# Decentralized, Adaptive, Look-Ahead Particle Filtering

Mohamed Osama Ahmed[*], Pouyan T. Bibalan, Nando de Freitas and Simon Fauvel
CS and ECE Departments
University of British Columbia
Vancouver, Canada
{mohameda,pouyant,simonf}@ece.ubc.ca and nando@cs.ubc.ca

March 13, 2012

**Abstract**

The decentralized particle filter (DPF) was proposed recently to increase the level of parallelism of particle filtering. Given a decomposition of the state space into two nested sets of variables, the DPF uses a particle filter to sample the first set and then conditions on this sample to generate a set of samples for the second set of variables. The DPF can be understood as a variant of the popular Rao-Blackwellized particle filter (RBPF), where the second step is carried out using Monte Carlo approximations instead of analytical inference. As a result, the range of applications of the DPF is broader than the one for the RBPF. In this paper, we improve the DPF in two ways. First, we derive a Monte Carlo approximation of the optimal proposal distribution and, consequently, design and implement a more efficient look-ahead DPF. Although the decentralized filters were initially designed to capitalize on parallel implementation, we show that the look-ahead DPF can outperform the standard particle filter even on a single machine. Second, we propose the use of bandit algorithms to automatically configure the state space decomposition of the DPF.

## 1 Introduction

Without a doubt, Rao-Blackwellization has proved to be the most successful technique for enabling particle filters to solve high-dimensional dynamic inference problems, see for example [1, 2, 3, 4] and the many citations to those papers. When applying Rao-Blackwellization to particle filtering, one decomposes the state space into two groups of variables. The first group of variables is sampled with a particle filter. Then one conditions on these samples to compute the sufficient statistics of the second group of variables analytically. If a decomposition exists such that the dimension of the sampled variables is small while the dimension of the analytical variables is large, then one can effectively solve high dimensional problems. An example of this, of great practical relevance, is the application of RBPFs to jump-Markov linear Gaussian systems [5, 6, 7, 8].

These works have however left open some important questions: (i) What happens if there is no analytical expression for the distribution of the second group of variables? (ii) Is there a reason for deriving an approximate Rao-Blackwellized Particle Filter (RBPF) in this case? (iii) Instead of

[*]Authorship in alphabetical order.

only two groups of variables, can one use successive nesting of more than two groups? (iv) How do we decompose the state space automatically?

Chen *et. al.* (2011) have recently provided answers to questions (i) and (ii). They designed a new particle filter, which they named the Decentralized Particle Filter (DPF), that is effectively an RBPF, but with the difference that the distribution of the second group of variables is also approximated by a conditional particle filter. That is, one uses a particle filter to sample the first group and then conditions on these samples to sample the second group with a conditional particle filter. They provide an important reason for doing this: increased parallelization.

The resampling step is one of the computational bottlenecks in parallel implementations of particle filters in graphics processing units (GPUs) and field-programmable gate arrays (FPGAs) [10, 11, 12]. By decomposing the state space, the DPF allows for more efficient, local in the state space, resampling. Chen and colleagues have demonstrated this advantage of DPFs over standard PFs. Moreover, with increased interest in the deployment of particle filters for large scale applications, such as the analysis of streaming news [13], algorithms that capitalize on decompositions of the space space are of great research interest.

Chen *et. al.* suggest the use of Gaussian approximations in order to manage computation. In this paper, we show that it is possible to avoid these Gaussian approximations without significant loss of performance. Moreover, we show that it is possible to obtain a pure Monte Carlo approximation of the optimal importance distribution (optimal proposal). This Monte Carlo approximation enables us to design and implement a look-ahead filter, where the sampling and resampling steps can be swapped. In the context of exact Rao-Blackwellization, this look-ahead strategy is described in detail in [5] and was first suggested in [1]. In [5], it was clear that the look-ahead RBPF performed significantly better than the PF and RBPF algorithms in practical domains. In our context, the derivation of a look-ahead strategy is a bit more tricky as it involves additional Monte Carlo approximations. However, as we will see in the experiments, the look-ahead strategy still results in substantial improvements over the standard DPF.

Our final contribution is to answer, to some extent and for the first time, question (iv). This question was posed more than ten years ago and continues appearing in the future work sections of papers on the topic, including the DPF paper. Our solution involves the usage of online bandit algorithms [14] to decide the order in which variables should be sampled. Question (iii) is still open, but we conjecture that the improvements introduced in [9] and here will lead to it being answered in the near future.

The paper is organized as follows. Section 2 describes the models, poses the inference problems, and provides a brief description of the DPF. The section ends with a description of the proposed look ahead (LA)-DPF algorithm. Section 3 presents the automatic state decomposition strategy. The PF, DPF and LA-DPF are compared in the experiments of Section 4. We conclude the paper in Section 5.

$$\begin{tikzcd}
\boxed{p(x_{0:t}|y_{0:t-1})} \arrow[d] & & & \boxed{p(z_t|x_{0:t}, y_{0:t-1})} \arrow[dll] \arrow[dd] \\
\boxed{p(x_{0:t}|y_{0:t})} \arrow[dd] & \boxed{p(y_t|x_{0:t}, y_{0:t-1})} \arrow[l] & \boxed{p(y_t|x_t, z_t)} \arrow[l] \arrow[dr] & \\
 & & & \boxed{p(z_t|x_{0:t}, y_{0:t})} \arrow[dd] \\
\boxed{p(x_{0:t+1}|y_{0:t})} & \boxed{p(x_{t+1}|x_{0:t}, y_{0:t})} \arrow[l] & \boxed{p(x_{t+1}|x_t, z_t)} \arrow[l] \arrow[dr] & \\
 & & & \boxed{p(z_t|x_{0:t+1}, y_{0:t})} \arrow[d] \\
 & & \boxed{p(z_{t+1}|x_{t:t+1}, z_t)} \arrow[r] & \boxed{p(z_{t+1}|x_{0:t+1}, y_{0:t})}
\end{tikzcd}$$

Figure 1: *Flow of distributions that need to be computed in order to solve the nested filtering problem.*

## 2 Decentralized Particle Filter

The state space is decomposed into two groups of variables ($x_t \in \mathcal{X} \subseteq \mathbb{R}^{n_x}, z_t \in \mathcal{Z} \subseteq \mathbb{R}^{n_z}$), which are governed by the following latent, dynamic state space model with observations $y_t \in \mathcal{Y} \subseteq \mathbb{R}^{n_y}$:

$$\begin{aligned} x_{t+1} &= f_t^x(x_t, z_t, v_t^x) \\ z_{t+1} &= f_t^z(x_{t:t+1}, z_t, v_t^z) \\ y_t &= h_t(x_t, z_t, e_t), \end{aligned} \tag{1}$$

where $x_{t:t+1} = (x_t, x_{t+1})$, $v_t = (v_t^x, v_t^z)$ and $e_t$ are noise processes, and $f(\cdot)$ and $h(\cdot)$ are nonlinear mappings. This model can be equivalently expressed in terms of the initial distributions $p(x_0)$ and $p(z_0|x_0)$, the transition distributions $p(x_{t+1}|x_t, z_t)$ and $p(z_{t+1}|x_{t:t+1}, z_t)$ and the observation model $p(y_t|x_t, z_t)$. We assume that the parameters of these distributions are known and focus on the inference problem.

