IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 5, 2020, accepted August 18, 2020, date of publication August 24, 2020, date of current version September 9, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3019226

New Branching Filters With Explicit Negative Dependence

MICHAEL A. KOURITZIN[1], ANNE MACKAY[2], AND NICOLAS VELLONE-SCOTT[2]
[1]Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, AB T6G 2G1, Canada
[2]Department of Mathematics, UQAM, Montreal, QC H2X 3Y7, Canada

Corresponding author: Anne MacKay (mackay.anne@uqam.ca)

This work was supported in part by the NSERC Discovery Grant 04869 and Grant 203089, and in part by the FRQNT Research Support for New Academics under Grant 253180.

ABSTRACT Particle filters are used to solve nonlinear filtering problems. We focus on the sampling step of a particle filter and present new algorithms that introduce explicit negative dependence between the number of particles reassigned at each location, with the goal of improving the performance of the filtering algorithm. We review partial and complete sampling in the context of both interacting and branching filters, that is, when the number of particles stays constant through all steps and when it does not. In particular, we use the quick simulation field algorithm to reproduce the variance structure induced by the minimal variance filter and create a new filtering algorithm. A numerical example is used to assess the performance of the new algorithms.

INDEX TERMS Nonlinear filtering, sequential Monte Carlo, Bernoulli sampling, interacting filters, branching filters.

I. INTRODUCTION

Noisy, stochastic phenomena are ubiquitous in nature, and attempts to understand such phenomena are diverse and span disciplines. Yet, the task is difficult because observations are often incomplete with abundant random noise. Nonlinear filtering is one popular way of modeling such phenomena that accounts for both of these factors. Often, we represent an original signal by a sequence of random variables $(X_t)_{t\geq 0}$, and its observations by $(Y_t)_{t\geq 0}$, which are modeled as some function of $(X_t)_{t\geq 0}$, corrupted by noise: $Y_t = h(X_{t-1}) + V_t$, where V_t is a noise term. The goal is to obtain an approximation for the best least-squares estimator, $\mathrm{E}[f(X_t) \mid Y_t, \ldots, Y_1]$, where f is a measurable, bounded function. Sequential Monte Carlo (SMC) methods are an important class of algorithms that achieve this goal, relying heavily on a sampling step in the process. This step is the focus of the present article.

Nonlinear filtering has a number of applications in different domains. Since the 60's, it has been a staple of the defense, search and rescue, and aerospace industries. Later, it also became an essential tool in image processing. In mathematical finance, it is widely used to calibrate models with unobservable factors or variables (volatilities, credit risk, instantaneous interest rates) using only observable quantities such as asset prices (see for example [14], [17], [18], [23]). Nonlinear filtering also finds applications in fields as diverse as biology [31], aeronautics [39] and epidemiology [13].

The associate editor coordinating the review of this manuscript and approving it for publication was Liang Hu.

A. MOTIVATION

We adopt the treatment of nonlinear filtering in [11] and [20]. Let $(\Omega, \mathcal{F}, P)$ be a probability space. Let a non-observable signal be represented by $(X_t)_{t\in\mathbb{N}} : \Omega \to E$, where E is some complete, separable metric space. In this case, t could represent discrete time steps. Typically, it is assumed that X is a Markov process with a specified initial distribution $p(x_0)$ and evolution equation that, given X_t, returns X_{t+1} up to some noise term. Our indirect observations of the non-observable signal are of the form

$$Y_t = h(X_{t-1}) + V_t,$$

where $h : E \to \mathbb{R}^d$ is a known, measurable function, and the V_t are independent, random vectors with a common, strictly positive, bounded density g that is independent of $(X_t)_{t\geq 0}$. For simplicity, we also define

$$\mathbf{Y}_t := (Y_1, \ldots, Y_t),$$
$$\mathbf{X}_t := (X_1, \ldots, X_t).$$

Given this setting, we try to estimate, track and predict the signal based on distorted, corrupted partial observations. In order to recall the concepts underlying particle filtering, both the signal and the observations will be seen as random variables; the values of the observations will only be fixed when algorithms serving to compute estimates are presented. Throughout the paper, random variables are denoted by uppercase letters, while we use lowercase letters to represent realizations thereof.

Our goal is to develop an estimator for the conditional probabilities $P(X_t \in A|\mathbf{Y}_t)$ for all Borel sets A, or equivalently, for the conditional expectations $\mathrm{E}[f(X_t)|\mathbf{Y}_t]$, the best least-squares approximation of $f(X_t)$ given all observations thus far, for all $f : E \to \mathbb{R}$ that are bounded, measurable functions. Clearly, it would be convenient if we could directly and easily sample from the posterior distribution $p(\mathbf{X}_t|\mathbf{Y}_t)$, but the computational complexity of such a method is usually too great [11].

Sequential Monte Carlo (SMC) methods comprise an important class of algorithms that approximate the conditional probabilities $P(X_t \in A \mid \mathbf{Y}_t)$, or equivalently, $\mathrm{E}[f(X_t)|\mathbf{Y}_t]$. This is done by sequentially incorporating the observations Y_t in the computation of the filters via the Bayes formula and importance sampling. The system of simulated particles that approximates the conditional distribution of the signal is updated with each new observation (see for example Chapter 10 of [1] for more details on particle filters).

In our framework, the importance distribution used to generate the particle system is a probability measure Q, under which we assume that the signal and observation process $\{(X_t, Y_t), t = 0, 1, \ldots\}$ has the same distribution as the signal and noise process $\{(X_t, V_t), t = 0, 1, \ldots\}$ does under P. It follows that under Q, the importance density of the observations is g and the observations and the signal are independent. The information given by the observations is incorporated into the likelihood process $\{L_t, t = 0, 1, \ldots\}$ defined by

$$L_t = \prod_{j=1}^{t} \alpha_j(X_{j-1}),$$

with

$$\alpha_j(x) = \frac{g(Y_j - h(x))}{g(Y_j)},$$

for $t = 1, 2, \ldots$ and $L_0 = 1$, so that $L_t = L_{t-1}\alpha_t(X_{t-1})$. The likelihood process can be used to obtain the probability measure P from Q using Girsanov's theorem. It follows that the unnormalized filters of interest are given by $\mathrm{E}^Q[L_t f(X_t)|\mathbf{Y}_t]$, where $\mathrm{E}^Q[\cdot]$ denotes the expectation under the Q measure, which is approximated using the generated particle system.

Each time step of a SMC algorithm is comprised of two parts. The first one is the mutation step, where the particles are evolved using the transition density under the importance distribution Q; this allows for the computation of the approximated filter. After a new observation is incorporated, the second step involves sampling from the empirical distribution of the particles in order to avoid weight degeneracy, which can negatively impact the performance of the algorithm.

The focus of this article is the aforementioned sampling step, where low-weight particles are eliminated and replaced by average weight ones. This step is necessary because the variance of the weights increases over time. In practice, as the particle system develops, there tend to be a few particles with very high weights and a lot of particles with very low weights. This leads to either wasted computational power, when many low-weight particles are propagated, or to poor estimates of the conditional expectation.

The sampling step is an important factor in the speed of SMC algorithms, and it can easily become performance-limiting when it is poorly conceived. Aside from the actual number of operations in the algorithm itself, a sampling method can also affect performance by influencing the number of particles propagated (this is the case in so-called branching particle filters). If the number of particles in the system grows too large, then more operations will have to be performed.

An ideal sampling method should even weights without introducing excess noise nor computations. Various existing sampling algorithms seek to strike such a balance between variance reduction and execution speed. The new algorithms introduced in this article were built with such a goal in mind.

B. PREVIOUS WORK

The bootstrap particle filter [15] was an important step in the development of fast SMC methods. Numerous improvements were made over the years to the sampling step of the bootstrap filter. Residual sampling is introduced in [29] to reduce sampling noise and execution time. Stratified sampling is introduced in [19] to reduce the variance of the uniform random variables involved in sampling. Combined sampling, a combination of residual and stratified sampling, is discussed in [10]. The minimal variance algorithm, which we discuss further in this work, is presented in [4]. All these sampling methods keep the number of particles in the system constant throughout the time steps. In this work, we will refer to algorithms based on such sampling methods as *interacting filters*. See [5] for a survey for convergence results and [8] for a recent exposition on interacting particle filters.

Other sampling algorithms return a random number of particles, and when they are incorporated in an SMC algorithm, the number of particles can vary at each time step. Such algorithms are often referred to as *branching filters*. Previous work on branching particle filters includes [2], [6], [9], [20]. Contrary to beliefs about particle instability (i.e., particle numbers either exploding or going to zero), [20] shows that a branching particle filter can be stable if the correct normalizing constant is used. Still, the stability in particle numbers, and indirectly the performance of the algorithm, is affected by the variance of the number of branches (or offspring) assigned to each particle in the sampling step, and by the variance of the total number of particles at each step. Adding negative

dependence between the number of offspring assigned to each particle can help reduce this latter variance.

Intuitively, two random variables that are negatively dependent tend to move in opposite directions; if one takes a small value, the probability that the other one takes a large value is increased. This idea is closely related to negative lower orthant dependence, which occurs when two random variables X_1 and X_2 satisfy $P(X_1 \leq x_1, X_2 \leq x_2) < P(X_1 \leq x_1)$ $P(X_2 \leq x_2)$ for all x_1, x_2. An imperfect but widely used measure of dependence is the coefficient of correlation, which we use later in this article. For more details on negative dependence, we refer the reader to Chapter 5 of [32]. In our context, inducing negative dependence between the number of offspring at adjacent locations ensures that the particles are more evenly distributed among all locations. To optimize this effect, it is desirable to maximize the negative correlation induced between the different locations. The concept of maximal negative dependence is well studied in two dimensions; a pair of countermonotonic random variable attains such a maximum. We say that two random variables (X_1, X_2) are *countermonotonic* if there exist a random variable Z and real functions f and g, with f increasing and g decreasing, such that (X_1, X_2) has the same distribution as $(f(Z), g(Z))$ (see for example [33]). The concept of extreme negative dependence is not clearly defined for vectors of dimension higher than 2. In this general case, a type of extreme negative dependence structure for random vectors is joint mixability. We say that a random vector $X = (X_1, \ldots, X_d)$ is *jointly mixable* (see [38]) if $P(\sum_{i=1}^{d} X_i = K) = 1$ for some $K \in \mathbb{R}$. In this article, we exploit both concepts to identify and induce negative dependence between the number of offspring redistributed to each particle. In [20], negative dependence was produced by stratification using the Yates-Fisher shuffle or partial shuffle consisting of some exchanges.

A key step in the new algorithms presented in this work is the generation of correlated discrete random fields. Indeed, we wish to generate a sequence of random variables with specified marginal probability mass functions and a given covariance matrix. The fact that our random variables must be discrete precludes the use of the variety of popular methods to generate Gaussian random variables (for example, [30], [34], [35], [37]). The methods that best fulfilled our criteria were the quick simulation fields (QSF) method of [26] and the list sequential sampling of [3]. These sampling methods are incorporated in specific branching algorithms in order to speed them up while keeping some control on the variance of the total number of particles in the system.

C. UNIQUE CONTRIBUTIONS

Many of the refinements to the bootstrap algorithm involve the use of residues, the use of negative correlation in the sampling step or the use of partial sampling. All classes of improvements reduce sampling noise and improve performance. However, the best ways to implement the negative dependence and partial sampling are unknown. In this work, we create new sampling algorithms by considering novel methods to induce negative dependence between the number of particles reassigned to each particle location in the sampling step. We also show that the use of partial sampling can be implemented in the new filters, as well as in existing interacting and branching filters, for performance improvement. This is explored in an extensive numerical experiment, through which we also show that our branching algorithms maintain respectable particle control.

Although our work is somewhat similar to [10] and [27], those two works do not consider branching filters, nor QSF and the list sequential method of [3] as ways to generate negatively dependent particles.

