Summer 8-12-2014

Data Assimilation Based on Sequential Monte Carlo Methods for Dynamic Data Driven Simulation

Haidong Xue

DATA ASSIMILATION BASED ON SEQUENTIAL MONTE CARLO METHODS FOR DYNAMIC DATA DRIVEN SIMULATION

by

HAIDONG XUE

Under the Direction of Xiaolin Hu

ABSTRACT

Simulation models are widely used for studying and predicting dynamic behaviors of complex systems. Inaccurate simulation results are often inevitable due to imperfect model and inaccurate inputs. With the advances of sensor technology, it is possible to collect large amount of real time observation data from real systems during simulations. This gives rise to a new paradigm of Dynamic Data Driven Simulation (DDDS) where a simulation system dynamically assimilates real time observation data into a running model to improve simulation results. Data assimilation for DDDS is a challenging task because sophisticated simulation models often have: 1) nonlinear non-Gaussian behavior 2) non-analytical expressions of involved probability density functions 3) high dimensional state space 4) high computation cost. Due to these properties, most existing data assimilation methods fail to effectively support data assimilation for DDDS in one way or another.

This work develops algorithms and software to perform data assimilation for dynamic data driven simulation through non-parametric statistic inference based on sequential Monte Carlo (SMC) methods (also called particle filters). A bootstrap particle filter based data assimilation framework is firstly developed, where the proposal distribution is constructed from simulation models and statistical cores of noises. The bootstrap particle filter-based framework is relatively easy to implement. However, it is ineffective when the uncertainty of simulation models is much larger than the observation model (i.e. peaked likelihood) or when rare events happen. To improve the effectiveness of data assimilation, a new data assimilation framework, named as the SenSim framework, is then proposed, which has a more advanced proposal distribution that uses knowledge from both simulation models and sensor readings. Both the bootstrap particle filter-based framework and the SenSim framework are applied and evaluated in two case studies: wildfire spread simulation, and lane-based traffic simulation. Experimental results demonstrate the effectiveness of the proposed data assimilation methods. A software package is also created to encapsulate the different components of SMC methods for supporting data assimilation of general simulation models.

INDEX WORDS: Data assimilation, Dynamic data driven simulation, Sequential Monte Carlo, Particle filter, Spatial-temporal systems, Statistical inference

DATA ASSIMILATION BASED ON SEQUENTIAL MONTE CARLO METHODS FOR DYNAMIC DATA DRIVEN SIMULATION

by

HAIDONG XUE

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2014

DATA ASSIMILATION BASED ON SEQUENTIAL MONTE CARLO METHODS FOR DYNAMIC DATA DRIVEN SIMULATION

by

HAIDONG XUE

Committee Chair: Xiaolin Hu

Committee: Guantao Chen

WenZhan Song

Rajshekhar Sunderraman

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

August 2014

DEDICATION

This manuscript is dedicated to my father, Jiang Xue, who has always been supporting me in all aspects. I also especially dedicate this work to my sweet wife, He Gong, who is the joy of my life, and to my grandparents, Dongming Chen, Zhengyang Xue, Baoying Lin, and Delin Chen, who made me believe in achievement instead of success.

ACKNOWLEDGEMENTS

I sincerely appreciate the help from my committee, Dr. Xiaolin Hu, Dr. Rajshekhar Sunderraman, Dr. WenZhan Song, and Dr. Guantao Chen. I especially express my gratitude to my advisor Dr. Xiaolin Hu for his advice on all my research projects and publications during my Ph.D. study.

I am grateful to the members of my research group (the Systems Integrated Modeling and Simulation Lab) for their support in writing of this manuscript and in coding of the experiments.

I also thank my collaborator in the wildfire research, Nathan A. Dahl, from the Center for Analysis and Prediction of Storms, Oklahoma University, for the assistance of atmosphere modeling.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Simulation models are widely constructed and used to predict behavior of real systems. However, due to computational abstractions, flawed model inputs, discretization of involved continuous variables, or data alignment and feedback delay among coupled simulation models, errors are inevitable. When systems are complex, the errors may increase in a non-linear way, and largely limit the prediction ability of simulation models.

With advances of sensor and sensor network technologies, real time observation data from real systems are often available when executing simulation models and they may help improve prediction results; to utilize information from observations, various data assimilation techniques have been developed and employed to reduce simulation errors. Data assimilation is generally comprehended as the procedure combining observation data with models to produce improved estimates of interested variables (Bouttier and Courtier 1999, Reichle 2008). It is also a critical aspect of the Dynamic Data Driven Applications Systems (DDDAS) paradigm as advocated in (Darema 2004). More specifically, from the view of probability theory, it is the process for calculating posterior probability distributions of variables in interest, given prior distributions from simulation models and data from sensor observations, that is, a Bayesian inference procedure; for example, the process described in (van Leeuwen and Evensen 1996).

Under the dynamic state-model assumptions, the posterior distributions can be derived in an iterative manner following Bayesian theorem as shown in (Doucet and Johansen 2011). However, when a system is sufficiently complex, the representation and integral calculation of involved probability distributions are problematic in practice. To avoid these problems, some popular methods assume systems are with linear functions of system transition and measurement, and Gaussian random components. Posterior distributions can then be calculated in very efficient

ways, for example, the procedure of the Kalman filter (KF) (Kalman 1960). When the linearity assumption is violated, various linearization techniques have been proposed, and resulted in many KF variants. Examples are extended Kalman filter, uncented Kalman filter (Julier and Uhlmann 2004), and ensemble Kalman filter (Evensen 1994). Another approach is to only derive a maximum a posteriori estimation (MAP) instead of calculating a full posterior probability density function (pdf). This class of methods is known as variational assimilation, and they construct cost functions from models and observations and find MAPs by solving the corresponding optimization problems. Widely used variational methods include 3D-VAR, 4D-VAR (Talagrand and Courtier 1987), and Physical-space Statistical Analysis System (PSAS) (Cohn, da Silva et al. 1998).

Complex simulation models usually have strong non-Gaussianity and non-linearity behavior. For these models, the above methods then fail to find accurate full posterior distributions. Monte Carlo (MC) methods are often used to approach the distributions. The first class of MC methods is Markov Chain Monte Carlo (MCMC) methods (e.g., Metropolis-Hastings algorithm (Metropolis, Rosenbluth et al. 1953) and Gibbs sampler (Geman and Geman 1984)). Although they are widely used for approaching high dimensional probability distributions, their efficiency is a problem when applied to complex models as discussed in (Fearnhead 2008). Specifically, in a data assimilation application, their computational complexity increases with the total amount of observation data received that is generally unacceptable since it continuously increases with time. In contrast, the complexity of the second type of MC methods, sequential Monte Carlo (SMC) methods (the basic algorithm and various improvements can be found in (Cappe, Godsill et al. 2007) and (Doucet and Johansen 2011)), depends only on the number of employed parti-

cles. SMC methods then offer more flexible and practical solutions for non-linear or non-Gaussian data assimilation.

A typical SMC method requires analytical forms of system transition distributions and proposal distributions for the sampling and weight updating steps. However, they are not always available, especially in the context of complex system modeling. For example, in agent based simulation, it is common that only the behavior of each agent is defined and it is difficult (if possible) to derive the overall system dynamic equations, thus making it almost impossible to find the analytical expression of the overall system transition distribution. The absence of those analytical forms then challenges the application of SMC data assimilation. A frequently used solution is to set the proposal distribution the same as the system transition distribution, as in the bootstrap filter (Gordon, Salmond et al. 1993) and the condensation filter (Isard and Blake 1998). The sampling step is then only driven by simulation models, and the weight updating step does not require density values from transition and proposal distributions. Although this solution can straightforwardly perform SMC data assimilation on non-analytical simulation models, it significantly limits the effectiveness of SMC methods because the generated samples have no knowledge from the most recent observations. In turn, when the uncertainty in simulation models is much larger than the observation model (i.e. peaked likelihood), or when rare events happen, this method usually achieves poor results.

This dissertation proposes SMC-based data assimilation frameworks for non-analytical complex simulation models. It first presents the methods of constructing transition distributions from simulations, and analytical observation distributions from sensor readings. They then form a bootstrap filter based data assimilation framework. To solve the problem of the bootstrap framework with peaked likelihood or rare events, a more effective data assimilation framework

is proposed, consisting of two new components: an effective proposal distribution that uses knowledge from simulation models and sensor readings, and a method that updates importance weights using the kernel method. This work also introduces how the sensor reading correlation is modeled and utilized, and how two flexible software packages are designed and implemented for SMC data assimilation.

This dissertation is organized as follows: section 2 summaries related work; section 3 reviews the statistic inference background of sequential Monte Carlo methods; in section 4, the bootstrap filter based data assimilation framework is introduced; methods for constructing transition distributions and observation distributions are discussed; case studies are carried out on wildfire spread simulation; in section 5, "sensor monitored spatial-temporal systems" are defined, and the SenSim data assimilation framework is proposed with SenSim proposal and kernel weight updating methods; experiments on wildfire spread data assimilation and lane-based traffic data assimilation are performed by both the SenSim framework and the bootstrap filter framework, and results are displayed and compared. Section 6 displays how the sensor reading correlation is modeled and integrated into the developed frameworks; section 7 discusses the design and implementation of the software packages of SMC data assimilation; section 8 draws the conclusions, and points out future research directions.

2 BACKGROUND AND RELATED WORK

2.1 Dynamic Data Driven Simulation

Dynamic Data Driven Applications Systems (DDDAS) paradigm is established in (Darema 2004). "In DDDAS, instrumentation data and executing application models of these systems become a dynamic feedback control loop, whereby measurement data are dynamically incorporated into an executing model of the system in order to improve the accuracy of the model (or simulation), or to speed-up the simulation, and in reverse the executing application model controls the instrumentation process to guide the measurement process." The first aspect of DDDAS defines Dynamic Data Driven Simulation (DDDS) where a simulation system dynamically assimilates real time observation data into a running model to improve simulation results. This manuscript focuses on developing algorithms to support DDDS.

2.2 Data Assimilation

When analytical forms of models are available, various of data assimilation methods can be found in literature as summarized in (Bouttier and Courtier 1999) and (Reichle 2008). Recent developments and applications are also reported in many domains: (Mandel, Beezley et al. 2009) used enhanced versions of the ensemble Kalman filter to estimate fire states; (Reichle, Kumar et al. 2010) investigated the potential of assimilating satellite retrievals of land surface temperature for the Catchment land surface model and the Noah land surface model; (Buehner, Houtekamer et al. 2009) compared variational methods with the ensemble Kalman filter in the context of global deterministic numerical weather prediction when assimilating meteorological observations; (Lopez 2011) assimilated National Centers for Environmental Prediction state IV precipitation data in European Centre for Medium-Range Weather Forecasts; (Sakov, Counillon et al.

2012) presented the latest progresses of the coupled ocean-sea ice data assimilation system for the North Atlantic Ocean and Arctic.

2.3 SMC Methods

In strongly non-linear non-Gaussian scenarios, SMC methods are often employed. For example, to improve short term hydrologic forecasting, (Noh, Tachikawa et al. 2011) applied sequential Monte Carlo methods on a process-based distributed hydrologic model; (González, Blanco et al. 2009) applied SMC methods to improve robot localization by assimilating Ultra-Wide-Band range measurements; (Wang, Yang et al. 2009) used an extended SMC method to track objects in video sequence where HSV color histogram based observation is constructed from the current image frame; (Ahmed, Rutten et al. 2010) employed SMC methods to estimate positions of moving targets by sensor measured signals from a wireless network.

Less research has been conducted on applying SMC methods to systems with only non-analytical simulation models, and a common working method is to set the proposal distribution the same as the transition distribution that dates back to early versions of SMC methods. For example, in the bootstrap filter (Gordon, Salmond et al. 1993), samples of involved random components are firstly drawn, then passed through system transition black-boxes, and samples of system states are obtained. A similar method, called "factored sampling", is used in the condensation filter (Isard and Blake 1998). As widely recognized, this choice largely lower the ability of a SMC method since it excludes the knowledge of the most recent observations in the sampling step.

To utilize recent observation data in the SMC sampling step, proposal distributions need to be carefully designed. For specific applications, many design techniques are proposed. For example, (Kyriakides, Morrell et al. 2008) used motion constraints to build proposals for the multi-

ple target tracking problem; in video tracking, (Kyriakides, Morrell et al. 2008) developed an algorithm to find appropriate variances in proposal distributions, and (Lao, Zhu et al. 2009) incorporated video measurement confidence to tune proposals; (Zhai, Yeary et al. 2009) employed multiple transition models and state partition to help construct proposals; (Saha and Gustafsson 2012) utilized noise dependency to derive the optimal proposal for signal processing applications. Also, efficient SMC methods can be constructed when reasonably including recent observations into proposal distribution even when the model is extremely complex; examples can be found in (van Leeuwen 2010).

Two generic SMC proposal design methods have been proposed: the extended Kalman filter proposal (De Freitas, Niranjan et al. 2000) and the unscented Kalman filter proposal (Merwe, Doucet et al. 2000). For each SMC particle, an extended or unscented Kalman filter is maintained, and in each sampling step of a SMC method, the Kalman filter is first updated, then producing samples for the SMC sampling step. Those samples are then driven both by observations and model outputs, and then much closer to the optimal proposal distribution than those of a system transition distribution. However, as Kalman filter based methods, they are limited by the assumption of Gaussian system states; when the optimal proposal distribution is far away from a Gaussian distribution (it is especially likely to happen in the context of complex systems), their approximations then potentially have large errors. Also, the Jacobians of an extended Kalman filter is difficult to be obtained when analytical system transition functions are unknown; meanwhile, the computation load of an unscented Kalman filter is proportional to the dimension of system states, it is then also not practical for complex systems that often have very high dimensions.

2.4 Sensor Reading Correlation

When sensors are unevenly deployed, their records may be highly spatially correlated. Research work has been performed to model the correlation and use it to improve many aspects of wireless sensor network. (Vuran, Akan et al. 2004) modeled the physical phenomenon of sensor observation as a multivariate Gaussian variable with an independent Gaussian noise on each sensor. (Berger, Oliveira et al. 2001) summarized the four families of correlation models from which correlation can be estimated from spatial distance. (Vuran and Akyildiz 2006) developed a correlation based MAC protocol to reduce collisions and energy consumption. Methods for efficiently selecting active sensors based on sensor correlation can be found in (Zoghi and Kahaei 2009) and (Shah and Bozyigit 2007). Similarly, by correlation knowledge, (SunHee and Shahabi 2005) and (Yingqi and Lee 2006) respectively proposed algorithms for in-network aggregation and link quality estimation.

2.5 DEVS-FIRE and Its Data Assimilation

In this manuscript, much research work has been carried out on DEVS-FIRE data assimilation. DEVS-FIRE is a wildfire spread and containment integrated simulation model (Ntaimo, Hu et al. 2008, Hu, Sun et al. 2012). The fire spread model of DEVS-FIRE is a discrete event simulation model based on DEVS (Zeigler, Kim et al. 2000). It simulates a fire as a two dimensional cellular automata, where each cell defines its fire behavior along with the interactions with other cells. The overall behavior of all the grids then represents the spreading behavior of a fire. However, the analytical expression of the system dynamics cannot to be derived. Given a current fire state, DEVS-FIRE can be used as a black-box to tell a fire state at a later time point:

$$fire_{t+\Delta t} = DEVSFIRE(fire_t, \theta_t, \Delta t),$$

where $fire_t$ and $fire_{t+\Delta t}$ are the fire states at time t and $t + \Delta t$, θ_t is the vector of other model inputs.

Most of the DEVS-FIRE data assimilation work of this dissertation has already been published: a SMC data assimilation method for a wildfire prediction model using the bootstrap filter was presented in (Xue, Gu et al. 2012); a specific instance of the SenSim proposal distribution was shown in (Xue and Hu 2013); applications of the sensor reading observation model and its sensor correlation estimation method can be found in (Xue and Hu 2012).

2.6 Traffic Simulators and Their Data Assimilation

Much research work has been performed on traffic system modeling, and various simulators can be found in literature. They can be roughly classified into two categories: macro-level simulators and micro-level simulators. Marco-level simulators model traffic systems as continua of vehicles. For example, (Payne 1971, Cremer 1979, Byrne, United et al. 1982, Wunderlich 1995, Treiber and Kesting 2010). They focus on higher level system states, such as spatial densities and average velocities. Also, analytical system dynamics are usually available.

On the other hand, micro-level simulators model behavior of individual vehicles. For example, the simulators in (Fellendorf and Vortisch 2001, Park and Schneeberger 2003, Gomes, May et al. 2004, Hunter, Fujimoto et al. 2006). Using them, detailed vehicle behavior can be simulated, such as lane-changing, accident avoidance, drivers' erroneous judgments and reaction time. Interactions of individual vehicles also create overall behavior of the system, but the analytical expressions of system behavior are hard to be constructed.

Most of existing traffic system data assimilation researches uses macro-level simulators, and aims at estimating high level system states. With analytical traffic flow models and Gaussian assumption on random components, Kalman Filter based methods are widely employed. For ex-

ample, the extended Kalman Filter in (Wang and Papageorgiou 2005, Tampère and Immers 2007, Wang, Papageorgiou et al. 2007, Schreiter, Van Hinsbergen et al. 2010), the unscented Kalman Filter in (Mihaylova, Boel et al. 2006), and the ensemble Kalman Filter in (Work, Tossavainen et al. 2008).

