged over locations. For large d, DaC has the smallest variability among the three algorithms.

DaC with fixed-cost lightweight mixture resampling gives better results, in general, than those of the adaptive lightweight mixture resampling. However, the cost of the latter is considerably smaller, making the adaptive version still manageable for large N, whereas the fixed-cost lightweight mixture resampling becomes too costly for large N and large d. As discussed in Section 3.4, the computational cost of both versions of DaC could be reduced using GPUs. In particular, both W_1 and KS decay more quickly with N for DaC without

adaptation than they do for DaC with adaptive lightweight mixture resampling. The decay with N is less evident for NSMC.

4.2. Spatial model

We consider a model on a 2D-lattice in which the latent dynamics are simple, but the observation structure is challenging. The components of X_t are indexed by the vertices $v \in V$ of a lattice, where $V = \{1, \ldots, d\}^2$, and follow a simple linear evolution $X_t(v) = X_{t-1}(v) + U_t(v)$, where $U_t(v) \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma_x^2)$. The observations model is $Y_t = X_t + V_t$, where we take V_t to be jointly t-distributed with $\nu = 10$ degrees of freedom, mean zero, and precision structure encapsulating a spatial component. Let D denote the graph distance. Then, the entry in row v and column j of the precision matrix Σ^{-1} is given by $(\Sigma^{-1})_{vj} = \tau^{D(j,v)}$ if $D(j, v) \leq r_y$, and zero otherwise. We obtain data from the model above with $\sigma_x^2 = 1$, $\tau = -0.25$, $r_y = 1$, and $t = 10$. The observation density does not factorize, and therefore NSMC and STPF cannot be applied (at least without approximating g with, e.g., a Gaussian or discarding the covariance information). To validate the correctness of the algorithm, we compare the DaC results with those of the standard bootstrap PF in Appendix S4 on a small lattice, and found the agreement to be excellent.

To decompose the 2D lattice into a binary tree, we use the decomposition described in Lindsten et al. (2017, Sec. 5.1), which recursively connects the vertices, first horizontally and then vertically. To evaluate the performance of the algorithm, we consider the filtering means obtained with 50 repetitions of the DaC-SMC algorithm on an 8×8 and a 16×16 grid for $N = 100, 500, 1000$, and 5000. To show how the standard bootstrap particle filter struggles with higher-dimensional problems, we run a bootstrap PF with $N = 10^5$ particles for the 8×8 grid. Figure 4 reports the filtering means for a corner node and an interior node of the lattice. Both DaC approaches are in agreement, however, the adaptive version seems to provide slightly less variable results. The behavior for the different nodes is similar. As observed for the linear Gaussian model, the DaC algorithm with adaptive lightweight mixture resampling has a lower cost than its non-adaptive counterpart, and remains feasible for large N (e.g., $N = 5000$).

Unsurprisingly, the bootstrap PF struggles to recover the filtering means and provides high variance estimates for node $(1, 1)$, and collapses completely for node $(8, 6)$, failing to recover the filtering mean.

The box plots in Figure 4 show the variance of the estimator provided by the DaC algorithm. For small N, the decay in variance seems to be more pronounced (at least for the adaptive version of the DaC algorithm) than it is for large N. This is consistent with the expected decay of the standard deviation from the variance expansions in Kuntz, Crucinio and Johansen (2023), where for small N, the higher-order contributions to the variance are not yet negligible. However,