The paper is divided as follows. In Section II, we compare complete and partial sampling, and discuss the implementation of interacting and branching algorithms in each context. Section III presents a review of existing methods as further motivation for our new algorithms, which are also outlined in this section. Numerical experiments are presented in Section IV, and concluding remarks are in Section V.

II. COMPLETE AND PARTIAL SAMPLING

We denote by $\boldsymbol{N}_t \in \mathbb{N}$ the number of particles in the filter at the end of the t-th time step, after sampling; $\boldsymbol{N}_t$ may vary across time steps, for example when branching filters are used (this is further explained below). Let $(x_{i,t})_{i=1}^{\boldsymbol{N}_{t-1}}$ be the collection of $\boldsymbol{N}_{t-1}$ independent samples from $p(\cdot \mid x_{i,t-1})$, the transition density of X, generated at the beginning of the t-th time step. At $t = 0$, we consider $(x_{i,0})_{i=1}^{\boldsymbol{N}_0}$ where each $x_{i,0}$ is sampled from p_0.

Each particle is assigned a likelihood $\ell_{i,t}$ that weighs the particle based on how well it approximates the original signal according to the observations. This likelihood is given by $\ell_{i,0} = 1$ and

$$\ell_{i,t} := \alpha_t(x_{i,t-1})\ell_{i,t-1}$$

for $t > 0$, with

$$\alpha_t(x_{i,t-1}) := \frac{g(y_t - h(x_{i,t-1}))}{g(y_t)},$$

where g is the common, strictly positive, bounded density of V_t.

The sampling step in particle filters is often necessary to avoid weight degeneracy, that is, to avoid ending up with a few particles having extremely high weight compared to the rest. In most common sampling methods, all particles are redistributed (interacting filters) or branched (branching filters) randomly. The probability that a particle appears in the new sample is proportional to its weight $\ell_{i,t}$.

Remark 1: Technically, we consider one-step predictor filters by incorporating the observation equation $Y_t = h(X_{t-1}) + V_t$ instead of the more commonly used tracking filter observation model $Y_t = h(X_t) + V_t$. Our setting describes the important situation where one must estimate the signal prior to receiving the current observation compared to the usual assumption that you may use the current observation in the signal estimate. All the algorithms and

analysis given herein transfer seamlessly to the more common tracking filter setting by simply replacing $\alpha_t(X_{t-1})$ with $\alpha_t(X_t)$ and $\alpha_t(x_{i,t-1})$ with $\alpha_t(x_{i,t})$ everywhere. The algorithms would stay the same except one has to evolve the particles prior to calculating the weights, i.e. switch steps 5 and 6 in Algorithm 1, to handle this α_t argument change. There are two reasons why the one-step predictor was considered: i) It is a very important setting; ii) It is consistent with the pathspace convergence and other empirical results in [21] and [20] respectively. The pathspace convergence results obtained in [21] would have been, at least notationally, more difficult if the tracking observation model had been used.

In the rest of this section, we compare the situation where all particles are resampled at each time step to the one where only a subset of the particles is considered for sampling.

A. COMPLETE SAMPLING

We use the term *complete sampling* to refer to algorithms in which all the particles are resampled. That is, the *sampling set* at time t, denoted by C_t, is $\{1, \ldots, \boldsymbol{N}_{t-1}\}$. We further define the normalized weights associated with $(x_{i,t-1})_{i=1}^{\boldsymbol{N}_{t-1}}$ by

$$a_{i,t} = \frac{\ell_{i,t}}{\sum_{i \in C_t} \ell_{i,t}} \tag{1}$$

for $i \in C_t$, so that $\sum_{i \in C_t} a_{i,t} = 1$.

Complete sampling can be performed via *interacting* or *branching* algorithms. In this work, interacting sampling procedures refer to algorithms that redistribute the same number of particles as there were in the original sample, such as the multinomial or stratified sampling algorithms of [10] or the minimal variance algorithm of [4]. Since the same type of sampler is used throughout the time steps, the number of particles used in the filter remains constant, that is, $\boldsymbol{N}_t = \boldsymbol{N}_0$ for all t.

We call branching samplers the algorithms in which each particle can be split into a random number of offspring, which are then used as the starting point for the next time step, without a guarantee that the number of particles remains constant across all time steps. In complete sampling algorithms, the average number of offsprings that each particle has is proportional to its normalized weight $a_{i,t}$. The main difference between interacting and branching samplers is the constant particle requirement for interacting algorithms and the flexibility in particle numbers for branching samplers.

Filters that make use of branching samplers (herein called branching filters) have been criticized for particle instability. However, by forcing the expected number of particles to be equal to the initial number of particles $\boldsymbol{N}_0$ at each time step, the algorithms of [20] show increased particle stability. In this work, the branching samplers we propose are based on the same idea, but we add explicit negative dependence between the branching decisions from one particle to the other to further control the variation in the number of particles.

B. PARTIAL SAMPLING

As stated previously, it may not be necessary to sample all particles at every time step. Indeed, results in [20] show that it may be advantageous to leave alone particles whose weights are neither too big nor too small as we avoid introducing excess sampling noise. If a weight is too small, we would like to eliminate that particle; if the weight is large, that particle should likely have multiple offspring. All of this must be done without introducing bias. There are other possible partial sampling schemes [12], but here, for simplicity, we use the following one from [20].

We define the sampling set C_t as

$$\left\{ i \in \{1 \ldots, \boldsymbol{N}_{t-1}\} : \ell_{i,t} \notin (r^{-1}\bar{\ell}_t, r\bar{\ell}_t) \right\}, \tag{2}$$

for some $r \geq 1$ fixed and where

$$\bar{\ell}_t := \frac{1}{\boldsymbol{N}_0} \sum_{i=1}^{\boldsymbol{N}_{t-1}} \ell_{i,t}.$$

In other words, we resample the particles whose likelihood is smaller than $r^{-1}\bar{\ell}_t$ or larger than $r\bar{\ell}_t$. That is, if its likelihood falls outside of a given interval around the $\boldsymbol{N}_0$-average $\bar{\ell}_t$, it gets resampled. Using the $\boldsymbol{N}_0$-average rather than dividing the sum of the likelihoods by $\boldsymbol{N}_{t-1}$ helps to keep the number of particles stable.

Interacting and branching samplers are applied differently to the partial sampling set C_t. Indeed, interacting samplers use the normalized weights associated with the sampling set as defined by (1) and randomly redistribute the number of particles in C_t to the locations of the resampled particles. The normalized weights are such that $\sum_{i \in C_t} a_{i,t} = 1$, that is, the sum of the normalized weights associated to the resampled particles should sum to 1. We also observe that the normalized weights used in interacting sampler algorithms depend on C_t. In particular, the normalized weight $a_{i,t}$ for a given particle will change if complete, rather than partial, sampling is used (given that the sampling set is affected), or if the parameter r is modified.

In contrast, even when only a subset of particles are branched, branching samplers consider the weight of the particle as a proportion of the sum of the weights of all particles, resampled or not. To each resampled particle, branching samplers assign a number of offspring based on the ratio $\ell_{i,t}/\bar{\ell}_t$. This is true whether the sampling set contains all the particles or not; the composition of the sampling set does not change the distribution of the number of offspring. In other words, since

$$\frac{\ell_{i,t}}{\bar{\ell}_t} = \frac{\ell_{i,t}}{\sum_{i=1}^{\boldsymbol{N}_{t-1}} \ell_{i,t}} \times \boldsymbol{N}_0,$$

the number of offsprings assigned to each particle is proportional to the weights normalized by the sum of *all* the particles' weights, not only those in the sampling set.

This subtle but important difference between interacting and branching samplers affect their implementation in the general filtering procedure. We conclude this section with

two important remarks that summarize the differences and similarities between the different types of algorithms.

Remark 2: In all cases, except for partial branching samplers, the weights normalized over the sampling set, $(a_i)_{i\in C_t}$, are used for sampling. This is indeed also the case in complete branching samplers; when the sampling set contains all the particles, we observe that for all $i \in \{1, \dots, N_{t-1}\}$,

$$\frac{\ell_{i,t}}{\bar{\ell}_t} = \frac{N_0 \ell_{i,t}}{\sum_{i=1}^{N_{t-1}} \ell_{i,t}} = N_0 a_{i,t}.$$

This similarity between interacting and branching complete samplers is further explained in Section III.

Remark 3: For both interacting and branching algorithms, complete sampling can be seen as a special case of partial sampling (using $r = 1$), so one only needs to implement the partial sampling algorithm while ensuring that setting r to 1 is possible. However, our numerical results will show that complete sampling is rarely the best choice so $r > 1$ should generally be used.

C. IMPLEMENTATION

In light of the observations above, a general partial sampling algorithm is given by Algorithm 1 (below). The first part of the algorithm (lines 2 to 15) describes the mutation step and the treatment of the non-sampled particles; it is identical for interacting or branching filters. In the algorithm, and going forward, for $a \in \mathbb{R}$, we denote $\lfloor a \rfloor = \max\{z \in \mathbb{Z} : z \le a\}$ and $\{a\} = a - \lfloor a \rfloor$.

The "if" statement on line 16 splits the algorithm into two. Lines 16 to 21 refer to the use of an interacting sampler. Line 18 is intentionally left vague, as it should be replaced with a specific sampler. In the next sections, we discuss sampling algorithms in further details. Interacting samplers will be presented as subroutines that take as input the particles and their weights, $(\hat{x}_{i,t}, \hat{\ell}_{i,t})_{i\in C_t}$ and return a vector of the same length containing the sampled values.

Lines 22 to 32 are used for branching samplers. Line 23 can be replaced by one of the branching algorithms presented in the next section. Herein, branching samplers take as input a vector of probabilities $(\{\ell_{i,t}/\bar{\ell}_t\})_{i\in C_t}$ corresponding to each of the particles in C_t, and return a vector of Bernoulli random variables $(\rho_{i,t})_{i\in C_t}$ of the same length, where each $\rho_{i,t}$ takes the value 0 or 1, that is $\rho_{i,t} \in \{0, 1\}$. The number of offspring assigned to location i_t (and therefore given value $\hat{x}_{i,t}$) is given by $\lfloor \ell_{i,t}/\bar{\ell}_t \rfloor + \rho_{i,t}$, and the weights at these locations is reset to the N_0-average. These steps correspond to lines 26 to 29. In the next sections, we detail the branching algorithms that can replace line 23.