The goal of inference is to recursively estimate the posterior distribution $p(z_t, x_{0:t}|y_{0:t})$. Using the following factorization:

$$p(z_t, x_{0:t}|y_{0:t}) = p(z_t|x_{0:t}, y_{0:t})p(x_{0:t}|y_{0:t}), \tag{2}$$

with $x_{0:t} = (x_0, \ldots, x_t)$, this filtering problem can be split into the two nested subproblems of recursively estimating (1) $p(x_{0:t}|y_{0:t})$ and (2) $p(z_t|x_{0:t}, y_{0:t})$. The way in which these subproblems interact with each other is depicted in Figure 1. The diagram shows the necessary steps for implementing the optimal filter in this nested setting. However, except in very specific cases, there is no analytic solution to this filtering problem. Therefore, a numerical algorithm must be employed. The DPF is one such algorithm. It handles the two nested subproblems using particle filters: subproblem 1 is dealt with using a PF with $N_x$ particles, and subproblem 2 is dealt with using $N_x$ PFs with $N_z$ particles each. The DPF solves these two nested subproblems in 7 steps [9], as illustrated in Figure 2. We only give a brief overview of the main steps of the DPF and state the important equations required for deriving our look-ahead DPF algorithm. We refer the reader to [9] for all the mathematical details involved in deriving the DPF algorithm and to [15] for an introduction to particle filtering.

We describe the 7 steps briefly. Assume that we have Monte Carlo approximations of the

**Initialize** the particles $\tilde{x}_0^{(i)} \sim p(x_0), i = 1, \ldots, N_x$, and for each particle $\tilde{x}_0^{(i)}$, the particles $\tilde{z}_0^{(i,j)} \sim p(z_0|\tilde{x}_0^{(i)})$, $j = 1, \ldots, N_z$. Initialize $r_1$.

*At each time* ($t \geq 0$)*, perform the following 7 steps:*

1. **Measurement update of** $x_{0:t}$ **given** $y_t$. Calculate the importance weights $w_t^{(i)}$, $i = 1, \ldots, N_z$ according to:
$$w_t^{(i)} \propto \frac{p_{N_z}(y_t|\tilde{x}_{0:t}^{(i)}, y_{0:t-1})p_{N_z}(\tilde{x}_t^{(i)}|x_{0:t-1}^{(i)}, y_{0:t-1})}{\pi(\tilde{x}_t^{(i)}|x_{0:t-1}^{(i)}, y_{0:t-1})}; \quad \sum_{i=1}^{N_x} w_t^{(i)} = 1\,.$$
2. **Resample** $\{\tilde{x}_{0:t}^{(i)}, \bar{z}_t^{(i,1)}, \tilde{r}_t^{(i,1)}, \ldots, \bar{z}_t^{(i,N_z)}, \tilde{r}_t^{(i,N_z)}\}$, $i = 1, \ldots, N_x$ according to $w_t^{(i)}$ to generate particles $\{x_{0:t}^{(i)}, \bar{z}_t^{(i,1)}, r_t^{(i,1)}, \ldots, \bar{z}_t^{(i,N_z)}, r_t^{(i,N_z)}\}$, $i = 1, \ldots, N_x$.
3. **Measurement update of** $z_t$ **given** $y_t$. For $i = 1, \ldots, N_x$, the importance weights $\bar{q}_t^{(i,j)}$, $j = 1, \ldots, N_z$, are evaluated according to:
$$\bar{q}_t^{(i,j)} \propto p(y_t|x_t^{(i)}, \bar{z}_t^{(i,j)})r_t^{(i,j)}; \quad \sum_{j=1}^{N_z} \bar{q}_t^{(i,j)} = 1\,.$$
4. **Propose particles** $\tilde{x}_{t+1}^{(i)}$, $i = 1, \ldots, N_x$ according to the proposal function $\pi(x_{t+1}|x_{0:t}^{(i)}, y_{0:t})$.
5. **Measurement update of** $z_t$ **given** $\tilde{x}_{t+1}$. For $i = 1, \ldots, N_x$, the importance weights $q_t^{(i,j)}$, $j = 1, \ldots, N_z$, are evaluated according to:
$$q_t^{(i,j)} \propto p(y_t|x_t^{(i)}, \bar{z}_t^{(i,j)})p(\tilde{x}_{t+1}^{(i)}|x_t^{(i)}, \bar{z}_t^{(i,j)})r_t^{(i,j)}; \quad \sum_{j=1}^{N_z} q_t^{(i,j)} = 1\,.$$
6. **Resample** $\bar{z}_t^{(i,j)}, i = 1, \ldots, N_x, j = 1, \ldots, N_z$ according to $q_t^{(i,j)}$ to obtain $z_t^{(i,j)}$.
7. **Propose particles** $\tilde{z}_{t+1}^{(i,j)}, i = 1, \ldots, N_x, j = 1, \ldots, N_z$ according to the proposal function $\pi(z_{t+1}|\tilde{x}_{0:t+1}^{(i)}, y_{0:t})$. Set $\tilde{x}_{0:t+1}^{(i)} = (x_{0:t}^{(i)}, \tilde{x}_{t+1}^{(i)})$ and compute $\tilde{r}_{t+1}$ :
$$\tilde{r}_{t+1}^{(i,j)} = \frac{\tilde{p}_{N_z}(\tilde{z}_{t+1}^{(i,j)}|\tilde{x}_{0:t+1}^{(i)}, y_{0:t})}{\pi(\tilde{z}_{t+1}^{(i,j)}|\tilde{x}_{0:t+1}^{(i)}, y_{0:t})}$$

Figure 2: *The DPF algorithm.*

distributions of interest from the previous time step:

$$\begin{aligned} \tilde{p}_{N_x}(x_{0:t-1}|y_{0:t-1}) &= \frac{1}{N_x}\sum_{i=1}^{N_x} \delta(x_{0:t-1} - x_{0:t-1}^{(i)}) \\ p_{N_z}(z_{t-1}|x_{0:t-1}^{(i)}, y_{0:t-1}) &= \sum_{j=1}^{N_z} \bar{q}_{t-1}^{(i,j)} \delta(z_{t-1} - \bar{z}_{t-1}^{(i,j)}), \end{aligned} \tag{3}$$

where $\bar{q}_{t-1}^{(i,j)}$ is an importance weight defined in equation (7). Assume that we have samples $\tilde{x}_{0:t}^{(i)}|_{i=1}^{N_x}$ from a proposal distribution $\pi(\tilde{x}_t|x_{0:t-1}^{(i)}, y_{0:t-1})$. Then, importance sampling enables us to obtain the following approximation of the posterior distribution of $x_{0:t}$:

$$p_{N_x}(x_{0:t}|y_{0:t}) = \sum_{i=1}^{N_x} w_t^{(i)} \delta(x_{0:t} - \tilde{x}_{0:t}^{(i)}).$$

As in standard particle filtering, the importance weights $w_t^{(i)}$ are given by:

$$w_t^{(i)} \propto \frac{p_{N_z}(y_t|\tilde{x}_{0:t}^{(i)}, y_{0:t-1}) p_{N_z}(\tilde{x}_t^{(i)}|x_{0:t-1}^{(i)}, y_{0:t-1})}{\pi(\tilde{x}_t^{(i)}|x_{0:t-1}^{(i)}, y_{0:t-1})}.$$