Due to high complexity of vehicle behavior models, much less work has been reported on using micro-level simulators to estimate micro-level behaviors by data assimilation. The proposed SMC frameworks are designed to perform data assimilation for complex simulation models, and a micro-level traffic simulator, MovSim (Treiber, Hennecke et al. 2000, Kesting, Treiber et al. 2010, Treiber and Kesting 2010), is then selected as the second case study to test the data assimilation ability.

3 SEQUENTIAL MONTE CARLO STATISTICAL INFERENCE

Assimilating observation data into a simulation model can be considered as a Bayesian inference process under the assumptions of the dynamic state-space model. Sequential Monte Carlo (SMC) methods employ a collection of system state samples to approach the system state posterior distributions conditional on all received observation data, by iteratively evolving samples for one time step and updating their importance weights. Compared with other methods, such as Kalman filter (KF) based ones, SMC methods have no assumption on system transition functions, observation functions, or the probability distributions of involved random components, so they usually achieve better results when applied to strong non-linear or non-Gaussian systems. The mathematic foundation of SMC methods is summarized that lays a ground for the proposed data assimilation frameworks.

3.1 Dynamic State-Space Model

The dynamic state-space model expresses the relationship of system states and their observations. Firstly, system states are modeled as a time sequence $\{s_t | \, t \in \mathbb{N}\}$, and a *system state transition function* defines how a system state evolve with time as shown in equation (1):

$$s_t = f(s_{t-1}, u_t), \tag{1}$$

where f is a system state transition function (also referred as *system transition function* or *transition function*), s_{t-1} and s_t are system states at time point $t-1$ and t, and u_t is the vector of other inputs that may include random variables (e.g., noises). With the statistic kernel of u_t, en s_{t-1}, the next state s_t then becomes a random variable and can be described by a probability distribution (usually referred as *system state transition distribution* or *transition distribution)*:

$$p(s_t | s_{t-1}). \tag{2}$$

The state transition is then a first order Markov process, that is, state s_t depends only on s_{t-1}, and independent with other states given s_{t-1}.

For each system state at a time point, there exists a corresponding observation, and all observations form another time sequence $\{m_t | \, t \in \mathbb{N}\}$. System states are "hidden", that is, they cannot be directly observed, and can only be accessed from observation data through an observation function:

$$m_t = g(s_t, \, v_t), \tag{3}$$

where g is an observation function, s_t is a system state at time point t, v_t is the vector of other inputs that may include random variables, and m_t is the observation of s_t. Similarly, with the statistic kernel of v_t, m_t can be described as a probability distribution conditionally on s_t(usually referred as *observation distribution*):

$$p(m_t|s_t). \tag{4}$$

Given s_t, m_t is then independent from other measurements and states.

Because of the obstacles in Bayesian inference discussed in the next section, most existing algorithms assume g, f are linear or can be linearized, and s_t, u_t and v_t are Gaussian distributed. When they are strongly non-linear or non-Gaussian, SMC methods are usually employed.

3.2 Bayesian Inference on System States

Deriving system state posterior distributions (i.e. $p(s_{0:t}|m_{0:t})$, where $s_{0:t} = \{s_0, s_1, \dots, s_t\}$ and $m_{0:t} = \{m_0, m_1, \dots, m_t\}$) given all observations is the ultimate goal of data assimilation since they contain full statistic information of system states. Nonetheless, many applications may be only interested in a marginal distribution of the full posterior distribution, such as $p(s_t|m_{0:t})$ and ($p(s_{t'}|m_{0:t})$, where $0 \leq t' < t$.

If analytical expressions of the system state transition distribution and system state observation distribution are available, from an initial state distribution, following Bayesian rules, the recursive equations to calculate the posterior distribution are:

$$p(s_{0:t}|m_{0:t}) = \frac{p(s_{0:t}|m_{0:t-1})p(m_t|s_t)}{p(m_t|m_{0:t-1})}, \quad (5)$$

$$p(s_{0:t}|m_{0:t-1}) = p(s_{0:t-1}|m_{0:t-1})p(s_t|s_{t-1}), \quad (6)$$

$$p(m_t|m_{0:t-1}) = \int p(s_{0:t}|m_{0:t-1})p(m_t|s_t)ds_{0:t}, \quad (7)$$

where $p(s_{0:t-1}|m_{0:t-1})$ is the posterior state distribution at $t-1$, $p(m_t|s_t)$ is the observation distribution, $p(s_t|s_{t-1})$ is the system state transition distribution. Although, in certain cases, $p(s_{0:t}|m_{0:t})$ can be analytically calculated from those equations, it is usually intractable for the when the targeted systems are complex. Monte Carlo based methods are then employed to simulate this posterior distribution by a set of independent random samples.

Two classes of Monte Carlo methods exist for estimating $p(s_{0:t}|m_{0:t})$: Markov Chain Monte Carlo methods (MCMC) and sequential Monte Carlo (SMC) methods. The computational complexity of MCMC methods increases with t; on the other hand, the one of SMC methods depends only on the number of employed particles. As a result, SMC methods are more practical in applications where large amount of observation data arrive on-line. It should be noted that, with finite number of particles, as pointed out in (Doucet and Johansen 2011), SMC methods fail to approach $p(s_{0:t}|m_{0:t})$ when t is sufficiently large. However, SMC methods are still capable of estimating its marginal distributions $p(s_{t-L:t}|m_{0:t})$, where L is a constant number. In many applications, $p(s_t|m_{0:t})$ is the distribution of interest, SMC methods are then preferred.

3.3 Deriving System State Posterior Distributions through SMC Methods

Sequential Monte Carlo (SMC) methods are closely related to Monte Carlo methods and Importance Sampling (IS), and are usually identified as procedures of sequential Importance Sampling (SIS) with a resampling step after certain weight updating steps.

Although the SIS algorithm and resampling steps of importance sampling can be found in early literature (e.g. (Rubin 1988)), before (Gordon, Salmond et al. 1993) put them together, the SIS algorithm suffered from the sample degeneracy problem, and was then not useful in practice. Variants of SMC methods have been proposed from several research fields, including: the boot-strap filter (Gordon, Salmond et al. 1993) in signal processing, the condensation filter (Isard and Blake 1998, MacCormick and Blake 2000) in computer vision, the interacting particle system (Moral 1997, Crisan, Moral et al. 1999) and the Monte Carlo filter (Kitagawa 1996) in statistics. SIS with resampling steps approaches the posterior distribution based on Monte Carlo methods and Importance Sampling.

*3.3.1 **Monte Carlo Methods***

Monte Carlo methods approximate a probability density function by a set of independent samples $\{x^{(i)}|1 \leq i \leq N\}$ (also referred as "*particles*" or "*realizations*"):

$$p(x) \approx \frac{1}{N}\sum_{i=1}^{N} \delta(x - x^{(i)}), \tag{8}$$

where δ is the Dirac-delta function, and N is the number of samples. A asymptotically unbiased estimation of a function of x, $h(x)$, is then:

$$E(h(x)) \approx \frac{1}{N}\sum_{i=1}^{N} h(x^{(i)}). \tag{9}$$

3.3.2 ***Importance Sampling***

When it is hard to draw samples from $p(x)$, Importance Sampling (IS) is an alternative. There exists a probability density function $q(x)$ (often referred as *importance distribution*, *instrumental distribution* or *proposal distribution*) that for any x if $p(x) > 0$, $q(x) > 0$. The expectation of $h(x)$ over $p(x)$ is then the same as the expectation of $\frac{h(x)p(x)}{q(x)}$ over $q(x)$. As a result, a Monte Carlo estimation of $h(x)$ from samples of $q(x)$ is then:

$$E_{p(x)}(h(x)) = E_{q(x)}\left(\frac{h(x)p(x)}{q(x)}\right) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{p(x^{(i)})}{q(x^{(i)})}h(x^{(i)}). \tag{10}$$

It can be considered as $p(x)$ is approximated by weighted samples:

$$p(x) \approx w(x)\sum_{i=1}^{N}\delta(x - x^{(i)}), \tag{11}$$

$$w(x) = \frac{1}{N}\frac{p(x)}{q(x)}. \tag{12}$$

$w(x)$ is usually referred as *normalized importance weight function*, and can be approximated by samples of $q(x)$ as:

$$w(x) \approx \frac{w'(x)}{\sum_{i=1}^{N} w'(x^{(i)})}, \tag{13}$$

$$w'(x) \propto w(x), \tag{14}$$

where $w'(x)$ is the unnormalized importance weight function. $E_{p(x)}\big(h(x)\big)$ is then estimated as

$$E_{p(x)}(h(x)) \approx \sum_{i=1}^{N}\frac{w'(x^{(i)})}{\sum_{j=1}^{N} w'(x^{(j)})}h(x^{(i)})\delta(x - x^{(i)}), \tag{15}$$

where $\frac{w'(x^{(i)})}{\sum_{j=1}^{N} w'(x^{(j)})}$ is termed the *normalized importance weight*.

3.3.3 ***Sequential Importance Sampling***

To derive the weighted samples approximating $p(s_{0:t}|m_{0:t})$, IS can be sequentially applied. The resulting algorithm is called Sequential Importance Sampling (SIS).

In each iteration, for example the iteration at time t, it chooses an importance distribution:

$$q(s_{0:t}|m_{0:t}) = q(s_{0:t-1}|m_{0:t-1})q(s_t|s_{t-1}, m_t). \tag{16}$$

Given a sample from the previous step (i.e. the one of $q(s_{0:t-1}|m_{0:t-1})$, $s_{0:t-1}^{(i)}$) and the sample of $q(s_t|s_{t-1}^{(i)}, m_t)$, $s_t^{(i)}$, (where $s_{t-1}^{(i)}$ is the $t-1$ part of $s_{0:t-1}^{(i)}$), their union ($s_{0:t-1}^{(i)}$, $s_t^{(i)}$) is then a sample of $q(s_{0:t}|m_{0:t})$. In this way, starting from the initial samples at $q(s_0|m_0)$, one can iteratively draw samples of $q(s_{0:t}|m_{0:t})$. If importance weights could also be iteratively updated, one then obtains the weighted samples approaching each posterior distribution.

Inserting equation (6) into equation (5):

$$p(s_{0:t}|m_{0:t}) = \frac{p(s_{0:t-1}|m_{0:t-1})p(s_t|s_{t-1})p(m_t|s_t)}{p(m_t|m_{0:t-1})}. \tag{17}$$

Dividing both sides of equation (17) by the corresponding side of equation (16), an unnormalized importance weight function is found:

$$\frac{p(s_{0:t}|m_{0:t})}{q(s_{0:t}|m_{0:t})} = \frac{p(s_{0:t-1}|m_{0:t-1})}{q(s_{0:t-1}|m_{0:t-1})} \frac{1}{p(m_t|m_{0:t-1})} \frac{p(s_t|s_{t-1})p(m_t|s_t)}{q(s_t|s_{t-1},m_t)}. \tag{18}$$

Since $p(m_t|m_{0:t-1})$ is invariable with $s_{0:t}$, an unnormalized importance weight at a time step is then proportional to an unnormalized importance weight in the previous time step multiplied with $\frac{p(s_t|s_{t-1})p(m_t|s_t)}{q(s_t|s_{t-1},m_t)}$. Another unormalized importance weight at time t can then be defined as:

$$w'(s_{0:t}) = w(s_{0:t-1}) \frac{p(s_t|s_{t-1})p(m_t|s_t)}{q(s_t|s_{t-1},m_t)}. \tag{19}$$

where $w(s_{0:t-1})$ is the normalized importance weight at $t-1$ .The normalized importance weight of this step can be approximated by equation (13), that is, $w(s_{0:t}) \approx w'(s_{0:t})/\sum_{i=1}^{N} w'(s_{0:t}^{(i)})$.

Starting from an initial sample set of s_0 (usually drawn from $q(s_0|\, m_0)$), the samples up to t can then be iteratively constructed through the samples of $q(s_t|s_{t-1}, m_t)$, and their normalized importance weights can be iteratively updated through equations (19) and (13). This procedure forms the core of Sequential Importance Sampling (SIS) algorithm.

3.3.4 ***Resampling***

With the increase of t, the dimension of $p(s_{0:t}|m_{0:t})$ continually grow; since the number of employed samples is fixed in SIS, when t is sufficiently large, the samples will eventually fail to estimate $p(s_{0:t}|m_{0:t})$. From the view of sample weights, it means that sample degeneracy (most of sample weights are equally insignificant) will finally occur. To solve this problem, the resampling step (Rubin 1988) is introduced to SIS as in (Gordon, Salmond et al. 1993). When the variance of importance weights exceeds certain predefined threshold, a resampling step is invoked to drive samples to the areas with large probability density. The resulting algorithm is usually termed as the Sequential Importance Sampling with Resampling (SISR) algorithm.

A resampling step is to draw N samples from the SIS estimated posterior distribution (that is, $p'(s_{0:t}|m_{0:t}) = \frac{w'(s_{0:t})}{\sum_{i=1}^{N} w'\left(s_{0:t}^{(i)}\right)} \sum_{i=1}^{N} \delta(s_{0:t} - s_{0:t}^{(i)})$). With high probabilities, low weight samples are then removed and high weight samples are duplicated. The effect of sample degeneracy problem is then reduced.

This treatment puts more attention on recent states, and may harm the estimation for early states since the history paths of the removed samples are lost. Nonetheless, the SISR algorithm,

as shown in Algorithm 1, is still an efficient and effective algorithm to estimate $p(s_{t-L:t}|m_{0:t})$, where L is a constant number.

ALGORITHM 1. *Sequential Importance Sampling with Resampling (SISR)*

Input: $\{\boldsymbol{m_j}|\boldsymbol{0 \le j \le t}\}$; $\{\boldsymbol{q(s_j|s_{j-1}, m_j)}|\boldsymbol{1 \le j \le t}\}$; $\boldsymbol{q(s_0|\, m_0)}$; $\{\boldsymbol{p(s_j|s_{j-1})} \mid \boldsymbol{1 \le j \le t}\}$; $\boldsymbol{p(s_0)}$; $\{\boldsymbol{p(m_j|s_j)}|\boldsymbol{0 \le j \le t}\}$;

Output: $\{< s_{0:t}^{(i)}, w(s_{0:t}^{(i)}) > |1 \le i \le N\}$.

1. Initialization

Draw $\{s_0^{(i)}|1 \le i \le N\}$ from $q(s_0|\, m_0)$;

Repeat for $i = 1$ to N

$w'(s_0^{(i)}) \leftarrow \frac{p(s_0)p(m_0|s_0)}{q(s_0|\, m_0)}$;

Repeat for i $= 1$ to N

$w(s_0^{(i)}) \leftarrow \frac{w'(s_0^{(i)})}{\sum_{k=1}^{N} w'(s_0^{(k)})}$;

2. Iterative Sampling, Weight Updating and Resampling

Repeat for j = 1 to t

Repeat for i $= 1$ to N

Sampling (Prediction)

Draw $s_j^{(i)}$ from $q(s_j|s_{j-1}, m_j)$;

$s_{0:j}^{(i)} \leftarrow (s_{0:j-1}^{(i)}, s_j^{(i)})$;

Weight Updating (Correction)

$w'^{(s_{0:j}^{(i)})} \leftarrow w(s_{0:j-1}^{(i)}) \frac{p(s_j|s_{j-1})p(m_j|s_j)}{q(s_j|s_{j-1}, m_j)}$;

Resampling (Selection)

Draw $\{s_{0:j}^{(i)}|1 \le i \le N\}$ from $\sum_{i=1}^{N} \frac{w'(s_{0:j}^{(i)})}{\sum_{k=1}^{N} w'(s_{0:j}^{(k)})} \delta(s_{0:j} - s_{0:j}^{(i)})$;

Set$\{w(s_{0:j}^{(i)}) = 1/N \,|1 \le i \le N\}$;

Return $\{< s_{0:t}^{(i)}, w(s_{0:t}^{(i)}) > |1 \le i \le N\}$.

3.3.5 ***Proposal Distributions of SMC Methods***

The design of proposal distributions is critical for a SMC data assimilation application. From equation (19), it is easy to observe that, starting from an equally weighted sample set, the proposal distributions proportional to $p(s_t|s_{t-1})p(m_t|s_t)$ minimize the importance weight variance, and then give the optimal structure of proposal distributions as discussed in (Cappe, Godsill et al. 2007) and (Doucet and Johansen 2011):

$$q^{opt}(s_t|s_{t-1}, m_t) \propto p(s_t|s_{t-1})p(m_t|s_t). \quad (20)$$

Given the recent observation and a system state sample, if $p(s_t|s_{t-1})p(m_t|s_t)$ can be evaluated, Markov Chain Monte Carlo (MCMC) methods can be employed to draw samples to approach the optimal distribution. However, a MCMC component challenges the efficiency of a SMC method because the time complexity of MCMC does not dependent only on the number of particles. It needs a large number of burn-in iterations as stated in (Godsill and Clapp 2001); moreover, in certain cases, it is difficult to know when a burn-in phase is finished. Furthermore, if the analytical form of $p(s_t|s_{t-1})$ is unknown, MCMC does not work since $p(s_t|s_{t-1})p(m_t|s_t)$ is hard to be evaluated. Although directly drawing samples from the optimal proposal distribution is often impractical, it still shows a direction for designing effective proposal distributions, that is, when the sample has both high likelihood and transition density value, it also has a high density value in the optimal proposal distribution.