Algorithm 1 General SMC Algorithm

1: **procedure** SMC(N_0, T, r)
▷ $T \in \mathbb{N}$ is the number of time steps
▷ Initialize particles
2: $\ell_{i,0} = 1, x_{i,0} \sim p_0$, for all $i \in \{1, \cdots, N_0\}$
3: **for** $t \in \{1, \dots, T\}$ **do**
4: **for** $i \in \{1, \dots, N_{t-1}\}$ **do**
▷ Calculate the weights
5: $\hat{\ell}_{i,t} = \ell_{i,t-1}\alpha_t(x_{i,t-1})$
▷ Evolve the particles
6: $\hat{x}_{i,t} \sim p(X_t | X_{t-1} = x_{i,t-1})$
7: **end for**
▷ Estimate the conditional expectation
8: $\sum_{i=1}^{N_{t-1}} \hat{\ell}_{i,t} f(\hat{x}_{i,t}) / \sum_{i=1}^{N_{t-1}} \hat{\ell}_{i,t}$
▷ Get N_0-average
9: $\bar{\ell}_t = \frac{1}{N_0} \sum_{i=1}^{N_{t-1}} \hat{\ell}_{i,t}$
▷ Determine the sampling set
10: $C_t = \{i : \hat{\ell}_{i,t} \notin (\frac{1}{r}\bar{\ell}_t, r\bar{\ell}_t)\}$
▷ Count the particles for next iteration
11: $N_t = 0$
12: **for** $i \notin C_t$ **do**
▷ Non resampled particles
13: $N_t = N_t + 1$
14: $(x_{N_t,t}, \ell_{N_t,t}) = (\hat{x}_{i,t}, \hat{\ell}_{i,t})$
15: **end for**
16: **if** Interacting Filter **then**
17: Get normalized weights $(a_{i,t})_{i\in C_t}$
18: Sample $(x_{i,t})_{i=N_t+1}^{N_{t-1}}$ from $(\hat{x}_{i,t})_{i\in C_t}$
19: $\ell_{i,t} = \bar{\ell}_t$ for $i \in \{N_t + 1, \dots, N_{t-1}\}$
20: $N_t = N_{t-1}$
21: **end if**
22: **if** Branching Filter **then**
23: Create mean $(\{\ell_{i,t}/\bar{\ell}_t\})_{i\in C_t}$ Bernoullis $(\rho_{i,t})_{i\in C_t}$
24: **for** $i \in C_t$ **do**
25: $M_{i,t} = \lfloor \ell_{i,t}/\bar{\ell}_t \rfloor + \rho_{i,t}$
26: **for** $j = 1, \dots, M_{i,t}$ **do**
▷ Propagate offspring
27: $x_{N_t+j,t} = \hat{x}_{i,t}$
28: $\ell_{N_t+j,t} = \bar{\ell}_t$
29: **end for**
30: $N_t = N_t + M_{i,t}$
▷ Update particle number
31: **end for**
32: **end if**
33: **end for**
34: **end procedure**

III. NEW SAMPLING ALGORITHMS

In its simplest form, the sampling part of the filtering procedure consists essentially in sampling a fixed number n of independent random variables from the categorical (empirical) distribution obtained in the mutation step, or a subset thereof. In this section, we focus on a single time step, so we drop the reference to time t from the notation. We also let n denote the cardinality of the sampling set at the time step of interest. The n values $(x_i)_{i=1}^n$ have associated likelihoods $(\ell_i)_{i=1}^n$, from which normalized weights can be obtained as in (1), so that

$$a_i = \frac{\ell_i}{\sum_{i=1}^n \ell_i},$$

for $i \in \{1, \dots, n\}$.

A. REVIEW OF (SOME) EXISTING ALGORITHMS

In this section, we focus on complete sampling (i.e. we assume that the sampling set contains all the particles), since it better highlights the common theory behind interacting and branching samplers. A discussion on the application of branching algorithms in partial sampling is provided at the end of the section.

1) MULTINOMIAL BOOTSTRAP

Multinomial bootstrap (see [15]) is arguably the simplest sampling algorithm, and refers to the simulation of n independent random variables, each one taking the value x_i with probability a_i, $i = 1, \dots, n$, so that each random variable is drawn from a categorical distribution. The result of these n independent draws can be expressed as a vector $M = (M_1, \dots, M_n)$, where each M_i takes a value in $\{0, 1, \dots, n\}$ and represents the number of times the value x_i is drawn, so that $\sum_{i=1}^{n} M_i = n$.

The vector M has a multinomial distribution with parameters $(n, a_1, \dots, a_n)$, so the random variables $M_1, \dots, M_n$ are clearly not independent. Indeed, it can easily be shown that

$$\mathrm{Cov}(M_i, M_j) = -n a_i a_j$$

for $i, j \in \{1, \dots, n\}$, $i \neq j$. In other words, the simplest multinomial bootstrap procedure results in negatively correlated numbers of offspring at any two different locations i and j.

2) REDUCING SAMPLING NOISE

To improve the performance of the particle filter, it is desirable to control the variance of two quantities (among others):

- The number of particles redistributed to each each individual site, M_i; and
- The total number of particles reassigned, $N =: \sum_{i=1}^{n} M_i$.

While reducing the variance of each M_i also reduces the variance of N, the opposite is not necessarily true. For example, as explained above, multinomial bootstrap yields a constant total number of offspring n, thus attaining the smallest possible variance for N, but does not control the variance of each M_i. We remark that in this case, the random vector M is *jointly mixable*, since $P\left(\sum_{i=1}^{n} M_i = n\right) = 1$. As mentioned above, joint mixability is a type of extreme negative dependence.

Even when $P(N = n) = 1$, as in interacting filters, reducing the variance of each M_i while retaining unbiasedness is desirable in order to improve the performance of the filtering procedure (see [1] for more details). Many well-known interacting algorithms, such as residual sampling [29], stratified sampling [19], as well as combined (interacting) sampling [10] reduce the variance of the individual number of particles assigned to each site. Our implementation of these algorithms is presented for reference in the appendix.

If one focuses only on decreasing the variance of the M_i's and relaxes the constraint that N is almost surely constant, then one can attain the lowest possible variance for each M_i. Indeed, in order to keep the filtering procedure free of bias, sampling must be done so that $\mathrm{E}[M_i] = n a_i$ (interacting samplers) or $\mathrm{E}[M_i] = \ell_i / \bar{\ell}$ (branching samplers) for each i. If we define by $\mathcal{A}_a$ the set of integer-valued random variables with expectation $a \in \mathbb{R}$, it can be shown (see for example Exercise 9.1 of [1]) that the random variable $Y \in \mathcal{A}_a$ that attains the lowest possible variance is given by

$$Y = \begin{cases} \lfloor a \rfloor, & \text{with probability } 1 - \{a\} \\ \lfloor a \rfloor + 1, & \text{with probability } \{a\}, \end{cases}$$

and has variance $\{a\}(1 - \{a\})$. We recall that $\lfloor x \rfloor$ denotes the floor of x, that is $\lfloor x \rfloor =: \max\{z \in \mathbb{Z} : z \leq x\}$, and $\{x\} =: x - \lfloor x \rfloor$, for $x \in \mathbb{R}^+$.

It follows that the lowest possible variance for the individual number of offspring at each location i is given by $\{n a_i\}(1 - \{n a_i\})$, and can be attained by letting

$$M_i = \lfloor n a_i \rfloor + \mathbb{1}_{\{U_i < \{n a_i\}\}}, \tag{3}$$

where $U_1, \dots, U_n$ are Uniform random variables on $[0, 1]$. The uniform random variables do not need to be independent to achieve this lower bound.

Branching-type procedures for sampling are based on this idea and attain the lowest possible variance for each M_i by allocating $\lfloor n a_i \rfloor$ offspring with probability $1 - \{n a_i\}$, and $\lfloor n a_i \rfloor + 1$ offspring with probability $\{n a_i\}$ at each location i. If this operation is performed independently at each location (for example using (3)), then $P(N = n) \neq 1$ and

$$\mathrm{Var}(N) = \sum_{i=1}^{n} \{n a_i\}(1 - \{n a_i\})$$

Unless $\{n a_i\} = 0$ for each i, $\mathrm{Var}(N)$ is strictly positive.

Remark 4: In our implementation of branching filters (see Algorithm 1), the number of particles observed at the beginning of a given step, $\boldsymbol{N}_t$ may differ from the initial number of particles $\boldsymbol{N}_0$. However, we use $\boldsymbol{N}_0$ to determine the number of offspring assigned to each particle; that is, we set $M_i = \lfloor \boldsymbol{N}_0\, a_i \rfloor + \mathbb{1}_{\{U_i < \{\boldsymbol{N}_0 a_i\}\}}$ to increase the stability of the number of particles through time. For simplicity, in this section, we assume $\boldsymbol{N}_t = \boldsymbol{N}_0$, but the discussion remains valid in the context of Algorithm 1, since it is possible to show that $E[\boldsymbol{N}_t] = \boldsymbol{N}_0\ \forall t$.

If, for each i, we let $M_i = \lfloor n a_i \rfloor + I_i$, where I_i has marginal Bernoulli distribution with mean $\{n a_i\}$, then the variance of N can be reduced by adding negative dependence between the I_i's. This is done in [20] via stratification. The resulting combined branching algorithm is presented for reference in the appendix. In Section III-C, we introduce new branching algorithms that explicitly induce negative correlation between the I_i's.

There exists a specific dependence structure for the vector $I = (I_i)_{i=1}^{n}$ (or equivalently, for the M) that allows both individual variances of each M_i and the variance of N to be minimal. It stems from the *minimal variance* algorithm introduced by [4] (see also [1]) and ensures that $\mathrm{Var}(M_i) = \{n a_i\}(1 - \{n a_i\})$ for each $i \in \{1, \dots, n\}$ and that $P(N = n) = 1$,

so that $\mathrm{Var}(N) = 0$.[1] [1] also explain that the resulting random vector minimizes $\mathrm{Var}\left(\sum_{i=j}^{n} M_i\right)$ and $\mathrm{Var}\left(\sum_{i=1}^{j} M_i\right)$ for all $j \in \{1, \ldots, n\}$, among all n-dimensional integer-valued vectors with expectation $(na_1, \ldots, na_n)$.

The minimal variance algorithm keeps track of the number of offspring left to distribute and compares it with its theoretical average, in order to assign the offspring in a way that minimizes the variance. The algorithm is implemented via embedded "if" statements.

Remark 5: The minimal variance algorithm induces a vector of jointly mixable Bernoulli random variables. Indeed, $\sum_{i=1}^{n} \lfloor na_i \rfloor$ offspring are distributed in a deterministic manner, since each location i receives $\lfloor na_i \rfloor$ offspring with probability 1. Since the algorithm ensures that $N = n$ almost surely, the number of offspring that are randomly re-distributed must be equal to $n - \sum_{i=1}^{n} \lfloor na_i \rfloor$. In other words, $\sum_{i=1}^{n} I_i = n - \sum_{i=1}^{n} \lfloor na_i \rfloor$ a.s., and therefore the vector I is jointly mixable.

Our first algorithm, presented in Section III-B, is inspired by the minimal variance algorithm. The use of "if" statements is replaced with the Quick Simulation Field (QSF) algorithm of [26], which allows quick simulation of a joint distribution with given marginals and correlation structure.

3) SPECIAL CASE: PARTIAL SAMPLING WITH BRANCHING ALGORITHMS

The discussion above does not apply directly to the case where branching samplers are used for partial sampling, since in this case, the sampling weights $\ell_i/\bar{\ell}$ do not sum to 1. Indeed, recall from Algorithm 1 that for $i \in C$, where C denotes the sampling set,

$$M_i = \lfloor \ell_i/\bar{\ell} \rfloor + \rho_i,$$

where ρ_i is a Bernoulli with mean $\{\ell_i/\bar{\ell}\}$. It follows that the expected value of the total number of offspring assigned to particles in the sampling set is

$$\mathrm{E}\left[\sum_{i \in C} M_i\right] = \frac{\sum_{i \in C} \ell_i}{\bar{\ell}}.$$

Recall that $\bar{\ell}$, the N_0-average, is calculated using *all* the particles at a given time step. It follows that the expected number of offspring is not necessarily equal to the number of particles in the sampling set, and so partial branching samplers cannot be included in the previous discussion. Nonetheless, branching algorithms developed using the ideas presented above can also be applied to partial sampling; this is demonstrated empirically in Section IV.

B. QSF-BASED MINIMAL VARIANCE ALGORITHM

The QSF algorithm (introduced in [24], see also [25], [26]) sequentially generates random variables with pre-specified marginal probability mass functions and covariance matrix, by modeling the problem as the simulation of random vertices on a graph with given edge weights. The vertices of the graph correspond to the random variables to simulate, while the edge weights correspond to the covariances between a pair of random variables/vertices. The full version of the algorithm uses auxiliary marginal distributions in order to enlarge the set of reproducible joint distributions. In this article, we use a simplified version of the algorithm to generate our random variables, which we recall here.