However, unlike in simple Markov processes, we cannot exploit conditional independence in a trivial manner so as to simplify the numerator. Instead, we express the quantities in the numerator in terms of the following marginals

$$\begin{aligned} p(x_t|x_{0:t-1}, y_{0:t-1}) &= \int p(x_t|x_{t-1}, z_{t-1}) p(z_{t-1}|x_{0:t-1}, y_{0:t-1}) dz_{t-1} \\ p(y_t|x_{0:t}, y_{0:t-1}) &= \int p(y_t|x_t, z_t) p(z_t|x_{0:t}, y_{0:t-1}) dz_t \end{aligned}$$

and approximate them with the following Monte Carlo estimates:

$$p_{N_z}(\tilde{x}_t|x_{0:t-1}^{(i)}, y_{0:t-1}) = \sum_{j=1}^{N_z} \bar{q}_{t-1}^{(i,j)} p(\tilde{x}_t|x_{t-1}^{(i)}, \bar{z}_{t-1}^{(i,j)}) \quad (4)$$

$$p_{N_z}(y_t|\tilde{x}_{0:t}^{(i)}, y_{0:t-1}) = \sum_{j=1}^{N_z} \tilde{r}_t^{(i,j)} p(y_t|\tilde{x}_t, \tilde{z}_t^{(i,j)}) / \sum_{j=1}^{N_z} \tilde{r}_t^{(i,j)}. \quad (5)$$

The first expression is a simple Monte Carlo estimate obtained by replacing $p(z_{t-1}|x_{0:t-1}, y_{0:t-1})$ with its approximation $p_{N_z}(z_{t-1}|x_{0:t-1}^{(i)}, y_{0:t-1})$. In the second expression, we use importance sampling to approximate the integral. In particular, we assume that we have already computed the importance weight $\tilde{r}_t^{(i,j)}$, defined as follows:

$$\tilde{r}_t^{(i,j)} = \frac{\tilde{p}_{N_z}(\tilde{z}_t^{(i,j)}|\tilde{x}_{0:t}^{(i)}, y_{0:t-1})}{\pi(\tilde{z}_t^{(i,j)}|\tilde{x}_{0:t}^{(i)}, y_{0:t-1})}, \quad (6)$$

where $\tilde{z}_t^{(i,j)}$ are samples from the proposal mechanism $\pi(z_t|\tilde{x}_{0:t}^{(i)}, y_{0:t-1})$. Note that to compute this importance weight, we require an expression for $\tilde{p}_{N_z}(\tilde{z}_t^{(i,j)}|\tilde{x}_{0:t}^{(i)}, y_{0:t-1})$. To achieve this, we need to first obtain expressions for $p(z_t|x_{0:t}, y_{0:t})$ and $p(z_t|x_{0:t+1}, y_{0:t})$.

Using Bayes rule and conditional independence, we have:

$$p(z_t|x_{0:t}^{(i)}, y_{0:t}) \propto p(z_t|x_{0:t}^{(i)}, y_{0:t-1}) p(y_t|x_t^{(i)}, z_t).$$

Since $\pi(z_t|\tilde{x}_{0:t}^{(i)}, y_{0:t-1})$ is the proposal mechanism from which the samples $\bar{z}_t^{(i,j)}$ originated, importance sampling yields the following approximation:

$$p_{N_z}(z_t|x_{0:t}^{(i)}, y_{0:t}) = \sum_{j=1}^{N_z} \bar{q}_t^{(i,j)} \delta(z_t - \bar{z}_t^{(i,j)}),$$

where

$$\bar{q}_t^{(i,j)} \propto p(y_t|x_t^{(i)}, \bar{z}_t^{(i,j)})r_t^{(i,j)}. \tag{7}$$

Similarly, by two successive applications of Bayes rule, we have:

$$p(z_t|\tilde{x}_{0:t+1}^{(i)}, y_{0:t}) \propto p(z_t|x_{0:t}^{(i)}, y_{0:t-1})p(y_t|x_t^{(i)}, z_t)p(\tilde{x}_{t+1}^{(i)}|x_t^{(i)}, z_t).$$

Using importance sampling, the approximation for this distribution is given by:

$$p_{N_z}(z_t|\tilde{x}_{0:t+1}^{(i)}, y_{0:t}) = \sum_{j=1}^{N_z} q_t^{(i,j)}\delta(z_t - \bar{z}_t^{(i,j)}),$$

where

$$q_t^{(i,j)} \propto p(y_t|x_t^{(i)}, \bar{z}_t^{(i,j)})p(\tilde{x}_{t+1}^{(i)}|x_t^{(i)}, \bar{z}_t^{(i,j)})r_t^{(i,j)}. \tag{8}$$

Using marginalization and conditional independence, we have

$$p(z_{t+1}|x_{0:t+1}, y_{0:t}) = \int p(z_{t+1}|x_{t:t+1}, z_t)p(z_t|x_{0:t+1}, y_{0:t})dz_t.$$

A Monte Carlo approximation of this quantity results in the expression necessary for computing the numerator of $\tilde{r}_{t+1}$ :

$$\tilde{p}_{N_z}(z_{t+1}|\tilde{x}_{0:t+1}^{(i)}, y_{0:t}) = \frac{1}{N_z}\sum_{j=1}^{N_z} p(z_{t+1}|\tilde{x}_{t:t+1}^{(i)}, z_t^{(i,j)})$$

Note that in contrast to what is done in [9], no further approximation of (4) will be made in the remainder of the derivation. We refer the reader to the results section for more details on this.

In the algorithm, shown in Figure 2, the weights $\bar{q}_{t-1}$ and $\tilde{r}_t$ were computed in steps 3 and 7 respectively. Steps 2 and 6 are standard resampling steps [15]. One has to be careful keeping tracks of indices, tildes and bars, but aside from this, the algorithm follows easily from the standard importance sampling steps for particle filtering.

## 2.1 On the choice of proposal distribution

A common practice is to use the prior distributions as proposal distributions:

$$\pi(x_t|x_{0:t-1}^{(i)}, y_{0:t}) = p(x_t|x_{0:t-1}^{(i)}, y_{0:t-1})$$

and

$$\pi(z_t|x_{0:t}^{(i)}, y_{0:t}) = p(z_t|x_{0:t}^{(i)}, y_{0:t-1}).$$

These proposals are both intuitive and reduce the complexity of the derivations considerably. However, they do not take into account the current observation $y_t$. The original DPF algorithm uses these prior proposal distributions. However, a better choice of proposal for $x_t$ that takes into account the current observation $y_t$ is given by:

$$\pi(x_t|x_{0:t-1}^{(i)}, y_{0:t}) = p(x_t|x_{0:t-1}^{(i)}, y_{0:t}).$$

It can be shown that this importance distribution is optimal [16].