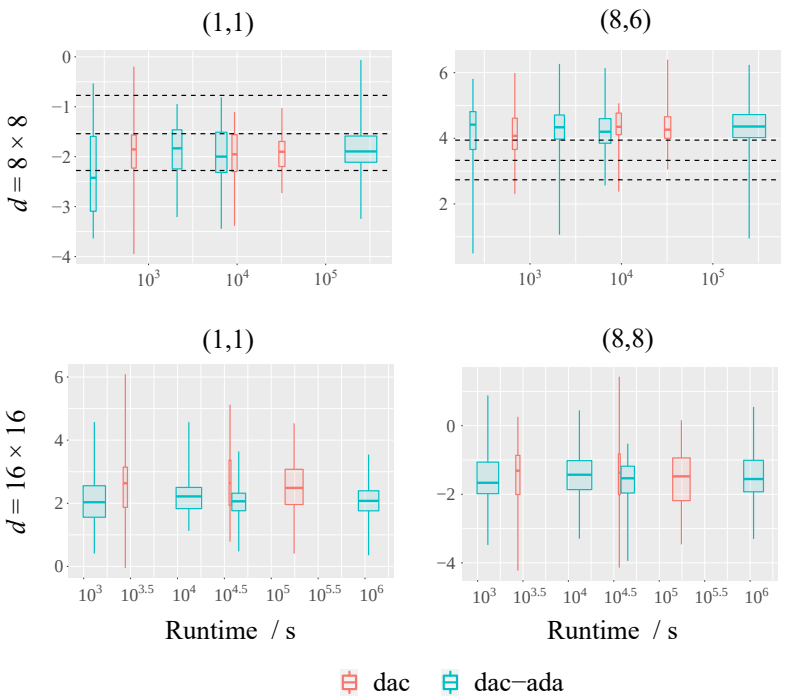

Figure 4. Filtering mean estimates for a corner node and a node in the middle of the grid for an 8×8 and a 16×16 lattice at time $t = 10$. The box plots from left to right report the distributions over 50 repetitions for $N = 100, 500, 1000$, and 5000. The results for the non-adaptive version of the DaC algorithm are not included for $N = 5000$, owing to the excessive cost. The reference lines for the 8×8 grid show the average value of the filtering mean estimate and the interquartile range obtained with 50 repetitions of a bootstrap PF with $N = 10^5$ particles.

we anticipate that a central limit theorem could be obtained by combining the results of Kuntz, Crucinio and Johansen (2023) with those for marginal PFs.

5. Discussion

We have introduced a novel sequential Monte Carlo algorithm, combining ideas from marginal PFs and the divide-and-conquer SMC algorithm to extend the latter to the filtering context. This algorithm is based on a novel space decomposition for high-dimensional SSMs that allows us to recursively merge low-dimensional marginals of the filtering distribution. Thus, we obtain the full filtering distribution, taking into account the mismatch between the product of marginals and the joint distributions using importance sampling. In principle, the DaC-SMC approach is amenable to distributed implementation, although its marginalization technique would necessitate significant communication from the node that computes the overall filtering distribution at time $t - 1$ to all nodes involved in computing at time t. This is left for future work.

In contrast to the nested SMC and the space-time PFs algorithms, the DaC-SMC approach to filtering can be applied when the marginals of the joint density (2.1) are not available analytically. The computational cost of this new approach grows polynomially with the number of particles N. However, this cost can be reduced by exploiting GPU routines to reduce the cost of computing the weights, as discussed in Section 3.4.

The results of the experiments in Section 4.1 show that DaC-SMC performs comparably with NSMC and STPF, with a runtime that remains competitive, even for large d (but small N), in contrast to STPFs. In addition, DaC-SMC can be applied to filtering problems that do not allow for factorization, as shown in Section 4.2 and Appendix S4. The variance decay shown in Figure 4 and Figure 2 in Appendix S4.1 suggests that this extended DaC-SMC achieves the same convergences rates as those of DaC-SMC (Kuntz, Crucinio and Johansen (2023)) for sufficiently large N. Thus, we anticipate that we can combine these techniques with those used to analyze the MPF in order to provide formal convergence results for the method developed here. The adaptive lightweight mixture resampling discussed in Section 3.3 is a promising route to further reduce the computational cost of DaC-SMC for filtering. However, as the experiments in Section 4 and Appendix S1 suggest, selecting the target ESS value that obtains the best trade-off between computational cost and accuracy is likely to be problem dependent. This raises interesting theoretical questions, which we leave for future work.

For challenging problems, it is likely that tempering and MCMC kernels are required for good performance. As discussed in Johansen (2015) and Guarniero, Johansen and Lee (2017), the smoothing and filtering distributions (i.e., $p(x_{1:t}|y_{1:t})$ and $p(x_t|y_{1:t})$, respectively) have significantly different support in the presence of informative observations, especially in high-dimensional settings. Thus, we expect that including the influence of future observations in the proposals and targets in Section 3.2 (as in the lookahead methods of Lin, Chen and Liu (2013), Guarniero, Johansen and Lee (2017), and Ruzayqat et al. (2022), among others) will lead to considerable improvements in the accuracy of the estimates and the development of good general-purpose filters for high-dimensional problems.

This work focuses on obtaining approximations of the filtering distribution for high-dimensional SSMs. Recently, studies have examined obtaining approximations of the smoothing distribution, which is a necessary component of parameter estimation algorithms (e.g., Finke and Singh (2017); Guarniero, Johansen and Lee (2017)). The DaC-SMC approach to filtering can be extended to tackle smoothing and parameter estimation dealing with the marginalization in (3.1). In principle, algorithms that directly approximate only marginals of smoothing distributions can be adapted to these settings (see, e.g., Gerber and Chopin (2017)). We leave this for future work.