4 BOOTSTRAP FILTER BASED DATA ASSIMILATION FRAMEWORK

4.1 The Dilemma of Sequential Monte Carlo Methods for Complex System Data Assimilation

As displayed in Algorithm 1, there are three components in the core algorithm of SMC:

Sampling: given a particle at time $t-1$ ($s_{t-1}^{(i)}$) , draw a sample ($s_t^{(i)}$)) from $q(s_t|s_{t-1}^{(i)}), m_t)$.

Weight updating: with samples $s_{t-1}^{(i)}$ and $s_t^{(i)}$, update the importance weights ing $w'(s_t^{(i)}) = w(s_{t-1}^{(i)}) \frac{p(s_t^{(i)}| s_{t-1}^{(i)}) p(m_t|s_t^{(i)})}{q(s_t^{(i)}| s_{t-1}^{(i)}, m_t)}$.

Resampling: draw N samples from the current sample represented posterior tion $\sum_{i=1}^{N} \frac{w'(s_t^{(i)})}{\sum_{k=1}^{N} w'(s_t^{(k)})} \delta(s_t - s_t^{(i)})$.

In the weight updating step, analytical forms of system transition distribution ($p(s_t|s_{t-1})$), proposal distribution ($q(s_t|s_{t-1}, m_t)$) and observation distribution ($p(m_t|s_t)$) are all required. Among them the observation distribution is used to calculate likelihoods, and it is often easy to construct since, given a system state, and the sensor reading is often distributed around the true observation value. However, for complex systems, it is hard to find the analytical forms of $p(s_t|s_{t-1})$ and $q(s_t|s_{t-1}, m_t)$ because system behavior could be quite non-linear, and the involved random components could be distributed in complex or unknown distributions.

To avoid the difficulty of weight updating, a widely used method is to set the proposal distribution the same as the transition distribution, that is, set $q(s_t|s_{t-1}, m_t)$= $p(s_t|s_{t-1})$. As a result, the weight updating equation then reduced to:

$$w'(s_t^{(i)}) = w(s_{t-1}^{(i)}) p(m_t|s_t^{(i)}). \tag{21}$$

With a constructed analytical observation distribution, importance weights then can be easily updated by likelihoods. Also, no effort is needed to design a proposal distribution since it is the same as the transition distribution. However, the chosen proposal distribution has no knowledge of the most recent observation. When the support of transition distribution is much larger than the likelihood function, or when the majority of the likelihood function locates in the tail of the transition distribution, samples draw from the transition distribution can barely cover the high density area of the optimal proposal distribution, and then result into poor estimation of the next posterior distribution.

In sum, when designing a proposal distribution making use of the most recent observations, it is hard to update the importance weight; when using transition distribution as the proposal distribution, the performance could be problematic in some cases. It is then a dilemma that can be found in many complex system data assimilation applications using SMC.

In the rest of this section, the framework using transition distributions as proposal distributions is introduced, and in section 5 the SenSim framework using the kernel method to work around the dilemma is proposed.

4.2 Architecture of the Bootstrap Filter

The SMC method choosing system transition distributions as proposal distributions is often termed as the bootstrap filer. Figure 4.1 displays a data assimilation framework based on the bootstrap filter. N system state samples are created as the initial particles with $1/N$ as the importance weights at the beginning of a data assimilation process. When observation data is arrived at time t, each of the particles is sampled to t. For each particle, likelihood is then calculated from the new state and the observation at t through an observation distribution, then updating the importance weight using equation (21).

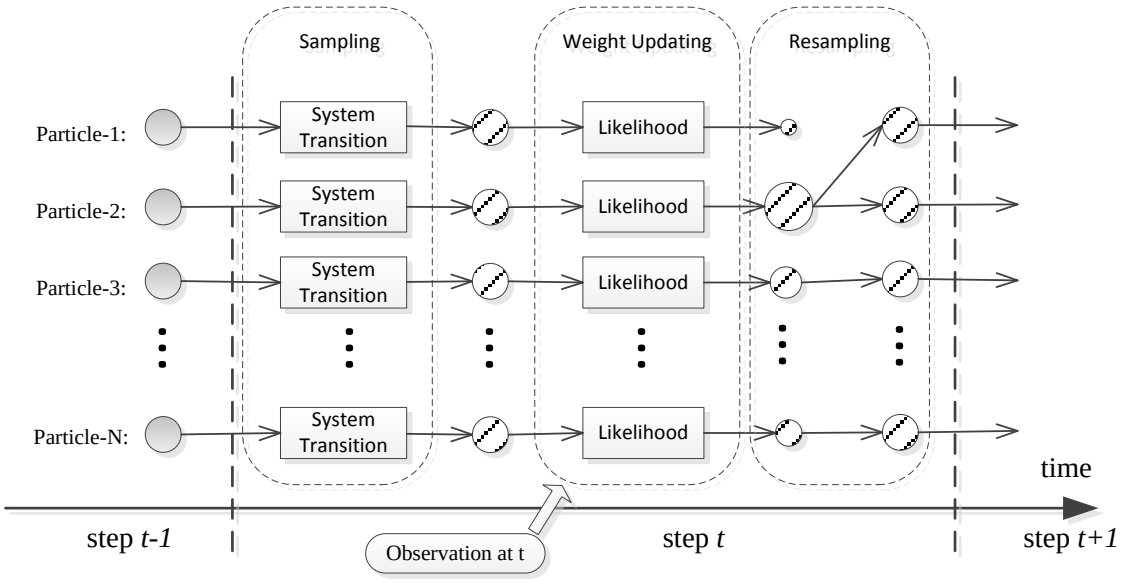


Figure 4.1 The bootstrap filter based SMC data assimilation framework

For complex simulation models, this manuscript proposes to use simulations to construct system transition distributions, and use sensor readings to construct observation distributions.

4.3 Simulation Based Transition Distribution

The snapshot of a simulation for a complex system contains most of the information of a system state. In this proposed framework, executing simulations are then used as system states in the bootstrap filter.

Given a simulation at $t-1$, it is straightforward to obtain a possible state at t by running the simulation model for one time step. However, for deterministic simulation models, it only produces one possible state; even with stochastic simulation models, the produced states are still from fixed model settings, and fail to cover all possible states in the true transition distribution. This work proposes to use distributions of random model parameters, noise of system states, and random moves of static model parameters to approach the system transition distribution.

A simulation model naturally defines a system transition functions $\mathcal{F}$:

$$s_t = \mathcal{F}(s_{t-1}, u_t),$$

where s_{t-1} and s_t are the system states at $t-1$ and t, and u_t represents parameters affecting the simulation results.

Three categories of uncertainty are identified: 1) parameters that are already random variables 2) noise indicating the difference between simulated state and true state 3) artificial added random moves on static parameters. A state sample is then drawn from all of them as described in Algorithm 2.

ALGORITHM 2. *Transition Sampling Using Simulation Models*

Input: simulation model $\boldsymbol{\mathcal{F}}$, system state $\boldsymbol{s_{t-1}}$, $\boldsymbol{p(u_{t,random})}$, $\boldsymbol{p(u_{t,static}|u_{t-1,static})}$, $\boldsymbol{p(noise|s_t)}$

Output: a system state sample s_t

1. Draw a sample $u_{t,random}$ for random model parameters from their distributions $p(u_{t,random})$,
2. Draw a sample $u_{t,static}$ for static model parameters from their random move distributions $p(u_{t,static}|u_{t-1,static})$,
3. Draw a noise sample $noise$ from its distribution $p(noise|s_t)$,
4. Run simulation model $s_t = \mathcal{F}(s_{t-1}, u_t)$, where $u_t \leftarrow u_{t,random}$, $u_{t,static}$ and *noise*,
5. Return the state s_t.

In applications, only a sub set of the random components may be used. For instance, if a stochastic simulation model has already covered the possible states incurred by noise, the noise sampling can then be excluded. Also, it should be noted that even all random components are considered, it is still hard to obtain samples with rare events since they are in the tail area of the involved random components.

4.4 Sensor Reading Based Observation Distribution

From Algorithm 1 and equation (21), it can be seen that to finish weight updating the probability density function of the observation distribution is required. A sensor reading based observation distribution is then proposed in this manuscript.

Given a sensor set $= \{c_i =< m_i, p_i, D_i > \mid p_i \in A, D_i \subseteq A, 1 \leq i \leq n_c\}$, where m_i is the sensor reading, p_i is the sensor location, and D_i is the sensor covered area. At time t, the observation data is then defined as a vector of all measured sensor readings:

$$m_t = [m_{t,i}]_{i=1}^{n_c}, \tag{22}$$

where $m_{t,i}$ is the measured sensor reading on the i-th sensor at time t, and n_c is the number of sensors.

With a *measurement function* $\mathcal{M}: S \times \mathbb{R}^2 \rightarrow \mathbb{R}^{n_r}$, a true sensor reading vector can be calculated as:

$$[r_i]_{i=1}^{n_c} = [\mathcal{M}(s_{t,D_i},\ p_i\ )]_{i=1}^{n_c}\ . \tag{23}$$

where s_{t,D_i} is the local state in D_i. With a defined difference on sensor readings, the observation distribution is constructed as a multivariate Gaussian distribution:

$$(m_t - [r_i]_{i=1}^{n_c}) \sim MN(0, \Sigma), \tag{24}$$

where $MN(0, \Sigma)$ is a multivariate Gaussian distribution with a zero mean vector and a covariance matrix $\Sigma = [\rho_{ij}]_{n_c \times n_c}$. Single sensor measurement error is expressed by the variance coefficients of Σ (that is, $\{{\sigma_i}^2 = \rho_{ii} | 1 \leq i \leq n_c\}$), and spatial correlation of sensor readings is expressed by the covariance coefficients of Σ (that is, $\{\rho_{ij} | 1 \leq i \leq n_c, 1 \leq j \leq n_c, i \neq j\}$).

When it is hard to determine ρ_{ij}, spatial distance based correlation model can be employed to estimate the covariance coefficients, that is, set ρ_{ij} as:

$$\rho_{ij} = \sigma_i \sigma_j\ Corr(d_{ij}), \tag{25}$$

where $Corr: \mathbb{R} \rightarrow [0, 1]$ is a spatial correlation estimation function, d_{ij} is the Euclidean distance from the i-th sensor to the j-th sensor. Four types of spatial correlation functions have been in-

troduced in (Berger, Oliveira et al. 2001), and the one best fitting the physical process of an specific application should be employed to estimate ρ_{ij}. More details can be found in section 6.

Given a system state at t, s_t, and a measured sensor reading vector, $m_t = [m_{t,i}]_{i=1}^{n_c}$, from equations (23) and (24), the observation distribution can then be derived as $MN([\mathcal{M}(s_{t,D_i}, p_i)]_{i=1}^{n_c}, \Sigma)$, that is:

$$p(m_t|s_t) = \frac{exp(-\frac{1}{2}\alpha_m{}'\Sigma^{-1}\alpha_m)}{(2\pi)^{n_r/2}|\Sigma|^{1/2}}, \tag{26}$$

$$\alpha_m = [m_{t,i}]_{i=1}^{n_c} - [\mathcal{M}(s_{t,D_i}, p_i)]_{i=1}^{n_c} = [\Delta_c(m_{t,i}, \mathcal{M}(s_{t,D_i}, p_i))]_{i=1}^{n_c}, \tag{27}$$

where Δ_c: $\mathbb{R}^{n_r} \times \mathbb{R}^{n_r} \to \mathbb{R}$ is the distance function of two sensor readings. With equations (26) and (27), it is then easy to evaluate the likelihood of a system state given the measured sensor readings.

4.5 Case Study on Data Assimilation for Wildfire Spread Simulation

In this section, the proposed bootstrap framework is tested with DEVS-FIRE (introduction of DEVS-FIRE can be found in section 2.5). Assuming ground temperature sensors are deployed in the area where a real wildfire happens and DEVS-FIRE is used to simulate the fire spread behavior, the data assimilation goal here is to use received sensor temperature readings to improve the DEVS-FIRE estimation of the current fire and the predictions on future fires.

*4.5.1 **Architecture of bootstrap filter based data assimilation for DEVS-FIRE***

Figure 4.2 shows the architecture of DEVS-FIRE data assimilation using the proposed bootstrap filter framework. The rectangle boxes represent the major components in one step, and the circles and rounded rectangles represent the data/variables. At time step *t*, the set of system state variables (particles), i.e., the fire states, from time step *t*-1 (denoted as $Fire_{t\text{-}1}$ in Figure 4.2) are fed into the system transition model that is constructed by DEVS-FIRE simulation and fire

front noise. The resulting fire state set is denoted as $Fire'_t$. To compute the importance weights of the particles, for each fire state in $Fire'_t$, all sensor temperatures are computed according to the empirical model in section 4.5.3. The set of temperatures of all fire states in $Fire'_t$ are denoted as M'_t. Then considering each temperature set in M'_t as the mean vector, a likelihood of the real temperature vector m_t (the temperatures collected from real time sensors) is calculated from a multivariate Gaussian distribution as discussed in section 4.4. The likelihoods are used to update importance weights for this bootstrap filter. After normalizing the weights of all particles, a resampling algorithm is applied to generate $Fire_t$, and it is the input of the next step.

In Figure 4.2, the time step t-1, t, and t+1 are used to indicate the stepwise nature of the algorithm. The actual time interval between two consecutive steps is usually defined by how often the sensor data is collected, for example, in every 20 minutes.

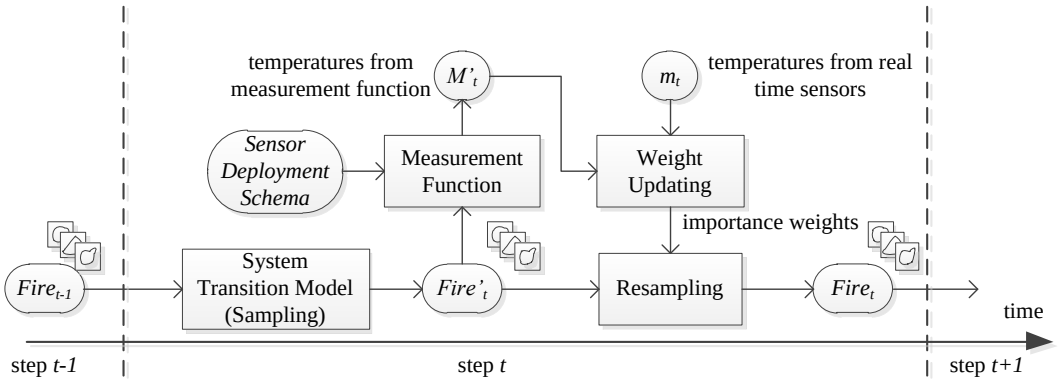


Figure 4.2 Wildfire data assimilation using a bootstrap filter

In this bootstrap filter, the set of fire states is represented by a set of particles. It starts by initializing N particles representing the initial fire states when the fire is ignited. Each particle's weight is initialized to 1/N. Then the algorithm goes through multiple iterations, each of which includes sampling, weight updating, and resampling stages. As discussed in section 4.1, different

from generic SMC algorithms, the sampling step ignores the current measurement data, and in this way the weight updating step can safely update importance weight by only likelihoods.

4.5.2 ***Construct the system transition distribution***

The system transition distribution is represented by a sampling algorithm. The goal of the sampling algorithm is to generate a fire state sample for the next time step given the current fire state. The samples drawn then approach the distribution $p(fire_t \mid fire_{t-1})$. Given $fire_{t-1}$, in order to draw a sample from $p(fire_t \mid fire_{t-1})$, DEVS-FIRE is first used. Specifically, for each particle, a DEVS-FIRE simulation is created starting from $t-1$ and run to t. The length of a time step is determined by how often the sensor data is collected. The DEVS-FIRE result is indicated by $\overline{fire_t}$. The fire perimeter (also called *fire front*) is extracted from $\overline{fire_t}$, and a graph noise is sampled and added to this fire front.

In order to add graph noise to the fire front, a fire front is divided into multiple segments (the number of segments is denoted as C_1), each of which consists of an equal number of burning cells. For each segment, a noise denoted as d_i is introduced that defines the change (in number of cells) inside or outside a cell along the direction from the ignition point to this cell. Different segments may have different noise, but all cells in the same segment share the same noise. Next, for each cell in a segment, it is moved to a new position that is d_i cells away. After reconnecting all the moved cells, a new fire front is formed (named as *the noised fire front*). The algorithm of drawing samples from $p(fire_t \mid fire_{t-1})$ is given in Algorithm 3. Figure 4.3 shows the results of four different runs of the algorithm for a given fire state. In each run, each noised fire front is compared with the original fire front of $\overline{fire_t}$.

ALGORITHM 3. *Fire Front Transition Sampling*

Input: The fire state at time step *t*-1 ($fire_{t-1}$), segment denominator (C_1), noise denominator (C_2), and noise variance (C_3).

Output: A sample of $p(fire_t \mid fire_{t-1})$.

1. $\overline{fire_t} = SF(fire_{t-1}, t-1)$;
2. Scan the fire front of $\overline{fire_t}$;
3. Divide the fire front into C_1 consecutive segments, denoted as $SEG_1, SEG_2, \ldots, SEG_{C_1}$;
4. Generate noise $d_1, d_2, \ldots, d_{C_1}$ for all the segments where
   $d_i \sim Gaussian\left(\frac{length\ of\ SEG_i}{C_2}, C_3\right)$;
5. For every burning cell *c* in segment SEG_i, move it to d_i cells away along the direction from the ignition point to the cell;
6. Connect all the segments according to the segment order to form a closed shape (referred to as the noised fire front), and set all the cells on the noised fire front to burning;
7. Set all the cells inside the noised fire front to burned and all the cells outside the noised fire front to unburned;
8. Return the fire state.