Using Bayes rule, the optimal proposal distribution can be written as:

$$\pi(x_t|x_{0:t-1}^{(i)}, y_{0:t}) \quad = \quad \frac{p(y_t|x_{0:t-1}^{(i)}, x_t, y_{0:t-1})p(x_t|x_{0:t-1}^{(i)}, y_{0:t-1})}{p(y_t|y_{0:t-1}, x_{0:t-1}^{(i)})}. \tag{9}$$

This expression results in the following simplification of the importance weights for $x$:

$$w_t^{(i)} = \frac{p(x_{0:t}|y_{0:t})}{\pi(x_{0:t}|y_{0:t})} \propto \frac{p(y_t|x_{0:t}, y_{0:t-1})p(x_t|x_{0:t-1}, y_{0:t-1})}{\pi(x_t|x_{0:t-1}, y_{0:t})} = p(y_t|y_{0:t-1}, x_{0:t-1}^{(i)}).$$

The predictive distribution $p(y_t|y_{0:t-1}, x_{0:t-1}^{(i)})$ can be expanded as follows:

$$p(y_t|y_{0:t-1}, x_{0:t-1}^{(i)}) = \iint p(y_t|x_t, z_t)p(z_t|y_{0:t-1}, x_{0:t-1}^{(i)}, x_t)p(x_t|y_{0:t-1}, x_{0:t-1}^{(i)})dx_t dz_t$$

Note that $w_t^{(i)}$ does not depend on the value of the sample drawn from $\pi(x_t|x_{0:t-1}^{(i)}, y_{0:t})$.

The optimal importance density suffers from two major drawbacks: it requires the ability to sample from $\pi(x_t|x_{0:t-1}^{(i)}, y_{0:t})$ and to evaluate the integral over the new states in the calculation of the importance weights. In general, both of these steps are hard. There are two cases when the use of the optimal importance density is possible: when $\pi(x_t|x_{0:t-1}^{(i)}, y_{0:t})$ is a member of a finite set, *e.g.* a jump-Markov linear system [1], or when $\pi(x_t|x_{0:t-1}^{(i)}, y_{0:t})$ is Gaussian [16].

In our case, we can use the Monte Carlo estimates (4) and (5) to obtain an approximation of $\pi(x_t|x_{0:t-1}^{(i)}, y_{0:t})$:

$$\begin{aligned}\hat{\pi}_{N_z}(x_t|x_{0:t-1}^{(i)}, y_{0:t}) \quad &= \quad \frac{p_{N_z}(y_t|x_{0:t-1}^{(i)}, x_t, y_{0:t-1})p_{N_z}(x_t|x_{0:t-1}^{(i)}, y_{0:t-1})}{p_{N_z}(y_t|y_{0:t-1}, x_{0:t-1}^{(i)})} \\ &= \quad \frac{\sum_{j=1}^{N_z} r_t^{(i,j)} p(y_t|x_t, z_t^{(i,j)}) / \sum_{j=1}^{N_z} r_t^{(i,j)} \cdot \sum_{j=1}^{N_z} q_{t-1}^{(i,j)} p(x_t|x_{t-1}^{(i)}, z_{t-1}^{(i,j)})}{p_{N_z}(y_t|y_{0:t-1}, x_{0:t-1}^{(i)})},\end{aligned} \tag{10}$$

where $p_{N_z}(y_t|y_{0:t-1}, x_{0:t-1}^{(i)})$ is a Monte Carlo approximation of $p(y_t|y_{0:t-1}, x_{0:t-1}^{(i)})$, which we derive next. First, we draw $N_x$ samples $\bar{\bar{x}}_t^{(m)}, m = 1, \ldots, N_x$ using $p_{N_z}(x_t|x_{0:t-1}^{(i)}, y_{0:t-1})$. Then, we draw $N_z$ corresponding samples $\bar{\bar{z}}_t^{(m,k)}, k = 1, \ldots, N_z$ according to $p_{N_z}(z_t|y_{0:t-1}, x_{0:t-1}^{(i)}, \bar{\bar{x}}_t^{(m)})$. Now, the weights can be expanded as follows:

$$\hat{w}_t^{(i)} \quad \propto \quad \frac{1}{N_x}\frac{1}{N_z}\sum_{m=1}^{N_x}\sum_{k=1}^{N_z} p(y_t|\bar{\bar{x}}_t^{(m)}, \bar{\bar{z}}_t^{(m,k)}). \tag{11}$$

The cost of the algorithm is $N_x \times N_z$. Since $\mathcal{X}$ and $\mathcal{Z}$ are lower dimensional than $\mathcal{X} \times \mathcal{Z}$, the hope is that $N_x \times N_z$ is still lower than the the number of particles $N$ required by a standard particle filter on the joint space $\mathcal{X} \times \mathcal{Z}$. Moreover, since typically $N_x$ and $N_z$ are much smaller than $N$, the resampling steps of the DPF are much cheaper than usual $O(N)$ cost of the standard PF. This can therefore result in significant computational gains when parallelizing the algorithm for real-time applications.

## 2.2 Look-ahead DPF algorithm

As mentioned in [5], one more improvement is possible when using the optimal proposal distribution. The optimal importance weights don't depend on $x_t$ or $z_t$, as we are in fact marginalizing over these variables. Therefore, we can swap the resampling and proposal steps. This enables us to resample (select the fittest) particles at time $t-1$ using information from the future time $t$. We refer the reader to Figure 3 for an intuitive diagram highlighting the benefits of this. Figure 4 illustrates the 4 steps of the look-ahead DPF algorithm.

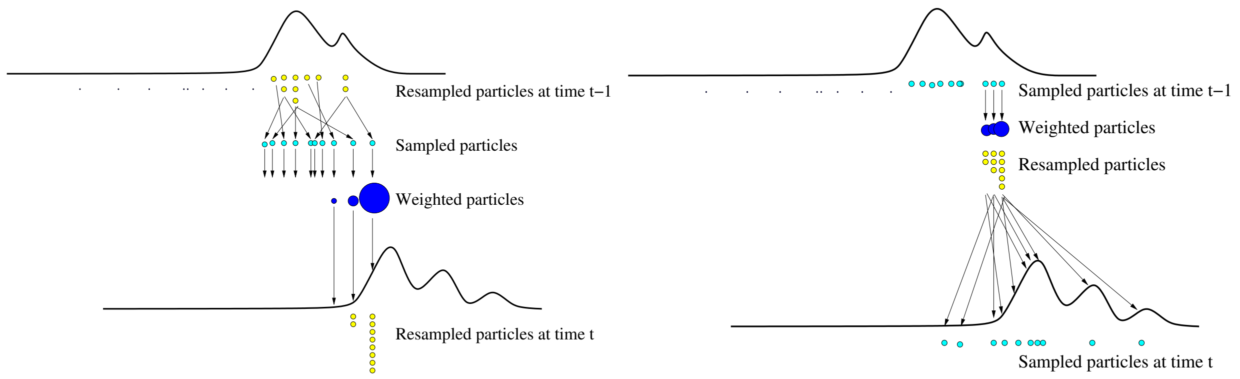

Resampled particles at time t−1
Sampled particles
Weighted particles
Resampled particles at time t
Sampled particles at time t−1
Weighted particles
Resampled particles
Sampled particles at time t

Figure 3: *PF (left) and look-ahead PF (right) algorithms for a continuous one-dimensional problem. For the PF, starting with the resampled particles at $t-1$, a new set of particles is proposed at time $t$. We compute the importance weight of each particle. Finally, we select the fittest particles according to their weights. Note the PF has failed to track the two modes appearing on the right of the filtering posterior distribution at time $t$. On the other hand, for the look-ahead filter, we first compute the importance weights. After resampling according to these weights, we propose new particles at time $t$. With this algorithm, we are more likely to propose particles in areas of high probability.*

**Initialize** the particles $\tilde{x}_0^{(i)} \sim p(x_0), i = 1, \ldots, N_x$, and for each particle $\tilde{x}_0^{(i)}$, the particles $\tilde{z}_0^{(i,j)} \sim p(z_0|\tilde{x}_0^{(i)}), j = 1, \ldots, N_z$.

