Supplementary material for 'Divide-and-Conquer with Sequential Monte Carlo'

F. Lindsten[1], A. M. Johansen[2], C. A. Naesseth[3], B. Kirkpatrick[4], T. B. Schön[1], J. A. D. Aston[5], and A. Bouchard-Côté[6]

June 13, 2016

Abstract

This document contains supplementary material for the paper 'Divide-and-Conquer with Sequential Monte Carlo'. In Section A we prove Propositions 1 and 2 and provide additional details on how these propositions can be extended to the more advanced D&C-SMC implementations discussed in Section 4 of the main paper. Section B provides detailed algorithmic descriptions of the (lightweight) mixture sampling method (Section 4.1 of the main paper) and tempering method (Section 4.2 of the main paper) in the context of the proposed D&C-SMC sampler. Sections C and D contain additional details and results for the two numerical examples presented in Sections 5.1 and 5.2 of the main paper, respectively. Finally, Section E provides an overview of the implementation used in Sections 5.2 and 5.3 of the main paper.

A Theoretical Properties

A.1 Proof of Proposition 1

We consider all of the random variables simulated in the running of the algorithm, following the approach of Andrieu et al. (2010) for standard SIR algorithms. A direct argument provides for the unbiasedness of the normalizing constant estimate by demonstrating that the ratio $\widehat{Z}_r^N/Z_r$ is equal to the Radon-Nikodým derivative of a particular extended target distribution to the joint sampling distribution of all of the random variables generated during the running of the algorithm (and is hence of unit expectation).

*1 Uppsala University, 2 University of Warwick, 3 Linköping University, 4 University of Miami, 5 University of Cambridge, 6 University of British Columbia. Address for Correspondence: Fredrik Lindsten, Division of Systems and Control, Department of Information Technology, Uppsala University, Box 337, 751 05 Uppsala, Sweden. Email: fredrik.lindsten@it.uu.se.

This qualitative argument continues to hold when annealing or mixture sampling is employed within the algorithm; the expressions involved in verifying this are cumbersome and so we provide explicit details only for the case in which T is a balanced binary tree (i.e. $|\mathcal{C}(t)| = 2$ for every non-leaf node). The more general case follows by the same argument *mutatis mutandis*. We note in particular that:

1. the extension to balanced trees of degree greater than two is trivial and that unbalanced trees may be addressed by the introduction of trivial dummy nodes (or directly, at the expense of further complicating the notation);

2. the use of annealing can be addressed by introducing additional nodes with single children;

3. the use of mixture sampling requires only the substitution of the appropriate weight expressions within the argument which follows;

4. the use of lightweight mixture sampling results in a two-stage procedure (first sampling independently mN offsprings for each child population, then resampling N particles from the mN matchings) but both of these stages can be handled analogously to below.

We assume that subpopulation h at depth d is obtained from subpopulations $2h-1$ and $2h$ at depth $d+1$. Let,

$$\mathbf{x}_{\langle D\rangle_2}^{1:N} := \{\mathbf{x}_{(d,h)}^i : (d,h) \in \langle D\rangle_2, i \in \{1,\ldots,N\}\}$$

be the collection containing N particles within each sub-population and let

$$\mathbf{a}_{\langle D\rangle_2'}^{1:N} := \{\mathbf{a}_{(d,h)}^i : (d,h) \in \langle D\rangle_2', i \in \{1,\ldots,N\}\}$$

be the ancestor indices associated with the resampling step; $\mathbf{a}_{(d,h)}^i$ is the ancestor of the i^{th} particle obtained in the resampling step associated with subpopulation h at level d of the tree.

First we specify sets in which multi-indices of particle populations live:

$$\langle D\rangle_2 = \bigcup_{d=0}^{D} \{d\} \otimes \{1,\ldots,2^d\}, \qquad \langle D\rangle_2' = \langle D\rangle_2 \setminus \{0\} \otimes \{1\}.$$

Thus, population h at depth d, where the root of the tree is at a depth of 0, may be identified as $(d,h) \in \langle D\rangle_2$ for any $d \in \{0,\ldots,D\}$.

The joint distribution from which variables are simulated during the running of the

algorithm may be written as:

$$\widetilde{q}\left(\mathrm{d}\mathbf{x}^{1:N}_{\langle D\rangle_2}, \mathrm{d}\mathbf{a}^{1:N}_{\langle D\rangle'_2}\right) = \prod_{h=1}^{2^D}\prod_{i=1}^{N} q_{(D,h)}(\mathrm{d}\mathbf{x}^i_{(D,h)}) \times \prod_{(d,h)\in\langle D\rangle'_2}\prod_{i=1}^{N} \boldsymbol{W}^{\mathbf{a}^i_{(d,h)}}_{(d,h)}\mathrm{d}\mathbf{a}^i_{(d,h)}$$
$$\times \prod_{(d,h)\in\langle D\rangle_2}\prod_{i=1}^{N}\delta_{\left(\mathbf{x}^{\mathbf{a}^i_{(d+1,2h-1)}}_{(d+1,2h-1)},\mathbf{x}^{\mathbf{a}^i_{(d+1,2h)}}_{(d+1,2h)}\right)}\left(\mathrm{d}(\mathbf{x}^i_{(d,h)} \setminus \widetilde{\mathbf{x}}^i_{(d,h)})\right) q_{(d,h)}(\mathrm{d}\widetilde{\mathbf{x}}^i_{(d,h)}|\mathbf{x}^i_{(d,h)} \setminus \widetilde{\mathbf{x}}^i_{(d,h)})$$

where $\boldsymbol{W}^i_{(d,h)}$ denotes the normalized (to sum to one within the realized sample) importance weight of particle i in subpopulation h of depth d. Note that this distribution is over $\left(\otimes_{(d,h)\in\langle D\rangle_2}\mathsf{X}_{(d,h)}\right)^N \otimes \{1,\ldots,N\}^{|\langle D\rangle'_2|N}$ and the $\mathrm{d}\mathbf{a}$ corresponds to a counting measure over the index set. The inclusion of the singular transition emphasizes the connection between this algorithm and the standard SIR algorithm.

It is convenient to define ancestral *trees* for our particles using the following recursive definition:

$$b^i_{(0,1)} = i \qquad \text{and} \qquad b^i_{(d,h)} = \begin{cases} b^i_{(d-1,(h+1)/2)} & d \text{ odd}, \\ \mathbf{a}^{b^i_{(d-1,h/2)}}_{(d,h)} & d \text{ even}, \end{cases}$$

the intention being that $\{b^i_{(d,h)} : (d,h) \in \langle D\rangle'_2\}$ contains the multi-indices of all particles which are ancestral to the i^{th} particle at the root.

It is also useful to define an auxiliary distribution over all of the sampled variables and an additional variable k which indicates a selected ancestral tree from the collection of N, just as in the particle MCMC context. Here we recall that $\pi_r = \gamma_r/Z_r$:

$$\widetilde{\pi}_r\left(\mathrm{d}\mathbf{x}^{1:N}_{\langle D\rangle_2}, \mathrm{d}\mathbf{a}^{1:N}_{\langle D\rangle'_2}, \mathrm{d}k\right) = \frac{\pi_r(\mathrm{d}\mathbf{x}^k_{(0,1)}) \prod_{(d,h)\in\langle D\rangle_2}\delta_{\mathbf{x}^{b^k_{(d,h)}}_{(d,h)}\setminus\widetilde{\mathbf{x}}^{b^k_{(d,h)}}_{(d,h)}}\left(\mathrm{d}\left(\mathbf{x}^{b^k_{(d+1,2h-1)}}_{(d+1,2h-1)}, \mathbf{x}^{b^k_{(d+1,2h)}}_{(d+1,2h-1)}\right)\right)}{N^{|\langle D\rangle_2|}}$$
$$\times \frac{\widetilde{q}\left(\mathrm{d}\mathbf{x}^{1:N}_{\langle D\rangle_2}, \mathrm{d}\mathbf{a}^{1:N}_{\langle D\rangle'_2}\right)}{\prod_{h=1}^{2^D} q_{(D,h)}(\mathrm{d}\mathbf{x}^{b^k_{(D,h)}}_{(D,h)}) \left(\prod_{(d,h)\in\langle D\rangle'_2}\boldsymbol{W}^{b^k_{(d,h)}}_{(d,h)}\right) \prod_{(d,h)\in\langle D\rangle_2} q_{(d,h)}(\mathrm{d}\widetilde{\mathbf{x}}^{b^k_{d,h}}_{(d,h)}|\mathbf{x}^{b^k_{d,h}}_{(d,h)} \setminus \widetilde{\mathbf{x}}^{b^k_{d,h}}_{(d,h)})}$$
$$\times \left(\prod_{(d,h)\in\langle D\rangle_2}\delta_{\left(\mathbf{x}^{b^k_{(d+1,2h-1)}}_{(d+1,2h-1)},\mathbf{x}^{b^k_{(d+1,2h)}}_{(d+1,2h-1)}\right)}\left(\mathrm{d}\left(\mathbf{x}^{b^k_{(d,h)}}_{(d,h)} \setminus \widetilde{\mathbf{x}}^{b^k_{(d,h)}}_{(d,h)}\right)\right)\right)^{-1} \mathrm{d}k$$

which can be straightforwardly established to be a properly-normalized probability. Note that the division by a product of singular measures should be interpreted simply as removing

the corresponding singular component of the numerator.

Augmenting the proposal distribution with $k \in \{1, \ldots, N\}$ obtained in the same way is convenient (and does not change the result which follows as $\widehat{Z}_r^N$ does not depend upon k):

$$\bar{q}\left(\mathrm{d}\mathbf{x}_{\langle D\rangle_2}^{1:N}, \mathrm{d}\mathbf{a}_{\langle D\rangle_2'}^{1:N}, \mathrm{d}k\right) = \widetilde{q}\left(\mathrm{d}\mathbf{x}_{\langle D\rangle_2}^{1:N}, \mathrm{d}\mathbf{a}_{\langle D\rangle_2'}^{1:N}\right) \boldsymbol{W}_{(0,1)}^k \mathrm{d}k.$$

It is straightforward to establish that $\widetilde{\pi}_r$ is absolutely continuous with respect to $\bar{q}$. Taking the Radon-Nikodým derivative yields the following useful result (the identification between $\mathbf{x}_{(0,1)}^k$ and its ancestors is taken to be implicit for notational simplicity; as this equality holds with probability one under the proposal distribution, this is sufficient to define the quantity of interest almost everywhere). We allow π_r and $q_{d,h}$ to denote the *densities* of the measures which they correspond to (with respect to an appropriate version of Lebesgue or counting measure)

$$\begin{aligned}
&\frac{d\widetilde{\pi}_r}{d\bar{q}}\left(\mathbf{x}_{\langle D\rangle_2}^{1:N}, \mathbf{a}_{\langle D\rangle_2'}^{1:N}, k\right) \\
&\quad= \frac{\pi_r\left(\mathbf{x}_{(0,1)}^k\right)}{N^{|\langle D\rangle_2|}} \frac{1}{\prod_{h=1}^{2^D} q_{(D,h)}(\mathbf{x}_{(D,h)}^{b_{(D,h)}^k}) \prod_{(d,h)\in\langle D\rangle_2} q_{(d,h)}(\widetilde{\mathbf{x}}_{(d,h)}^{b_{d,h}^k} | \mathbf{x}_{(d,h)}^{b_{d,h}^k} \setminus \widetilde{\mathbf{x}}_{(d,h)}^{b_{d,h}^k}) \prod_{(d,h)\in\langle D\rangle_2} \boldsymbol{W}_{(d,h)}^{b_{(d,h)}^k}} \\
&\quad= \frac{\pi_r\left(\mathbf{x}_{(0,1)}^k\right)}{\prod_{h=1}^{2^D} q_{(D,h)}(\mathbf{x}_{(D,h)}^{b_{(D,h)}^k}) \prod_{(d,h)\in\langle D\rangle_2} q_{(d,h)}(\widetilde{\mathbf{x}}_{(d,h)}^{b_{d,h}^k} | \mathbf{x}_{(d,h)}^{b_{d,h}^k} \setminus \widetilde{\mathbf{x}}_{(d,h)}^{b_{d,h}^k}) \prod_{(d,h)\in\langle D\rangle_2} \frac{\mathbf{w}_{(d,h)}^{b_{(d,h)}^k}}{\frac{1}{N}\sum_{i=1}^N \mathbf{w}_{(d,h)}^i}}
\end{aligned}$$

where $\mathbf{w}_{(d,h)}^i$ denotes the *unnormalized* importance weight of particle i in subpopulation h at depth d.