We consider a collection of n marginal probability mass function with finite support denoted by $\pi_i(\cdot)$, $i \in \{1, \ldots, n\}$. We let $\mu_i = \sum_{x_i \in \mathcal{X}_i} x_i \pi_i(x_i)$, where $\mathcal{X}_i$ denotes the support of $\pi_i(\cdot)$, and $s_{ii}^2 = \sum_{x_i \in \mathcal{X}_i} (x_i - \mu_i)^2 \pi_i(x_i)$, and introduce the following auxiliary functions:

$$g_i(x_i) = \frac{x_i - \mu_i}{s_{ii}^2}, \quad h(i) = \prod_{1 \le k \le i} \pi_k(x_k).$$

We also denote $\mathbf{X}_i = (X_1, \ldots, X_i)$ and $\mathbf{x_i} = (x_1, \ldots, x_i)$, $i \in \{1, \ldots, n\}$, with $\mathbf{X} = \mathbf{X}_n$. Following [26], we have that if $\{s_{ij}^2,\ i, j \in \{1, \ldots, n\}\}$ are numbers such that the right-hand side of

$$P(X_i = x_i | \mathbf{X_{i-1}} = \mathbf{x_{i-1}}) = \pi_i(x_i) \times \left[1 + \frac{g_i(x_i) h(i)}{P(\mathbf{X_{i-1}} = \mathbf{x_{i-1}})} \sum_{j=1}^{i-1} s_{ij}^2 g_j(x_j)\right], \quad (4)$$

is in $[0, 1]$, then the joint distribution of the random vector $\mathbf{X}$ obtained recursively using $P(X_1 = x_1) = \pi_1(x_1)$, (4) and

$$P(\mathbf{X} = \mathbf{x}) = \prod_{i=1}^{n} P(X_i = x_i | \mathbf{X_{i-1}} = \mathbf{x_{i-1}})$$

has marginal distributions $\pi_i(\cdot)$ and covariances $\mathrm{Cov}(X_i, X_j) = s_{ij}^2$, $i, j \in \{1, \ldots, n\}$.

The goal of our new algorithm is to reproduce the marginal distributions of M_i and $M_{1:i-1} =: \sum_{k=1}^{i-1} M_k$, as well as the correlations between the M_i's induced by the minimal variance algorithm. That is, following Section 9.2 of [1], we want

$$P(M_i = m_i) = \begin{cases} 1 - \{na_i\}, & m_i = \lfloor na_i \rfloor \\ \{na_i\}, & m_i = \lfloor na_i \rfloor + 1 \end{cases} \quad (5)$$

and

$$P(M_{1:i} = m_{1:i}) = \begin{cases} 1 - \{na_{1:i}\}, & m_{1:i} = \lfloor na_{1:i} \rfloor \\ \{na_{1:i}\}, & m_{1:i} = \lfloor na_{1:i} \rfloor + 1, \end{cases} \quad (6)$$

where $a_{1:i} = \sum_{k=1}^{i} a_k$. We also want to have the same covariances between $M_{1:i-1} =: \sum_{k=1}^{i-1} M_k$ and M_i, $i \in \{2, \ldots, n\}$ as those that result from the application of the minimal variance sampling algorithm, given in the following proposition.

Proposition 1: Let M_i denote the number of offspring assigned to the location i and denote $M_{i:j} = \sum_{k=i}^{j} M_k$ for $i, j \in \{1, \ldots, n\}$ with $i < j$. When the vector $M = (M_1, \ldots, M_n)$ is generated with the minimal variance algorithm, the covariance between $M_{1:i-1}$ and M_i is given by

$$Cov\,(M_{1:i-1}, M_i) = -(1 - \{na_{i:n}\})\{na_i\}, \quad (7)$$

[1] The pseudo-code for the specific version of the algorithm we discuss here is provided in the Appendix.

if $\{na_i\} + \{na_{i+1:n}\} < 1$, *and*

$$Cov\,(M_{1:i-1}, M_i) = -\{na_{i:n}\}(1 - \{na_i\}), \tag{8}$$

if $\{na_i\} + \{na_{i+1:n}\} \geq 1$, *where* $a_{i:n} =: \sum_{k=i}^{n} a_k$ *for all* $i \in \{1, \ldots, n\}$.

Proof: We let $\mathcal{U}_i = \sigma\left(\{M_j, j = 1, \ldots, i\}\right)$ for $i = 1, \ldots, n-1$. From Proposition 9.3 of [1], we have that $\mathrm{E}[M_i] = na_i$, $\mathrm{E}[M_{1:i-1}] = na_{1:i-1}$ and

$$\mathrm{E}[(M_i - na_i)|\mathcal{U}_{i-1}] = (M_{i:n} - na_{i:n})\frac{\{na_i\}}{\{na_{i:n}\}},$$

if $\{na_i\} + \{na_{i+1:n}\} < 1$, and

$$\mathrm{E}[(M_i - na_i)|\mathcal{U}_{i-1}] = (M_{i:n} - na_{i:n})\frac{1 - \{na_i\}}{1 - \{na_{i:n}\}},$$

if $\{na_i\} + \{na_{i+1:n}\} \geq 1$. Note that since $M_{i:n} = n - M_{1:i-1}$, $M_{i:n} \in \mathcal{U}_{i-1}$. We use these results to obtain

$$\begin{aligned}
&\mathrm{Cov}\,(M_{1:i-1}, M_i) \\
&\quad = \mathrm{E}[(M_{1:i-1} - na_{1:i-1})(M_i - na_i)] \\
&\quad = \mathrm{E}\,[\mathrm{E}[(M_{1:i-1} - na_{1:i-1})(M_i - na_i)|\mathcal{U}_{i-1}]] \\
&\quad = \mathrm{E}\,[(M_{1:i-1} - na_{1:i-1})\mathrm{E}[(M_i - na_i)|\mathcal{U}_{i-1}]] \\
&\quad = \begin{cases} \mathrm{E}\,[(M_{1:i-1} - na_{1:i-1})(M_{i:n} - na_{i:n})]\,\dfrac{\{na_i\}}{\{na_{i:n}\}}, \\ \qquad \text{if } \{na_i\} + \{na_{i+1:n}\} < 1 \\ \mathrm{E}\,[(M_{1:i-1} - na_{1:i-1})(M_{i:n} - na_{i:n})]\,\dfrac{1 - \{na_i\}}{1 - \{na_{i:n}\}}, \\ \qquad \text{if } \{na_i\} + \{na_{i+1:n}\} \geq 1. \end{cases}
\end{aligned}$$

Since $M_{i:n}$ has the minimal variance property, we have $\mathrm{Var}(M_{i:n}) = \{na_{i:n}\}(1 - \{na_{i:n}\})$. It follows that

$$\begin{aligned}
&\mathrm{E}\,[(M_{1:i-1} - na_{1:i-1})(M_{i:n} - na_{i:n})] \\
&\quad = \mathrm{E}\,[M_{1:i-1}M_{i:n}] - n^2 a_{1:i-1}a_{i:n} \\
&\quad = \mathrm{E}\,[(n - M_{i:n})M_{i:n}] - n^2 a_{1:i-1}a_{i:n} \\
&\quad = n\mathrm{E}\,[M_{i:n}] - \mathrm{E}\left[M_{i:n}^2\right] - n^2 a_{1:i-1}a_{i:n} \\
&\quad = n\mathrm{E}\,[M_{i:n}] - \mathrm{Var}(M_{i:n}) - (\mathrm{E}\,[M_{i:n}])^2 \\
&\qquad - n^2 a_{1:i-1}a_{i:n} \\
&\quad = -(1 - \{na_{i:n}\})\{na_{i:n}\},
\end{aligned}$$

where the last equality follows from $a_{1:i-1} + a_{i:n} = 1$. The result follows from considering the two cases $\{na_i\} + \{na_{i+1:n}\} < 1$ and $\{na_i\} + \{na_{i+1:n}\} \geq 1$. □

Since $\{na_i\} + \{na_{i+1:n}\} \in [0, 2)$, the covariance given in Proposition 1 above can be re-written as

$$\begin{aligned}
\mathrm{Cov}\,(M_{1:i-1}, M_i) = &-\{na_i\}(1 - \{na_{i:n}\}) \\
&+ (\{na_i\} - \{na_{i:n}\})\,[\{na_i\} + \{na_{i+1:n}\}]\,.
\end{aligned}$$

When used with the marginal distributions (5), (6) and covariances (7) and (8), (4) simplifies to

$$\begin{aligned}
&P(M_i = m_i|M_{1:i-1} = m_{1:i-1}) \\
&\quad = P(M_i = m_i) \\
&\qquad \times \left(1 + \mathrm{Cov}(M_{1:i-1}, M_i)\left(\frac{m_i - na_i}{\{na_i\}(1 - \{na_i\})}\right)\right. \\
&\qquad \left. \times \left(\frac{m_{1:i-1} - na_{1:i-1}}{\{na_{1:i-1}\}(1 - \{na_{1:i-1}\})}\right)\right).
\end{aligned}$$

In particular, we obtain

$$\begin{aligned}
&P(M_i = \lfloor na_i \rfloor + 1|M_{1:i-1} = m_{1:i-1}) \\
&\quad = \{na_i\} + \mathrm{Cov}(M_{1:i-1}, M_i) \\
&\qquad \times \left(\frac{m_{1:i-1} - na_{1:i-1}}{\{na_{1:i-1}\}(1 - \{na_{1:i-1}\})}\right),
\end{aligned} \tag{9}$$

with $\mathrm{Cov}(M_{1:i-1}, M_i)$ given by (7) and (8). The resulting QSF minimal variance algorithm is given by Algorithm 2. This subroutine can be inserted on line 18 of Algorithm 1.

Algorithm 2 QSF Minimal Variance

1: **procedure** QSF Minimal Variance($\boldsymbol{N}_t$, $\boldsymbol{N}_{t-1}$, $(\hat{x}_{i,t}, a_{i,t})_{i \in C_t}$)
▷ $\boldsymbol{N}_t$ is the number of particles not in C_t
▷ Compute the number of particles in C_t
2: $n = \boldsymbol{N}_{t-1} - \boldsymbol{N}_t$
▷ Initialize variables
3: $g = n$, $h = n$, $\bar{h} = 0$
4: Generate $(U_k)_{k=1}^{n-1}$ Uniforms(0,1).
5: **for** $i \in C_t$ **do**
▷ Compute $\mathrm{Cov}(M_{1:i-1,t}, M_{i,t})$
6: $\sigma = -\{na_{i,t}\}(1 - \{g\}) + (\{na_{i,t}\} - \{g\})\lfloor\{na_{i,t}\} + \{g - na_{i,t}\}\rfloor$
▷ Number of offspring at location i
7: $M_{i,t} = \lfloor na_{i,t} \rfloor + \mathbb{1}_{\{U_i < \{na_{i,t}\} + \sigma\bar{h}\}}$
8: **for** $k \in \{1, \cdots, M_{i,t}\}$ **do**
▷ Place offspring in new location
9: $x_{\boldsymbol{N}_{t-1}-h+k,t} = \hat{x}_{i,t}$
10: **end for**
▷ Update average number of particles to distribute
11: $g = g - na_{i,t}$
▷ Update number of particles to distribute
12: $h = h - M_{i,t}$
13: $\bar{h} = \frac{g-h}{\{n-g\}(1-\{n-g\})}$
14: **end for**
15: $M_n = h$
16: **end procedure**

The algorithm is built so that each M_i and $M_{1:i}$, for $i \in \{1, \ldots, n\}$ have minimal variance. The next proposition confirms this property.

Proposition 2: Let M_i *denote the number of offspring assigned to the location* i *and let* $M_{1:i} = \sum_{i=1}^{i} M_i$ *for* $i \in \{1, \ldots, n\}$. *When the vector* $M = (M_1, \ldots, M_n)$ *is generated with the QSF minimal variance algorithm, the following hold for each* i*:*

(a) $E[M_i] = na_i$;
(b) $E[(M_i - na_i)^2] = \{na_i\}(1 - \{na_i\})$;
(c) $E[M_{1:i}] = na_{1:i}$;
(d) $E[(M_{1:i} - na_{1:i})^2] = \{na_{1:i}\}(1 - \{na_{1:i}\})$.

Proof: Proposition 2 is proved by induction. First, we observe that when $i = 1$, $P(M_1 = \lfloor na_1 \rfloor + 1) = \{na_1\}$. It follows easily that $\mathrm{E}[M_1] = na_1$ and $\mathrm{Var}(M_1) = p(1-p) = \{na_1\}(1 - \{na_1\})$.