*At each time ($t \geq 0$), perform the following 4 steps:*

1. **Generate $N'_x$ particles** $\bar{\bar{x}}^{(m)} \sim p_{N_z}(x_t|x_{0:t-1}^{(i)}, y_{0:t-1}), m = 1, \ldots, N'_x$, and corresponding $N'_z$ particles $\bar{\bar{z}}_t^{(m,k)} \sim p(z_t|y_{0:t-1}, x_{0:t-1}^{(i)}, \bar{\bar{x}}_t^{(m)}), k = 1, \ldots, N'_z$.
2. **Compute the weights** $w_t^{(i)}$ according to (11).
3. **Resampling**. Multiply or discard particles $\{\tilde{x}_{0:t-1}^{(i)}, \tilde{z}_{0:t-1}^{(i,j)}\}$ using the importance weights $w_t^{(i)}$ to obtain $\{x_{0:t-1}^{(i)}, z_{0:t-1}^{(i,j)}\}$.
4. **Generate new particles**. Obtain $x_t^{(i)} \sim \pi(x_t|x_{0:t-1}^{(i)}, y_{0:t})$ and $z_t^{(i,j)} \sim p(z_t|x_{0:t}^{(i)}, y_{0:t-1})$.

Figure 4: *The look-ahead DPF algorithm.*

```
Initialization: Choose a real number $\eta > 0$. Set $G_i(0) = 0$ for $i = 1, \dots, K$.
Repeat for $t = 1, 2, \dots$:
    1. Choose action $i_t$ according to the distribution: $p_i(t) = \frac{\exp(\eta G_i(t-1))}{\sum_{j=1}^{K} \exp(\eta G_j(t-1))}$.
    2. Receive the reward vector $r(t)$ and score the gain $r_{i_t}(t)$.
    3. Set $G_i(t) = G_i(t-1) + r_i(t)$ for $i = 1, \dots, K$.
```

```
Initialization: Choose $\gamma \in (0, 1]$. Initialize Hedge($\eta$).
Repeat for $t = 1, 2, \dots$:
    1. Get the distribution $p(t)$ from Hedge.
    2. Select action $i_t$ to be $j$ with probability $\hat{p}_j(t) = (1-\gamma)p_j(t) + \frac{\gamma}{K}$.
    3. Receive reward $r_{i_t}(t) \in [0, 1]$.
    4. Feed the simulated reward $\hat{r}(t)$ back to Hedge, where $\hat{r}_j(t) = \begin{cases} \frac{r_{i_t}(t)}{\hat{p}_{i_t}(t)} & \text{if } j = i_t \\ 0 & \text{otherwise} \end{cases}$
```

Figure 5: *The Hedge (top) and Exp3 (bottom) algorithms [14].*

## 3 Automatic State Decomposition

The decision of how to decompose the state space of a given system plays a dramatic role in the performance of the DPF. In our setting, with only two nested subproblems, the decomposition problem reduces to deciding in which order the two groups of variables should be sampled. This order can have a large impact both on the execution time and on the overall accuracy of the algorithm. If the system is non-stationary, the optimal order can change over time. For this reason, we need to design algorithms to automatically choose the optimal order.

In this paper, we will adopt classical online bandit algorithms [14] to infer the sampling order. Note that these algorithms are however applicable to the more general problem of choosing state decompositions when splitting the state-space into more than two groups of variables.

The first algorithm we consider is Hedge (Figure 5). Hedge chooses action $i$ (out of $K$ possible actions) at time $t$ with probability proportional to $\exp(\eta G_i(t-1))$ where $\eta > 0$ is a memory parameter and $G_i(t) = \sum_{t'=1}^{t} r_i(t')$ is the cumulative reward scored by action $i$ from time 1 to $t$. Actions that repeatedly yield higher rewards quickly gain a higher probability of being selected. In our case, the actions are the different nesting orders. Different reward models are possible. We choose the closeness of the observations to the predicted observations as the reward measure. One drawback of Hedge is the fact that each action must be tried to pick the best action (Hedge assumes it has full information about the rewards). This is not ideal since it introduces a large overhead, especially when the number of actions is large. For this reason, we must introduce Exp3 [14]. Exp3 stands for "Exponential-weight algorithm for Exploration and Exploitation". This algorithm assumes it has only partial information about the reward vector. Exp3 only tries one action at each iteration, and hence it only has information about one reward. Exp3 calls Hedge as a subroutine. For each time interval $t$, Exp3 receives the probability vector $p(t)$ from Hedge, and it selects an action $i_t$ according to a new distribution, $\hat{p}(t)$, which is a mixture of $p(t)$ and the uniform distribution. Using $\hat{p}(t)$ ensures that each action gets tried over time. After receiving the reward $r_{i_t}(t)$ associated with the chosen action, Exp3 must simulate a full reward vector $\hat{r}(t)$ for Hedge. That is, it must fill in the reward for the actions that were not tried (since Hedge requires this information). The pseudo-code is shown in Figure 5. Both Hedge and Exp3 have vanishing regret [14].

## 4 Simulation Results

We conduct three experiments. The first experiment compares the performance of the look-ahead (LA)-DPF, bootstrap PF and DPF algorithms. To provide a meaningful comparison with results in the literature, we use the model benchmarks adopted in [9]. These are summarized in Table 1. The significant level of nonlinearity, multi-modality and non-stationarity in these models is sufficient to cause classical filtering algorithms, such as the extended Kalman filter, to fail. The second experiment investigates the difference between using a Gaussian approximation to evaluate (4), as opposed to our Monte Carlo strategy. In the third experiment, we study the behaviour of the automatic state ordering bandit methods.

Table 1: *Models used for testing the algorithms.*

| **Model 1: 2-dimensional example** | **Model 2: 4-dimensional example** |
|---|---|
| $x_{t+1} = x_t + \frac{z_t}{1+z_t^2} + v_t^x$ | $x_{1,t+1} = 0.5x_{1,t} + 8\sin t + v_t^{x_1}$ <br> $x_{2,t+1} = 0.4x_{1,t} + 0.5x_{2,t} + v_t^{x_2}$ |
| $z_{t+1} = x_t + 0.5z_t + \frac{25z_t}{1+z_t^2} + 8\cos(1.2t) + v_t^z$ | $z_{1,t+1} = z_{1,t} + \frac{z_{2,t}}{1+z_{2,t}^2} + v_t^{z_1}$ <br> $z_{2,t+1} = z_{1,t+1} + 0.5z_{2,t} + \frac{25z_{2,t}}{1+z_{2,t}^2} + 8\cos(1.2t) + v_t^{z_2}$ |
| $y_t = \arctan(x_t) + \frac{z_t^2}{20} + e_t$ | $y_t = \frac{x_{1,t}+x_{2,t}}{1+x_{1,t}^2} + \arctan(z_{1,t}) + \frac{z_{2,t}^2}{20} + e_t$ |
| $[x_0\ z_0]^T \sim \mathcal{N}(0, I_{2\times2})$ | $[x_0\ z_0]^T \sim \mathcal{N}(0, I_{4\times4})$ |
| $v_t \sim \mathcal{N}\left(0, \begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix}\right)$ | $v_t \sim \mathcal{N}\left(0, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0.1 & 10 \end{bmatrix}\right)$ |
| $e_t \sim \mathcal{N}(0,1)$ | $e_t \sim \mathcal{N}(0,1)$ |