We can then identify the product $\prod_{(d,h)\in\langle D\rangle_2} \mathbf{w}_{(d,h)}^{b_{(d,h)}^k}$ with the ratio of density π_r to the assorted proposal densities evaluated at the appropriate ancestors of the surviving particle multiplied by the normalizing constant Z_r (by construction; these are exactly the unnormalized weights). Furthermore, we have that

$$\prod_{(d,h)\in\langle D\rangle_2} \frac{1}{N}\sum_{i=1}^N \mathbf{w}_{(d,h)}^i = \widehat{Z}_r^N$$

which implies that $\widetilde{\pi}_r = (\widehat{Z}_r^N / Z_r)\bar{q}$. Consequently, we obtain the result as:

$$\mathbb{E}_{\bar{q}}[\widehat{Z}_r^N] = \mathbb{E}_{\bar{q}}[Z_r \widetilde{\pi}_r / \bar{q}] = Z_r \mathbb{E}_{\widetilde{\pi}_r}[1] = Z_r.$$

□

A.2 Consistency – Proof of Proposition 2

We now turn to consistency. We make a few amendments to the notation used in the main text. First, as we consider limits in the number of particles N, we add an index N the particles and their weights. Second, as in the preceding proof we use $\boldsymbol{W}$ to denote the normalized weights. For simplicity we also assume a perfect binary tree, $\mathcal{C}(t) = (t_1, t_2)$, where t_1 and t_2 denote the left and right children of node t. The proof in this case captures the essential features of m the general case. We base our argument on Douc and Moulines (2008) (henceforth, DM08). We will use the following definitions and assumptions.

1. We require that the Radon-Nikodým derivative

$$\frac{\mathrm{d}\gamma_t}{\mathrm{d}\ \gamma_{t_1} \otimes \gamma_{t_2} \otimes q_t}(\mathbf{x}_{t_1}, \mathbf{x}_{t_2}, \widetilde{\mathbf{x}}_t) < \infty, \tag{1}$$

   is well-defined and finite almost everywhere.

2. We define C_t to denote the collection of bounded integrable test functions as in DM08.

3. We also make the following assumption: there exists a constant C such that for all $t, \mathbf{x}_{t_1}, \mathbf{x}_{t_2}, \widetilde{\mathbf{x}}_t$:

$$\frac{\mathrm{d}\gamma_t}{\mathrm{d}\ \gamma_{t_1} \otimes \gamma_{t_2} \otimes q_t}(\mathbf{x}_{t_1}, \mathbf{x}_{t_2}, \widetilde{\mathbf{x}}_t) < C,$$

   This assumption is too strong to cover some of the algorithms and applications discussed in the main text. However the assumption simplifies exposition and could be relaxed.

4. We assume that multinomial resampling—or its mixture resampling variant—is performed at every step.

Proposition A1. *Under the above assumptions, the normalized weighted particles* $(\mathbf{x}_N^i, \boldsymbol{W}_N^i : i \in \{1, \ldots, N\})$, $\boldsymbol{W}_N^i = \mathbf{w}_N^i / \sum_j \mathbf{w}_N^j$, *obtained from* dc_smc*(r) are consistent with respect to* (π, C_r), *i.e.: for all test function* $f \in \mathsf{C}_r$, *as the number of particles* $N \to \infty$*:*

$$\sum_{i=1}^{N} \boldsymbol{W}_N^i f(\mathbf{x}_N^i) \xrightarrow{\mathbb{P}} \int f(\mathbf{x}) \pi(\mathrm{d}\mathbf{x}).$$

Lemma A1. *Let $\mathcal{F}$ denote a σ-algebra generated by a semi-ring $\mathcal{A}$, $\mathcal{F} = \sigma(\mathcal{A})$. Let π be a finite measure constructed using a Carathéodory extension based on $\mathcal{A}$. Then, given $\epsilon > 0$ and $E \in \mathcal{F}$, there is a finite collection of disjoint simple sets $R_1, \ldots, R_n$ that cover E and provide an ϵ-approximation of its measure:*

$$\mu(A) \leq \mu(E) + \epsilon,$$
$$A = \bigcup_{j=1}^{n} R_j \supset E.$$

Proof. From the definition of the Carathéodory outer measure, and the fact that it coincides with π on measurable sets such as E, we have a countable cover $R_1, R_2, \ldots$ with:

$$\mu(\widetilde{A}) \leq \mu(E) + \frac{\epsilon}{2},$$
$$\widetilde{A} = \bigcup_{j=1}^{\infty} R_j \supset E.$$

Moreover, since $\mu(E) < \infty$, the sum can be truncated to provide a finite ϵ-approximation. Finally, since $\mathcal{A}$ is a semi-ring, we can rewrite the finite cover as a disjoint finite cover. □

Lemma A2. *Let π_1, π_2 be finite measures, and f a measurable function on the product σ-algebra $\mathcal{F}_1 \otimes \mathcal{F}_2$. Then for all $\epsilon > 0$, there is a measurable set B such that $\pi_1 \otimes \pi_2(B) < \epsilon$, and outside of which f is uniformly approximated by simple functions on rectangles:*

$$\lim_{M \to \infty} \sup_{x \notin B} \left| f(x) - \sum_{m=1}^{M} c_m^M \mathbf{1}_{R_m^M}(x) \right| = 0,$$

for some $R_m^M = F_m^M \times G_m^M$, $F_m^M \in \mathcal{F}_1, G_m^M \in \mathcal{F}_2$.

Proof. Assuming $f \geq 0$ (if not, apply this argument to the positive and negative parts of f), we obtain that there exists simple functions f_M that converge almost everywhere to f:

$$\lim_{M \to \infty} f_M = f \quad \text{(a.s.)}$$
$$f_M = \sum_{m=1}^{M} c_m^M \mathbf{1}_{E_m^M},$$
$$E_m^M \in \mathcal{F}_1 \otimes \mathcal{F}_2$$

Next, we apply Lemma A1 to each E_m^M, with $\mu = \pi_1 \otimes \pi_2$. We set an exponentially decreasing

tolerance $\epsilon_m^M = \epsilon M^{-1} 2^{-M-1}$ so that:

$$\mu(A_m^M) \leq \mu(E_m^M) + \epsilon M^{-1} 2^{-M-1},$$
$$A_m^M = \bigcup_{j=1}^{n_m^M} R_{m,j}^M \supset E_m^M.$$

Note that outside of some bad set B_1, we now have pointwise convergence of simple functions on rectangles (since each A_m^M is a disjoint union of rectangles):

$$\lim_{M\to\infty} \sum_{m=1}^{M} c_m^M \mathbf{1}_{A_m^M}\Big|_{B_1{}^c} = f\Big|_{B_1{}^c} \quad \text{(a.s.)} \tag{2}$$
$$B_1 = \bigcup_{M=1}^{\infty} \bigcup_{m=1}^{M} (A_m^M \backslash E_m^M).$$

Note that:

$$\pi_1 \otimes \pi_2(B_1) \leq \sum_{M=1}^{\infty} \sum_{m=1}^{M} \epsilon M^{-1} 2^{-M-1}$$
$$= \epsilon \sum_{M=1}^{\infty} 2^{-M-1} = \frac{\epsilon}{2}.$$

Finally, pointwise convergence almost everywhere in Equation (2) implies by Egorov's theorem the existence of a set B_2 with $\pi_1 \otimes \pi_2(B_2) < \epsilon/2$ and such that convergence is uniform outside of $B = B_1 \cup B_2$. □

Lemma A3. *If $x < y + \beta_2$, then $\mathbb{P}(|X - x| > \beta_1) < \alpha$ implies $\mathbb{P}(X > \beta_1 + \beta_2 + y) < \alpha$.*

Proof. We prove that $\mathbb{P}(X > \beta_1 + \beta_2 + y) < \mathbb{P}(|X - x| > \beta_1)$ via the contrapositive on the events: $|X - x| \leq \beta_1 \Longrightarrow X - x \leq \beta_1 \Longrightarrow X \leq \beta_1 + \beta_2 + y$. □

Proof of Proposition A1. Let us label the nodes of the tree using a *height function* h defined to be equal to zero at the leaves, and to $h(t) = 1 + \max\{h(t_{t_1}), h(t_{t_2})\}$ for the internal nodes.

We proceed by induction on $h = 0, 1, 2, \ldots$, showing that for all t such that $h(t) \leq h$, the normalized weighted particles $(\mathbf{x}_{t,N}^i, \boldsymbol{W}_{t,N}^i)$ obtained from $\mathsf{dc\_smc}(t)$ are consistent with respect to (π, C_t). The base case it trivially true. Suppose t is one of the subtrees such that $h(t) = h$. Note that its two children t_{t_1} and t_{t_2} are such that $h(t_c) < h(t)$, so the induction hypothesis implies these two children populations of weighted particles $(\mathbf{x}_{t_1,N}^i, \boldsymbol{W}_{t_1,N}^i), (\mathbf{x}_{t_2,N}^i, \boldsymbol{W}_{t_2,N}^i)$ are consistent. We now show that we can adapt Theorem 1 of DM08 to establish that the weighted particles $(\mathbf{x}_{t,N}^i, \boldsymbol{W}_{t,N}^i)$ are consistent as well.

Note that for each simple f^M defined as in the proof of Lemma A2, we have:

$$\begin{aligned}
&\sum_{i=1}^{N}\sum_{j=1}^{N} \boldsymbol{W}^i_{t_1,N}\boldsymbol{W}^j_{t_2,N} f^M(\mathbf{x}^i_{t_1,N}, \mathbf{x}^i_{t_2,N}) \\
&\qquad = \sum_{m=1}^{M}\left(\sum_{i=1}^{N} \mathbf{w}^i_{t_1,N}\mathbf{1}_{A^M_m}(\mathbf{x}^i_{t_1,N})\right)\left(\sum_{j=1}^{N} \mathbf{w}^j_{t_2,N}\mathbf{1}_{B^M_m}(\mathbf{x}^j_{t_2,N})\right) \\
&\qquad \xrightarrow{\mathbb{P}} \sum_{m=1}^{M}\left(\int \mathbf{1}_{A^M_m}(\mathbf{x}_{t_1})\pi_{t_1}(\mathrm{d}\mathbf{x}_{t_1})\right)\left(\int \mathbf{1}_{B^M_m}(\mathbf{x}_{t_2})\pi_{t_2}(\mathrm{d}\mathbf{x}_{t_2})\right) \\
&\qquad = \iint f^M(\mathbf{x}_{t_1}, \mathbf{x}_{t_2})(\pi_{t_1}\otimes\pi_{t_2})(\mathrm{d}\mathbf{x}_{t_1}\times\mathrm{d}\mathbf{x}_{t_2}). \qquad (3)
\end{aligned}$$

Next, we show that this convergence in probability can be lifted from simple f^M to general bounded $\mathcal{F}_{t_1}\otimes\mathcal{F}_{t_2}$-measurable functions. To shorten the notation, let:

$$\begin{aligned}
\mu_A(f) &= \pi_{t_1}\otimes\pi_{t_2}(\mathbf{1}_A f) \\
\mu^N_A(f) &= \sum_{i=1}^{N}\sum_{j=1}^{N} \boldsymbol{W}^i_{t_1,N}\boldsymbol{W}^j_{t_2,N}\mathbf{1}_A f(\mathbf{x}^i_{t_1,N}, \mathbf{x}^i_{t_2,N}),
\end{aligned}$$

(and $\mu(f), \mu^N(f)$ are defined similarly but without the indicator restriction).

Let $\epsilon, \delta > 0$ be given. Using the result of Equation (3), first pick $\underline{N} > 0$ such that for all $N \geq \underline{N}$, $\mathbb{P}(|\mu^N_{B^c}(f^M) - \mu_{B^c}(f^M)| > \epsilon) < \delta/2$. Second, using Lemma A2 pick $B \in \mathcal{F}_{t_1}\otimes\mathcal{F}_{t_2}$ and $M > 0$ such that $\sup_{\mathbf{x}\notin B}|f^M(\mathbf{x}) - f(\mathbf{x})| < \epsilon/C$ and $\mu(B) < \epsilon/C$. This implies that both $|\mu_{B^c}(f^M) - \mu_{B^c}(f)|$ and $|\mu^N_{B^c}(f^M) - \mu^N_{B^c}(f)|$ are bounded above by ϵ. Third, using Lemma A1, pick a cover A of B, composed of a union of rectangles and such that $\mu(A) < \mu(B) + \epsilon/C$. Using Equation (3) again, pick $\underline{N}' > 0$ such that for all $N \geq \underline{N}'$, $\mathbb{P}(|\mu^N(A) - \mu(A)| > \epsilon/C) < \delta/2$. Applying Lemma A3 with $X = \mu^N(A), \beta_1 = \beta_2 = \epsilon/C$, $x = \mu(A)$, $\alpha = \delta/2$ and $y = \mu(B)$, we get $\mathbb{P}(\mu^N(A) > 3\epsilon/C) < \delta/2$.