For the induction step, we denote $\mathcal{U}_i$ as in the proof of Proposition 1 and let $(U_i)_{i=1}^n$ be i.i.d Uniform(0,1) random variables. To show (a), we note by (7), (8) and (9) that

$$M_i = \lfloor na_i \rfloor + \mathbb{1}_{\{U_i < \{na_i\} + \sigma\bar{h}\}},$$

with

$$\sigma = -(1 - \{na_{i:n}\})\{na_i\} \tag{10}$$

if $\{na_i\} + \{na_{i+1:n}\} < 1$, and

$$\sigma = -\{na_{i:n}\}(1 - \{na_i\}) \tag{11}$$

if $\{na_i\} + \{na_{i+1:n}\} \geq 1$, and

$$\bar{h} = \frac{M_{1:i-1} - na_{1:i-1}}{\{na_{1:i-1}\}(1 - \{na_{1:i-1}\})\}}.$$

It follows that

$$\mathrm{E}[M_i|\mathcal{U}_{i-1}] = \lfloor na_i \rfloor + \{na_i\} + \sigma\bar{h} = na_i + \sigma\bar{h},$$

and thus $\mathrm{E}[M_i] = na_i$ since $\mathrm{E}[\bar{h}] = 0$ by the induction hypothesis. To obtain (b), we first note that $M_i - na_i = -\{na_i\} + \mathbb{1}_{\{U_i < \{na_i\} + \sigma\bar{h}\}}$, so that

$$\begin{aligned}
&\mathrm{E}[(M_i - na_i)^2|\mathcal{U}_{i-1}] \\
&\quad = \{na_i\}^2 + (1 - 2\{na_i\})\mathrm{E}[\mathbb{1}_{\{U_i < \{na_i\} + \sigma\bar{h}\}}|\mathcal{U}_{i-1}] \\
&\quad = \{na_i\}^2 + (1 - 2\{na_i\})(\{na_i\} + \sigma\bar{h}).
\end{aligned}$$

Since $\mathrm{E}[\bar{h}] = 0$, it follows that

$$\begin{aligned}
\mathrm{E}[(M_i - na_i)^2] &= \{na_i\} - \{na_i\}^2 \\
&= \{na_i\}(1 - \{na_i\}).
\end{aligned}$$

To show (c), it suffices to observe that

$$\begin{aligned}
&\mathrm{E}[M_{1:i} - na_{1:i}|\mathcal{U}_{i-1}] \\
&\quad = \mathrm{E}[M_{1:i-1} + M_i - na_{1:i-1} - na_i|\mathcal{U}_{i-1}] \\
&\quad = M_{1:i-1} - na_{1:i-1} + \mathrm{E}[M_i - na_i|\mathcal{U}_{i-1}].
\end{aligned}$$

The result follows by (a) and the induction hypothesis.

To show (d), we first re-write $(M_{1:i} - na_{1:i})^2$ as

$$\begin{aligned}
&(M_{1:i-1} - na_{1:i-1})^2 + (M_i - na_i)^2 \\
&\qquad + 2(M_{1:i-1} - na_{1:i-1})(M_i - na_i),
\end{aligned}$$

so that

$$\begin{aligned}
&\mathrm{E}[(M_{1:i} - na_{1:i})^2|\mathcal{U}_{i-1}] \\
&\quad = \mathrm{E}[(M_{1:i-1} - na_{1:i-1})^2|\mathcal{U}_{i-1}] \\
&\qquad + \mathrm{E}[(M_i - na_i)^2|\mathcal{U}_{i-1}] \\
&\qquad 2\mathrm{E}[(M_{1:i-1} - na_{1:i-1})(M_i - na_i)|\mathcal{U}_{i-1}],
\end{aligned}$$

and thus

$$\begin{aligned}
&\mathrm{E}[(M_{1:i} - na_{1:i})^2] \\
&\quad = \{na_{1:i-1}\}(1 - \{na_{1:i-1}\}) \\
&\qquad + \{na_i\}(1 - \{na_i\}) + 2\,\mathrm{Cov}(M_{1:i-1}, M_i),
\end{aligned}$$

where the last equality is obtained using the induction hypothesis and (b). The QSF minimal variance is built so that $\mathrm{Cov}(M_{1:i-1}, M_i) = \sigma$, with σ given by (10) and (11), but this result can also easily be verified by induction. From this result, we have

$$\begin{aligned}
&\mathrm{E}[(M_{1:i} - na_{1:i})^2] \\
&\quad = \{na_{1:i-1}\}(1 - \{na_{1:i-1}\}) + \{na_i\}(1 - \{na_i\}) \\
&\qquad - 2(1 - \{na_{i:n}\})\{na_i\},
\end{aligned} \tag{12}$$

if $\{na_i\} + \{na_{i+1:n}\} < 1$, and

$$\begin{aligned}
&\mathrm{E}[(M_{1:i} - na_{1:i})^2] \\
&\quad = \{na_{1:i-1}\}(1 - \{na_{1:i-1}\}) + \{na_i\}(1 - \{na_i\}) \\
&\qquad - 2\{na_{i:n}\}(1 - \{na_i\}),
\end{aligned} \tag{13}$$

if $\{na_i\} + \{na_{i+1:n}\} \geq 1$. We consider the first case, $\{na_i\} + \{na_{i+1:n}\} < 1$, which is equivalent to

$$\{na_i\} + \{na_{i+1:n}\} = \{na_{i:n}\}. \tag{14}$$

We also observe that $\{na_{1:i-1}\} = 1 - \{na_{i:n}\}$ if $\{na_{i:n}\} \neq 0$, so that

$$\begin{aligned}
&\mathrm{E}[(M_{1:i} - na_{1:i})^2] \\
&\quad = \{na_{i:n}\}(1 - \{na_{i:n}\}) + \{na_i\}(1 - \{na_i\}) \\
&\qquad - 2(1 - \{na_{i:n}\})\{na_i\} \\
&\quad = (1 - \{na_{i:n}\})(\{na_{i:n}\} - \{na_i\}) \\
&\qquad + \{na_i\}(\{na_{i:n}\} - \{na_i\}) \\
&\quad = (\{na_{i:n}\} - \{na_i\})(1 - \{na_{i:n}\} + \{na_i\}) \\
&\quad = \{na_{i+1:n}\}(1 - \{na_{i+1:n}\}) \\
&\quad = (1 - \{na_{1:i}\})\{na_{1:i}\},
\end{aligned}$$

where the fourth equality holds by (14).

If $\{na_{i:n}\} = 0$, then by (14), $\{na_i\} = \{na_{i+1:n}\} = 0$ since both values must be non-negative. It follows from (12) and (13) that $\mathrm{E}[(M_{1:i} - na_{1:i})^2] = 0$. But since $\{na_{i+1:n}\} = 0$, then $\{na_{1:i}\} = 0$ and $(1 - \{na_{1:i}\})\{na_{1:i}\} = 0$, so $\mathrm{E}[(M_{1:i} - na_{1:i})^2] = (1 - \{na_{1:i}\})\{na_{1:i}\}$ holds.

The case (13) is handled similarly: $\{na_i\} + \{na_{i+1:n}\} \geq 1$ is equivalent to

$$\{na_i\} + \{na_{i+1:n}\} = \{na_{i:n}\} + 1, \tag{15}$$

which allows us to show that

$$\mathrm{E}[(M_{1:i} - na_{1:i})^2] = \{na_{1:i}\}(1 - \{na_{1:i}\})$$

when $\{na_{1:i-1}\} \neq 0$. If $\{na_{1:i-1}\} = 0$, then $\{na_{i:n}\} = 0$ and it follows easily from (12) and (13) that

$$\begin{aligned}
\mathrm{E}[(M_{1:i} - na_{1:i})^2] &= \{na_i\}(1 - \{na_i\}) \\
&= (1 - \{na_{i+1:n}\})\{na_{i+1:n}\} \\
&= \{na_{1:i}\}(1 - \{na_{1:i}\}),
\end{aligned}$$

where the second equality holds by (15) and the last equality is true whether $\{na_{i+1:n}\}$ is 0 or not. □

C. BRANCHING ALGORITHMS WITH NEGATIVE DEPENDENCE

The branching algorithms that we propose in this section ensure that the variance of the number of offspring at each location, M_i, remain minimal for all i. However, we relax the condition that the total number of particles remain stable through time. Such a relaxation is considered with the goal of reducing computational time of the sampling procedure.

1) ANTITHETIC VARIATES

The lowest possible correlation between two random variables with given marginals can only be attained if the random variables are countermonotonic.

Here we propose to correlate the I_i's two-by-two so that each couple has a countermonotonic dependence structure. We do so by simulating $n/2$ Uniform(0,1) random variables (or $(n+1)/2$ if n is odd), and by generating two countermonotonic Bernoullis using each of the uniform random variables. The pseudo-code for this method is given in Algorithm 3.

Algorithm 3 Antithetic Variates

procedure Antitethic($N_t, N_{t-1}, (\ell_{i,t})_{i\in C_t}, \bar{\ell}$)
▷ Compute number of particles in C_t
  $n = N_{t-1} - N_t$
▷ m is the number of Uniforms to simulate
  $m = \lfloor \frac{n}{2} \rfloor + 1$
  Generate $(U_i)_{i=1}^m$ Uniforms(0,1)
  for $i \in \{1, \cdots, m\}$ **do**
    $\rho_{2i-1,t} = \mathbb{1}_{\{U_i < \{\ell_{2i-1}/\bar{\ell}\}}$
    $\rho_{2i,t} = \mathbb{1}_{\{1-U_i < \{\ell_{2i-1}/\bar{\ell}\}}$
  end for
end procedure

2) LIST SEQUENTIAL SAMPLING

The sampling method of [3] can be used to generate a vector of dependent Bernoulli random variables with predetermined conditional correlations between each component. It is similar to a special case of the quick simulation fields algorithm of [26] applied to multivariate Bernoulli random variables, but it uses conditional covariances instead of unconditional ones.

The method stems from the experiment design and sampling literature; each Bernoulli to simulate can be seen as a unit which will either be sampled or not in the context of a survey. The inclusion probability of each unit is proportional to some quantity of interest and it can differ from one unit to the other. Introducing negative correlation between inclusion indicators can reduce the variability of the results of the survey.

The general idea of the method is to go through each unit one by one in a pre-specified order (so it is a type of *list sequential sampling*) and to decide whether or not this unit will be included in the sample; this is equivalent to simulating a Bernoulli, where 1 indicates inclusion of the unit. After each unit is sampled, the conditional laws (or inclusion probabilities) of all the yet-to-be-sampled units are updated, based on the value of the new simulated Bernoulli (or inclusion decision) and on correlations (or correlation-based weights) chosen by the sampler.

Therefore, for all $i \in \{1, \ldots, n\}$, if we let $p_i^{(0)} = \{\ell_i/\bar{\ell}\}$ be the unconditional Bernoulli parameter (or unconditional inclusion probability), the updated parameters are given by

$$p_j^{(i)} = p_j^{(i-1)} - (\rho_i - p_i^{(i-1)})\beta_{j-i}^{(i)},$$

for $j \geq i+1$. To ensure that the updated conditional inclusion probabilities $p_j^{(i)}$ remain in [0, 1], the weights $\beta_{j-i}^{(i)}$ must satisfy

$$-\min\left(\frac{1-p_j^{(i-1)}}{1-p_i^{(i-1)}}, \frac{p_j^{(i-1)}}{p_i^{(i-1)}}\right) \leq \beta_{j-i}^{(i)} \leq \min\left(\frac{p_j^{(i-1)}}{1-p_i^{(i-1)}}, \frac{1-p_j^{(i-1)}}{p_i^{(i-1)}}\right). \quad (16)$$

A non-zero weight creates dependence between the i^{th} and the j^{th} random variables. It is explained in [3] that negative correlations will be obtained by choosing positive weights $\beta_{j-i}^{(i)}$. The maximal weight that can be chosen is therefore the upper bound in (16), given that the sum of the total allocated weights remains smaller than or equal to 1. This list sequential method using maximal weights is implemented in Algorithm 4. The parameter m, to be selected by the user, indicates the maximum number of random variables that will be negatively correlated with each ρ_i.