Supplementary Material

The online Supplementary Material contains details of the tempering approach, and additional details and results on the experiments. An `R` package reproducing the experiments is available at `https://github.com/FrancescaCrucinio/Dac4filtering`.

Acknowledgments

FRC and AMJ acknowledge support from the EPSRC (grant # EP/R034710/1). AMJ acknowledges further support from the EPSRC (grant # EP/T004134/1) and the Lloyd's Register Foundation Programme on Data-Centric Engineering at the Alan Turing Institute. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising from this submission. No new data was created or analysed in this study. Data sharing is not applicable to this article.

References

Akyildiz, D., Crisan, D. and Miguez, J. (2022). Space-sequential particle filters for high-dimensional dynamical systems described by stochastic differential equations. *arXiv:2204.07680*.

Bengtsson, T., Bickel, P. and Li, B. (2008). Curse-of-dimensionality revisited: Collapse of the particle filter in very large scale systems. In *Probability and Statistics: Essays in Honor of David A. Freedman*, 316–334. Institute of Mathematical Statistics.

Beskos, A., Crisan, D., Jasra, A., Kamatani, K. and Zhou, Y. (2017). A stable particle filter for a class of high-dimensional state-space models. *Advances in Applied Probability* **49**, 24–48.

Carmi, A., Septier, F. and Godsill, S. J. (2012). The Gaussian mixture MCMC particle algorithm for dynamic cluster tracking. *Automatica* **48**, 2454–2467.

Charlier, B., Feydy, J., Glaunes, J. A., Collin, F.-D. and Durif, G. (2021). Kernel operations on the GPU, with autodiff, without memory overflows. *Journal of Machine Learning Research* **22**, 1–6.

Chib, S., Omori, Y. and Asai, M. (2009). Multivariate stochastic volatility. In *Handbook of Financial Time Series*, 365–400. Springer.

Chopin, N. and Papaspiliopoulos, O. (2020). *An Introduction to Sequential Monte Carlo*. Springer.

Clarté, G., Diez, A. and Feydy, J. (2022). Collective proposal distributions for nonlinear MCMC samplers: Mean-field theory and fast implementation. *Electronic Journal of Statistics* **22**, 6395–6460.

Corneflos, A., Chopin, N. and Särkkä, S. (2022). De-Sequentialized Monte Carlo: A parallel-in-time particle smoother. *Journal of Machine Learning Research* **23**, 1–39.

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

Doucet, A. and Johansen, A. M. (2011). A tutorial on particle filtering and smoothing: Fifteen years later. In *The Oxford Handbook of Nonlinear Filtering* (Edited by D. Crisan and B. Rozovsky), 656–704. Oxford University Press.

Evensen, G. (2009). *Data Assimilation*. 2nd Edition. Springer.

Finke, A. and Singh, S. S. (2017). Approximate smoothing and parameter estimation in high-dimensional state-space models. *IEEE Transactions on Signal Processing* **65**, 5982–5994.

Gerber, M. and Chopin, N. (2017). Convergence of sequential quasi-Monte Carlo smoothing algorithms. *Bernoulli* **23**, 2951–2987.

Gerber, M., Chopin, N. and Whiteley, N. (2019). Negative association, ordering and convergence of resampling methods. *The Annals of Statistics* **47**, 2236–2260.

Gilks, W. R. and Berzuini, C. (2001). Following a moving target–Monte Carlo inference for dynamic Bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **63**, 127–146.

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

Gray, A. and Moore, A. (2000). 'N-body' problems in statistical learning. In *Advances in Neural Information Processing Systems*, 521–527. NeurIPS.

Guarniero, P., Johansen, A. M. and Lee, A. (2017). The iterated auxiliary particle filter. *Journal of the American Statistical Association* **112**, 1636–1647.

Han, Y. and Nakamura, K. (2021). The application of Zig-Zag sampler in sequential Markov chain Monte Carlo. *arXiv:2111.10210*.

Jasra, A., Stephens, D. A., Doucet, A. and Tsagaris, T. (2010). Inference for Lévy-driven stochastic volatility models via adaptive sequential Monte Carlo. *Scandinavian Journal of Statistics* **38**, 1–22.

Johansen, A. M. (2015). On blocks, tempering and particle MCMC for systems identification. *IFAC-PapersOnLine* **48**, 969–974.

Kitagawa, G. (1996). Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics* **5**, 1–25.