In this algorithm, the purpose of the graph noise method is to model the overall simulation error, and generate a *noised fire front* from an existing fire front. The effectiveness of this method is supported by several features that are built into the method. First, the cells of the same segment have the same noise level and different segments can have different noise levels. This is based on the fact that, in wildfire spread simulations, cells in nearby regions of a fire front tend to have similar distance errors. Second, guaranteed by step 4, larger noise comes with lower probability. In this way, with a high probability, the belief of current particles is inherited by the successor particles in the next step. Both of these features can be seen from Figure 4.3. Finally, small fires have smaller noises, and the noise level is controlled by C_1, C_2 and C_3.

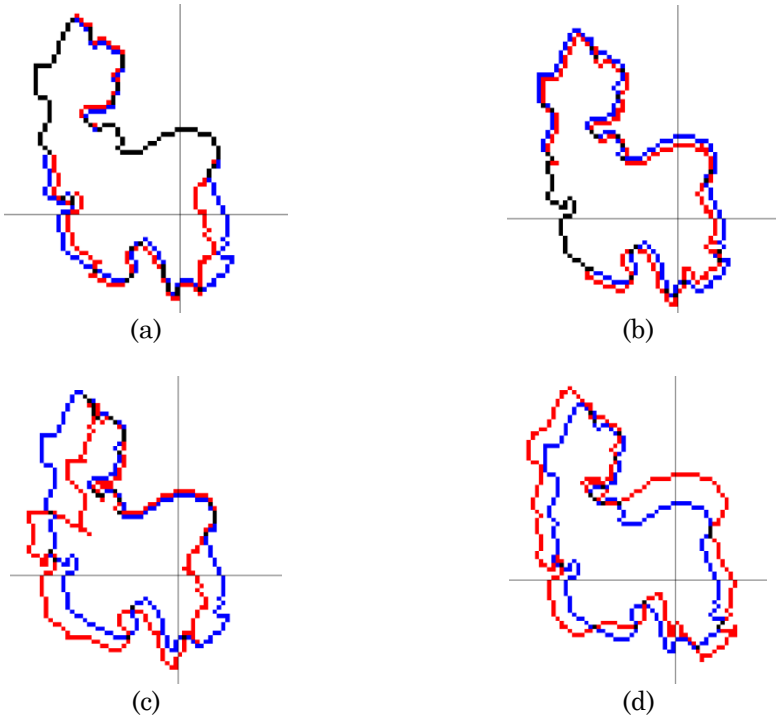
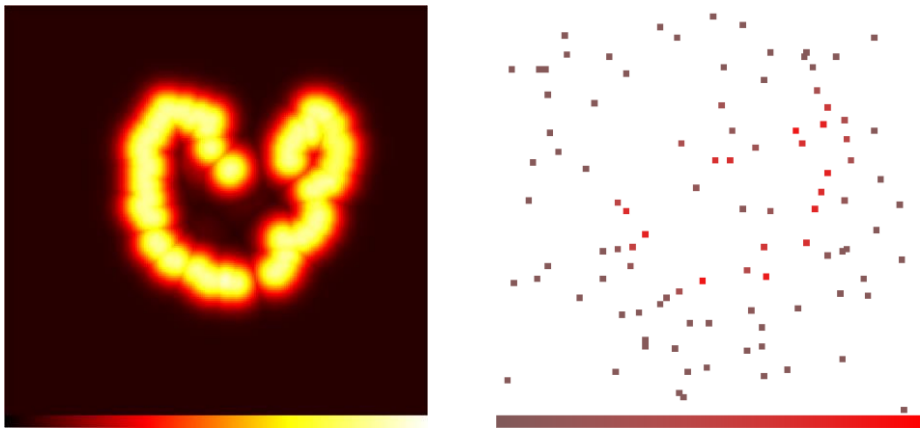


Figure 4.3 Noised fire fronts. In each of them, the original fire front of $\overline{fire_t}$ is shown in blue and black; the noised fire front is shown in red and black. (The black cells are on both the original fire front and the noised fire front.)

The fire shape error on the fire front could be caused by imprecise fuel data, terrain data, weather condition, fire model error and other uncertain elements affecting fire spreads. The assumption here is that the effect of these imprecise data and errors can be modeled by the fire front graph noise.

4.5.3 ***Construct the observation distribution***

The location of each temperature sensor is known, and each sensor reports the temperature every certain time interval. Due to measurement noise, for the same real temperature at the same location, a sensor may report different readings. Examples of sensor locations and temperature information are displayed in Figure 4.4.



(a) (b)

Figure 4.4 Examples of ground temperature sensor readings: (a) Real temperature map with temperature range [0℃, 400℃]; (b) Temperature readings of 100 randomly deployed sensors. (Each dot is a deployed temperature sensor, and its color from dark to bright indicates the temperature from 0 ℃ to 400 ℃.)

Given a fire state, each sensor reading is calculated. From a sensor location, for each fire cell, a temperature can be calculated from an empirical model:

$$T = T_c e^{\frac{-d^2}{2\sigma^2}} + T_a,$$

where T is the temperature of a sensor; T_c (°C) refers to the temperature rise above ambient temperature of the closest burning cell on the fire front; T_a denotes the ambient temperature (°C); d denotes the distance from the sensor to the closest burning cell; σ is a constant; T_c is calculated from

$$T_c = \frac{3.9FI^{2/3}}{h},$$

where FI is the fireline intensity of the burning cell (kW m^{-1}), and h is the height above ground (m). In the DEVS-FIRE simulation, the fireline intensity of a burning cell is computed from Rothermel's model at runtime. For ground temperature sensors, the height h would be their installation heights. The recorded sensor reading is the highest temperature among all the fire cells.

Figure 4.5 illustrates how the measurement function works in computing the temperature data of ground temperature sensors. It displays a simplified fire state, where 8 cells are burning (displayed in red) in a 9×9 cell space with each cell's resolution being 15 (m). The temperature sensors are regularly deployed in the cell space with one temperature sensor every three cells (in both horizontal and vertical directions). In Figure 4.5, the cells where sensors are deployed are displayed in gray. To illustrate how the temperature data are calculated, all burning cells' T_c is assumed to be 376 °C and the ambient temperature is assumed to be 27 °C. The temperature is then $T = 376e^{-d^2/2\sigma^2} + 27$ ($\sigma = 50$), where d is the distance from a sensor to its closest burning cell on the fire front. Based on this formula, the temperature data of all the sensors can be obtained, which are {149, 267, 267, 267, 386, 386, 267, 386, 371} (°C) indexed from left to right and from top to bottom. Note that in this example, the closest distances to the fire front for these sensors are {75, 45, 45, 45, 15, 15, 45, 15, 21} (m), and in this uniform fire space fire cells with these distances bring the highest temperatures.

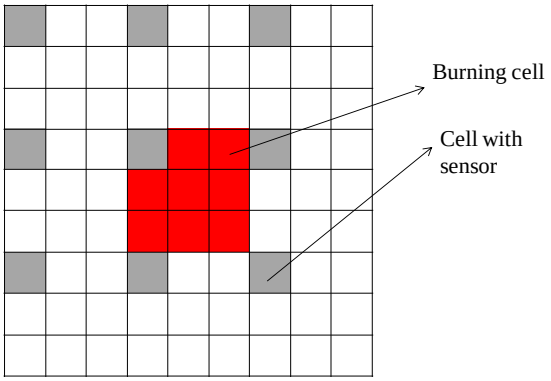


Figure 4.5 An example of the measurement model

A multivariate Gaussian distribution is used to model the observation distribution (as introduced in section 4.4, i.e.:

$$p(m_t \mid fire_t) = \frac{\exp(-\frac{1}{2}(m_t - MF(fire_t,t))'\Sigma^{-1}(m_t - MF(fire_t,t)))}{(2\pi)^{n_s/2}|\Sigma|^{1/2}},$$

where *MF* is the measurement function mapping a fire state to a set of temperatures according to the empirical model, Σ is the covariance matrix. Given the fire state of a particle and the current measurement, the likelihood can then be directly calculated.

4.5.4 ***Experiments of bootstrap filter data assimilation***

Experimental design. Identical-twin experiment is used to evaluate the data assimilation method. The purpose of identical-twin experiments is to study the assimilation in ideal situations and evaluate the proximity of the prediction to the true states in a controlled manner. In the identical-twin experiment, a simulation is first run, and the corresponding data are recorded. This simulation result is considered as "true"; therefore, the observation data obtained here are regarded as the real observation data (because they come from the "true" model). Consequently, observation data are used to improve simulation results using SMC methods, and then the effectiveness can be seen from if these estimated results are close to the "true" result.

Three terms are used: *real fire*, *filtered fire*, and *simulated fire*, to present the experimental results. A real fire is the simulated fire spread from which the real observation data are obtained. A simulated fire is the simulation result based on some "erroneous" data ("erroneous" in the sense that the data are different from those used in the real fire), for example, imprecise weather data. This is to represent the fact that wildfire simulations usually rely on imperfect data as compared to real wildfires. Finally, a filtered fire is the bootstrap filter enhanced simulation result based on the same "erroneous" data as in the simulated fire. The particle with the largest importance weight before resampling is chosen as the one with the filtered fire.

The differences between a real fire and a simulated fire are due to the imprecise data such as wind speed, wind direction, GIS data, and fuel model, used in the simulation. In these experiments, imprecise wind conditions (wind speed and wind direction) are used as the "erroneous" data. Table shows the configurations of four sets of experiments. The real wind speed and direction are 8 (mph) and 180 (degrees) with random variances added every 10 minutes. The variances for the wind speeds are in the range of –2 to 2 (mph) (denoted as 8±2 in the table), and the variances for the wind direction are in the range of -20 to 20 (degrees) (denoted as 180±20 in the table). Our first two experiment cases introduce errors to the wind speeds and make the wind directions to be exactly the same as the real wind direction. In case 1 the wind speed is randomly generated based on 6 (mph) with variances added in the range of –2 to 2 (mph). In case 2, the wind speed is randomly generated based on 10 (mph) with variances added in the range of –2 to 2 (mph). Our next two experiment cases introduce errors to the wind directions only: case 3 uses wind direction of 160 (degrees) with added variances in the range of ±30 (degrees); case 4 uses wind direction of 200 (degrees) with added variances in the range of ±30 (degrees). For wind directions, the degrees indicate the angle between the north direction clockwise to the direction from where the wind comes.

Table 4.1 Weather Data Used in Bootstrap Filter Experiments

<table>
<tr><td rowspan="2">Case</td><td colspan="2">Erroneous data</td><td colspan="2">Real data</td></tr>
<tr><td>Speed (mph)</td><td>Direction (degrees)</td><td>Speed (mph)</td><td>Direction (degrees)</td></tr>
<tr><td>1</td><td>6±2</td><td rowspan="2">No error</td><td rowspan="4">8±2</td><td rowspan="4">180±20</td></tr>
<tr><td>2</td><td>10±2</td></tr>
<tr><td>3</td><td rowspan="2">No error</td><td>160±30</td></tr>
<tr><td>4</td><td>200±30</td></tr>
</table>

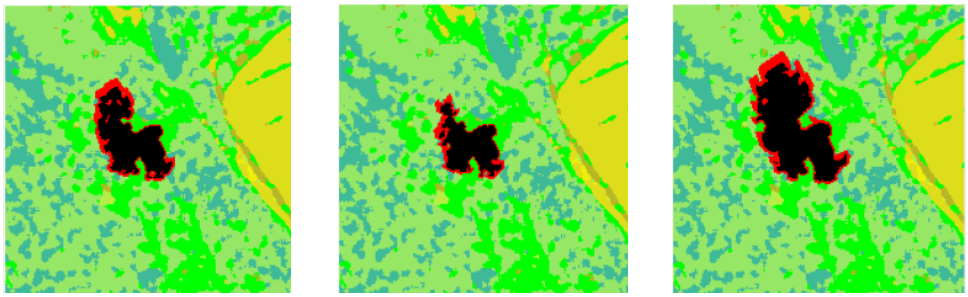
For all the experiment cases in Table 4.1, a regular sensor deployment schema is used where the sensors are regularly deployed with one sensor every 10 cells. All simulations use the

real world GIS data and fuel data. The cell space dimension is 200×200 and the cell size is 15 (m). Those data were acquired from Huntsville area, Texas, during the leaf-off season in March 2004 by M7 Visual Intelligence of Houston, Texas. The ignition point is set to the center point of the cell space for all of the simulations. The observation data (ground temperature sensor data) from the real fire are collected every 20 minutes. 50 particles are used in all SMC experiments and set C_1 to 6, C_2 to 20, and C_3 to 1 in Algorithm 3.

Experimental results with erroneous wind speeds. This set of experiments (case 1 and case 2) test the data assimilation results when the wind speed data have errors. Figure 4.6(a) displays the real fire after 8 steps (20 minutes each step) of the simulation. This real fire is used to generate the observation data for all our experiments. Figure 4.6(b) and Figure 4.6(c) show the simulated fires for case 1 and case 2 respectively for the same simulation duration (160 minutes). Note that all these fires are simulated from DEVS-FIRE using their corresponding data shown in Table 4.1. In the figures, the burning cells and the burned cells are displayed in red and black respectively. The other colors show different fuel types of cells. From the figures, it can be seen that the real fire and the simulated fires have large deviations due to the errors of the wind speeds. In case 1, the real fire spreads faster than the simulated fire because the real wind speeds (8±2 mph) are larger than the erroneous wind speeds (6±2 mph). In case 2, the real fire grows slower than the simulated fire since the real wind speeds (8±2 mph) are smaller than the erroneous wind speeds (10±2 mph).

By assimilating the observation data, the filtered fires are obtained. Figure 4.7 shows the results after 8 steps for case 1. Figure 4.7(a) shows the filtered fire; Figure 4.7(b) compares the real fire front (displayed in green) with the filtered fire front (displayed in blue. For comparison purpose, the simulated fire front (displayed in red) from Figure 4.6(b) is also shown in the figure.

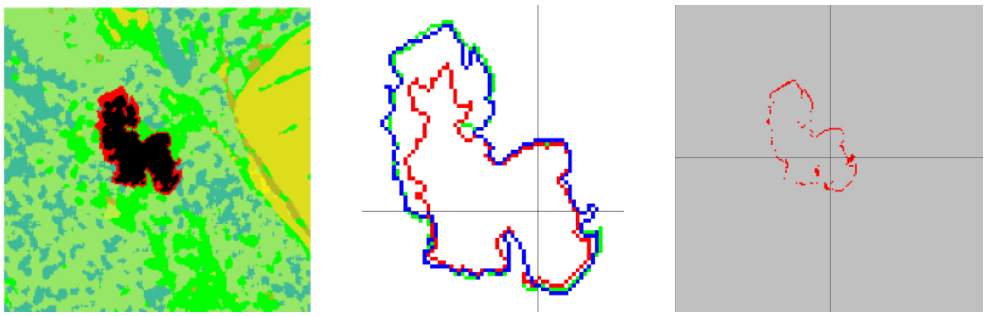
Figure 4.7(c) shows the cells that have mismatched states between the real fire and the filtered fire. From Figure 4.7(b), it is observed that the fire front of the filtered fire is much closer to the one of the real fire than that of the simulated fire. Specifically, due to the erroneous smaller wind speeds, the simulated fire is significantly smaller than the real fire at the head of the fire (the top part of the fire shape as shown in Figure 4.7). Using data assimilation, the filtered fire overcomes this problem, and matches the real fire front with smaller difference. Figure 4.7(c) confirms this and shows that the mismatched cells of the filtered fire are roughly evenly distributed along the real fire front.



(a) (b) (c)

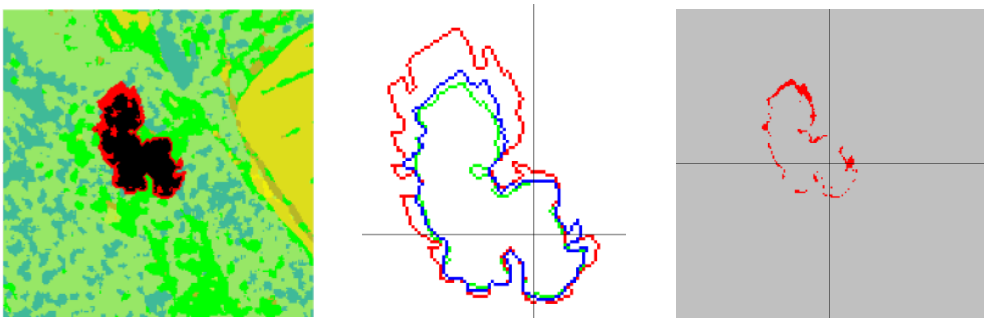
Figure 4.6 Real fire and simulated fires of bootstrap filter experiments (case 1 and 2). (a) Real fire after 160 minutes (average wind speed is 8 mph; average wind direction is 180 degrees); (b) Simulated fire for case 1 after 160 minutes (average wind speed is 6 mph; average wind direction is 180 degrees); (c) Simulated fire for case 2 after 160 minutes (average wind speed is 10 mph; average wind direction is 180 degrees).