Algorithm 4 List Sequential Sampling

procedure List sequential($N_t, m, (\ell_{i,t})_{i\in C_t}, \bar{\ell}$)
▷ Compute the initial Bernoulli parameters
  $(p_i)_{i\in C_t} = (\{\ell_i/\bar{\ell}\})_{i\in C_t}$
  for $i \in C_t$ **do**
    $\rho_i = \mathbb{1}_{\{U_i < p_i\}}$
▷ Record total allocated weight
    $\bar{\beta} = 0$
    for $j \in \{1, \ldots, m\}$ **do**
      $\beta_j = \min\left(\frac{p_{i+j}}{1-p_i}, \frac{1-p_{i+j}}{p_i}, 1-\bar{\beta}\right)$
      $p_{i+j} = p_{i+j} - (\rho_i - p_i)\beta_j$
      $\bar{\beta} = \min(\bar{\beta} + \beta_j, 1)$
    end for
  end for
end procedure

IV. NUMERICAL RESULTS

A. MODEL

To assess the speed and stability of the proposed algorithms, we test on a common model from the particle filtering literature (see for example [28]), given by

$$X_t = \frac{1}{2}X_{t-1} + \frac{25\,X_{t-1}}{1+X_{t-1}^2} + 8\cos(1.2(t-1)) + U_t$$
$$Y_t = \frac{X_{t-1}^2}{20} + V_t,$$

where U_t is normally distributed with mean 0 and variance 10 and V_t is standard Cauchy distributed.

For this model, we calculate the error as

$$error := \sqrt{\frac{1}{T}\sum_{k=1}^{T}(\pi_k^N(f) - f(X_k))^2}, \tag{17}$$

where f is defined as

$$f(x) := \begin{cases} 1000, & x > 1000 \\ -1000, & x < -1000 \\ x, & -1000 \le x \le 1000 \end{cases}$$

$\pi_k^N(f)$ is our estimate of $f(X_k)$.

B. METHODS

1) PROCEDURE

In all experiments, to determine an optimal value of r for sampling, we proceed in the following manner.

We set the number of time steps to be $T = 1000$ and the number of trials to be 1000. We generate 1000 random copies of the signal, one for each trial, at the beginning of the experiments. We use these random copies for all experiments so as to be consistent when evaluating different algorithms and evaluating values of r for a given algorithm. The values of r we study range from 1 to 6. This range was chosen based on preliminary experiments to ensure that it includes the optimal value for each algorithm.

For each algorithm and for each value of r, we seek the execution time of the algorithm for a fixed performance. To do so, for each algorithm, we specify an initial number of particles $\boldsymbol{N}_0 = 150$ and run the algorithm for 1000 trials. At the end of each trial, we calculate the error. If the average error over all trials is lower than the specified threshold, we then accept that average time. If the error is above the threshold, we increment the number of particles by 10 and repeat the experiment.

All the algorithms are implemented according to the pseudo-code included in this article. For the list sequential sampling (Algorithm 4), we set $m = 3$ to increase the amount of dependence between the particle locations without slowing down the algorithm too much.

We set the error threshold to be 14. These values are based on preliminary experiments which determined an error that was not so high to be unachievable given our limited computational resources, but not so low as to be reached trivially by all algorithms. All run times are recorded in milliseconds.

2) SUMMARY STATISTICS

The main summary statistics used to evaluate particle variation are standard deviation range and the average maximum and minimum number of particles. For every trial i and over all time steps in that trial, denote the maximum number of particles by $N_{max,i}$, the minimum number of particles by $N_{min,i}$, and the average number of particles by $\overline{N}_i$. Let $\overline{N}_{max}$ denote the average of all $N_{max,i}$'s, $\overline{N}_{min}$ the average of all $N_{min,i}$'s, and $\overline{N}$ the average of all $\overline{N}_i$'s. We report both $\overline{N}_{max}$ and $\overline{N}_{min}$.

For each trial, the standard deviation of the number of particles is calculated after all T time steps, and then averaged across all the trials. The resulting statistics is denoted σ_N. We then calculate the standard deviation range. The intuition for the standard deviation range is that we wish to know the range of the number of particles corresponding to two standard deviations both below and above the mean. To obtain the standard deviation range, we divide 4 times the standard deviation σ_N by $\overline{N}$:

$$\Delta_\sigma := \frac{4\sigma_N}{\overline{N}}.$$

3) CONFIGURATION

All simulations were coded in C++ and run in RStudio using Rcpp on a PC with a Dual Intel Xeon Processor E5-2650 v2 and 64 GB of RAM.

C. RESULTS

1) BRANCHING PARTICLE STABILITY

An important concern about branching methods is particle control. Here, we assess the variation in the number of particles resulting from the branching algorithms we consider.

We first observe that in all three cases, Δ_σ increases with r. Increasing r means that fewer particles are branched, since increasing r widens the interval in which the particles are left untouched. As r increases, the weights of the particles considered for branching become either very high (and will likely create a large number of offspring) or very low (and will die off with a high probability). This explains the larger variation in the number of particles when r increases.

We also observe that a higher number of initial particles $\boldsymbol{N}_0$ contributes to stabilizing the number of particles throughout the filter. Comparing Fig. 1a and 1b shows that the effect becomes significant for higher values of r. This trend confirms similar particle stability experiments in [20]. The risk of the number of particles exploding decreases as $\boldsymbol{N}_0$ increases. This result might be surprising for those who believe that branching algorithms are doomed to particle instability, but is not so surprising once we note that, as described in [20], the expected number of particles at time t for our branching algorithms is always the initial number of particles $\boldsymbol{N}_0$ rather than the previous number of particles $\boldsymbol{N}_{t-1}$.

For lower values of r, antithetic variates and list sequential sampling appear to offer significantly higher particle stability than the combined branching algorithm. This may be due to the explicit negative dependence structure between the number of particles at certain locations. As r increases, this difference disappears, and combined branching offers greater stability. We find in the next section that the optimal parameter r for these three algorithms fall between 2.05 and 3.50, depending on the method. In this range, all three algorithms perform very similarly in terms of particle stability.

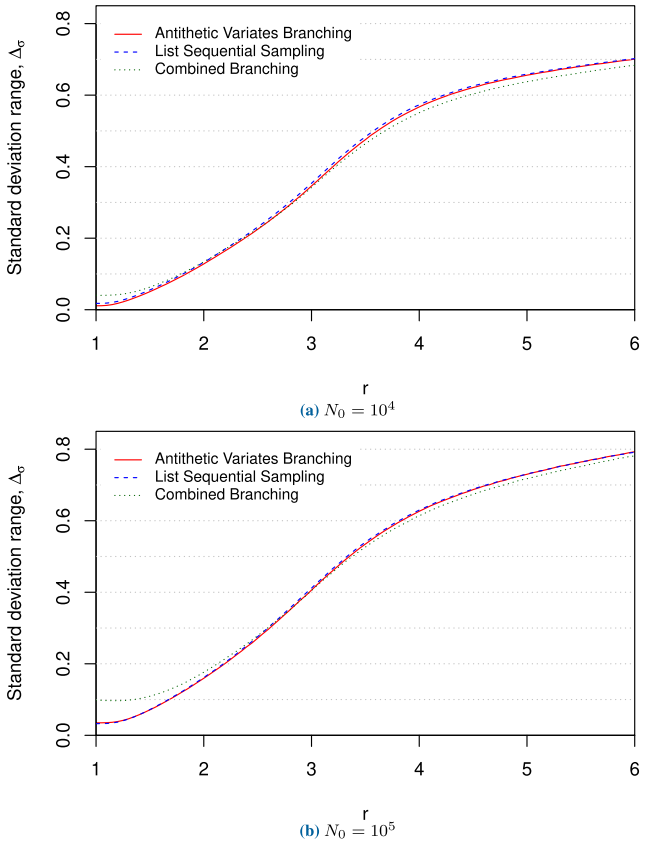


FIGURE 1. Δ_σ as a function of r for branching algorithms.

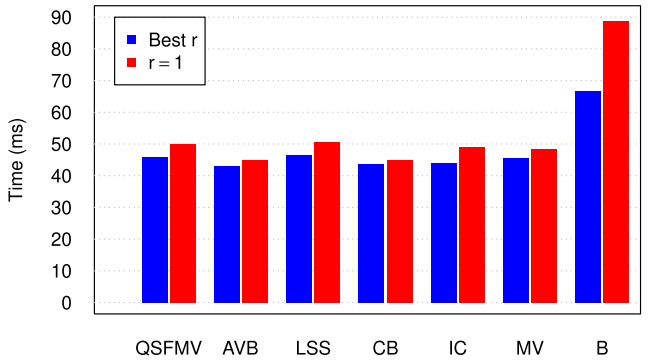


FIGURE 2. Comparison of the run times for best results (Table 1). Key: QSFMV = QSF minimal variance, AVB = antithetic variates branching, LSS = list sequential sampling, CB = combined branching, IC = interacting combined, MV = minimal variance, B = bootstrap.

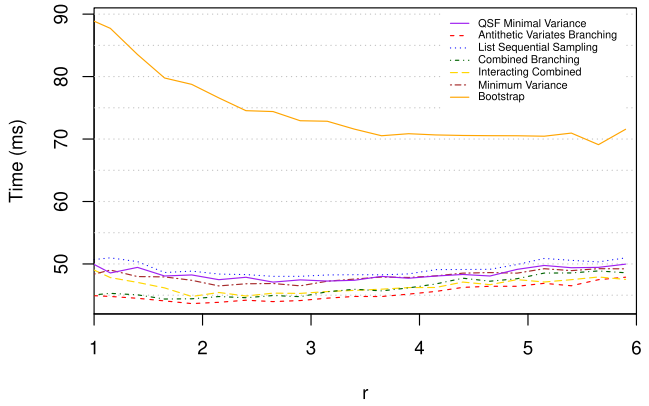


FIGURE 3. Run time as a function of r (all algorithms).

It should be noted that while particle stability resulting from branching methods is exemplified here in a particular context, previous experiments in different settings have also lead to similar conclusions. In particular, [22] uses sequential branching Monte Carlo to simulate asset prices in the context of the Heston stochastic volatility model (see [16]) and show similar particle stability. Such property was also observed when using branching particle filters to calibrate the Heston model in [36]. We are therefore confident that the particle stability property we observe in the numerical example presented here translates to a variety of problems.

2) PERFORMANCE RESULTS AND DISCUSSION

Table 1 presents the results of the experiment described in Section IV-B1. As expected, the basic bootstrap is slower than the other algorithms and requires a higher number of initial particles to reach a similar precision. Fig. 3 shows that execution time of the basic bootstrap algorithm can be reduced by partial sampling, but remains high. It is interesting to note that the best value r for the bootstrap algorithm is much higher at 5.65 than for all the other algorithms (between 2.05 and 3.50). Such a high value results in much fewer particles being sampled, which speeds up the algorithm. Although 5.65 is identified as the optimal r, Fig. 3 shows that the run time of the algorithm (for the fixed performance that we selected) remains around the same level for values of r between 3.5 and 6.

The antithetic variates branching algorithm is fastest overall, both when considering the best performing r (in Table 1 and Fig. 2). Fig. 3 and 4 show that this result is robust for most values of r considered. This result is not surprising given the low computational complexity of the antithetic variates algorithm, combined with the fact that it requires the simulation of half as many random variables as there are particles to branch. Nonetheless, except for the bootstrap, the other algorithms considered are at most 10% slower than the antithetic variates. These run times are also not particularly sensitive to the choice of r, although they tend to increase for high values of r.

It is important to remark that the problem considered here is chosen from the interacting particle filter literature, and that our results do not contradict other empirical results showing greater outperformance by branching algorithms in problems chosen to highlight the benefits of branching (see for example [21]).