### 4.1 Comparison between LA-DPF, DPF, and PF

The objective of this experiment is to evaluate the performance of the proposed LA-DPF algorithm with respect to the existing DPF and standard PF algorithms. The simulations were carried out for 250 time intervals ($t = 1, \ldots, 250$) with a varying number of particles $N_x$ and $N_z$. The accuracy of the state estimate was measured using the root mean square error (RMSE) between the true state and the state estimate. The results were averaged over 500 Monte Carlo simulations. As in [9], the number of particles for the regular PF is set to $N_x(N_z + 1)$ for meaningful comparison.

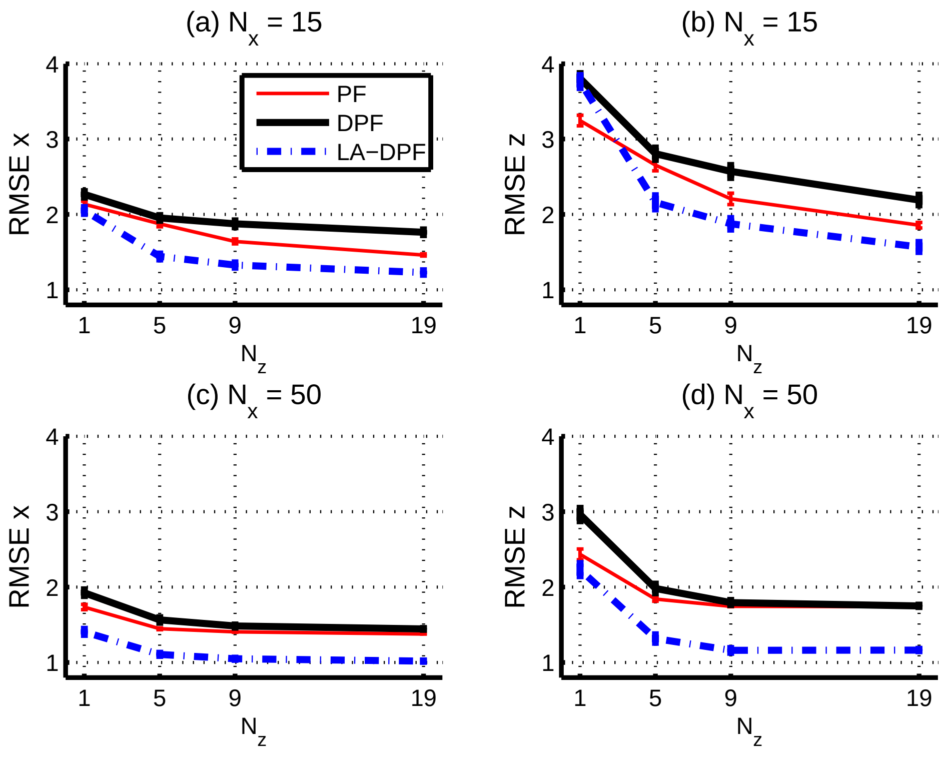


Figure 6: *Comparison among the PF, DPF and LA-DPF algorithms for model 1. The plots show the RMSE (mean and error bars) for the states $x$ and $z$ respectively as a function of the number of particles. Note that even in a sequential implementation, LA-DPF outperforms the standard PF in terms of this measure.*

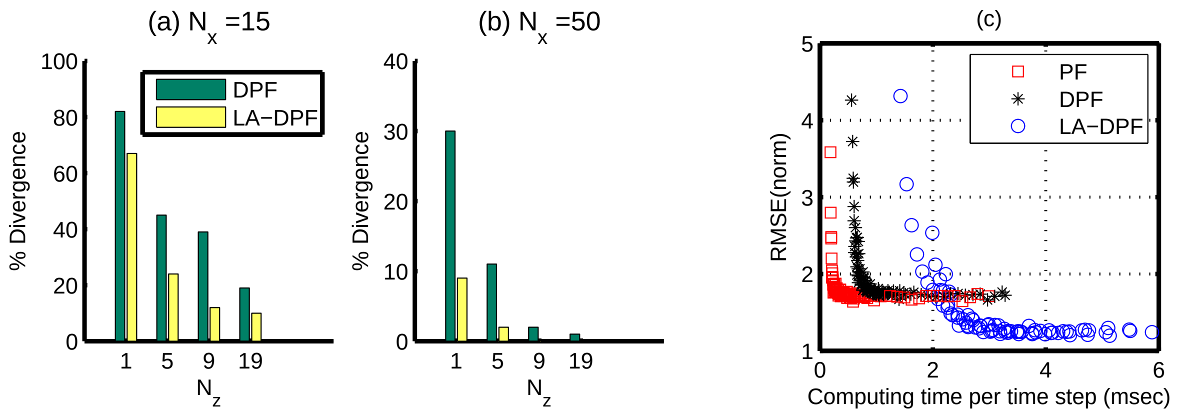


Figure 7: *Comparison of LA-DPF against DPF in terms of $\{(a),(b)\}$ divergence rate for different numbers of particles and (c) attained RMSE as a function of computing power.*

The results are summarized in Figures 6 to 9. Figure 6 shows the RMSE (mean and 90% error bars) for the $x$ and $z$ states for different numbers of particles with model 1. It is clear from these plots that although PF can have lower error than DPF in a serial implementation of the algorithms