From these choices, we obtain that for all $N > \max\{\underline{N}, \underline{N}'\}$:

$$\mathbb{P}(|\mu^N(f) - \mu(f)| > 4\epsilon) \leq \mathbb{P}\left(\mu^N_B(f) > 4\epsilon\right) + \mathbb{P}\left(D_1 + D_2 + D_3 + \mu_B(f) > 4\epsilon\right),$$

where:

$$\begin{aligned}
D_1 &= |\mu^N_{B^c}(f) - \mu^N_{B^c}(f^M)| \\
D_2 &= |\mu^N_{B^c}(f^M) - \mu_{B^c}(f^M)| \\
D_3 &= |\mu_{B^c}(f^M) - \mu_{B^c}(f)|.
\end{aligned}$$

Therefore:

$$\begin{aligned}\mathbb{P}(|\mu^N(f) - \mu(f)| > 4\epsilon) &\leq \mathbb{P}(\mu^N(B) > 4\epsilon/C) + \mathbb{P}(D_2 > \epsilon) \\ &\leq \mathbb{P}(\mu^N(A) > 3\epsilon/C) + \delta/2 \\ &\leq \delta.\end{aligned}$$

Next, we use the fact that resampling is performed at every iteration. First consider the case of simple multinomial resampling. Condition 4, and Theorem 3 from DM08, allow us to view resampling as reducing the N^2 particles into N unweighted particles. We plug in the following quantities in their notation:

$$\begin{aligned}M_N &= N^2 \\ \widetilde{M}_N &= N \\ \xi_{N,(i,j)} &= (\mathbf{x}^i_{t_1,N}, \mathbf{x}^j_{t_2,N}) \\ \omega_{N,(i,j)} &= \boldsymbol{W}^i_{t_1,N} \boldsymbol{W}^j_{t_2,N}.\end{aligned}$$

This yields that the N particles obtained from resampling N times from the particle approximation of $\pi_{t_1} \times \pi_{t_2}$ creates a consistent approximation. Theorem 1 from DM08 closes the induction argument. The use of mixture resampling is addressed with an additional application of Theorem 1 of DM08 with a trivial Markov kernel and the Radon-Nikodym derivative obtained from the properly normalised mixture-sampling weights prior to such the use of Theorem 3 (together, of course, with the use of the appropriate weight expressions throughout the argument). □

B D&C-SMC algorithm

In Algorithms B1 and B2 we provide pseudo-code for D&C-SMC using mixture sampling and annealing, and D&C-SMC using lightweight mixture sampling and annealing, respectively (see Sections 4.1 and 4.2 of the main paper).

C Supplement on the square-lattice MRF models

C.1 Additional numerical results for the Ising model

In this section we provide some additional numerical results for the Ising model presented in Section 5.1 of the main paper. The Matlab code used to run the various samplers for the Ising model is available at `https://github.com/freli005/divide-and-conquer-smc`.

Algorithm B1 $\mathsf{dc\_smc}(t)$ – D&C-SMC (ann+mix), using mixture sampling and annealing

1. (a) For $c \in \mathcal{C}(t)$, $(\{\mathbf{x}_c^i, \mathbf{w}_c^i\}_{i=1}^N, \widehat{Z}_c^N) \leftarrow \mathsf{dc\_smc}(c)$.

   (b) For $i = 1$ to N, draw $(\check{\mathbf{x}}_{c_1}^i, \ldots, \check{\mathbf{x}}_{c_C}^i)$ from

$$Q_t(\mathrm{d}\mathbf{x}_{c_1}, \ldots, \mathrm{d}\mathbf{x}_{c_C}) = \sum_{i_1=1}^N \cdots \sum_{i_C=1}^N \frac{v_t(i_1, \ldots, i_C)\delta_{(\mathbf{x}_{c_1}^{i_1}, \ldots, \mathbf{x}_{c_C}^{i_C})}(\mathrm{d}\mathbf{x}_{c_1}, \ldots, \mathrm{d}\mathbf{x}_{c_C})}{\sum_{j_1=1}^N \cdots \sum_{j_C=1}^N v_t(j_1, \ldots, j_C)},$$

   where

$$v_t(i_1, \ldots, i_C) = \left(\prod_{c \in \mathcal{C}(t)} \mathbf{w}_c^{i_c}\right) \check{\pi}_t(\mathbf{x}_{c_1}^{i_1}, \ldots, \mathbf{x}_{c_C}^{i_C}) \Big/ \prod_{c \in \mathcal{C}(t)} \pi_c(\mathbf{x}_c^{i_c}),$$

   and where $\check{\pi}_t$ is chosen by the user (e.g., according to Equation (8) of the main paper).

   (c) Compute

$$\widehat{Z}_t^N = \left(\prod_{c \in \mathcal{C}(t)} \widehat{Z}_c^N \left\{\frac{1}{N}\sum_{i=1}^N \mathbf{w}_c^i\right\}^{-1}\right) \left\{\frac{1}{N^C}\sum_{i_1=1}^N \cdots \sum_{i_C=1}^N v_t(i_1, \ldots, i_C)\right\}.$$

2. (a) For $i = 1$ to N, if $\tilde{\mathsf{X}}_t \neq \emptyset$, simulate $\widetilde{\mathbf{x}}_t^i \sim q_t(\cdot \,|\, \check{\mathbf{x}}_{c_1}^i, \ldots, \check{\mathbf{x}}_{c_C}^i)$ where $(c_1, c_2, \ldots, c_C) = \mathcal{C}(t)$; else $\widetilde{\mathbf{x}}_t^i \leftarrow \emptyset$.

   (b) For $i = 1$ to N, set $\mathbf{x}_{t,0}^i = (\check{\mathbf{x}}_{c_1}^i, \ldots, \check{\mathbf{x}}_{c_C}^i, \widetilde{\mathbf{x}}_t^i)$ and $\mathbf{w}_{t,0}^i = 1$.

   (c) For SMC sampler iteration $j = 1$ to n_t: (N.B. $\gamma_{t,0} = \check{\pi}_t q_t$.)

      i. For $i = 1$ to N, compute $\mathbf{w}_{t,j}^i = \mathbf{w}_{t,j-1}^i \gamma_{t,j}(\mathbf{x}_{t,j-1}^i)/\gamma_{t,j-1}(\mathbf{x}_{t,j-1}^i)$.

      ii. Optionally, resample $\{\mathbf{x}_{t,j-1}^i, \mathbf{w}_{t,j}^i\}_{i=1}^N$:

         - Update $\widehat{Z}_t^N \leftarrow \widehat{Z}_t^N \times \{\frac{1}{N}\sum_{i=1}^N \mathbf{w}_{t,j}^i\}$.
         - Override the notation $\{\mathbf{x}_{t,j-1}^i, \mathbf{w}_{t,j}^i\}_{i=1}^N$ to refer to the resampled particle system.

      iii. For $i = 1$ to N, draw $\mathbf{x}_{t,j}^i \sim K_{t,j}(\mathbf{x}_{t,j-1}^i, \cdot)$ using a $\pi_{t,j}$-reversible Markov kernel $K_{t,j}$.

   (d) For $i = 1$ to N, set $\mathbf{x}_t^i = \mathbf{x}_{t,n_t}^i$ and $\mathbf{w}_t^i = \mathbf{w}_{t,n_t}^i$.

3. Update $\widehat{Z}_t^N \leftarrow \widehat{Z}_t^N \times \{\frac{1}{N}\sum_{i=1}^N \mathbf{w}_t^i\}$.

4. Return $(\{\mathbf{x}_t^i, \mathbf{w}_t^i\}_{i=1}^N, \widehat{Z}_t^N)$.

Algorithm B2 $\mathsf{dc\_smc}(t)$ – D&C-SMC (ann+lw), using lightweight mixture sampling and annealing

1. (a) For $c \in \mathcal{C}(t)$, $(\{\mathbf{x}_c^i, \mathbf{w}_c^i\}_{i=1}^N, \widehat{Z}_c^N) \leftarrow \mathsf{dc\_smc}(c)$.
   (b) For $c \in \mathcal{C}(t)$, sample mN times with replacement from $\{\mathbf{x}_c^i, \mathbf{w}_c^i\}_{i=1}^N$. Override the notation and let $\{\mathbf{x}_c^i\}_{i=1}^{mN}$ refer to the resampled particles.
   (c) Form the mN matchings $\{(\mathbf{x}_{c_1}^i, \ldots, \mathbf{x}_{c_C}^i)\}_{i=1}^{mN}$ and compute the weights
   $$v_t^i = \breve{\pi}_t(\mathbf{x}_{c_1}^i, \ldots, \mathbf{x}_{c_C}^i) \Big/ \prod_{c \in \mathcal{C}(t)} \pi_c(\mathbf{x}_c^i), \qquad i = 1, \ldots, mN$$
   where $(c_1, c_2, \ldots, c_C) = \mathcal{C}(t)$.
   (d) Sample with replacement N particles $\{(\check{\mathbf{x}}_{c_1}^i, \ldots, \check{\mathbf{x}}_{c_C}^i)\}_{i=1}^N$ from the mN weighted particles $\{(\mathbf{x}_{c_1}^j, \ldots, \mathbf{x}_{c_C}^j), v_t^j\}_{j=1}^{mN}$.
   (e) Compute
   $$\widehat{Z}_t^N = \left(\prod_{c \in \mathcal{C}(t)} \widehat{Z}_c^N \right) \left\{ \frac{1}{mN} \sum_{i=1}^{mN} v_t^i \right\}.$$
2. (a) For $i = 1$ to N, if $\tilde{\mathsf{X}}_t \neq \emptyset$, simulate $\widetilde{\mathbf{x}}_t^i \sim q_t(\cdot \,|\, \check{\mathbf{x}}_{c_1}^i, \ldots, \check{\mathbf{x}}_{c_C}^i)$ where $(c_1, c_2, \ldots, c_C) = \mathcal{C}(t)$; else $\widetilde{\mathbf{x}}_t^i \leftarrow \emptyset$.
   (b) For $i = 1$ to N, set $\mathbf{x}_{t,0}^i = (\check{\mathbf{x}}_{c_1}^i, \ldots, \check{\mathbf{x}}_{c_C}^i, \widetilde{\mathbf{x}}_t^i)$ and $\mathbf{w}_{t,0}^i = 1$.
   (c) For SMC sampler iteration $j = 1$ to n_t: (N.B. $\gamma_{t,0} = \breve{\pi}_t q_t$.)
      i. For $i = 1$ to N, compute $\mathbf{w}_{t,j}^i = \mathbf{w}_{t,j-1}^i \gamma_{t,j}(\mathbf{x}_{t,j-1}^i) / \gamma_{t,j-1}(\mathbf{x}_{t,j-1}^i)$.
      ii. Optionally, resample $\{\mathbf{x}_{t,j-1}^i, \mathbf{w}_{t,j}^i\}_{i=1}^N$:
         - Update $\widehat{Z}_t^N \leftarrow \widehat{Z}_t^N \times \{\frac{1}{N} \sum_{i=1}^N \mathbf{w}_{t,j}^i\}$.
         - Override the notation $\{\mathbf{x}_{t,j-1}^i, \mathbf{w}_{t,j}^i\}_{i=1}^N$ to refer to the resampled particle system.
      iii. For $i = 1$ to N, draw $\mathbf{x}_{t,j}^i \sim K_{t,j}(\mathbf{x}_{t,j-1}^i, \cdot)$ using a $\pi_{t,j}$-reversible Markov kernel $K_{t,j}$.
   (d) For $i = 1$ to N, set $\mathbf{x}_t^i = \mathbf{x}_{t,n_t}^i$ and $\mathbf{w}_t^i = \mathbf{w}_{t,n_t}^i$.
3. Update $\widehat{Z}_t^N \leftarrow \widehat{Z}_t^N \times \{\frac{1}{N} \sum_{i=1}^N \mathbf{w}_t^i\}$.
4. Return $(\{\mathbf{x}_t^i, \mathbf{w}_t^i\}_{i=1}^N, \widehat{Z}_t^N)$.

Please see the `README.md` file in the repository for details on how to run the simulations.

First, in Figures 1–2 we give results complementing those shown in Figure 6 of the main paper with the estimates for D&C-SIR , D&C-SMC (mix), and D&C-SMC (mix+ann). D&C-SMC (mix) and D&C-SMC (mix+ann) use the same numbers of particles as in the main paper: $N \in \{2^6, 2^7, \ldots, 2^{11}\}$, whereas D&C-SIR uses $N \in \{2^{10}, 2^{11}, \ldots, 2^{15}\}$ to more closely match its computational cost with the other methods (with more than 2^{15} particles we ran into memory limitations).

We also show results for a random scan blocked Gibbs sampler with a block size of 3×3 sites. That is, at each step of the blocked Gibbs sampler we choose a random lattice index (i, j) uniformly on $\{1, \ldots, M\}^2$ and update the sites with row indices $\{i, i+1, i+2\}$ and column indices $\{j, j+1, j+2\}$ (periodic at the boundaries). We run the sampler for 2^{13} iterations, discarding the first 2^9 as burn-in—here, one iteration consists of $\lfloor (64/3)^2 \rfloor$ individual block update steps, so that we on average update each lattice site once per iteration. We also tried block sizes 2×2 and 4×4 with inferior results (in particular for the latter which resulted in a large computational overhead, suggesting that even larger block sizes will be inefficient as well). Thus, the results for 2×2 and 4×4 are not shown here. Finally, for easier comparison with Figure 6 of the main paper we also repeat the results for D&C-SMC (ann+lw) in Figures 1–2.