All algorithms can be sped up by partial sampling, as shown in Fig. 2. The time improvement is most noticeable for the basic bootstrap algorithm. A slight U-shape can be observed in Fig. 4 for all algorithms except the bootstrap, indicating that values of r between 2 and 4 are optimal for this problem, with performance worsening for r outside of this range. We hypothesize that this observation is the result

TABLE 1. Best performance for each algorithm.

Algorithm	r	N_0	Error	Time (ms)	Δ_σ	$\bar{N}_{\min}$	$\bar{N}_{\max}$
QSF minimal variance	2.65	260	13.998	45.993	0.0	260.0	260.0
Antithetic variates branching	2.05	260	13.999	43.072	0.234	163.5	339.7
List sequential sampling	3.50	280	13.995	46.605	0.622	115.8	421.4
Combined branching	2.45	260	13.989	43.720	0.358	138.5	359.1
Interacting combined	2.45	260	13.993	44.084	0.0	260.0	260.0
Minimal variance	2.05	250	13.997	45.717	0.0	250.0	250.0
Bootstrap	5.65	310	13.998	66.862	0.0	310.0	310.0

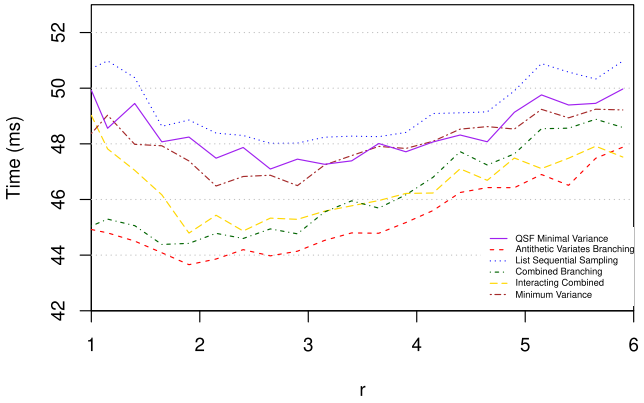


FIGURE 4. Run time as a function of r (faster algorithms).

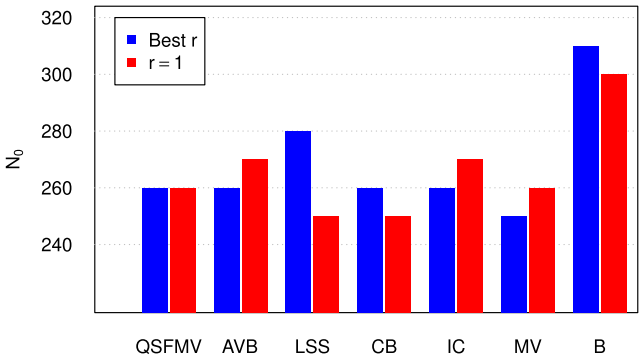
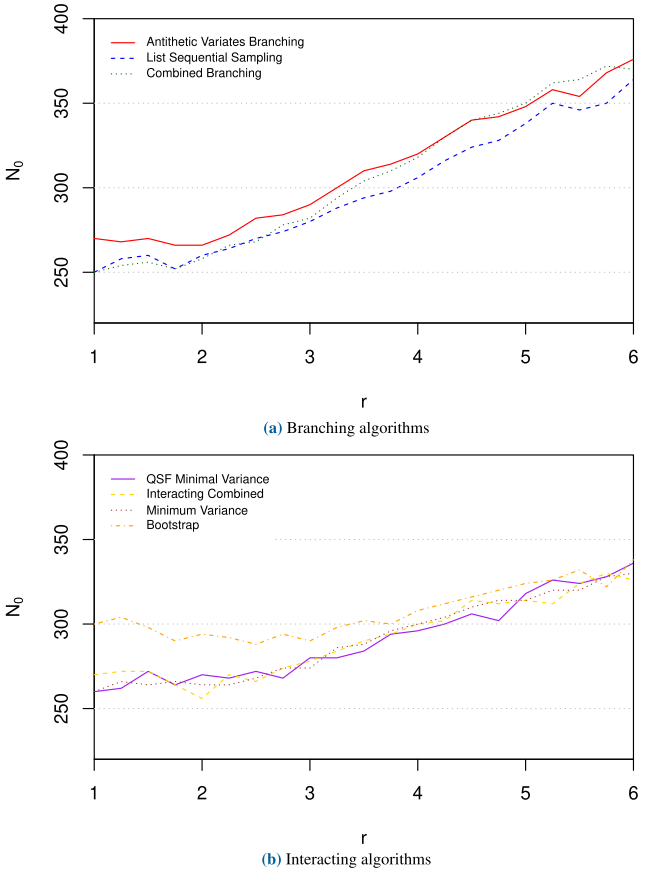


FIGURE 5. Comparison of initial particle numbers (N_0) used to obtain best results (Table 1). Key: QSFMV = QSF Minimal Variance, AVB = antithetic variates branching, LSS = list sequential sampling, CB = combined branching, IC = interacting combined, MV = minimal variance, B = boostrap.



(a) Branching algorithms

(b) Interacting algorithms

FIGURE 6. Initial number of particles N_0 as a function of r.

of two phenomenons. First, when r is close to 1, too many particles are sampled, which introduces unnecessary noise in the system. In other words, we are unnecessarily perturbing particles that already accurately represent the signal. When r is too large, not enough sampling is taking place, which leaves low-quality particles untouched. Further insight can be obtained from examining Fig. 6: there seems to be a tradeoff between the value of r and the initial number of particles N_0 required to reach the error threshold. When r is high, fewer computational resources are expended sampling particles, but more particles are needed to reach the error threshold. When r is low, we possibly require fewer particles to reach the error threshold, but need to sample more of them.

The new algorithms we introduced in this article fare at least as well as existing ones. However, it appears from Fig. 3 that the list sequential sampling algorithm may be slightly slower overall. The negative dependence it creates between the number of particles at different locations may be too weak to offset the added computational complexity. In contrast, the antithetic variates branching induces negative dependence between particles only two-by-two. As shown in Fig. 6, this somewhat less sophisticated negative dependence structure seems to require a slightly higher initial number of particles N_0. However, the simplicity of its implementation makes it faster, which makes up for the increased number of particles.

It could also be argued that the version of the combined branching algorithm we implemented is new, since we modified it to remove the shuffle. Our results show that the possible bias induced by this omission does not worsen the performance, and that the modified algorithm performs similarly to the branching algorithms that include explicit negative dependence.

Our new implementation of the minimal variance dependence structure structure, which uses quick simulation fields rather than embedded "if" statements, performs as well as the minimal variance algorithm of [7]. When the best value for r is considered (in Table 1), both run times are very close. This result holds across values of r, see Fig. 4.

While minimizing the variance of the total number of particles is theoretically desirable, our results show that relaxing this requirement and letting N vary may result in better performing algorithms. The increased variance in the number of particles can be offset by the reduction in run time, especially when the variance of the particles at each location is kept minimal. In the problem we consider here, the performance of the branching algorithms is therefore not impaired by the variation in the number of particles.

V. CONCLUDING REMARKS

We proposed new sampling algorithms that explicitly create negative dependence between the number of particles reallocated to each location in the sampling step of a particle filter. One of these new algorithms establishes a novel way to impose the so-called minimal variance dependence structure via quick simulation fields. Our method may be easier to understand and our numerical experiments show that it performs at least as well as the original minimal variance algorithm. In order to improve the performance of the filtering procedure, all three new algorithms keep the variance of the number of particles *at each location* minimal, but some allow for variation in the total number of particles. Numerical results show that explicit negative dependence can be implemented efficiently and results in high performing algorithms.

We also recall the partial sampling scheme of [20] and propose to implement it in a wide range of interacting and branching procedures. In all the cases we considered, partial sampling reduces the run time of filtering algorithms for a fixed performance. We also show that the introduction of partial resampling into interacting particle filters significantly closes the gap between them and branching particle filters.

Although our numerical results are model specific, they contribute to the growing body of literature (see [20], [22]) providing evidence that branching particle filters are stable when they are correctly implemented. Further testing and application of branching filters and branching Monte Carlo methods in general is desirable to further the analysis of these methods.

APPENDIX

This section contains some of the algorithms mentioned in the main part of the text. They are the algorithms that are implemented for comparison purposes in Section IV. Similarly to the algorithms presented in Section III, we drop the reference to time to simplify the algorithms.

Algorithm 5 Multinomial Bootstrap

procedure Bootstrap($\boldsymbol{N}_t, \boldsymbol{N}_{t-1}, (\hat{x}_{i,t}, a_{i,t})_{i\in C_t}$)
▷ Compute the number of particles in C_t
  $n = \boldsymbol{N}_{t-1} - \boldsymbol{N}_t$
▷ Compute cumulative probabilities
  $p_i = \sum_{k=1}^{i} a_{i,t}$ for $i \in C_t$
  $V_{n+1} = 1$
  $j = n - 1$
  for $i \in \{n, \ldots, 1\}$ **do**
    Generate U_i a Uniform(0,1)
    $V_i = (U_i)^{\frac{1}{i}} V_{i+1}$
    while $V_i \le p_j$ **do**
      $j = j - 1$
    end while
    $x_{\boldsymbol{N}_t+k,t} = \hat{x}_{j+1,t}$
  end for
end procedure

Algorithm 6 Interacting Combined Filter

procedure Interacting Combined($\boldsymbol{N}_t, \boldsymbol{N}_{t-1}, (\hat{x}_{i,t}, a_{i,t})_{i\in C_t}$)
▷ Compute the number of particles in C_t
  $n = \boldsymbol{N}_{t-1} - \boldsymbol{N}_t$
▷ Record particles assigned deterministically
  $S = 0$
  for $i \in C$ **do**
    $k = 0$
▷ Deterministically assign particles
    while $k < \lfloor n a_{i,t} \rfloor$ **do**
      $k = k + 1$
      $x_{N_t+S+k} = \hat{x}_{i,t}$
    end while
    $S = S + k$
  end for
▷ Number of particles left to assign
  $m = n - S$
  $j = 1$
  for $i \in \{1, \ldots, n\}$ **do**
    $p_i = \sum_{k=1}^{i} \frac{\{n a_{i,t}\}}{m}$
  end for
  for $k \in \{1, \ldots m\}$ **do**
▷ Use stratified uniforms
    Generate U_k, a Uniform$\left(\frac{k-1}{m}, \frac{k}{m}\right)$
    while $U_k \ge p_j$ **do**
      $j = j + 1$
    end while
    $x_{\boldsymbol{N}_t+S+k,t} = \hat{x}_{j,t}$
  end for
end procedure

Algorithm 7 Minimal Variance

procedure Minimal Variance($N_t, N_{t-1}, (\hat{x}_{i,t}, a_{i,t})_{i\in C_t}$)
▷ Compute the number of particles in C_t
  $n = N_{t-1} - N_t$
  $g = n, h = n$
  for $i \in C_t$ **do**
▷ Deterministic number of offspring at location i.
    $M_{i,t} = \lfloor na_{i,t} \rfloor$
    Generate Uniform(0, 1,) U_i
    if $\{na_{i,t}\} + \{g - na_{i,t}\} < 1$ and $U_i\{g\} \leq \{na_{i,t}\}$ **then**
      $M_{i,t} = M_{i,t} + h - \lfloor g \rfloor$
    else if $\{na_{i,t}\} + \{g - na_{i,t}\} \geq 1$ and $U_i(1 - \{g\}) \geq \{na_{i,t}\} - \{g\}$ **then**
      $M_{i,t} = M_{i,t} + h - \lfloor g \rfloor$
    else if $\{na_{i,t}\} + \{g - na_{i,t}\} \geq 1$ and $U_i(1 - \{g\}) < \{na_{i,t}\} - \{g\}$ **then**
      $M_{i,t} = M_{i,t} + 1$
    end if
    for $k \in \{1, \ldots, M_{i,t}\}$ **do**
      $x_{N_{t-1}-h+k,t} = \hat{x}_{i,t}$
    end for
    $h = h - M_{i,t}$
    $g = g - a_{i,t}$
  end for
end procedure

Algorithm 8 Combined Branching

procedure Combined Branching($N_t, N_{t-1}, (l_{i,t})_{i\in C_t}, \bar{\ell}$)
▷ Compute the number of particles in C_t
  $n = N_{t-1} - N_t$
  for $i \in C_t$ **do**
    Generate U_i, a Uniform$\left(\frac{i-1}{n}, \frac{i}{n}\right)$
    $\rho_{i,t} = \mathbb{1}_{\{U_i \leq \{\ell_{i,t}/\bar{\ell}\}}$
  end for
end procedure

The basic multinomial bootstrap algorithm [15] is detailed in Algorithm 5. The interacting combined sampler [10] is a combination of the residual [29] and stratified [19] methods. The implementation we consider is given in Algorithm 6. The minimal variance algorithm was introduced in [4]. In Algorithm 7, we present modified version of the one in [1], given in [20]. Finally, a modified version of the combined branching algorithm of [20], against which we compare our new branching algorithms, is detailed in Algorithm 8. The modification rests in the fact that we do not perform a full random permutation, as in the original version of the algorithm. Rather, we rely upon the natural particle reorderings in Algorithm 1 as a result of sampling to permute the particles.