Klaas, M., Briers, M., De Freitas, N., Doucet, A., Maskell, S. and Lang, D. (2006). Fast particle smoothing: If I had a million particles. In *Proceedings of the 23rd International Conference on Machine Learning*, 481–488. ICML.

Klaas, M., De Freitas, N. and Doucet, A. (2005). Toward practical N^2 Monte Carlo: The marginal particle filter. In *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*, 308–315. UAI.

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

Kuntz, J., Crucinio, F. R. and Johansen, A. M. (2022). Product-form estimators: Exploiting independence to scale up Monte Carlo. *Statistics and Computing* **32**, 1–22.

Kuntz, J., Crucinio, F. R. and Johansen, A. M. (2023). Divide-and-conquer Sequential Monte Carlo: Properties and limit theorems. *The Annals of Applied Probability*. In press.

Lang, D., Klaas, M. and de Freitas, N. (2005). *Empirical Testing of Fast Kernel Density Estimation Algorithms*. Technical report TR2005-03. University of British Columbia.

Lei, J., Bickel, P. and Snyder, C. (2010). Comparison of ensemble Kalman filters under non-Gaussianity. *Monthly Weather Review* **138**, 1293–1306.

Lin, M., Chen, R. and Liu, J. S. (2013). Lookahead strategies for sequential Monte Carlo. *Statistical Science* **28**, 69–94.

Lin, M. T., Zhang, J. L., Cheng, Q. and Chen, R. (2005). Independent particle filters. *Journal of the American Statistical Association* **100**, 1412–1421.

Lindsten, F., Johansen, A. M., Næsseth, C. A., Kirkpatrick, B., Schön, T. B., Aston, J. A. D. et al. (2017). Divide-and-Conquer with sequential Monte Carlo. *Journal of Computational and Graphical Statistics* **26**, 445–458.

Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing*. Springer.

Næsseth, C. A., Lindsten, F. and Schön, T. B. (2015). Nested sequential Monte Carlo methods. In *Proceedings of the 32nd International Conference on Machine Learning*, 1292–1301.

Næsseth, C. A., Lindsten, F. and Schön, T. B. (2019). High-dimensional filtering using nested sequential Monte Carlo. *IEEE Transactions on Signal Processing* **67**, 4177–4188.

Pal, S. and Coates, M. (2018). Sequential MCMC with the discrete bouncy particle sampler. In *IEEE Statistical Signal Processing Workshop (SSP)*, 663–667. IEEE.

Pitt, M. K. and Shephard, N. (1999). Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association* **94**, 590–599.

Rebeschini, P. and Van Handel, R. (2015). Can local particle filters beat the curse of dimensionality? *Annals of Applied Probability* **25**, 2809–2866.

Ruzayqat, H., Er-Raiy, A., Beskos, A., Crisan, D., Jasra, A. and Kantas, N. (2022). A lagged particle filter for stable filtering of certain high-dimensional state-space models. *SIAM/ASA Journal on Uncertainty Quantification* **10**, 1130–1161.

Septier, F., Pang, S. K., Carmi, A. and Godsill, S. (2009). On MCMC-based particle methods for Bayesian filtering: Application to multitarget tracking. In *3rd IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, 360–363. IEEE.

Septier, F. and Peters, G. W. (2016). Langevin and Hamiltonian based sequential MCMC for efficient Bayesian filtering in high-dimensional spaces. *IEEE Journal of Selected Topics in Signal Processing* **10**, 312–327.

Vallender, S. S. (1974). Calculation of the Wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications* **18**, 784–786.

Wang, L., Wang, S. and Bouchard-Côté, A. (2020). An annealed sequential Monte Carlo method for Bayesian phylogenetics. *Systematic Biology* **69**, 155–183.

Xu, Y. and Jasra, A. (2019). Particle filters for inference of high-dimensional multivariate stochastic volatility models with cross-leverage effects. *Foundations of Data Science* **1**, 61–85.

Zhou, Y., Johansen, A. M. and Aston, J. A. (2016). Toward automatic model comparison: An adaptive sequential Monte Carlo approach. *Journal of Computational and Graphical Statistics* **25**, 701–726.

Francesca R. Crucinio

Department of Statistics, University of Warwick, Coventry CV4 7AL, UK.

E-mail: francesca.crucinio@gmail.com

Adam M. Johansen

Department of Statistics, University of Warwick, Coventry CV4 7AL, UK.

E-mail: a.m.johansen@warwick.ac.uk

(Received July 2022; accepted February 2023)