Comparing Figure 2 with Figure 6 of the main paper shows that the blocked Gibbs sampler performs somewhat better than the single-flip MH sampler (but still worse than the D&C-SMC methods with annealing). It should be noted that all of the annealed D&C-SMC methods that we have considered still use single-flip MH updates for the MCMC kernels of their annealing processes. However, it would of course be possible to use, e.g., blocked Gibbs updates also together with the annealed D&C-SMC methods, which is expected to improve their performances even further. Indeed, any improvements made on the MCMC sampler can be directly used together with the annealed D&C-SMC samplers as well.

Next, we repeated the numerical evaluation described in Section 5.1 of the main paper for different inverse temperatures of the Ising model: $\beta = 0.40$ (90% of critical inverse temperature) and $\beta = 0.48$ (110% of critical inverse temperature). The results are given in Figures 3–6 and Figures 7–10, respectively. (To avoid overly cluttered figures, we keep the separation of the different methods under comparison into two groups as before, with the first group consisting of SMC, D&C-SMC (ann), D&C-SMC (ann+lw) and single-flip MH, and the second group consisting of D&C-SIR , D&C-SMC (mix), D&C-SMC (ann+mix) and blocked Gibbs (3×3). The results for D&C-SMC (ann+lw) are plotted in both figures for easier comparison.)

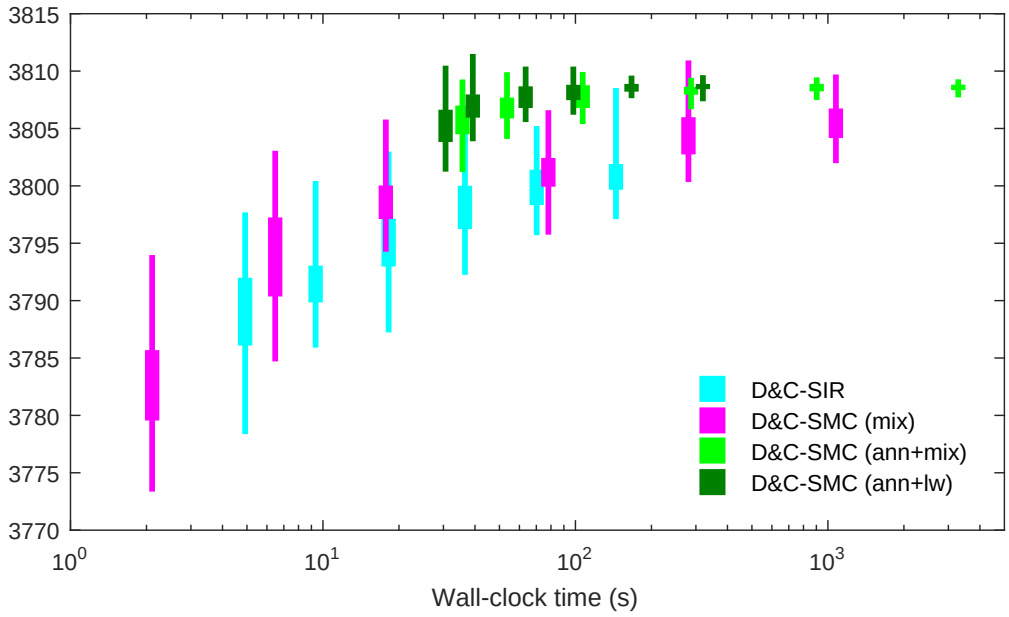

Figure 1: Box-plots (min, max, and inter-quartile) of estimates of $\log Z$ for the Ising model with $\beta = 0.44$ over 50 runs of each sampler (excluding the blocked Gibbs sampler which does not readily provide an estimate of $\log Z$). The boxes, as plotted from left to right, correspond to increasing number of particles N. This figure is best viewed in color.

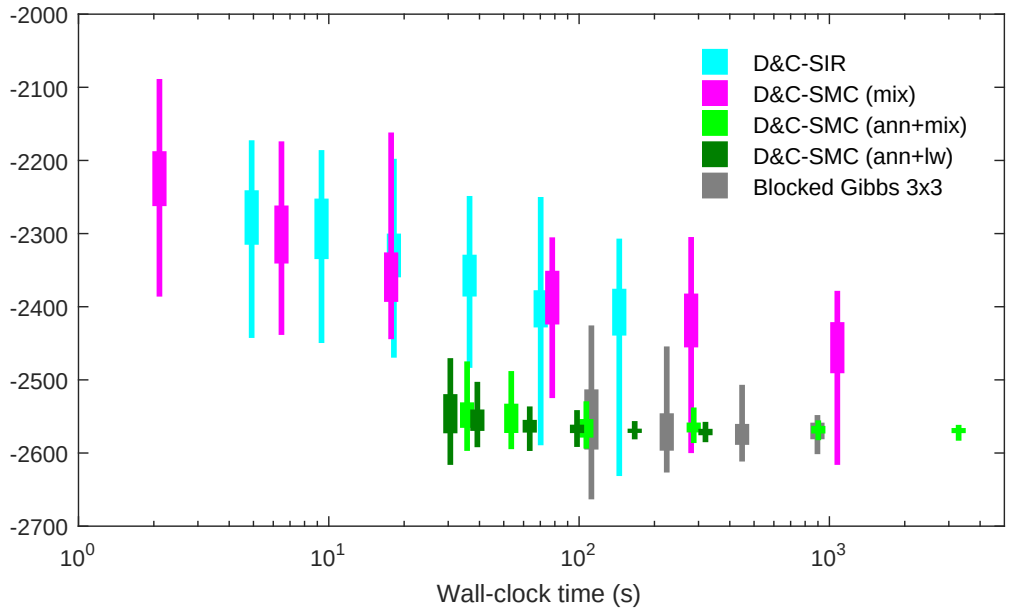

Figure 2: Box-plots (min, max, and inter-quartile) of estimates of $\mathbb{E}[E(\mathbf{x})]$ for the Ising model with $\beta = 0.44$ over 50 runs of each sampler. The boxes, as plotted from left to right, correspond to increasing number of particles N (or number of MCMC iterations for the blocked Gibbs sampler). This figure is best viewed in color.

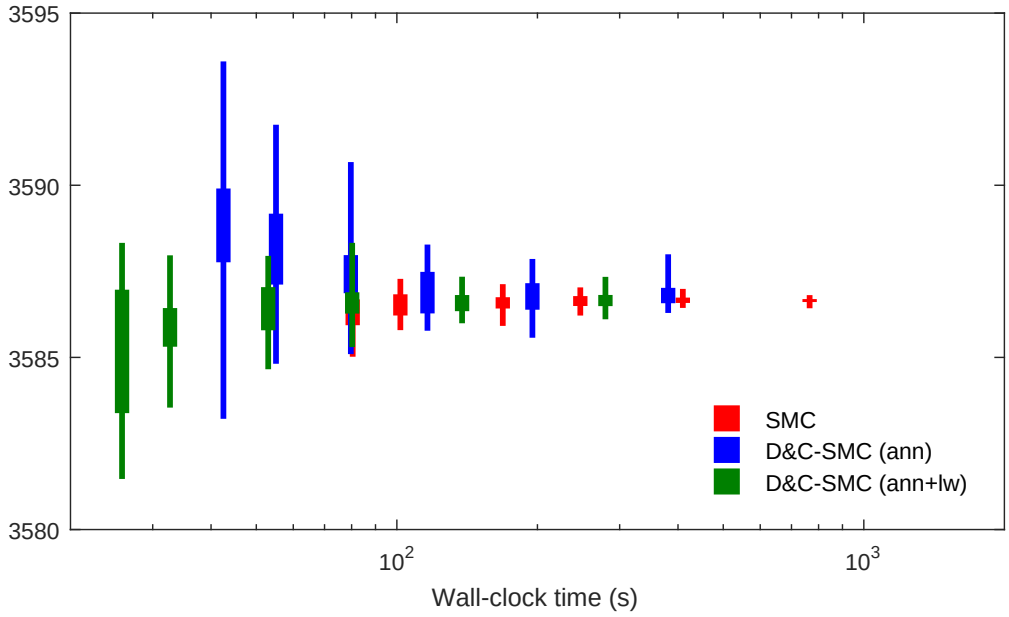

Figure 3: Box-plots (min, max, and inter-quartile) of estimates of $\log Z$ for the Ising model with $\beta = 0.40$ over 50 runs of each sampler in "group 1" (excluding the single-flip MH sampler which does not readily provide an estimate of $\log Z$). The boxes, as plotted from left to right, correspond to increasing number of particles N. This figure is best viewed in color.

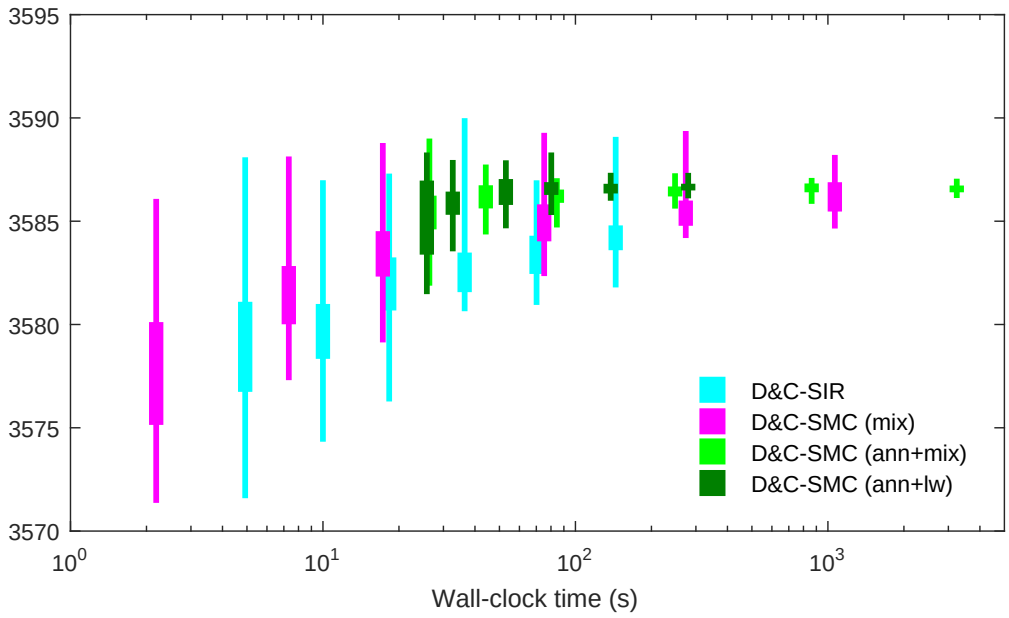

Figure 4: Box-plots (min, max, and inter-quartile) of estimates of $\log Z$ for the Ising model with $\beta = 0.40$ over 50 runs of each sampler in "group 2" (excluding the blocked Gibbs sampler which does not readily provide an estimate of $\log Z$). The boxes, as plotted from left to right, correspond to increasing number of particles N. This figure is best viewed in color.

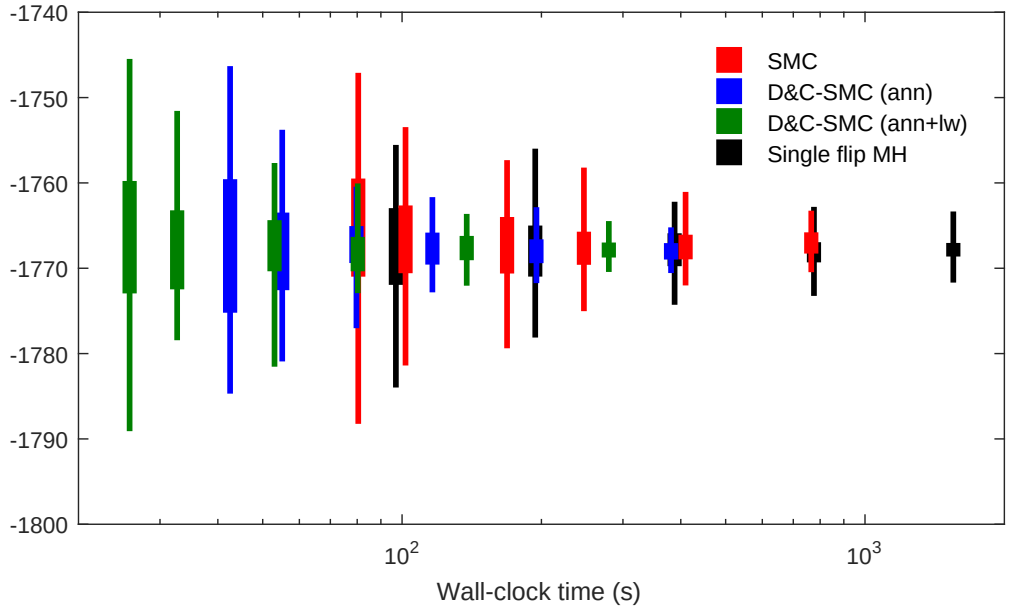

Figure 5: Box-plots (min, max, and inter-quartile) of estimates of $\mathbb{E}[E(\mathbf{x})]$ for the Ising model with $\beta = 0.40$ over 50 runs of each sampler in "group 1". The boxes, as plotted from left to right, correspond to increasing number of particles N (or number of MCMC iterations for the single-flip MH sampler). This figure is best viewed in color.