ACKNOWLEDGMENT

The authors thank two anonymous referees for their useful comments which helped improve the paper.

REFERENCES

[1] A. Bain and D. Crisan, *Fundamentals Stochastic Filtering*. New York, NY, USA: Springer, 2009.
[2] D. J. Ballantyne, H. Y. Chan, and M. A. Kouritzin, "A branching particle based nonlinear filter for multi-target tracking," in *Proc. Int. Conf. Inf. Fusion, Montreal*. Citeseer, 2001.
[3] L. Bondesson and D. Thorburn, "A list sequential sampling method suitable for real-time sampling," *Scandin. J. Statist.*, vol. 35, no. 3, pp. 466–483, 2008.
[4] D. Crisan, "Particle filters—A theoretical perspective," in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, N. Gordon, eds. New York, NY, USA: Springer, 2001. pp. 17–41
[5] D. Crisan and A. Doucet, "A survey of convergence results on particle filtering methods for practitioners," *IEEE Trans. Signal Process.*, vol. 50, no. 3, pp. 736–746, Mar. 2002.
[6] D. Crisan and T. Lyons, "Nonlinear filtering and measure-valued processes," *Probab. Theory Rel. Fields*, vol. 109, no. 2, pp. 217–244, 1997.
[7] D. Crisan and T. Lyons, "Minimal entropy approximations and optimal algorithms for the filtering problem," *Monte Carlo Methods Appl.*, vol. 8, no. 4, pp. 343–356, 2002.
[8] P. D. Moral and A. Doucet, "Particle methods: An introduction with applications," in *ESAIM: Proceedings*, vol. 44. Les Ulis, France: EDP Sciences, Jan. 2014, pp. 1–46.
[9] P. Del Moral and L. Miclo, "Branching and interacting particle systems approximations of Feynman-Kac formulae with applications to nonlinear filtering," in *Séminaire de Probabilités XXXIV*, J. Azéma, M. Ledoux, M. Émery, and M. Yor, eds. Berlin, Germany: Springer, 2000, pp. 1–145.
[10] R. Douc, O. Cappé, and E. Moulines, "Comparison of resampling schemes for particle filtering," in *Proc. 4th Int. Symp. Image Signal Process. Anal. (ISPA)*, 2005, pp. 64–69.
[11] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," in *The Oxford Handbook of Nonlinear Filtering*. New York, NY, USA: Oxford Univ. Press, 2009.
[12] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statist. Comput.*, vol. 10, no. 3, pp. 197–208.
[13] M. Fasiolo, N. Pya, and S. N. Wood, "A comparison of inferential methods for highly nonlinear state space models in ecology and epidemiology," *Stat. Sci.*, vol. 31, no. 1, pp. 96–118, Feb. 2016.
[14] R. Frey and W. Runggaldier, "Pricing credit derivatives under incomplete information: A nonlinear-filtering approach," *Finance Stochastics*, vol. 14, no. 4, pp. 495–526, Dec. 2010.
[15] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEEE Proc. F Radar Signal Process.*, vol. 140, no. 2, p. 107, 1993.
[16] S. L. Heston, "A closed-form solution for options with stochastic volatility with applications to bond and currency options," *Rev. Financial Stud.*, vol. 6, no. 2, pp. 327–343, Apr. 1993.
[17] A. Javaheri, *Inside Volatility Filtering: Secrets Skew*. Hoboken, NJ, USA: Wiley, 2015.
[18] M. S. Johannes, N. G. Polson, and J. R. Stroud, "Optimal filtering of jump diffusions: Extracting latent states from asset prices," *Rev. Financial Stud.*, vol. 22, no. 7, pp. 2759–2799, Jul. 2009.
[19] G. Kitagawa, "Monte Carlo filter and smoother for non-Gaussian nonlinear state space models," *J. Comput. Graph. Statist.*, vol. 5, no. 1, pp. 1–25, Mar. 1996.
[20] M. A. Kouritzin, "Residual and stratified branching particle filters," *Comput. Statist. Data Anal.*, vol. 111, pp. 145–165, Jul. 2017.
[21] M. A. Kouritzin, "Convergence rates for residual branching particle filters," *J. Math. Anal. Appl.*, vol. 449, no. 2, pp. 1053–1093, May 2017.
[22] M. A. Kouritzin and A. MacKay, "Branching particle pricers with Heston examples," *Int. J. Theor. Appl. Finance*, vol. 23, no. 01, Feb. 2020, Art. no. 2050003.
[23] M. A. Kouritzin and Y. Zeng, "Bayesian model selection via filtering for a class of micro-movement models of asset price," *Int. J. Theor. Appl. Finance*, vol. 8, no. 1, pp. 97–121, Jan. 2005.
[24] M. A. Kouritzin, F. Newton, and B. Wu, "On random field CAPTCHA generation," *IEEE Trans. Image Process.*, vol. 22, no. 4, pp. 1656–1666, Apr. 2013.

[25] M. A. Kouritzin, F. Newton, and B. Wu, "A graph theoretic approach to simulation and classification," *Comput. Statist. Data Anal.*, vol. 70, pp. 281–294, Feb. 2014.
[26] M. A. Kouritzin, F. Newton, and B. Wu, "A flexible, real-time algorithm for simulating correlated random fields and its properties," *J. Math. Statist.*, vol. 13, no. 3, pp. 197–208, Mar. 2017.
[27] T. Li, M. Bolic, and P. M. Djuric, "Resampling methods for particle filtering: Classification, implementation, and strategies," *IEEE Signal Process. Mag.*, vol. 32, no. 3, pp. 70–86, May 2015.
[28] T.-C. Li, G. Villarrubia, S.-D. Sun, J. M. Corchado, and J. Bajo, "Resampling methods for particle filtering: Identical distribution, a new method, and comparable study," *Frontiers Inf. Technol. Electron. Eng.*, vol. 16, no. 11, pp. 969–984, Nov. 2015.
[29] J. S. Liu and R. Chen, "Sequential Monte Carlo methods for dynamic systems," *J. Amer. Stat. Assoc.*, vol. 93, no. 443, pp. 1032–1044, Aug. 1998.
[30] A. Mantoglou and J. L. Wilson, "The turning bands method for simulation of random fields using line generation by a spectral method," *Water Resour. Res.*, vol. 18, no. 5, pp. 1379–1394, Oct. 1982.
[31] V. Maroulas and A. Nebenführ, "Tracking rapid intracellular movements: A Bayesian random set approach," *Ann. Appl. Statist.*, vol. 9, no. 2, pp. 926–949, Jun. 2015.
[32] R. Nelsen, *An Introduction to Copulas*. New York, NY, USA: Springer, 2007.
[33] G. Puccetti and R. Wang, "Extremal dependence concepts," *Stat. Sci.*, vol. 30, no. 4, pp. 9485–9517, 2015.
[34] M. J. L. Robin, A. L. Gutjahr, E. A. Sudicky, and J. L. Wilson, "Cross-correlated random field generation with the direct Fourier transform method," *Water Resour. Res.*, vol. 29, no. 7, pp. 2385–2397, Jul. 1993.
[35] M. Shinozuka, "Simulation of multivariate and multidimensional random processes," *J. Acoust. Soc. Amer.*, vol. 49, no. 1B, pp. 357–368, Jan. 1971.
[36] N. Vellone-Scott, "Calibration du modèle de Heston avec filtres à particles et ré-échantillonnage par branchement," M.S. thesis, Dept. Math., UQAM, Montreal, QC, Canada, 2020.
[37] M. Vořechovský, "Simulation of simply cross correlated random fields by series expansion methods," *Struct. Saf.*, vol. 30, no. 4, pp. 337–363, Jul. 2008.
[38] B. Wang and R. Wang, "Joint mixability," *Math. Oper. Res.*, vol. 41, no. 3, pp. 808–826, Aug. 2016.
[39] X. Yu, J. Li, and J. Xu, "Nonlinear filtering in unknown measurement noise and target tracking system by variational Bayesian inference," *Aerosp. Sci. Technol.*, vol. 84, pp. 37–55, Jan. 2019.

MICHAEL A. KOURITZIN was born in Vancouver, Canada, in 1964. He received the B.A.Sc. and Ph.D. degrees in electrical engineering from the University of Waterloo, Canada, in 1987 and 1991, respectively.

From 1990 to 1992, he was an Assistant Professor in electrical and computer engineering with the University of Waterloo. He was an NSERC and DAAD Sponsored Research Assistant with the Institut für Mathematische Stochastik, University of Freiburg, Germany, and the Department of Mathematics, Carleton University, Canada, in 1993 and 1994, respectively. He was a Postdoctoral Fellow with the Institute of Mathematics and its Applications, University of Minnesota, USA, in 1995, where he also worked with Lockheed Martin Corporation. In 1997 and 2001, he became an Associate and a Full Professor of mathematics and statistics with the University of Alberta. He has also held positions at Toulouse University, France; the University of Wisconsin, Madison, USA; HEC Montreal, Canada; The Hong Kong Polytechnic University, Hong Kong; the University of Copenhagen, Denmark; the University of Beijing, China; and the South University of Science and Technology, China. He ran the MITACS Canadian National Center of Excellence across three universities interacting with five companies from 1998 to 2005. He has published over 70 articles in mathematics, statistics, and engineering; and authored five patents.

Dr. Kouritzin won the PIMS Canadian Outreach Prize in 2001.

ANNE MACKAY was born in Quebec City, QC, Canada, in 1984. She received the B.Sc. degree in actuarial science from Université Laval, in 2007, the M.Sc. degree in mathematics from Concordia University, in 2011, and the Ph.D. degree in actuarial science from the University of Waterloo, in 2014.

From 2014 to 2016, she was a Postdoctoral Fellow at Risklab, ETH Zurich. From 2016 to 2020, she was an Assistant Professor with the Université du Québec à Montréal, where she has been an Associate Professor, since 2020. Her research interests include numerical methods in finance and insurance, and quantitative risk management of long-term financial risk.

Dr. MacKay has been a Fellow of the Society of Actuaries since 2012 and an Associate of the Canadian Institute of Actuaries since 2019.

NICOLAS VELLONE-SCOTT was born in Laval, QC, Canada, in 1995. He received the B.Sc. degree in actuarial science from the Université du Québec à Montréal, in 2017, where he is currently pursuing the M.Sc. degree in mathematics with a concentration in financial and actuarial mathematics, under the supervision of Prof. Anne MacKay and Prof. Clarence Simard.

From 2018 to 2020, he was a Teaching and a Research Assistant with the Université du Québec à Montréal. His research interest includes numerical methods in finance and insurance.

Mr. Vellone-Scott's awards and honors include scholarships from Fonds de recherche du Québec–Nature et technologies (FRQNT) and the Canadian Institute of Actuaries.