Figure 4.8 shows the results after 8 steps for case 2 with similar display arrangement as in Figure 4.7. As shown in Figure 4.8(b), because in case 2 the wind speeds are generally larger than the real wind speeds, the simulated fire grows much larger than the real fire does. Figure 4.8 (b) and Figure 4.8(c) show that, after assimilating sensor data by the SMC method, simulation results are significantly improved since the difference between the real fire and the filtered fire is much smaller than the one between the real fire and the simulated fire.



(a) (b) (c)

Figure 4.7 Comparisons of the simulated fire and the filtered fire of bootstrap filter experiment (case 1). (a) Filtered fire; (b) Fire fronts of the real fire (displayed in green), simulated fire (displayed in red) and filtered fire (displayed in blue); (c) Mismatched cells (displayed in red) between the real fire and the filtered fire.



(a) (b) (c)

Figure 4.8 Comparisons of the simulated fire and the filtered fire of bootstrap filter experiment (case 2). (a) Filtered fire; (b) Fire fronts of the real fire (displayed in green), simulated fire (displayed in red) and filtered fire (displayed in blue); (c) Mismatched cells (displayed in red) between the real fire and the filtered fire.

To quantitatively show the data assimilation results, Figure 4.9(a) and Figure 4.9(b) show the fire perimeters and burned areas of the real fire, the simulated fire, and the filtered fire for case 1 from time step 1 to 8. Figure 4.9(c) and Figure 4.9(d) show the fire perimeters and burned areas of the real fire, the simulated fire, and the filtered fire for case 2 from time step 1 to 8. In both cases, 10 independent runs are carried out of the data assimilation experiments, and display the average of the results in the figures. For case 1, after the simulation is finished, the standard

deviations of the filtered fires from the 10 runs are 0.36 km for perimeter and 1.26 hectares for burned area. For case 2, the standard deviations are 0.51 km for perimeter and 2.84 hectares for burned area. These figures show that the differences between the real fire's perimeters and the filtered fires' perimeters are smaller than those for the simulated fires. The same trend holds true for the burned areas.

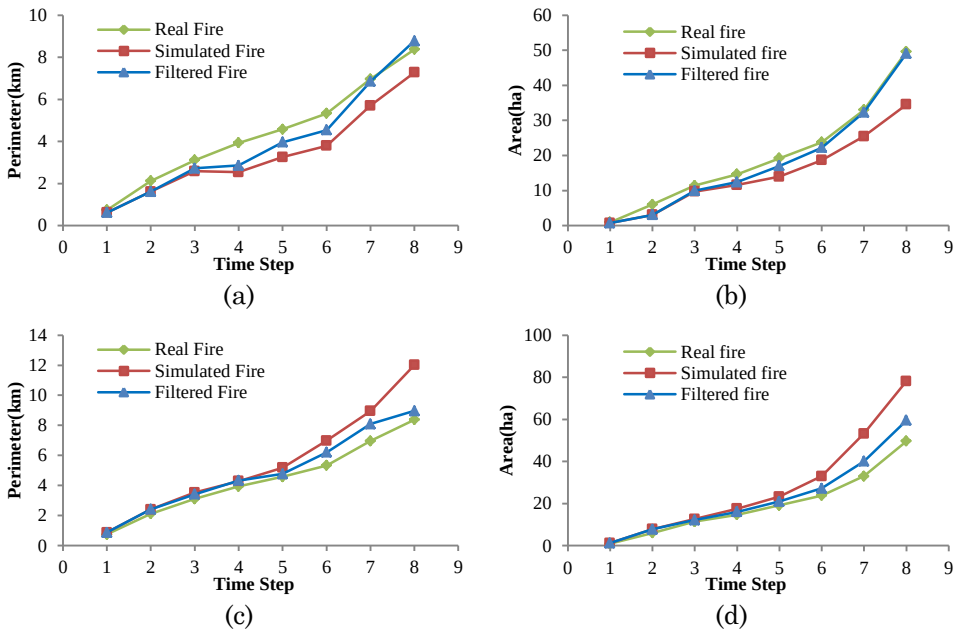


Figure 4.9 Perimeters and burned areas of the real fire, simulated fires, and filtered fires of bootstrap filter experiment (case 1 and 2). (a) Perimeters for case 1; (b) Burned areas for case 1; (c) Perimeters for case 2; (d) Burned areas for case 2.

Mismatched cells (i.e. the symmetric set differences) are used to calculate the error between a real fire and a simulated fire or filter fire. Figure 4.10 shows the symmetric set differences of the simulated fire (compared to the real fire) and that of the filtered fire (compared to the real fire) in case 1 (Figure 4.10(a)) and case 2 (Figure 4.10(b)) from time step 1 to 8. In these figures, the values of the filtered fire are the averages of 10 independent runs. The horizontal axis represents the time step, and the vertical axis represents the symmetric set difference value in

terms of the number of cells. From the figures, it can be seen that the symmetric set differences of the filtered fires are smaller than those of the simulated fires after step 2. With the increase of time step, (i.e., when more sensor data are assimilated), the difference between the simulated fire and the filtered fire becomes more and more notable. At step 8, the symmetric set difference of the filtered fire is less than half of the symmetric set difference of the simulated fire. This experiment demonstrates the effectiveness of the data assimilation method in wildfire spread simulation when using imprecise wind speeds.

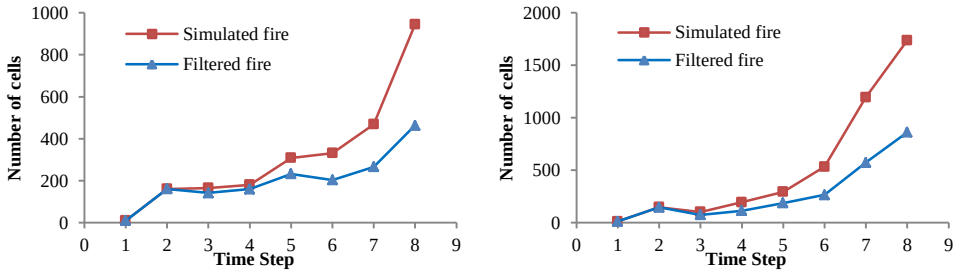
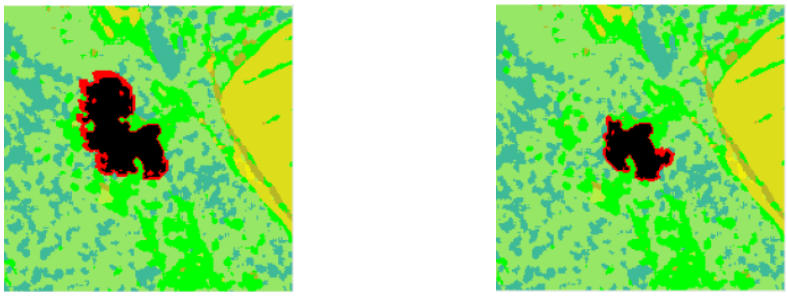


Figure 4.10 Errors of bootstrap filter experiment (case 1 and 2). (a) Case 1; (b) Case 2.

Experimental results with erroneous wind directions. This set of experiments (case 3 and case 4) examines the data assimilation results when the wind direction data have errors. DEVS-FIRE is used to obtain the simulated fires using the wind direction data in Table 4.1. Figure 4.11(a) and Figure 4.11(b) show the fire growth of the two simulated fires for case 3 and case 4 after 160 minutes.

Same as before, the temperature sensor data are dynamically assimilated into the simulations. The data assimilation results are displayed in Figure 4.12 for case 3, and Figure 4.13 for case 4. As can be seen, because of the incorrect wind directions, there are large differences between the real fire and the simulated fires. From Figure 4.12(b) one can see that the simulated fire grows much faster than the real fire. This is because the fuel types along the wind direction

(northwest) of the simulated fire make it easier for the fire to spread fast. In Figure 4.13(b), the simulated fire is smaller than the real fire because the fuel types along the wind direction (north-east) in case 4 are harder for the fire to spread. In both cases, the differences between the filtered fires and the real fire are much smaller than those of the simulated fire, as can be seen from Figure 4.12(b) and Figure 4.13(b).

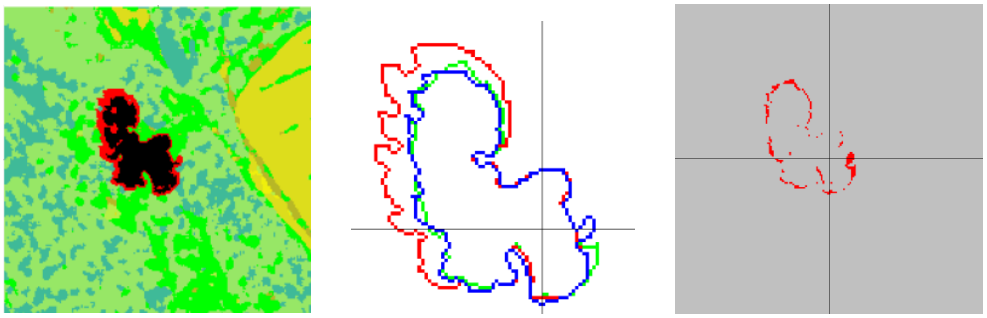


(a) (b)

Figure 4.11 Simulated fires of bootstrap filter (case 3 and 4). (a) Simulated fire for case 3 after 160 minutes (average wind speed is 8 mph; average wind direction is 160 degrees); (b) Simulated fire for case 4 after 160 minutes (average wind speed is 8 mph; average wind direction is 200 degrees).

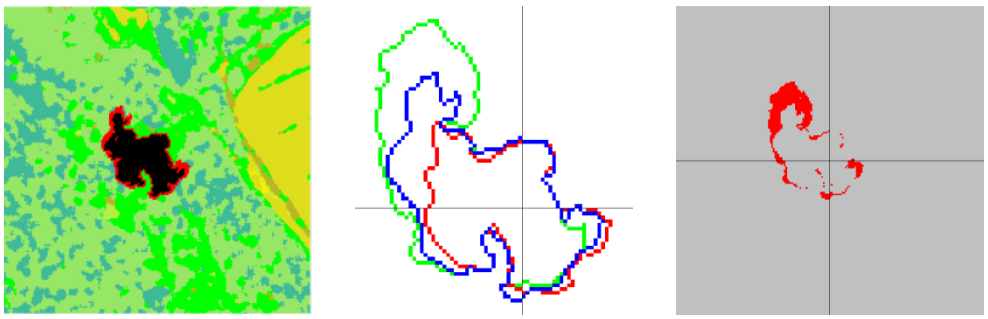
However, from Figure 4.13(b), it should be noticed that the data assimilation in case 4 is not as effective as in cases 1-3, although it still gives improved result compared to the simulated fire. The reason behind this is that, in case 4, the wind direction in the filtered fire is towards northeast, where it is hard for a fire to spread due to the fuel data in that direction, whereas the wind direction in the real fire is towards north where it is easy to spread. This difference of wind direction results in significantly different fire spreading behavior, which is also reflected by the large difference between the real fire shape and the simulated fire shape as shown in Figure 4.13(b). In this situation, the developed data assimilation method does not work as effectively to keep track of the real fire as in other cases, where there are relatively smaller differences between the real fire and the simulated fires. This indicates a future research task to develop techniques to improve data assimilation results. For example, the current method does not differenti-

ate different regions when modeling the system transition noise. A more advanced method may differentiate different parts of the fire and add graph noise according to the fire spreading behavior (e.g., higher level of graph noise for faster spreading fire regions).



(a) (b) (c)

Figure 4.12 Comparisons of the simulated fire and the filtered fire of bootstrap filter (case 3). (a) Filtered fire; (b) Fire fronts of the real fire (displayed in green), simulated fire (displayed in red) and filtered fire (displayed in blue); (c) Mismatched cells (displayed in red) between the real fire and the filtered fire.



(a) (b) (c)

Figure 4.13 Comparisons of the simulated fire and the filtered fire of bootstrap filter (case 4). (a) Filtered fire; (b) Fire fronts of the real fire (displayed in green), simulated fire (displayed in red) and filtered fire (displayed in blue); (c) Mismatched cells between the real fire and the filtered fire.

Figure 4.14 displays the perimeters and burned areas of the real fires, the simulated fires, and the corresponding filtered fires for case 3 and case 4. In both cases, 10 independent runs of the data assimilation experiments are carried out to show the averaged results for the filtered fires in the figures. In case 3, the standard deviations of the filtered fires in the last step are 0.36

km for perimeter and 1.30 hectares for burned area. In case 4, the standard deviations in the last step are 0.67 km for perimeter and 3.73 hectares for burned area. In both case 3 (Figure 4.14(a) and Figure 4.14(b)) and case 4 (Figure 4.14(c) and Figure 4.14(d)), compared to the ones of the simulated fires the perimeters and burned areas of the filtered fires are closer to those of the real fire.

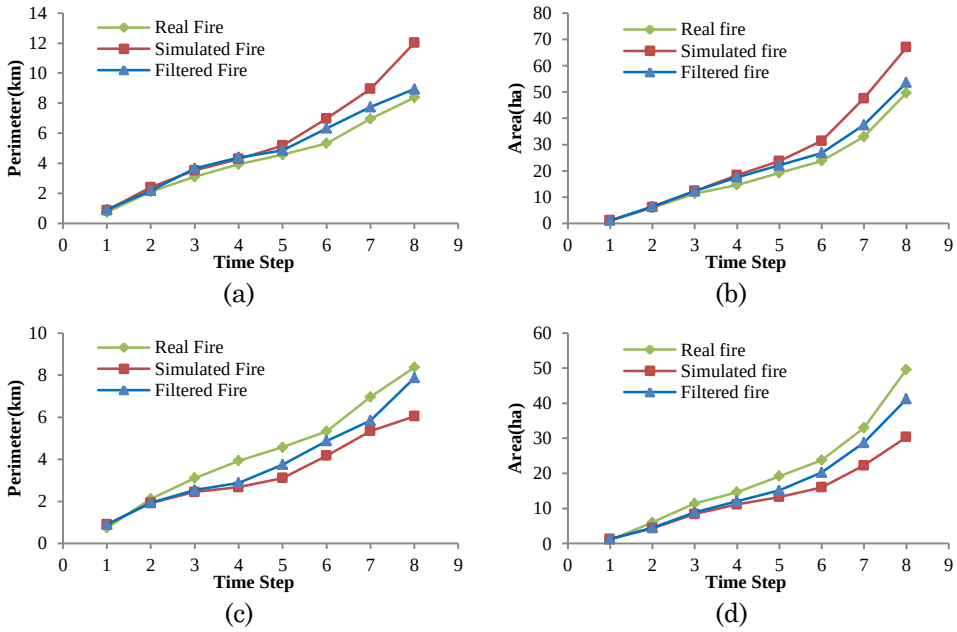


Figure 4.14 Perimeters and burned areas of the real fire, simulated fires, and filtered fires of bootstrap filter experiment (case 1 and 2). (a) Perimeters for case 3; (b) Burned areas for case 3; (c) Perimeters for case 4; (d) Burned areas for case 4.

To quantitatively see the effect of data assimilation on fire shape, Figure 4.15 shows the symmetric set differences for case 3 (Figure 4.15(a)) and case 4 (Figure 4.15(b)) from time step 1 to 8. These results are coincident with those in Figure 4.12 and Figure 4.13 From Figure 4.15, it can be observed that the symmetric set difference between the real fire and the filtered fire (displayed in blue) is smaller than that between the real fire and the simulated fire (displayed in red)

in both case 3 and case 4, even though the effectiveness in case 4 is lower as explained before. In general, this set of experiments show that the simulation results are improved after assimilating sensor data using the developed data assimilation method.

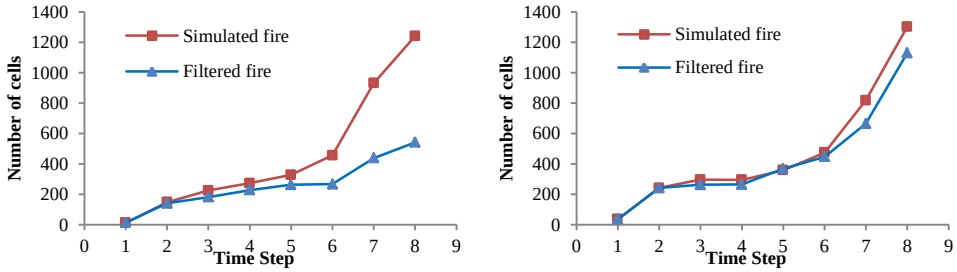


Figure 4.15 Symmetric set differences of bootstrap filter experiment (case 3 and 4). (a) Case 3; (b) Case 4.

*4.5.5 **Influence of Deployed Sensor Amount and Locations***

When sensors are relatively evenly deployed, the specific deployment and number of sensors also affect the results of data assimilation. Experiments are carried out using the same weather data of case 1 in Table 4.1. Five sensor deployment schemas (as shown in Figure 4.16): 1) regular deployment – one sensor per 10 cells (totally 400 sensors); 2) regular deployment – one sensor per 20 cells (totally 100 sensors); 3) regular deployment – one sensor per 40 cells (totally 25 sensors)); 4) randomly deployed 100 sensors 5) randomly deployed 100 sensors.