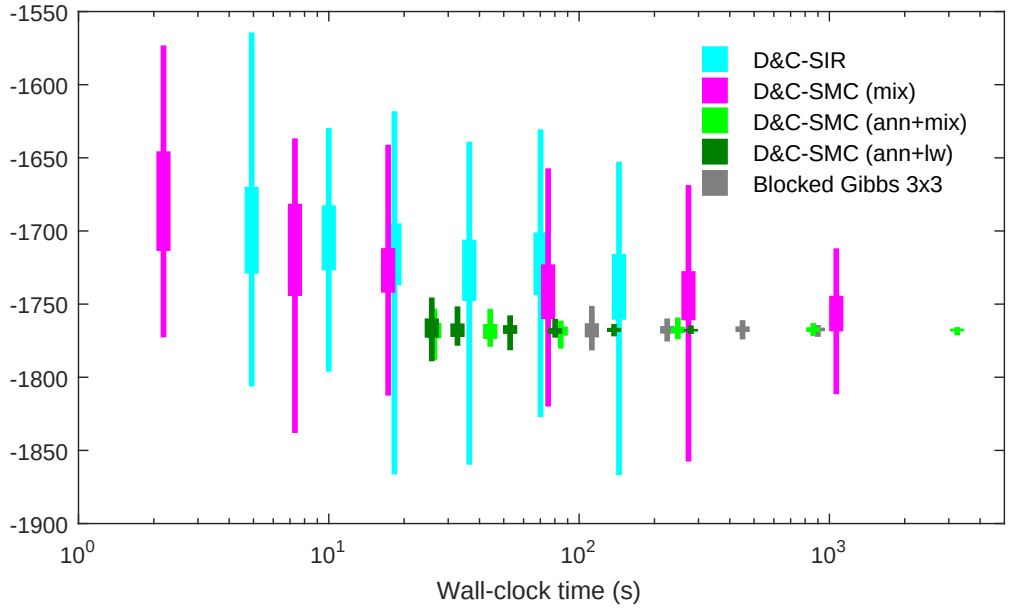

Figure 6: Box-plots (min, max, and inter-quartile) of estimates of $\mathbb{E}[E(\mathbf{x})]$ for the Ising model with $\beta = 0.40$ over 50 runs of each sampler in "group 2". The boxes, as plotted from left to right, correspond to increasing number of particles N (or number of MCMC iterations for the blocked Gibbs sampler). This figure is best viewed in color.

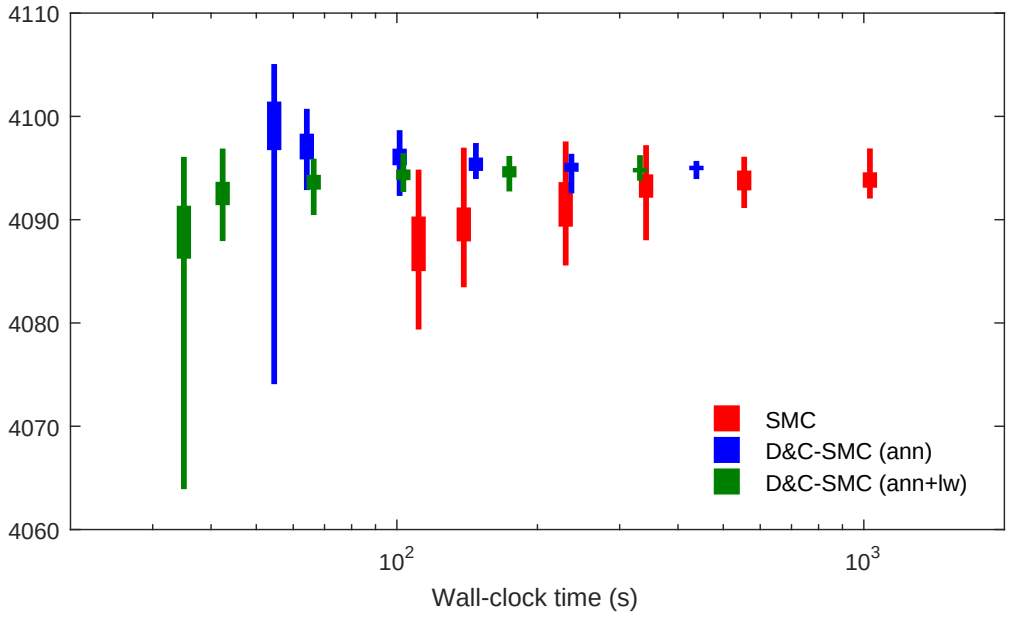

Figure 7: Box-plots (min, max, and inter-quartile) of estimates of $\log Z$ for the Ising model with $\beta = 0.48$ over 50 runs of each sampler in "group 1" (excluding the single-flip MH sampler which does not readily provide an estimate of $\log Z$). The boxes, as plotted from left to right, correspond to increasing number of particles N. This figure is best viewed in color.

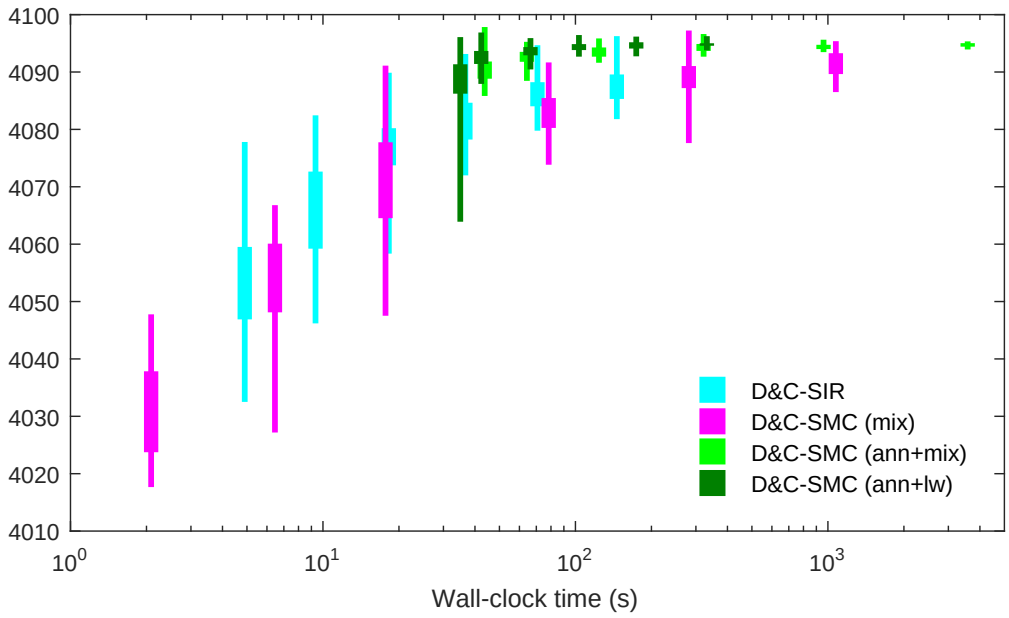

Figure 8: Box-plots (min, max, and inter-quartile) of estimates of $\log Z$ for the Ising model with $\beta = 0.48$ over 50 runs of each sampler in "group 2" (excluding the blocked Gibbs sampler which does not readily provide an estimate of $\log Z$). The boxes, as plotted from left to right, correspond to increasing number of particles N. This figure is best viewed in color.

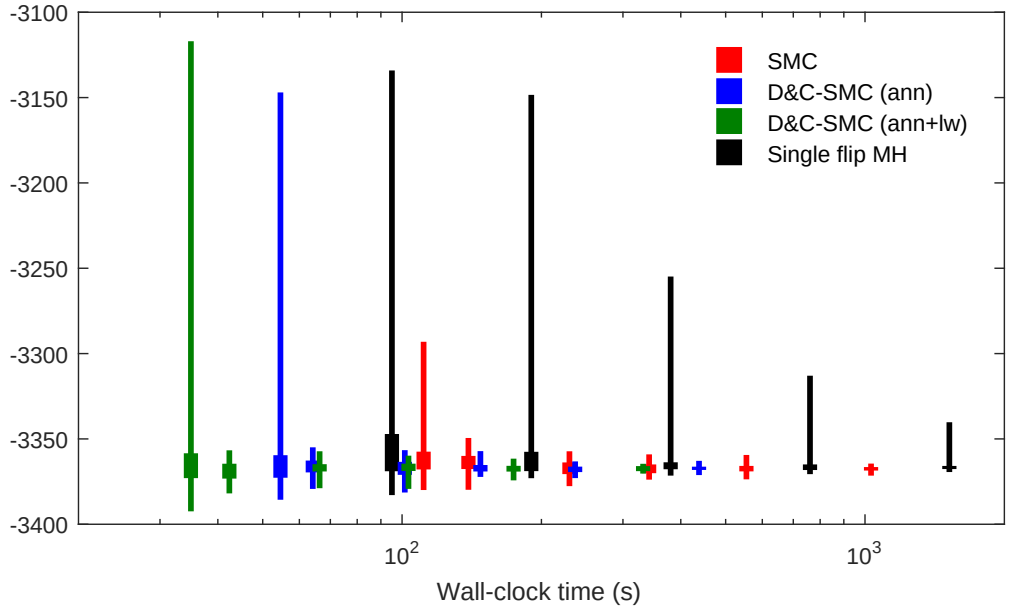

Figure 9: Box-plots (min, max, and inter-quartile) of estimates of $\mathbb{E}[E(\mathbf{x})]$ for the Ising model with $\beta = 0.48$ over 50 runs of each sampler in "group 1". The boxes, as plotted from left to right, correspond to increasing number of particles N (or number of MCMC iterations for the single-flip MH sampler). This figure is best viewed in color.

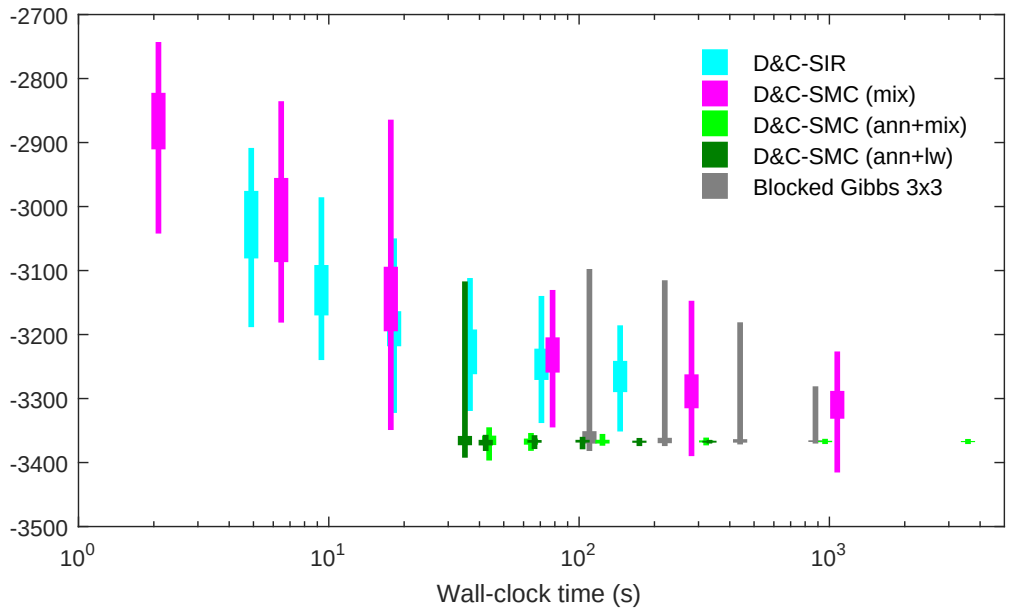

Figure 10: Box-plots (min, max, and inter-quartile) of estimates of $\mathbb{E}[E(\mathbf{x})]$ for the Ising model with $\beta = 0.48$ over 50 runs of each sampler in "group 2". The boxes, as plotted from left to right, correspond to increasing number of particles N (or number of MCMC iterations for the blocked Gibbs sampler). This figure is best viewed in color.