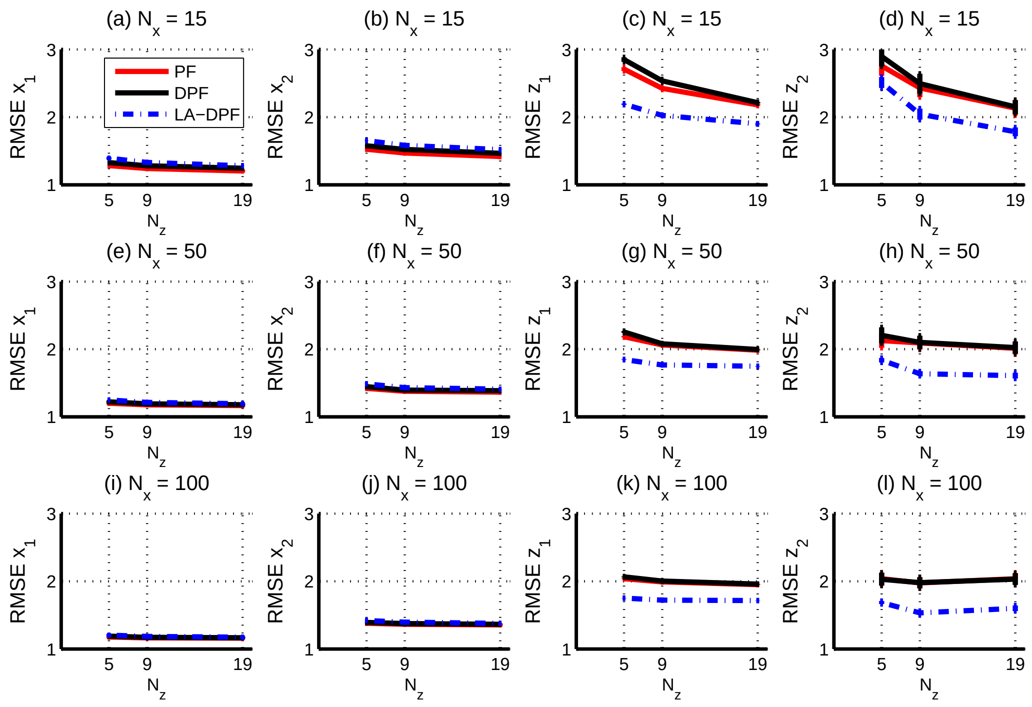


Figure 8: *Comparison among the PF, DPF and LA-DPF algorithms for model 2. The plots show the RMSE (mean and error bars) for the states $x_1$, $x_2$, $z_1$ and $z_2$ as a function of the number of particles. Note that LA-DPF outperforms both DPF and PF.*

(as also reported in [9]), the same is not true for LA-DPF. LA-DPF does significantly better as a function of the number of particles. Plots (a) and (b) of Figure 7 illustrate that for the same number of particles, the divergence rate of the DPF is higher than the one for LA-DPF. The difference is more significant for higher $N_x$. All methods work well for a reasonable number of particles. Figure 7 (c) is a scatter plot for runs with multiple random $N_x$ and $N_z$ values. It illustrates the accuracy versus computation time trade-off for DPF and LA-DPF. The latter can attain much lower error than the PF and DPF variants. We should point out that both DPF and LA-DPF can equally benefit from parallel implementation as they follow the same state-decomposition strategy to reduce the cost of resampling. We should also note that for some transition models, the execution time of LA-DPF could be decreased even further using kd-trees as proposed in [17].

Figures 8 and 9 are the results for Model 2, and are analogous to Figures 6 and 7, respectively. Again, it can be seen that LA-DPF performs significantly better than PF and DPF (or at worst, the performance is equivalent). Also, the divergence rate is much smaller for the LA-DPF than for the DPF. Similar conclusions to the ones for Model 1 can be drawn when it comes to the accuracy versus computation trade-off.

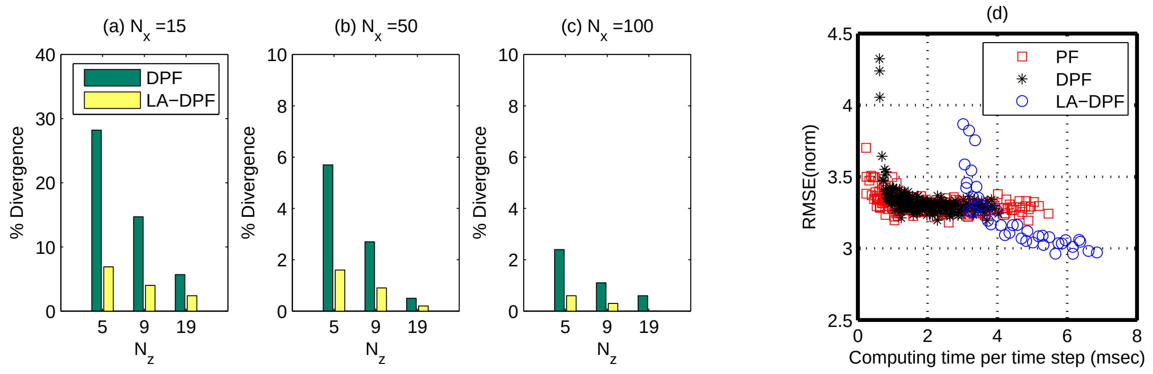


Figure 9: *The first 3 columns show the divergence plots for model 2. Again, for any number of particles the DPF has a higher likelihood of diverging than the LA-DPF. The rightmost column compares the LA-DPF against the PF and DPF in terms of attained RMSE as a function of computing power.*

Table 2: *Average RMSE for DPF (Gaussian) and DPF (Monte Carlo) algorithms,* $N_x = 100$, $N_z = 19$ *(100 runs)*

| | Gaussian approximation | Monte Carlo approximation |
|---|---|---|
| RMSE$_x$ | 1.3197 | 1.3216 |
| RMSE$_z$ | 1.1705 | 1.1640 |

## 4.2 Comparison between DPF with Gaussian and Monte Carlo approximations

We note that in [9], a Gaussian approximation is used to estimate the proposal distribution (4). It is however possible to use Monte Carlo approximations instead of Gaussian approximations. We carried experiments to verify that both approaches yield similar results for the model of Table 1. We ran the DPF with Gaussian approximation and the DPF with Monte Carlo approximation to compare their respective RMSEs. The experiment was performed 100 times, with $N_x = 100$ and $N_z = 19$. Table 2 shows the results. The results in terms of the RMSE are equivalent (within 1% of one another). However, the Monte Carlo method has the advantage of being universal in the sense that it does not require any assumption on the types of distributions that we are dealing with. It should be expected that for non-standard noise models, Gaussian approximation could result in very poor performance.

## 4.3 Automatic state decomposition for the two dimensional example

In this experiment, we use Hedge and Exp3 to choose the optimal sampling order among two actions (action 1= sampling $x$ first, then $z$, action 2 = sampling $z$ first, then $x$). We tried each action separately and verified that this ordering has a large effect on the RMSE.

We designed the reward function to be $r_{i_t} = \frac{\alpha}{\alpha+\epsilon_t^2}(0 < r_{i_t} \leq 1)$, where $\alpha$ is a small number (say 0.001) and $\epsilon_t$ is the difference between the one-step-ahead predicted observation and the actual observation. Other functionals of $\epsilon$ could also work just as easily.

Figure 10 shows the evolution of the probability for the two actions as a function of time for $\gamma = 0.2$ and $\eta = 0.5$ for both algorithms. The algorithms are able to converge to the action with the lowest RMSE. As expected, Hedge converges faster and more smoothly than Exp3. However, as the state space grows, Hedge becomes much less efficient than Exp3 because it relies on trying every action at each time step. Therefore, Exp3 is preferred.