To show the impact, mismatched cells of the simulated fire and the filtered fires from the 3 regular deployments are displayed in Figure 4.17. Compared with the simulated fire (Figure 4.17 (a)), all of the filtered fires (Figure 4.17 (b)-(c)) have fewer mismatched cells. Moreover, as the sensor number increases, it can be clearly seen that the number of mismatched cells decreases. This is confirmed by Figure 4.18, which shows the quantitative result of symmetric set difference of the simulated fire and the three filtered fires. Filtered fires all give better results (less

symmetric set difference) than the simulated fire does. Also, more sensors generally produce better results. When sensors are regularly deployed, with the increase of employed sensor amount, more information of the real system is collected through sensor readings.

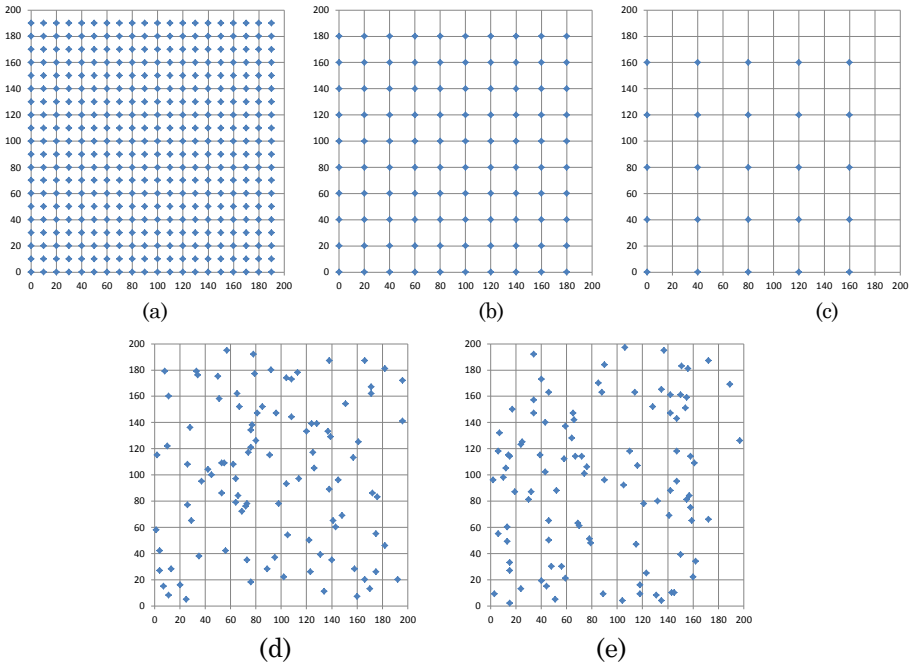


Figure 4.16 Sensor deployments (a) Regularly deployed 400 sensors; (b) Regularly deployed 100 sensors; (c) Regularly deployed 25 sensors; (d) Randomly deployed 100 sensors; (e) Randomly deployed 100 sensors.

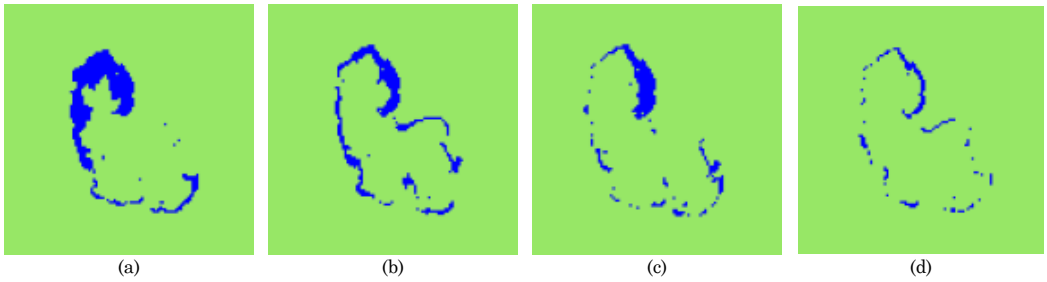


Figure 4.17 Mismatched cells (a) Simulated fire; (b) Filtered fire with 25 regularly deployed sensors; (c) Filtered fire with 100 regularly deployed sensors; (d) Filtered fire with 400 regularly deployed sensors.

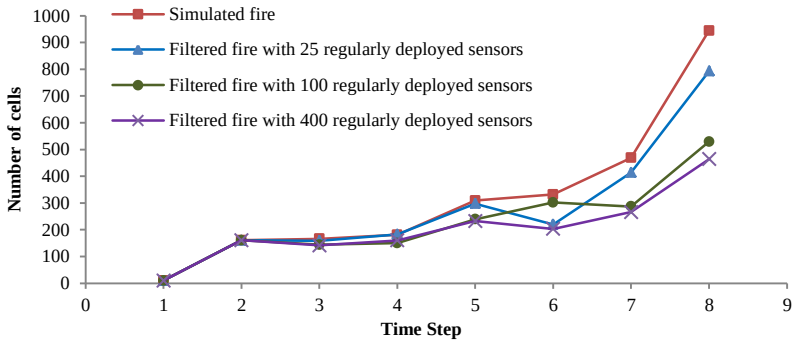


Figure 4.18 Symmetric set difference for the simulated fire and the filtered fires with regular sensor deployment schemas.

To show the influence of sensor locations, Figure 4.19 compares the symmetric set difference of the simulated fire and three filtered fires that use the same number of sensors but with different sensor deployments (one regular deployment and two random deployments, all with 100 sensors). As expected, all the filtered fires have smaller errors than that of the simulated fire. Among the filtered fires, the random deployment shown in Figure 4.16(e) gives the least improved results, and the random deployment of Figure 4.16 (d) produces the most improved result for most time steps (except for the last step). The results from the regular deployment lie in between the above two. Comparing the three deployment schemas as shown in Figure 4.16 (b), Figure 4.16 (d) and Figure 4.16 (e), one can see that Figure 4.16 (e) has more empty spaces (not covered by the sensors) in the center area of the cell space where the fire is ignited. As a result, the sensors in Figure 4.16(e) carry less information about the spreading of the real fire in many of the time steps. This explains why sensor deployment in Figure 4.16(e) gives the least improved result. On the contrary, the sensor deployment in Figure 4.16(d) has more sensors in the fire spreading area, and thus produces most improved result. This experiment shows that sensor locations affect the amount of information from sensors, and thus have impact in data assimila-

tion results. With a fixed number of sensors, how to design strategies to deploy the sensors in an effective manner could be a future research direction.

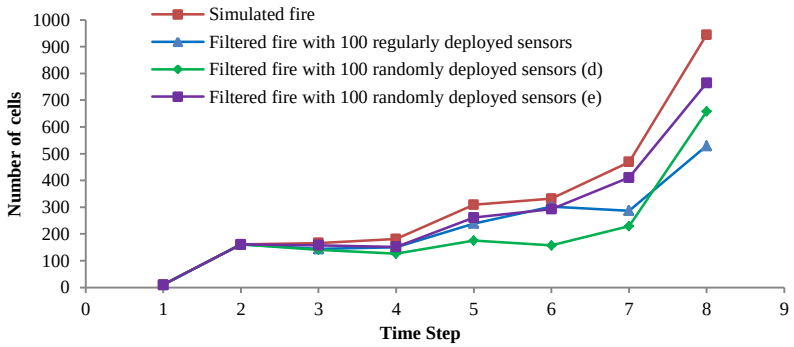


Figure 4.19 Symmetric set difference for the simulated fire and the filtered fires with different deployment schemas.

More case studies of this bootstrap filter based framework on wildfire spread simulation and lane-based traffic simulation are carried out in section 5.5 and 5.6, where they are also compared with the results produced by the SenSim framework.

5 SENSIM DATA ASSIMILATION FRAMEWORK

In this section, the SMC data assimilation framework for *sensor monitored spatial-temporal systems* is presented. It contains an analytical distribution of observation distribution $p(m_t|s_t)$, an effective proposal distribution $q(s_t|s_{t-1}, m_t)$ (named *SenSim* proposal), and a kernel method based importance weight estimation method. The proposed framework is named as "*SenSim framework*", and it uses the above components to assimilate observation data with the SISR algorithm.

5.1 Sensor Monitored Spatial-Temporal Systems

As presented in section 3, the analytical expressions of $p(s_t|s_{t-1})$, $p(m_t|s_t)$ and $q(s_t|s_{t-1}, m_t)$ are used in equation (19) to update importance weights. However, they may not be available in the case of very complex simulation models. In this section, a special class of systems is defined, named as "*sensor monitored spatial-temporal systems"*. With these systems and their assumptions defined below, section 5.2 and 5.3 build each SMC component to perform data assimilation with sensor readings.

A *sensor monitored spatial-temporal system* is defined as a seven-tuple: $< A, S, \mathcal{F}, \mathcal{M}, C, \Delta_s, \Delta_c >$. A is a two dimensional area on which system states are defined (termed as *system area*); S is the state space, which is the set of all possible system states; $\mathcal{F}: S \times \mathbb{R}^{n_u} \to S$ is the *system transition function*, and it determines the advance of a system state:

$$s_t = \mathcal{F}(s_{t-1}, u_t), \tag{28}$$

where s_{t-1}, s_t are system states at $t-1$ and t, and u_t is a n_u dimension vector indicating other inputs that may contain random variables; $\mathcal{M}: S \times \mathbb{R}^2 \to \mathbb{R}^{n_r}$ is the *measurement function*, and it determines the true sensor reading given a system state and a sensor position:

$$r = \mathcal{M}(s_{t,A_m},\ p\ ), \tag{29}$$

where r is a true sensor reading that is a n_r dimension vector, $p \in A$ is a sensor location, $A_m \subseteq A$ is a sub-area in A, and s_{t,A_m} is the local system state at time t of the area A_m; $C = \{c_i =< m_i, p_i, D_i > \mid p_i \in A, D_i \subseteq A, 1 \le i \le n_c\}$ is the sensor set, where, for the i-th sensor (c_i), m_i is the measured sensor reading, p_i is the sensor location, and D_i is the detection area; $\Delta_s: S \times S \to \mathbb{R}$ is the distance function of two system states; $\Delta_c: \mathbb{R}^{n_r} \times \mathbb{R}^{n_r} \to \mathbb{R}$ is the distance function of two sensor readings. These distance functions describe how different of two system states and two sensor readings. In the proposed methods, Δ_s is used to update importance weights, and $\Delta_c$is used to construct the observation model.

The following assumptions are made on *sensor monitored spatial-temporal systems*:

(1) Given a partition of A, $\{A_i | A_i \subseteq A, \forall k \neq l, A_k \cap A_l = \emptyset, 1 \le i \le n_A, \bigcup_{i=1}^{n_A} A_i = A\}$, a system state $s \in S$ can be broken into and reconstructed from n_A local states $\{s_{A_i} | 1 \le i \le n_A\}$, where s_{A_i} is the local system state in A_i;

(2) It is easy to draw samples from $p(u_t)$;

(3) The analytical forms of $\mathcal{F}$ and $\mathcal{M}$ are unknown, but they can be used as black-boxes;

(4) It is easy to draw samples from $p(s_{A_T} | T,\ s')$, where T is a sensor cluster covering A_T, s' is the simulation model generated system state, and s_{A_T} is the local system state in A_T. This distribution is termed as *local sensor proposal* in this manuscript.

Given such a *sensor monitored spatial-temporal system*, the transition distribution and observation distribution proposed in section 4.3 and 4.4 can be straightforwardly constructed. In section 5.2 and 5.3, a more effective proposal distribution and a weight updating method are introduced.

5.2 SenSim Proposal

As discussed in section 3.3.5, the optimal proposal distribution is proportional to $p(s_t|s_{t-1})p(m_t|s_t)$, and it implies that a system state with both high transition probability density and high likelihood is likely to have a high probability density in the optimal proposal distribution. Following this principle, a three-step sampling algorithm to generate system state samples that potentially have a high density value of $p(s_t|s_{t-1})$ and a high likelihood of $p(m_t|s_t)$ is proposed. This algorithm utilizes the knowledge from both sensor readings and simulation models, so the corresponding proposal distribution is named as "*SenSim* proposal". Given the measured sensor readings at t, m_t, and a system state sample at $t-1$, $\mathrm{s}_{t-1}^{(i)}$, it generates the possible system states at t as follows.

5.2.1 ***Step-1: Simulation Model Generated States***

Similar to the bootstrap filter, a input sample for the transition function is drawn from $p(u_t)$, indicated as $u_t^{(i)}$ which may include random parameters, static parameters and noise. The system transition function (that is, equation (28)) is then employed to generate a system state at t:

$$s'^{(i)}_t = \mathcal{F}(\mathrm{s}_{t-1}^{(i)}, u_t^{(i)}).$$

It is equivalent to drawing a sample from the system transition distribution $p(s_t|s_{t-1})$. $s'^{(i)}_t$ then only holds the belief of $\mathcal{F}$ that is commonly implemented by simulation models for non-

analytical complex systems. The likelihood of this sample, $p(m_t|s'^{(i)}_t)$, may be very small, especially when $p(m_t|s_t)$ has a much narrower support than $p(s_t|s_{t-1})$ (that is, a peaked likelihood), or the high value area of $p(m_t|s_t)$ is inside the tail area of $p(s_t|s_{t-1})$ (that is, a rare event).

5.2.2 ***Step-2: Sensor Reading Generated Local States***

The *system area A* is partitioned into a collection of independent sub areas:

$$\{A_j|1 \le j \le n_A\} \cup \{A_{cmp}\},$$

where A_j is covered by a cluster of sensor T_j, $A_{cmp} = A - \bigcup_{j=1}^{n_T} A_j$, and n_A is the number of local areas covered by sensors. $s'^{(i)}_t$ can then be divided into a collection of local states according to the partition:

$$\{s'^{(i)}_{t,A_j}|1 \le j \le n_A\} \cup \{s'^{(i)}_{t,\,A_{cmp}}\}.$$

For each local area A_j, a local state sample $s''^{(i)}_{t,A_j}$ is then drawn from the *local sensor proposal* $p(s_{A_j}|T_j,\ s'^{(i)}_t)$ (as defined in section 5.1).

As a result, on each local area A_j, there are two possible local states: $s'^{(i)}_{t,A_j}$ from simulation models, and $s''^{(i)}_{t,A_j}$ from sensor readings. $s'^{(i)}_{t,A_j}$ is more likely to bring a higher transition density value but lower likelihood than those of $s''^{(i)}_{t,A_j}$; on the other hand, $s''^{(i)}_{t,A_j}$ is inclined to produce high likelihoods but low transition density values. Thus, in certain cases, such as a simulation model has much less uncertainty than a *local sensor proposal*, $s''^{(i)}_{t,A_j}$ is more likely to fail; in other cases, such as a rare event happens, only $s''^{(i)}_{t,A_j}$ may work. It is then important to let both

of them have certain chance to appear in the final system state sample set at t, and the last step makes it happen.

*5.2.3 **Step-3: Sampling Local States***

Depending on the accuracy and uncertainty levels of the simulation models and sensor readings, for each local area A_j, confidence levels (in the range of $[0, 1]$) are defined as $c_{sen}^{A_j}$ and $c_{sim}^{A_j}$, to model the confidence in sensor readings and simulation models. For each A_j, a local state sample $s_{t,A_j}^{(i)}$ is drawn from:

$$p(s_{t,A_j}|s'^{(i)}_{t,A_j}, s''^{(i)}_{t,A_j}) = \frac{c_{sim}^{A_j}}{c_{sen}^{A_j}+c_{sim}^{A_j}}\delta(s_{t,A_j} - s'^{(i)}_{t,A_j}) + \frac{c_{sen}^{A_j}}{c_{sen}^{A_j}+c_{sim}^{A_j}}\delta(s_{t,A_j} - s''^{(i)}_{t,A_j}) \quad (30)$$

A possible system state sample, $s''^{(i)}_t$, is reconstructed from:

$$\{s_{t,A_j}^{(i)}\}_{j=1}^{n_A} \cup \{s'^{(i)}_{t,A_{cmp}}\}. \quad (31)$$

By an overall sensor reading confidence c_{sen} and an overall simulation model dence c_{sim}, a final system state sample is drawn from simulation model generated $s'^{(i)}_t$ and sensor reading affected $s''^{(i)}_t$:

$$p(s_t|s_{t-1}^{(i)}, m_t) = \frac{c_{sim}}{c_{sen}+c_{sim}}\delta(s_t - s'^{(i)}_t) + \frac{c_{sen}}{c_{sen}+c_{sim}}\delta(s_t - s''^{(i)}_t).$$

SenSim proposal sampling is summarized in Algorithm 4.

ALGORITHM 4. *Drawing Samples from Sensim Proposal*

Input: $\boldsymbol{s_{t-1}^{(i)}}$, $\boldsymbol{m_t}$, $\{\boldsymbol{c_{sen}^{A_j}}\}_{j=1}^{j=n_A}$, $\{\boldsymbol{c_{sim}^{A_j}}\}_{j=1}^{j=n_A}$, $\boldsymbol{c_{sen}}$, $\boldsymbol{c_{sim}}$;

Output: $\boldsymbol{s_t^{(i)}}$.