C.2 Square-lattice MRF with continuous latent variables

In this section we evaluate the proposed D&C-SMC method on a second square-lattice MRF example. We consider a toy-model consisting of a latent Gaussian field $\mathbf{x} \in \mathbb{R}^{M \times M}$ ($M = 32$), with nearest-neighbour interactions and periodic boundary conditions:

$$p(\mathbf{x}) \propto \exp\left(-\frac{1}{2}\left\{\sum_{(k,\ell)\in\mathcal{E}} \lambda_1(\widetilde{\mathbf{x}}_k - \widetilde{\mathbf{x}}_\ell)^2 + \lambda_2 \sum_{k\in\mathcal{V}} \widetilde{\mathbf{x}}_k^2\right\}\right), \tag{4}$$

with $\mathcal{E}$ and $\mathcal{V}$ being the edge set and vertex set of the MRF, respectively. We set $\lambda_1 = 10$ and $\lambda_2 = 0.01$ for the interaction strength and node potential, respectively. To each node of the MRF we also attach an observed variable y_k, conditionally distributed according to $p(y_k \,|\, \widetilde{\mathbf{x}}_k) = \mathcal{N}(y_k \,|\, \widetilde{\mathbf{x}}_k^2, 0.05^2)$, $k \in \mathcal{V}$. The target distribution for the samplers is then given by the Bayesian posterior distribution $p(\mathbf{x} \,|\, \mathbf{y})$, where $\mathbf{y} = \{y_k\}_{k\in\mathcal{V}}$.

We simulate a batch of data from the model and apply the following inference methods to sample from the posterior distribution (see Section 5.1 of the main paper for explanations): SMC, D&C-SIR , D&C-SMC (ann), D&C-SMC (mix), D&C-SMC (ann+mix), as well as a single-site Metropolis-within-Gibbs sampler with a random walk proposal. We did not try D&C-SMC (ann+lw) for this example, but note that this method is essentially an intermediate between D&C-SMC (ann) and D&C-SMC (ann+mix).

We initialise all methods by sampling independently for each $k \in \mathcal{V}$ from a distribution given by,

$$\mu_k(\widetilde{\mathbf{x}}_k) \propto \mathcal{N}(y_k \,|\, \widetilde{\mathbf{x}}_k^2, 0.05^2) \exp\left(-\frac{\lambda_2}{2}\widetilde{\mathbf{x}}_k^2\right), \tag{5}$$

which we compute to high precision by (one-dimensional) adaptive quadrature. This allows us to focus the evaluation on the difficulties arising from the interactions between neighbouring sites, since (5) is exactly the posterior distribution of $\widetilde{\mathbf{x}}_k$ if we neglect all interactions. Note that we observe the square of the latent variables $\widetilde{\mathbf{x}}_k$ (plus noise). Consequently, the distributions (5), as well as the marginal posteriors $p(\widetilde{\mathbf{x}}_k \,|\, \mathbf{y})$, tend to be bimodal whenever $|\widetilde{\mathbf{x}}_k|$ is large (relative to the observation variance). This poses significant difficulties for the single site MH sampler and the standard SMC sampler, as we shall see below.

All algorithmic settings are the same as in Section 5.1 of the main paper, with the exception that the MCMC kernel used for the annealed methods (and for the single site MH sampler) uses a Gaussian random walk proposal with standard deviation 0.132 (chosen by manual tuning; we obtained an average acceptance rate of 0.6–0.7 for all methods). It is interesting to note that we only needed to make small modification to the code used for the Ising model, changing only the initialisation procedure, the energy function, and the MCMC

kernel.

Figures 11 and 12 show results on the estimates of the log-normalising-constant and the expected energy for the posterior distribution, obtained from 50 runs of each algorithm on a single data set. The results for the single site random-walk MH sampler are not shown since the method fails completely to converge to samples from the correct posterior distribution (inter-quartile range in the estimates of the expected energy over the 50 independent runs of the MCMC sampler is more than 7 000; cf., Figure 12).

We also note a superior performance of all D&C-SMC samplers compared to SMC, which is attributed to the fact that the D&C-SMC samplers are better able to handle the multimodality of the initial distributions than the "standard" SMC approach considered: in standard SMC we initialise a single particle population by simulating N times from the product measure $\otimes_{k\in\mathcal{V}}\mu_k(\mathrm{d}\widetilde{\mathbf{x}}_k)$, where μ_k is given by (5). Since many of the μ_k-s are expected to be multimodal, this will likely result in particles that have very low posterior probability under the "correct model" (i.e., with the interactions taken into account). Indeed, any neighbouring pair $(\widetilde{\mathbf{x}}_k, \widetilde{\mathbf{x}}_\ell)$ where $\widetilde{\mathbf{x}}_k$ is drawn from the "positive mode" and $\widetilde{\mathbf{x}}_\ell$ is drawn from the "negative mode" will have low probability under the prior (4). Furthermore, since we use local MCMC moves to update the particle positions, the method is not able to correct for this in a sufficiently efficient manner during the annealing process. The D&C-SMC samplers are much better suited to handling this difficulty, since they make use of *multiple* particle populations at initialisation, one for each site. The interactions are thereafter gradually taken into account in the merge steps of the algorithm. This improvement comes without any application-specific implementation effort.

To further illustrate the ability of D&C-SMC to capture the multimodality of the marginal posteriors, we show in Figure 13 the empirical cumulative distribution functions for the samplers for four sites with increasing $|\widetilde{\mathbf{x}}_k|$. The results are for $N = 2\,048$ ($N = 32\,768$ for D&C-SIR) and each line correspond to one of the 50 independent runs. It is clear that the D&C-SMC samplers maintains the multimodality of the posterior much better than standard SMC.

D Supplement on the hierarchical Bayesian model

This section contains additional details and results for the hierarchical Bayesian logistic regression considered in Section 5.2 of the main paper.

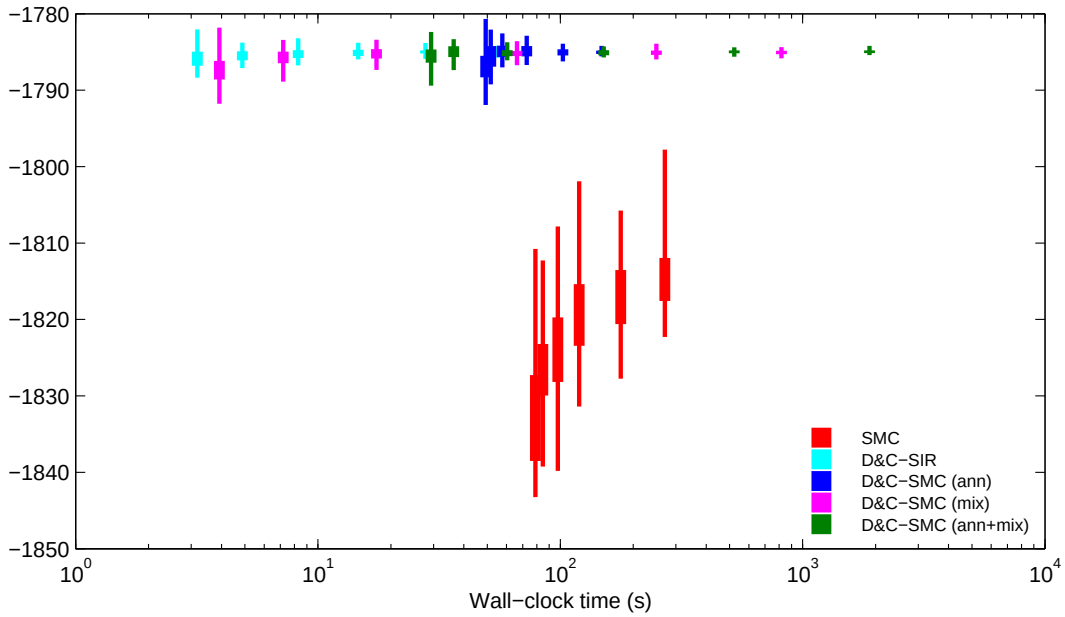

Figure 11: Box-plots of estimates of $\log Z$ over 50 runs of each sampler. The boxes, as plotted from left to right, correspond to increasing number of particles $N = 2^6$ to 2^{11} ($N = 2^{10}$ to 2^{15} for D&C-SIR). This figure is best viewed in color.

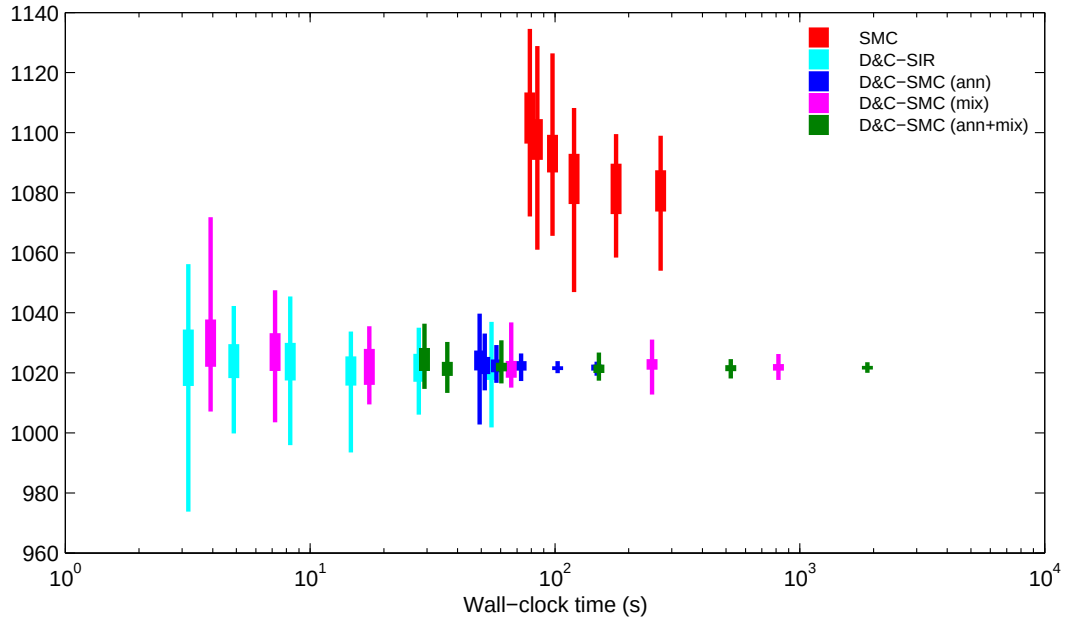

Figure 12: Box-plots of estimates of $\mathbb{E}[E(\mathbf{x})]$ over 50 runs of each sampler. The boxes, as plotted from left to right, correspond to increasing number of particles $N = 2^6$ to 2^{11} ($N = 2^{10}$ to 2^{15} for D&C-SIR). This figure is best viewed in color.

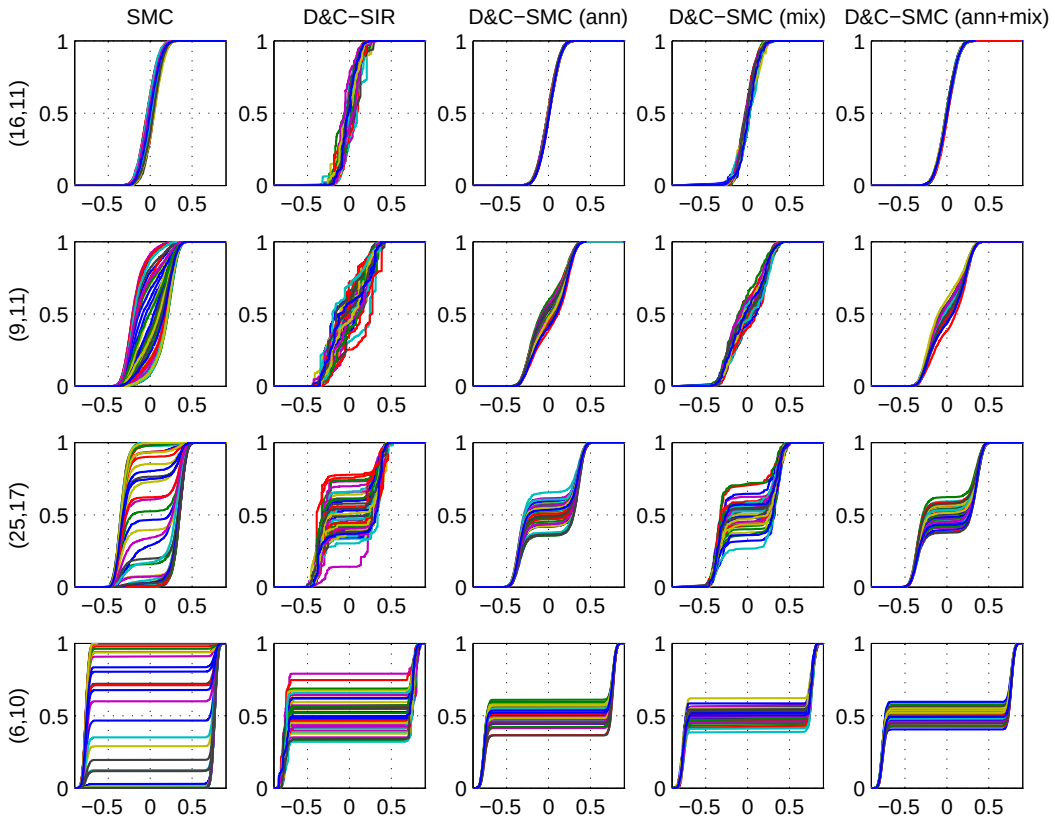

Figure 13: Empirical cumulative distribution functions for the marginal posteriors $p(\widetilde{\mathbf{x}}_k \,|\, \mathbf{y})$ for four sites with increasing $|\widetilde{\mathbf{x}}_k|$ (from top to bottom). Each line correspond to one of the 50 independent runs of each algorithm.