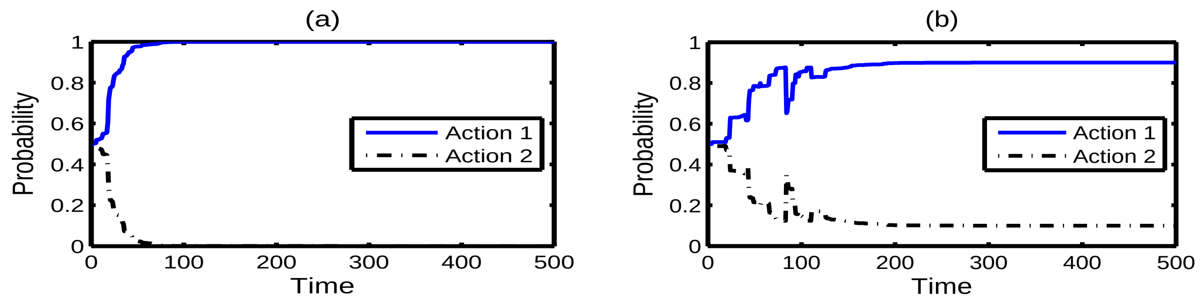
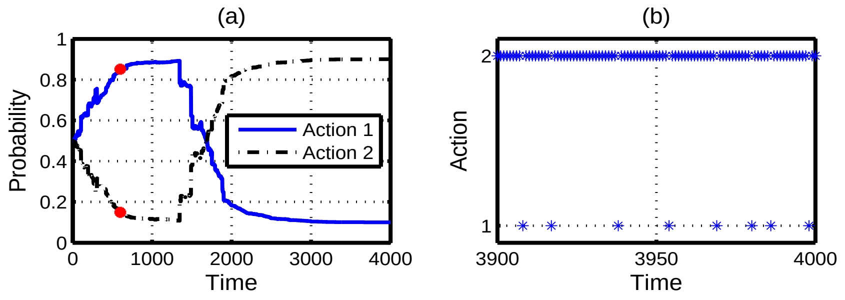


Figure 10: *Automatic state decomposition using (a) Hedge and (b) Exp3.*

In another experiment, we introduce a change point at $t = 600$, so as to also assess whether the bandit algorithm can adapt. For the change-point, we simply swap the transition models for model 1. We observed the same behavior with model 2.

Figure 11 (a) shows the evolution of the probability for the two actions as a function of time for $\gamma = 0.2$ and $\eta = 0.5$. The algorithm is able to converge to the action with the lowest RMSE. After the change-point, the algorithm is able to gradually adapt. The rate of this adaptation is controlled by the hyper-parameters of the control algorithm. It should be noted that we chose a very dramatic artificial change-point, which perturbs stability of the particle filters significantly. In practical applications we would expect more gradual model drift and, hence, better adaptation. Plot (b) shows the action sequence for the last 100 time steps and Table (c) compares the RMSE values incurred by Exp3 and the fixed action policies for the last 1000 time steps. As expected, Exp3 achieves a better result than the worst action. If we know our setting is stationary, then we can stop adapting and attain the same RMSE as the best action.



(c)
$3000 < t \leq 4000$

| Case | RMSE |
|---|---|
| action 1 | 0.8485 |
| action 2 | 0.5788 |
| Exp3 | 0.7564 |

Figure 11: *Automatic state decomposition using Exp3 when introducing a change point. a) Evolution of probability of each action, where the red dot indicates the time step at which we switch the models, b) Action sequence for the last 100 time steps, c) Comparison of the RMSE for action 1, action 2, and Exp3 for the last 1000 time steps.*

## 5 Concluding Remarks and Future Work

The DPF algorithm is a new ingenious particle filter that holds great promise. In this paper, we proposed two algorithmic improvements: a look-ahead formulation and an automatic state-decomposition strategy. The look-ahead strategy performed remarkably well. Even though the original motivation for the DPF was to improve the parallelization level of particle filters, our experiments show that the look-ahead strategy works better than the widely used PF algorithm even in a serial implementation.

In the experiments we also assessed the performance of the state-decomposition strategy using Exp3. The simple demonstration made it clear that it is possible to use bandits to automatically configure the filter. However, we also must point out that this set up was simple enough that Exp3 could handle it. As we move on to more sophisticated partitioning schemes, it will become necessary to adopt more powerful control strategies using correlated bandit strategies or Bayesian optimization; see for example [18, 19, 20, 21, 22].

The immediate future work directions are to test the look-ahead DPF on practical settings and to carry out an empirical evaluation using GPUs. A longer term goal is to increase the level of partitioning (having more than two levels of nesting) of the state space. How such a strategy behaves in high-dimensions is of great interest. Another long term goal is to capitalize on the ideas proposed here to distribute the observations across multiple cores. That is, both the states and the observations should be decomposed for greater applicability to vast streaming datasets.

## Acknowledgements

We would like to thank Arnaud Doucet, Alex Smola and Anthony Lee for useful discussions on this topic, which to a large extent shaped this paper. This work was supported by NSERC.

## References

[1] A. Doucet, N.J. Gordon, and V. Krishnamurthy. Particle filters for state estimation of jump Markov linear systems. *IEEE Transactions on Signal Processing*, 49(3):613–624, 2001.

[2] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Uncertainty in Artificial Intelligence*, pages 176–183, 2000.

[3] N. de Freitas. Rao-Blackwellised particle filtering for fault diagnosis. In *IEEE Aerospace Conference*, volume 4, pages 1767–1772, 2002.

[4] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: a factored solution to the simultaneous localization and mapping problem. In *National Conference on Artificial Intelligence (AAAI)*, pages 593–598, 2002.

[5] N. de Freitas, R. Dearden, F. Hutter, R. Morales-Menendez, J. Mutch, and D. Poole. Diagnosis by a waiter and a Mars explorer. *Proceedings of the IEEE*, 92(3):455–468, 2004.

[6] T. Schon, F. Gustafsson, and P.-J. Nordlund. Marginalized particle filters for mixed linear/nonlinear state-space models. *IEEE Transactions on Signal Processing*, 53(7):2279–2289, 2005.

[7] F. Caron, M. Davy, E. Duflos, and P. Vanheeghe. Particle filtering for multisensor data fusion with switching observation models: Application to land vehicle positioning. *IEEE Transactions on Signal Processing*, 55(6):2703 –2719, 2007.

[8] L. Liao, D. J. Patterson, D. Fox, and H. Kautz. Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311–331, 2007.

[9] T. Chen, T.B. Schon, H. Ohlsson, and L. Ljung. Decentralized particle filter with arbitrary state decomposition. *IEEE Transactions on Signal Processing*, 59(2):465–478, 2011.

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

[11] J. Míguez. Analysis of parallelizable resampling algorithms for particle filtering. *Signal Processing*, 87(12):3155–3174, 2007.

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

[13] A. Ahmed, Q. Ho, J. Eisenstein, E. Xing, A. J. Smola, and C. H. Teo. Unified analysis of streaming news. In *World Wide Web conference*, pages 267–276. ACM, 2011.

[14] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. Technical Report NC2-TR-1998-025, NeuroCOLT2 Technical Report Series, 1998.

[15] A. Doucet, N. de Freitas, and N. Gordon. An introduction to sequential Monte Carlo methods. *Sequential Monte Carlo methods in practice*, pages 3–14, 2001.

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

[17] M. Klaas, N. De Freitas, and A. Doucet. Toward practical $N^2$ Monte Carlo: The marginal particle filter. *Uncertainty in Artificial Intelligence*, 2005.

[18] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *World Wide Web Conference*, pages 661–670, 2010.

[19] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions with application to active user modeling and hierarchical reinforcement learning. eprint arXiv:1012.2599, arXiv, 2010.

[20] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning*, 2010.

[21] D. Lizotte, R. Greiner, and D. Schuurmans. An experimental methodology for response surface optimization methods. *Journal of Global Optimization*, pages 1–38, 2011.

[22] F. Hutter. *Automating the Configuration of Algorithms for Solving Hard Computational Problems.* PhD thesis, University of British Columbia, Vancouver, Canada, 2009.