1. Sampling from the simulation model

   Draw $\boldsymbol{u_t^{(i)}}$ from $\boldsymbol{p(u_t)}$;

   $\boldsymbol{s'^{(i)}_t \leftarrow \mathcal{F}(s_{t-1}^{(i)}, u_t^{(j)})}$;

2. Sampling from sensor readings

   Partition $\boldsymbol{s'^{(i)}_t}$ to $\{\boldsymbol{s'^{(i)}_{t,A_j}} | \boldsymbol{1 \le j \le n_A}\} \cup \{\boldsymbol{s'^{(i)}_{t,A_{cmp}}}\}$;

   Repeat for $\boldsymbol{j}$ = 1 to $\boldsymbol{n_A}$

      Draw $\boldsymbol{s''^{(i)}_{t,A_j}}$ from $\boldsymbol{p(s_{t,A_j}^{(i)} | T_j, s'^{(i)}_t)}$;

3. Sampling local states

   Repeat for $\boldsymbol{j}$ = 1 to $\boldsymbol{n_A}$

      Draw $\boldsymbol{s_{t,A_j}^{(i)}}$ from $\boldsymbol{\frac{c_{sim}^{A_j}}{c_{sen}^{A_j}+c_{sim}^{A_j}} \delta(s_{t,A_j} - s'^{(i)}_{t,A_j}) + \frac{c_{sen}^{A_j}}{c_{sen}^{A_j}+c_{sim}^{A_j}} \delta(s_{t,A_j} - s''^{(i)}_{t,A_j})}$;

   $\boldsymbol{\mathbf{s}''^{(i)}_t \leftarrow \{s_{t,A_j}^{(i)}\}_{j=1}^{n_A} \cup \{s'^{(i)}_{t,A_{cmp}}\}}$;

   Draw $\boldsymbol{s_t^{(i)}}$ from $\boldsymbol{\frac{c_{sim}}{c_{sen}+c_{sim}} \delta(s_t - \mathbf{s}'^{(i)}_t) + \frac{c_{sen}}{c_{sen}+c_{sim}} \delta(s_t - \mathbf{s}''^{(i)}_t)}$.

Return $\boldsymbol{s_t^{(i)}}$.

5.3 Kernel Method Based Weight Updating

When a system state sample at time t, $s_t^{(i)}$, is obtained, equation (19) is used to update the importance weight. For *sensor monitored spatial-temporal systems*, although a convenient $p(m_t|s_t)$ is constructed, applying equation (19) is still problematic since analytical forms of $p(s_t|s_{t-1})$ and $q(s_t|s_{t-1}, m_t)$ are still unknown. This manuscript is then proposes to use another layer of samples to estimate both $p(s_t^{(i)}|s_{t-1}^{(i)})$ and $q(s_t^{(i)}|s_{t-1}^{(i)}, m_t)$ by the kernel method.

The kernel method is a non-parametric method for density function estimation as presented in (Rosenblatt 1956). It was also used to improve the resampling step of SMC methods by reconstructing posterior distributions from particles, examples can be found in (Musso, Oudjane et al. 2001) and (Cheng and Ansari 2005). In this paper, it is employed to estimate the values of system transition and proposal probability density functions.

Given a set of samples, $\{x_i\}_{i=1}^{n}$, of a probability density function $p(x)$, the kernel method estimates this function as:

$$p(x) \approx \frac{1}{nh^{n_x}} \sum_{i=1}^{n} K(\frac{x-x_i}{h}), \tag{32}$$

where $K: \mathbb{R}^{n_x} \to \mathbb{R}$ is a symmetric probability density function (named *kernel function*), n_x is the number of dimensions of x, and $h > 0$ is the bandwidth. Optimal *K* and *h* are the ones minimizing the error from the true density function to the kernel estimated density function.

The samples in a SMC methods can be used to estimate the system state posterior density function, but they are not the samples approaching the density functions of ($p(s_t|s_{t-1})$ and $q(s_t|s_{t-1}, m_t)$); for each of them extra *M* samples from $p(s_t|s_{t-1}^{(i)})$ and $q(s_t|s_{t-1}^{(i)}, m_t)$ are then needed to estimate $p(s_t^{(i)}|s_{t-1}^{(i)})$ and $q(s_t^{(i)}|s_{t-1}^{(i)}, m_t)$.

When $s_t^{(i)}$ is ready, after *M* executions of Algorithm 4, a sample set, $\{\dot{s}_t^{(k)}\}_{k=1}^{M}$, is then formed for $q(s_t|s_{t-1}^{(i)}, m_t)$; similarly, only applying the first step "sampling from the simulation model" of Algorithm 4, the sample set, $\{\ddot{s}_t^{(k)}\}_{k=1}^{M}$, approaching $p(s_t|s_{t-1}^{(i)})$ can be obtained. Plugging each sample set into equation (32) with $s_t^{(i)}$, estimates of $p(s_t^{(i)}|s_{t-1}^{(i)})$ and $q(s_t^{(i)}|s_{t-1}^{(i)}, m_t)$ are then obtained.

With this method, the "weight updating" step of Algorithm 1 can be completed. In the context of complex system data assimilation, the extra computation load is mostly from the executions of simulation models (that is, the implementation of the $\mathcal{F}$ function). Since each execution of Algorithm 4 can generate two samples, one for $p(s_t^{(i)}|s_{t-1}^{(i)})$ and one for $q(s_t^{(i)}|s_{t-1}^{(i)}, m_t)$. The total extra computation load incurred is $N \times M$ executions of the simulation model. The computation complexity is still independent with t. Also, when using equation (31) to estimate a

density value from a sample set, samples can be used in a sequential way. That is, after $K(\frac{x-x_i}{h})$ is calculated, x_i can be discarded. As a result, the extra memory space incurred by this kernel method is *O*(1) that is critical for memory-consuming simulation models where particle numbers are limited by the size of memory.

5.4 Weight Updating for High Dimensional Systems

With enough samples, the kernel method produces sound estimates of $p(s_t^{(i)}|s_{t-1}^{(i)})$ and $q(s_t^{(i)}|s_{t-1}^{(i)}, m_t)$, and then correctly updates importance weights for the proposed SenSim framework. However, when system dimensions are high, the estimation generated by limited number of samples could be inaccurate.

As expressed in equation (19), an importance weight is updated by three factors: the density of the observation distribution (i.e. the likelihood), the density of the transition distribution, and the density of the proposal distribution. The likelihood shows how the particle is consistent with observation data; the transition density shows how the particle is consistent with the state in the last time step; the proposal density shows how possible a particle is generated. Higher levels of consistence with observation data and previous states increase importance weights; on the other hand, the more easily a particle is generated, a lower weight it receives.

In the proposed SMC data assimilation method, the likelihood is not affected by the kernel method, but the other two may receive poor estimates from the kernel method with insufficient number of samples. They may in turn generate inaccurate importance weights and fail the data assimilation method.

To avoid the problem incurred by inaccurate kernel method density estimation, this manuscript proposes to use constant numbers to indicate the consistence levels of a particle to the previous state (i.e. estimate the transition density), and how possible a particle is drawn (i.e. es-

timate the proposal density). The SenSim proposal generates two types of particles: the ones only from simulation models and the ones affected by sensor readings. For the first type of particles, a predefined large constant number is used as the transition density; for second type of particles, a predefined small constant number is used as the transition density. The parameter of the SenSim proposal gives the basic information about how possible a particle is generated. The first type of particles are generated with a probability of $\frac{c_{sim}}{c_{sen}+c_{sim}}$, and the second type of particles are with a probability of $\frac{c_{sen}}{c_{sen}+c_{sim}}$. They are then used as proposal densities to update importance weights. The procedure of estimating importance weights for high dimensional systems is summarized in Algorithm 5, where *L* and *H* are two constant numbers and $H > L$.

ALGORITHM 5. *Estimate Importance Weights for High Dimensional Systems*

Input: $\boldsymbol{s_t^{(i)}}$, $\boldsymbol{m_t}$, $\boldsymbol{c_{sen}}$, $\boldsymbol{c_{sim}}$, $\boldsymbol{H}$, $\boldsymbol{L}$, $\boldsymbol{w(s_{t-1}^{(i)})}$

Output: $\boldsymbol{w(s_t^{(i)})}$.

if $\boldsymbol{s_t^{(i)}}$ is affected by sensor readings, $\boldsymbol{w(s_t^{(i)}) \leftarrow w(s_{t-1}^{(i)})\ \frac{L}{c_{sen}} p(m_t|s_t^{(i)})}$;

else $\boldsymbol{(s_t^{(i)}) \leftarrow w(s_{t-1}^{(i)})\ \frac{H}{c_{sim}} p(m_t|s_t^{(i)})}$;

Return $\boldsymbol{w(s_t^{(i)})}$.

In the corner case where $c_{sim} = 0$, particles all have the same high number as the transition density, and 1 as the proposal density, it then downgrades to a bootstrap filter. In the case where $c_{sen} = 0$, particles all have the same low number as the transition density, and 1 as the proposal density. The importance weights are also determined only by likelihoods. When the SenSim proposal is carefully designed that most of particles do not break the constraints of system states, the SMC method still effectively select particles that both consistent with both simulation models and observation data, and then effectively improve simulation results.

5.5 Case Study on Data Assimilation for Wildfire Spread Simulation

This section performs another set of data assimilation experiments on wildfire spread simulation using both the bootstrap filter based data assimilation framework, and the SenSim framework.

Each element of a *sensor monitored spatial-temporal system* ($< A, S, \mathcal{F}, \mathcal{M}, C, \Delta_s, \Delta_c >$) for DEVS-FIRE data assimilation is first constructed. The *system area A* is the target fire area; the system state space S is all the possible fire states of DEVS-FIRE; the system transition function $\mathcal{F}$ is the DEVS-FIRE fire spread simulation model; based on the temperature empirical model in (Mandel, Bennethum et al. 2008), the *measurement function* $\mathcal{M}$ is defined as:

$$\mathcal{M}(s_{t,D_i},\ p_i) = \max_{1 \le j \le n_g}(T_c^{(j)}\, e^{\frac{-{d_{ij}}^2}{\sigma^2}} + T_a)\ ,$$

where D_i is detection area of the i-th sensor, s_{t,D_i} is corresponding local fire state at t, p_i is the location of the i-th sensor, j is the index of a fire cell in D_i, n_g is the number of fire cells in D_i, $T_c^{(j)}$ is the temperature rise above ambient temperature of the j-th fire cell, T_a is the ambient temperature, d_{ij} is the Euclidean distance from p_i to position of j-th fire cell, and σ is a constant set to 50 (m); for each sensor in sensor set C, the sensor location is predefined, the sensor detection area is set to the whole fire area, and the measured sensor reading is received every certain time interval; the system state distance function Δ_s returns the number of fire cells with different cell states; the sensor reading distance function Δ_c is defined as $\Delta_c = |t_1 - t_2|$, where t_1 and t_2 are the temperature readings from two temperature sensors.

When sensor temperature readings are received, the *local sensor proposal* $p(s_{A_j}|\ T,\ s'_{D_T})$ is defined as follows. Firstly, a simple *possible fire area* is determined. With a predefine a hot

threshold, $thre_h$, and a cool threshold, $thre_c$ ($thre_c < thre_h$). Sensors are classified into three categories: hot sensors,

$$C_{hot}=\{c|\ c \in C\ and\ c.m \geq\ thre_h\};$$

cool sensors,

$$C_{cool} = \{c|\ c \in C\ and\ c.m \leq thre_c\};$$

other sensors,

$$C_{other}=\{c|\ c \in C\ and\ thre_c < c.m < thre_h\}.$$

For sensors in C_{hot}, a Gaussian distributed turn-on radius is defined, $radius_{on}$, indicating the possible burning area around a sensor. The mean of $radius_{on}$ is proportional to the sensor temperature, and the variance is predefined:

$$radius_{on} \sim N(\alpha_{on}\ c.m, \sigma_{on}),$$

where α_{on} is a constant number, $c.m$ is the temperature reading of a sensor, and σ_{on} is the variance. Similarly, for the sensors in C_{cool}, a turn-off radius, $radius_{off}$, is defined, and it Gaussian distributed as:

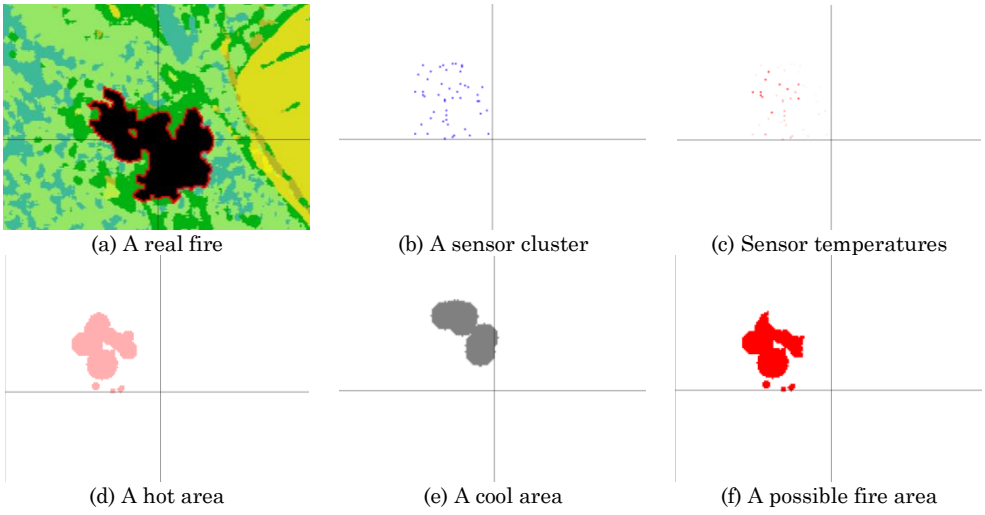
$$radius_{off} \sim N(\mu_{off}, \sigma_{off}),$$

where both μ_{off} and σ_{off} are predefined. With each $radius_{on}$, sensors in C_{hot} determine areas with possible fires (*hot areas*); with each $radius_{off}$, sensors in C_{cool} then determine areas with no fire (*cool areas*). Subtracting the union of all *cool areas* from the union of all *hot areas*, a *possible fire area* is found. An example is shown in Figure 5.1.

With the determined *possible fire area* and the union of all *cool areas*, a local fire state (that is, s_{A_j}) is generated by the help of DEVS-FIRE generated local state s'_{A_j}. Start from s'_{A_j},

for each sensor, if it is in C_{cool}, it turns off all the fire inside its cool area; if it is in C_{hot} and the intersection of its *hot area* has non-empty intersection with the *possible fire area*, it turns on this intersection. When turning on an intersection (say A_{int}), if there are burning cells in $s'_{A_{int} \cap A_j}$, the fire state of $A_{int} \cap A_j$ is set to $s'_{A_{int} \cap A_j}$; if there is no fire in $s'_{A_{int} \cap D_T}$, the whole intersection is set to burning.

After the above components ($< A, S, \mathcal{F}, \mathcal{M}, C, \Delta_s, \Delta_c >$ and $p(s_{A_j} | T, s')$) are defined, two groups of data assimilation experiments are performed. In one of them, sampling step is complete by drawing samples for wind speed and direction random moves, and executing DEVS-FIRE. Importance weights are only updated by likelihoods. It is the bootstrap framework based data assimilation. In the other group, system state samples are drawn by the SenSim proposal, and weights are updated by the kernel method, and it is then the SenSim framework based data assimilation.



(a) A real fire (b) A sensor cluster (c) Sensor temperatures

(d) A hot area (e) A cool area (f) A possible fire area

Figure 5.1 A possible fire area generated from sensor readings

In all experiments, fires are simulated with 40k fire cells (200×200) for 3 hours ignited from a single cell with cell coordinates (120, 18). The cell size is 15 (m) by 15 (m). Airborne

LiDAR (Light Detection and Ranging) (Wagner, Ullrich et al. 2004) raster-based terrain data is used as the GIS input of DEVS-FIRE. The fuel data is from Huntsville area, Texas, during the leaf-off season in March 2004. During the simulations, wind speed and wind directions are considered as random variables.

Sensor data and true fire states are generated from the identical twin paradigm. 1000 sensors randomly deployed in the whole fire area, and they report temperature readings of the real fire every 1200 seconds.

Another collection of fire simulations with erroneous settings are started as the simulated fires, and sensor readings are then assimilated. The settings of five data assimilation experiments are shown in Table 4.1.

Table 5.1 Settings of Real and Simulated Fire Systems for Identical Twin Experiments

	Real Fire	Simulated Fire
Case 1	Wind: 5m/s, 125 degrees	Wind: 6m/s, 105 degrees
Case 2	Wind: 5m/s, 125 degrees	Wind: 15m/s, 115 degrees
Case 3	Wind: 5m/s, 125 degrees	Wind: 4m/s, 305 degrees
Case 4	Wind: 5m/s, 125 degrees; a new ignition (100, 150) added at 4800s	Wind: 6m/s, 105 degrees
Case 5	Wind: 5m/s, 125 degrees	Wind: 6m/s, 105 degrees; unknown initial ignition

In Case 1, there are moderate wind speed and direction errors; in Case 2, the wind speed is significantly larger than the one of the real fire; in Case 3, there is a significant wind direction error; in Case 4, other than the moderate wind speed and direction error, a rare event happens in the real fire that a cell (100, 150) is ignited at 4800s; in Case 5, the simulated fire has a no-fire initial state and is with moderate wind speed and direction errors.