D.1 Data pre-processing

The data was downloaded from `https://data.cityofnewyork.us/Education/NYS-Math-Test-Results-By-Grade-2006-2011-School-Le/jufi-gzgp` on May 26, 2014. It contains New York City Results on the *New York State Mathematics Tests*, Grades 3-8. We used data from grade 3 only.

Data is available for the years 2006–2011. We removed years 2010 and 2011, since the documentation attached to the above URL indicates that: "Starting in 2010, NYSED changed the scale score required to meet each of the proficiency levels, increasing the number of questions students needed to answer correctly to meet proficiency."

Each row in the dataset contains a school code, a year, a grade, the number of students tested, and summary statistics on their grades. We use the last column of these summary statistics, which corresponds to the number of students that obtained a score higher than a fixed threshold.

Moreover, for each school code, we were able to extract its school district. We removed the data from the schools in School District 75. This is motivated by the specialized character of School District 75: "District 75 provides citywide educational, vocational, and behavior support programs for students who are on the autism spectrum, have significant cognitive delays, are severely emotionally challenged, sensory impaired and/or multiply disabled." (`http://schools.nyc.gov/Academics/SpecialEducation/D75/AboutD75/default.htm`)

For each school district we can also extract its county, one of Manhattan, Bronx, Kings, Queens, Richmond (some of these correspond to NYC boroughs with the same name, while Kings corresponds to Brooklyn; Richmond, to Staten Island; and Bronx, to The Bronx). The pre-processing steps can be reproduced using the script `scripts/prepare-data.sh` in the repository.

D.2 Additional information on the experimental setup

We checked correctness of the implementations by verifying on small examples that the posterior distributions obtained from all four methods (the three baselines and ours) become arbitrarily close when increasing the number of particles or MCMC iterations. We also subjected our Gibbs implementation to the Prior/Prior-Posterior correctness test of Geweke (2004).

The precise command line arguments used in the experiments as well as the scripts used for creating the plots can be found at `https://github.com/alexandrebouchard/divide-and-conquer-smc-experiments`. To describe in more detail the methods we compared to, we introduce the following notation: given a permutation $t_1, t_2, \ldots$ of the index set T, we set Ξ_j to θ_{t_j} if t_j is a leaf, or to $\sigma^2_{t_j}$ otherwise. The three baseline methods are:

Gibbs: A Metropolis-within-Gibbs algorithm, proposing a change on a single variable, using a normal proposal of unit variance. As with D&C-SIR , we marginalize the internal θ_t parameters. More precisely, we first sample a permutation $t_1, t_2, \ldots$ uniformly at random, then using the Metropolis-within-Gibbs kernel, we sample Ξ_1, followed by Ξ_2, and so, until we sample $\Xi_{|T|}$, at which point we sample an indendent permutation uniformly at random and restart the process. The Java implementation is available at `https://github.com/alexandrebouchard/divide-and-conquer-smc`, in the package "prototype.mcmc". We use a burn-in of 10% and collect sample each time we resample a permutation of the nodes.

Stan: An open-source implementation of the Hamiltonian Monte Carlo algorithm (Neal, 2011), more precisely of the No U-Turn Sampler (Hoffman and Gelman, 2012), which adaptively selects the number of "leap frog" steps. Stan generates efficient C++ code. In contrast to the other methods, we did not implement marginalization of the internal θ_t parameters. Stan includes a Kalman inference engine, however it is limited to chain-shaped graphical models as of version 2.6.0. We use the default setting for the burn-in and adaptation period (50%) and a thinning of 10. In other words, we store one sample every 10 accept-reject rounds. Since a large number of Hamilton Monte Carlo iterations was needed, this was need to make storage manageable. We verified using an ACF plot that this did not significantly penalize this method.

STD: A standard (single population) bootstrap filter with the intermediate distributions being sub-forests incrementally built in post-order. More precisely, let $t_1, t_2, \ldots$ denote a fixed post-order traversal of the nodes in the set T. Note that a prefix of this traversal, $t_1, t_2, \ldots, t_k$, corresponds to a forest, with a corresponding parameter vector $\Xi_{1:k} = (\Xi_1, \Xi_2, \ldots, \Xi_k)$. We use a standard SIR algorithm to sample $\Xi_{1:1}, \Xi_{1:2}, \Xi_{1:3}, \ldots$, using the canonical sequence of intermediate distributions. Again, as with D&C-SIR , we marginalize the internal θ_t parameters. To propose an additional parameter Ξ_i given its children, we use the same proposal distributions as those we used for D&C-SIR (described in Section 5.2 of the main paper). The Java implementation is available at `https://github.com/alexandrebouchard/divide-and-conquer-smc`.

D.3 Additional results (serial computation)

D.3.1 Posterior densities

We show in Figure 15 the posterior densities of the parameters σ_t^2, for t ranging in the nodes in the two top levels of the tree, and for the four methods with different numbers of particles or MCMC iterations. The plots support that 1 000 particles are sufficient for D&C-SMC

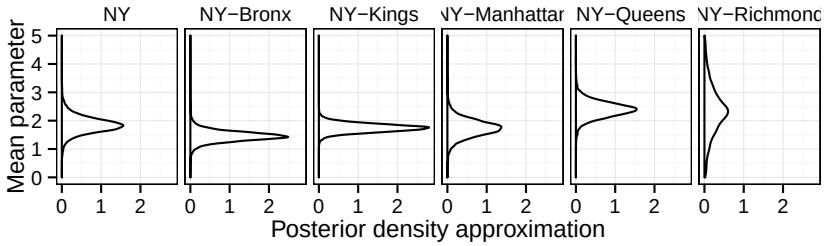

Figure 14: Posterior densities for the parameters θ_t with D&C-SIR ran with 10 000 particles. The values of the internal θ_t are marginalized during inference, but can be easily reinstantiated from the samples as a post-processing step.

to output a reasonable approximation (see rows DC-N), while a larger number of MCMC iterations seem required for Gibbs (GIBBS-N) or Stan (STAN-N). Note however that DC-N and STD-N performed similarly in this first set of results.

Next, we show in Figure 16 a comparison of the wall clock times for the different algorithm configurations. These experiments were performed on Intel Xeon E5430 quad-core processors, running at 2.66 GHz. While the detailed rankings of the methods are implementation and architecture dependent, the results show that the DC-1 000 was computed in approximately one minute, while all the reasonably mixed MCMC methods required at least one order of magnitude more time. For example, running 2 000 Stan iterations (1 000 burn-in + 1 000 samples) took around 6 hours. All Stan running times exclude the one-time model compilation.

The results for both SMC and MCMC are all performed on a single thread. We attribute the high computational efficiency of the SMC methods to the favorable memory locality of the particle arrays. This is supported by the non-linearity of the running time going from 1000 to 10 000 particles (the theoretical running time is linear in the number of particles).

Note that for any given particle budget, our implementation of D&C-SMC was slightly faster than STD. This could be caused by implementation details related to the fact that the particle datastructure for D&C-SMC are simpler than those required by STD (the particles being forests in STD, instead of trees for D&C-SMC).

These results further demonstrate that the model and data used in this section yield a challenging posterior inference problem.

D.3.2 Additional results on the normalization constant

In this section, we use an example where the true value of the normalization constant Z can be computed to validate our implementation of D&C-SMC . The model we use for this purpose consists in a tree shaped graphical model with a finite set of state at each

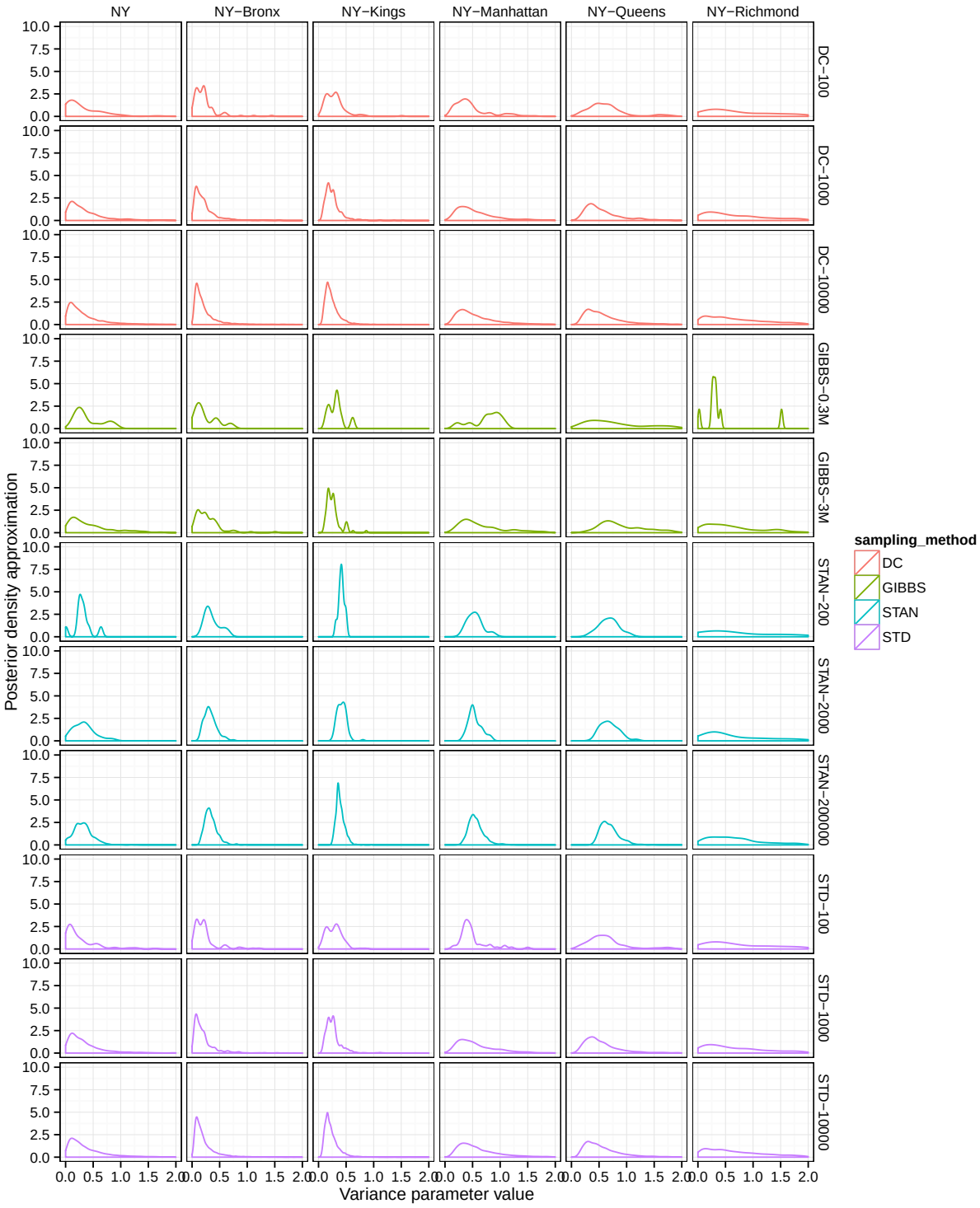

Figure 15: Posterior densities for the parameters σ_t^2. The rows index different methods and numbers of particles/iterations. The columns index different parameters t (only the first two levels shown).

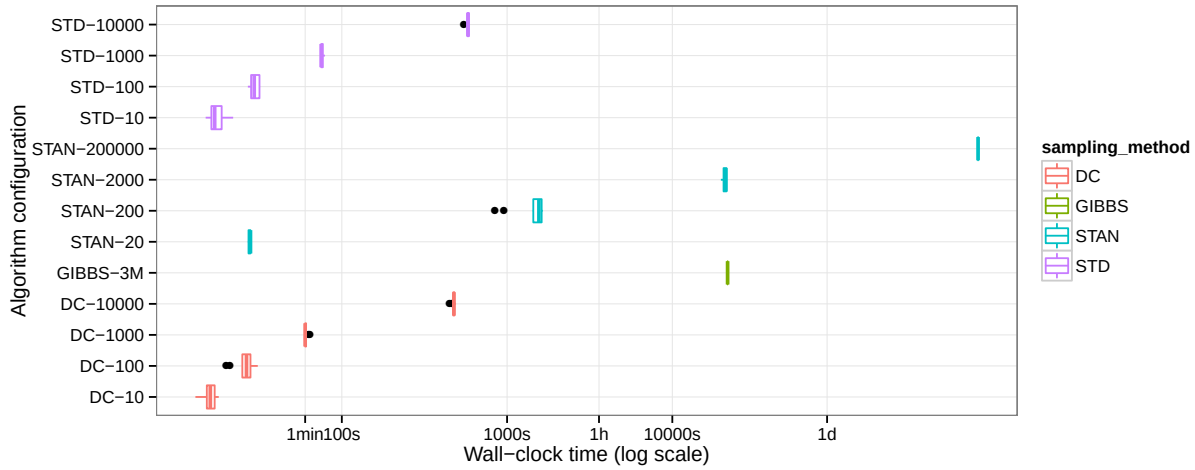

Figure 16: Comparison of the wall clock times for the different algorithm configurations. Note that the time axis is in log scale, and that iterations/particles were incremented exponentially. All experiments were replicated 10 times (varying only the Monte Carlo random seed), with the exception of the largest experiments taking more than 6 hours, which we ran only once (GIBBS-3M and STAN-200000).

node. We use a perfect binary tree of depth five and two states at each nodes. Only the leaves are observed. The details of this model are described in the file `README.md` of the code repository. While the state space has cardinality exponential in the depth of the tree, dynamic programming can be used to efficiently compute the value of Z (which corresponds in this case to the probability of the observations). For actual data analysis, D&C-SMC would not be advantegeous in this particular model, but again our goal here is to validate our implementation of the D&C-SMC recursions and normalization computation.

We fix the data, and vary the number of particles used, from 10 to 81 920 by doubling. Each configuration is repeated 100 times with different random seeds. The results are shown in Figure 17. They support that the means of the estimates converge to the true value, and that the variances of the estimates converge to zero. Moreover, the theory (specifically, combining Jensen's inequality with Proposition 1) predicts that the convergence is from below, i.e. that the means are no greater than the true value. This is again what we observe in Figure 17.

D.4 Additional results (distributed computation)

In this section, we provide a proof-of-concept demonstrating the suitability of D&C-SMC to distributed computing environments (supplementing the discussion in Section 5.3 of the main paper).

In this discussion, it is useful to emphasize the usual distinction between *parallel* architectures, where a single computer is composed of several cores sharing fast access to memory,

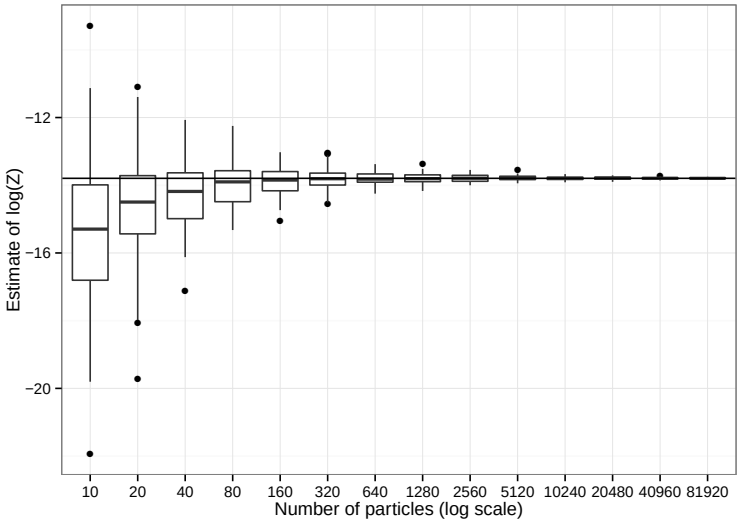

Figure 17: Estimates of the log normalization constant on a discrete model where the true value can be computed (horizontal line). These results are used as a validation of our computer implementation.

and *distributed* architectures, where several computers (nodes) are connected by a relatively slower network.

Parallelizing SMC is typically done in the proposal step, at the granularity of particles: when sampling from q or reweighting is expensive, or both, it becomes advantageous to parallelize these operations. However, the resampling step requires frequent communication, making this strategy less attractive in a distributed setting (but see for example Bolic et al. (2005); Jun et al. (2012); Vergé et al. (2015); Lee and Whiteley (2014); Whiteley et al. (2015) for alternatives).

In contrast, distributed computation in D&C-SMC can be scheduled to incur low communication costs. The main idea is to split the work at the granularity of populations, instead of the more standard particle granularity. In the following, we describe a simple strategy to decrease communication costs when the D&C-SMC recursion is computed in parallel at several locations of the tree $t \in T$. To simplify the exposition, we assume that the tree is binary.

To describe our distribution strategy, we introduce the notion of the *depth* of a vertex in the tree $t \in T$. We set the depth to be zero at the root r, $\text{depth}(r) = 0$, and recursively, for each vertex v with parent v', $\text{depth}(v) = \text{depth}(v') + 1$. We consider the largest depth d such that the number of vertices at that depth is greater or equal to the number c of compute nodes available. For each vertex v such that $\text{depth}(v) = d$, we assign the computation of all the D&C-SMC recursions for the vertices under v to the same computer.

With this architecture, communication is only required at edges of the tree T connecting

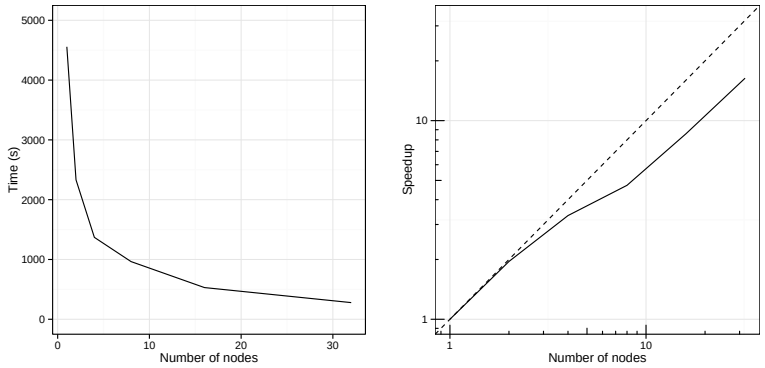

Figure 18: (a) Wall-clock times for the distributed D&C-SMC algorithm. (b) Speedup for the distributed D&C-SMC algorithm (log scale).

nodes above depth d. This implies that the number of particles transmitted over the network will be $O(cN)$. In contrast, naively distributing using a particle granularity would require $O(c|T|N)$ particle transmissions. Moreover, the population granularity strategy can be done in a completely decentralized fashion.

Based on this method, we have implemented an open source library for performing distributed, decentralized D&C-SMC computation. Thanks to its decentralized nature, the package is simple to use: nodes discover each other automatically and only require a rough estimate on the number of nodes available, to specify the value d discussed above. However, the algorithm can be made fault tolerant, in the sense that if some of the compute nodes fail, the computation will still be completed.

We applied this algorithm to the New York Mathematics Test dataset. We varied the number of active nodes in the cluster in $\{1, 2, 4, 8, 16, 32\}$, using 100 000 particles in all cases. The output of the distributed algorithm is identical in all cases, so it suffices to compare wall-clock times. The compute nodes consist in Xeon X5650 2.66GHz processors connected by a non-blocking Infiniband 4X QDR network. We show the results in Figure 18.

Each node used a single thread for simplicity, but note that combining parallelism and distribution can be done by parallelizing either at the level of particles, or populations with depths greater than d.

E Using and extending our code base

We provide here an overview of the implementation used in Sections 5.2 and 5.3 of the main paper. The goal is to help replicate experiments, and to show how our code base can be used to approach other models. The code is available at
`https://github.com/alexandrebouchard/divide-and-conquer-smc`. Note that the file

`README.md` in this repository might be more up to date at the time of reading. It also contains more detailed instructions on installing and building the software.

The code base actually contains two implementations of the basic D&C-SIR method:

1. One implementation has a specific model hard coded in it, namely the hierarchical model with a binomial emission model used in Section 5.2 of the main paper.

2. A second implementation offers convenient interfaces to apply the algorithm to other models. This second implementation also offers distributed and parallel computing functionalities. This second implementation is used in Section 5.3 of the main paper.

E.1 Running D&C-SMC on the binary emission hierarchical model (implementation 1)

- Checked out the repository and build the code using `gradle installApp`.
- Download and prepare the data by typing `scripts/prepare-data.sh` from the root of the repository (this requires the command wget). This writes the preprocessed data in `data/preprocessedNYSData.csv`.
- Run the software using `scripts/run-simple.sh -inputData data/preprocessedNYSData.csv -nParticles 1000`.
- Note: for the plots to work, you need to have R in your PATH variable (more precisely, Rscript).
- Various output files are written in `results/latest/`

The main interest of this implementation is for replicating the results in Section 5.2 of the main paper. To do so, see the following separate public repository, `https://github.com/alexandrebouchard/divide-and-conquer-smc-experiments` which contains the exact sets of options used to run the experiments as well as the plotting scripts. To extend D&C-SMC to other models, use the second implementation instead, described next.

E.2 Running parallel and distributed D&C-SMC on the binary emission hierarchical model (implementation 2)

- Checked out the repository and build the code using `gradle installApp`.

- Download and prepare the data by typing `scripts/prepare-data.sh` from the root of the repository (this requires the command wget). This writes the preprocessed data in `data/preprocessedNYSData.csv`.

- Run the software using
`scripts/run-distributed.sh -dataFile data/preprocessedNYSData.csv`

E.2.1 Distributed and parallel computing options

To run in a distributed fashion, simply issue the same command line on different machines. The machines will discover each other (via multicast) and distribute the work dynamically. You can see in the output a variable `nWorkers=[integer]` showing the number of nodes cooperating. For testing purpose, you can do this on one machine by opening two terminals (however, there is a better way to get multi-threading, see below).

The machines will only cooperate if all the command line options match exactly. You can check for example that running
`scripts/run-distributed.sh -dataFile data/preprocessedNYSData.csv -nParticles 2000` will not cooperate with
`scripts/run-distributed.sh -dataFile data/preprocessedNYSData.csv -nParticles 1000`. You can also force groups of machines to avoid cooperation by using the `-clusterSubGroup [integer]`

To use multiple threads within one machine, use `-nThreadsPerNode [integer]`. This can be used in conjunction with a distributed computation, or without.

E.2.2 Additional options

If you want the machines to wait each other, use `-minimumNumberOfClusterMembersToStart [integer]`. This will cause machines to wait to start until this number of machines is gathered or until `maximumTimeToWaitInMinutes` minutes has elapsed. Whether these options are used or not, machines can always join at any point in the execution of the algorithm. The waiting options were used for wall-clock running time analysis purpose. In most cases, these options can be ignored.

When the number of nodes in the D&C-SMC tree is much larger than the number of threads times the number of nodes, it is useful to coarsen the granularity of the subtasks. To do so, you can use `-maximumDistributionDepth [integer]`, which alters the way subtasks are created as follows: instead of the basic initial tasks being the leaves of the tree, they are the subtrees rooted at a depth from the root given by `maximumDistributionDepth`. The subtrees under are computed serially within each node/thread.

E.3 Using parallel and distributed D&C-SMC with your model

First, build a class to encode each individual particle. The only requirement is that it should implement `Serializable`.

Second, build a class to encode nodes in the tree. The only requirement is that you should override hashcode and equals to ensure that each node in the tree be unique (i.e. not `equals(...)` with any other node, as they will be inserted in a hashtable), and reproducible (i.e. the default implementation of hashcode will depend on the memory location and will be different from machine to machine). A reasonable default implementation is in `prototype.Node`.

Finally, the main step consists in providing code that proposes, i.e. merges sub-populations. This is done by creating a class implementing `dc.DCProposal`. This class will be responsible for both proposing, and providing a *log* weight update for the proposal (i.e. taking care of computing the ratio in step 2(c) of Algorithm 2).

See `README.md` in the repository for a step by step example on a simple model.

References

Andrieu, C., Doucet, A., and Holenstein, R. (2010). Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B*, 72(3):269–342.

Bolic, M., Djuric, P. M., and Hong, S. (2005). Resampling algorithms and architectures for distributed particle filters. *IEEE Transactions on Signal Processing*, 53(7):2442–2450.

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

Geweke, J. (2004). Getting it right. *Journal of the American Statistical Association*, 99(467):799–804.

Hoffman, M. and Gelman, A. (2012). The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*.

Jun, S., Wang, L., and Bouchard-Côté, A. (2012). Entangled Monte Carlo. In *Advances in Neural Information Processing Systems 25 (NIPS)*, volume 25, pages 2735–2743.

Lee, A. and Whiteley, N. (2014). Forest resampling for distributed sequential Monte Carlo. arXiv.org, arXiv:1406.6010.

Neal, R. M. (2011). MCMC using Hamiltonian dynamics. In Brooks, S., Gelman, A., Jones, G. L., and Meng, X.-L., editors, *Handbook of Markov Chain Monte Carlo*, pages 113–162. Chapman & Hall/CRC.

Vergé, C., Dubarry, C., Del Moral, P., and Moulines, E. (2015). On parallel implementation of sequential monte carlo methods: the island particle model. *Statistics and Computing*, 25(2):243–260.

Whiteley, N., Lee, A., and Heine, K. (2015). On the role of interaction in sequential Monte Carlo algorithms. *Bernoulli*. In press.