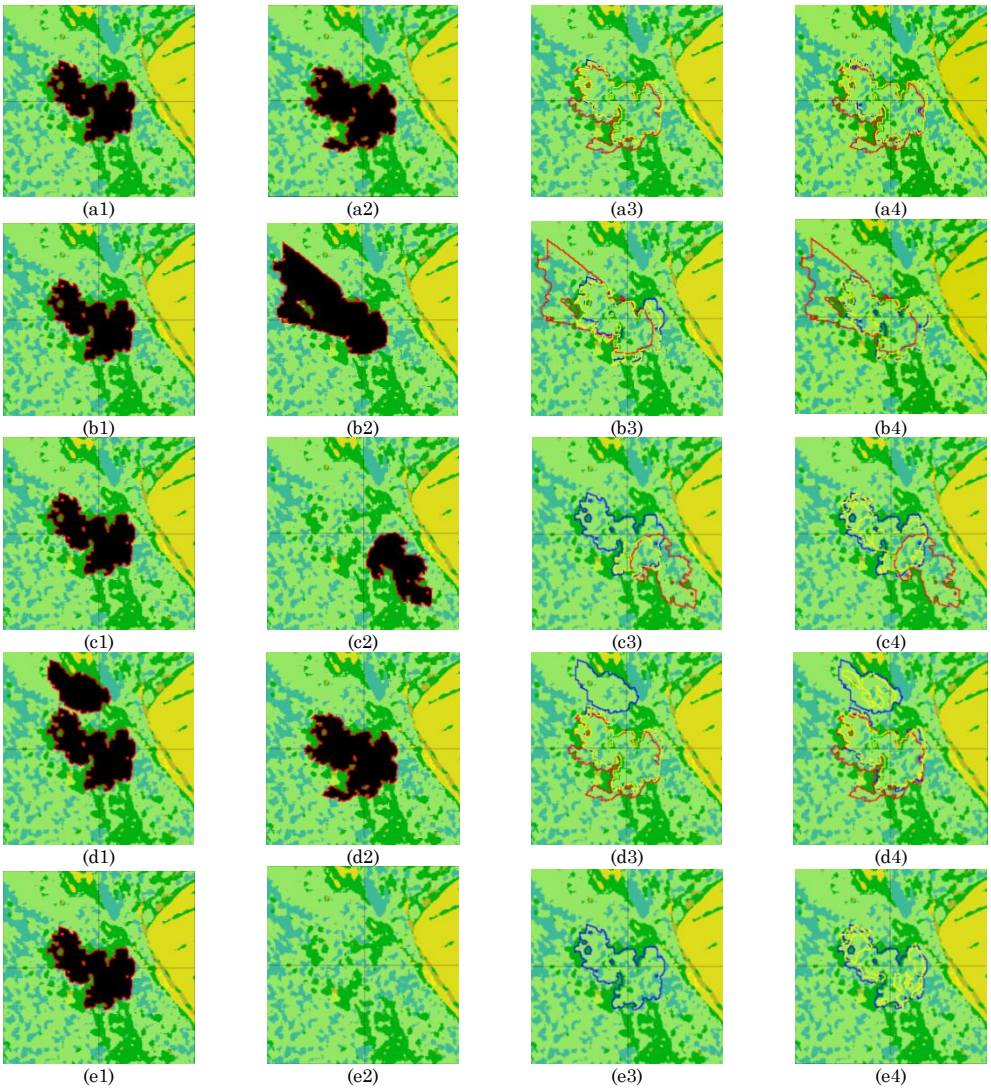


Figure 5.2 Graphical results of the SenSim framework data assimilation and the bootstrap filter data assimilation. Graphical results of the *SenSim framework* data assimilation and the bootstrap filter data assimilation. From Case 1 to Case 5, (a1), (b1), (c1), (d1), (e1) are the real fires; (a2), (b2), (c2), (d2), (e2) are the simulated fires; (a3), (b3), (c3), (d3), (e3) are the data assimilation results of the bootstrap filter; (a4), (b4), (c4), (d4), (e4) are the data assimilation results of the *SenSim framework*. (In data assimilation results, the sensor reading improved fire fronts are in yellow; the real fire fronts are in blue; the simulated fire fronts are in red.)

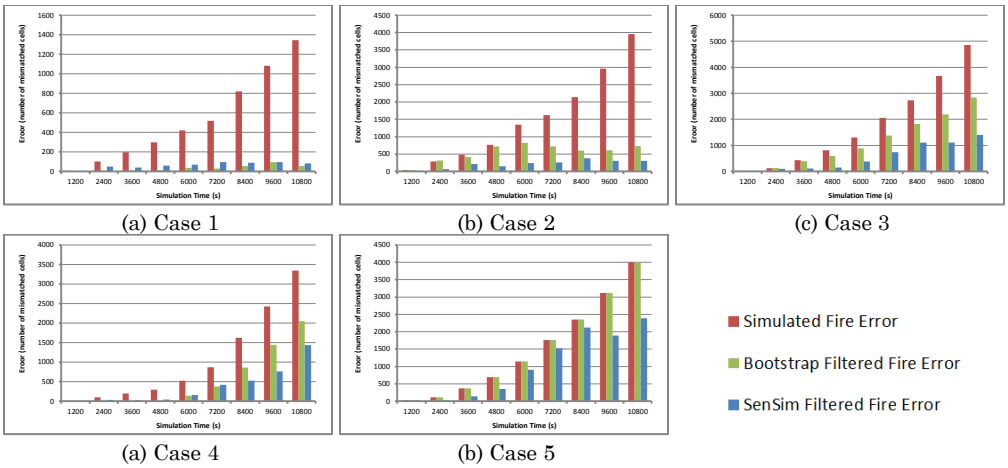


Figure 5.3 Numerical results of the SenSim framework data assimilation and the bootstrap filter data assimilation

Data assimilation using the SenSim framework and the bootstrap framework are then performed, and graphical results of fire states at 10800s for these five groups are displayed in Figure 5.2.

When the simulation model have only moderate errors (that is, Case 1), the *SenSim framework* and the bootstrap filter have comparable results. However, when the simulation error becomes significant, for example, the erroneous wind speed in Case 2 and wind direction in Case 3, benefited by early usage of sensor readings, the SenSim framework achieves much better results than those of the bootstrap framework. Moreover, when a rare event happens (that is, the new ignited fire in Case 4), the bootstrap framework fails to estimate the new fire because in simulation model based system transition distribution, the probability of new ignited fires is very low.

Case 5 shows the SenSim framework ability of solving the problems when initial conditions are unknown. Without the initial ignition point, in most cases, DEVS-FIRE predicts this area as with no fire, and sensor readings cannot help to correct it by the bootstrap framework since it only selects good instances from DEVS-FIRE generated samples. On the contrary, the

initial fire location can be roughly estimated by sensor temperatures when using the SenSim framework, so as shown in Figure 5.2(e4), it still roughly tracks where the fire is. Numerical errors displayed in Figure 5.3 also confirm the above facts.

5.6 Case Study on Data Assimilation for Lane-Based Traffic Data Assimilation

This section displays a case study on lane-based traffic system data assimilation. The performed identical twin experiments use a micro-level traffic simulator as the simulation model. Both the bootstrap framework and the SenSim framework are employed to carry out experiments. Their experimental results are compared and analyzed.

5.6.1 ***Experiment settings***

Similar to the case study of wildfire data assimilation, each element of a *sensor monitored spatial-temporal system* ($< A, S, \mathcal{F}, \mathcal{M}, C, \Delta_s, \Delta_c >$) for MovSim is first constructed.

The area A is defined as a segment of highway with a length of 1600 (m); state space S is defined as all possible states represented by vehicles on this highway, including the number of vehicles, their positions, velocities and accelerations.

Transition function $\mathcal{F}$ is constructed using MovSim, artificially added accidents and random moves. Given a current state, by a low probability (10% in this work), an accident is randomly added; MovSim is then run for one step; all the vehicles on the road are then randomly moved to a close new locations as one piece.

$\mathcal{M}$ returns the number, average velocity, and average acceleration of vehicles in the near 100 (m); C contains 4 sensors, evenly deployed on this segment of highway, their monitored area is defined as a 100 (m) long segment.

This highway segment is further divided into four sub-segments, and state distance Δ_s is defined as the sum of the vehicle density difference over all the sub-segments.

Different from the wildfire data assimilation, $\mathcal{M}$ here reports a vector of three variables, Δ_c then calculate the normalized difference on each dimension, and return the weighted average. The weights here reflect the importance of each variable, and 0.5, 0.3, and 0.2 are chosen. As a result, vehicle numbers (with the weight of 0.5) is the most important observation, and the acceleration (with the weight of 0.2) is the least important one, and the velocity (with the weight of 0.3) is the one in the middle. A *sensor monitored spatial-temporal system* for this traffic data assimilation is displayed in Figure 5.4.

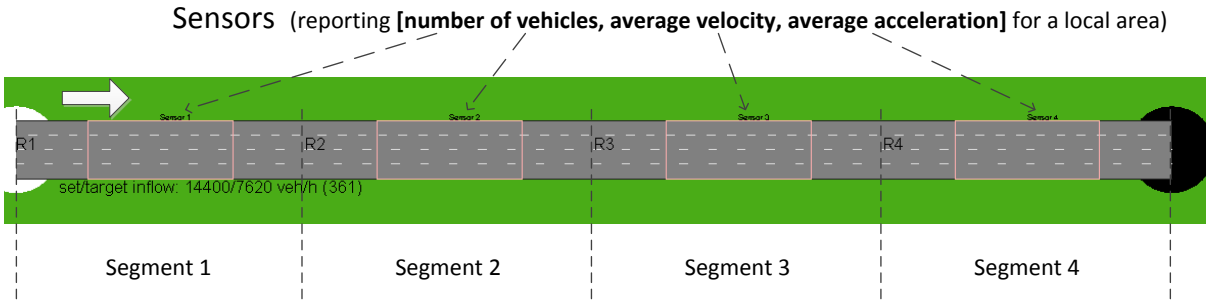


Figure 5.4 The setting of the lane-based traffic system data assimilation

The *local sensor proposal* is then constructed for the whole segment given the simulation result of MovSim, and the observations on all sensors. It is constructed in the following two steps:

1) *Remove Accidents*. For each sensor, if the average velocity is larger than 15m/s, and there is an accident in the sensor covered area, the accident is removed.
2) *Add Accident*s. In the reverse order of traffic flow, if the average velocity of the current sensor is lower than 8m/s, and there is no accident in the entire road segment, an accident is randomly placed from the left of the current sensor to the left of the senor on the right by a probability of 50%.

As an identical twin experiment, the real system is a MovSim simulation of 1800 seconds, where an accident is added at 250 seconds as the rare event of this system. The true states

at simulation time 180, 720, 1260 and 1800 seconds are shown in Figure 5.5. A stopped vehicle can be seen after 250 seconds. The vehicles on the left then become slower, and the left area has a significant larger density than the one on the right.

The simulated system is also a MovSim simulation with the same settings, but without the accident. As a result, the simulated system has very different predicted behavior than the real system after 250 seconds as shown in Figure 5.6. All vehicles are with high velocities, and there is no traffic jam.

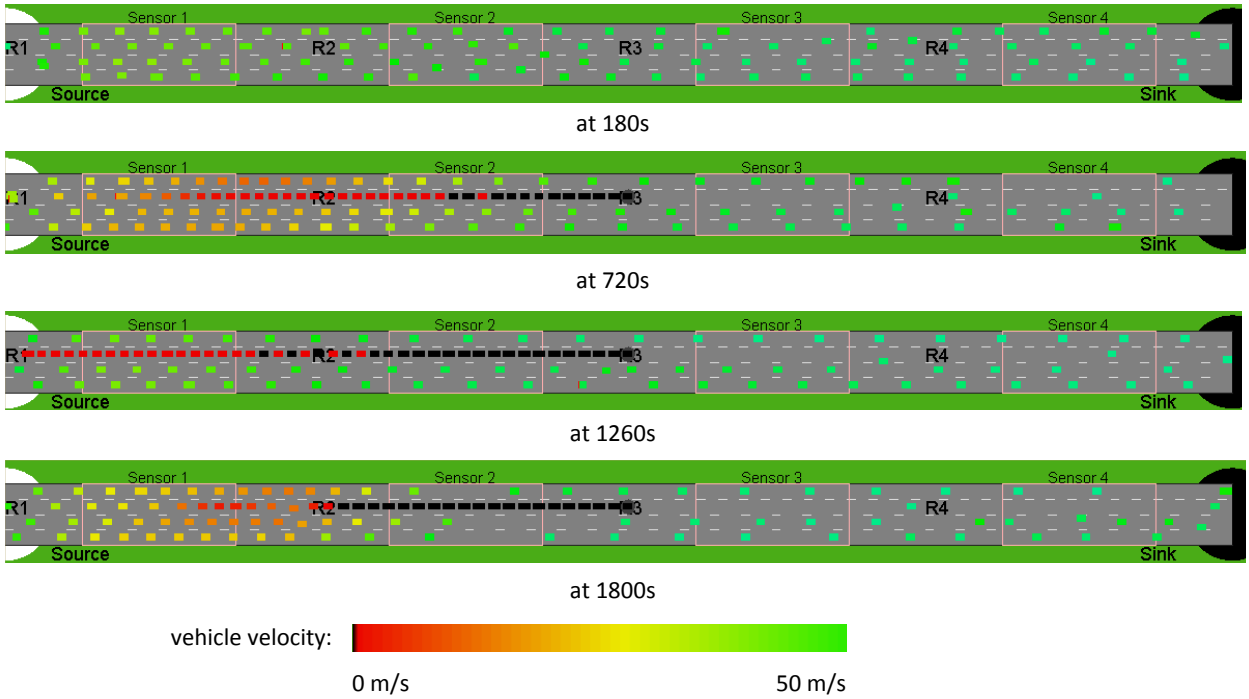


Figure 5.5 True states of the traffic system at 180s, 720s, 1260s and 1800s

In the following data assimilation experiments, every 180 seconds, sensor readings from the real system are assimilated into the simulated system by the bootstrap filter and the proposed SenSim framework. The first goal is to improve the simulation results on vehicle density in all the sub-segments; a much harder micro-level goal is to estimate where the accident is.

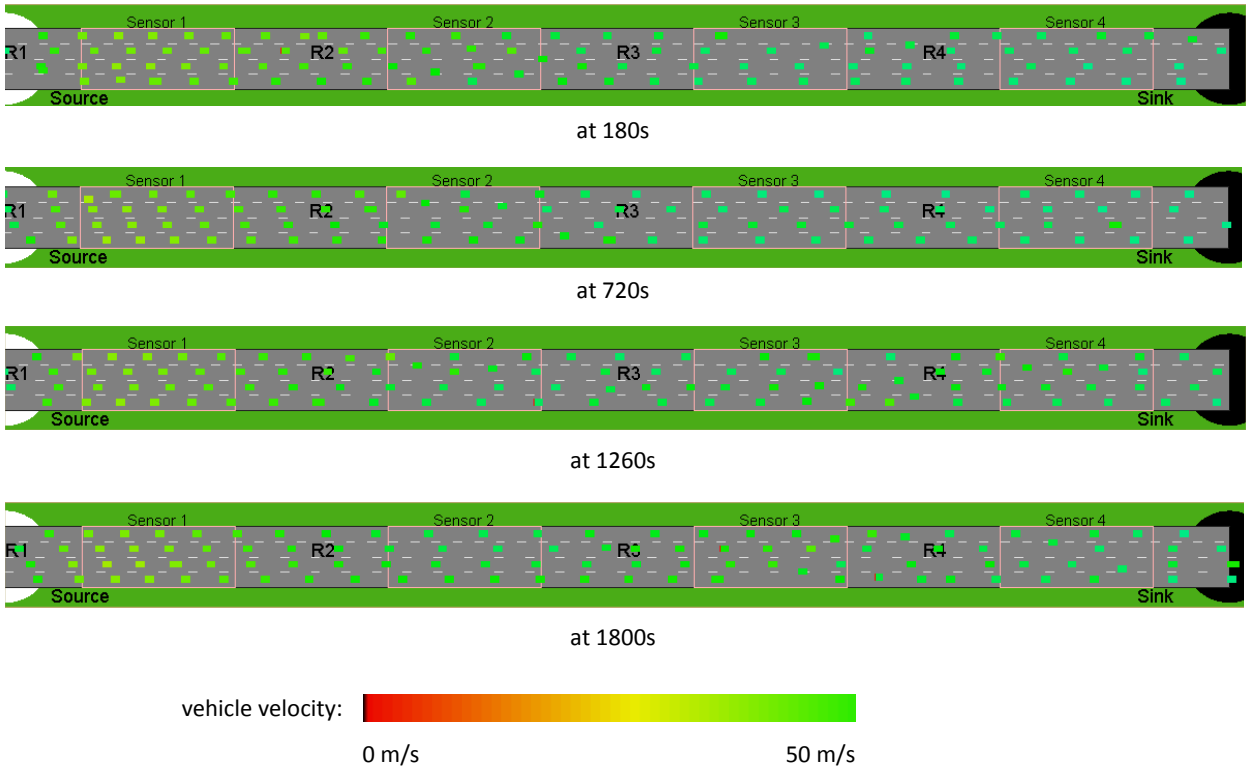


Figure 5.6 Simulated states of the traffic system at 180s, 720s, 1260s and 1800s

*5.6.2 **Experiments on Vehicle Density Estimation***

Data assimilation experiments are performed to reduce the overall error of vehicle density on all the sub-segments. The bootstrap filter data assimilation is first carried out with 10, 40 and 70 particles. Weighted average errors in each step of the particles are calculated and displayed in Figure 5.7. With 10 particles, the bootstrap filter achieved only limited improvement; with 40 and 70 particles, it reduced the error level to the range of about in [0.05, 1] after 1080 seconds.

SenSim data assimilation is then performed with the same amount of particles, and the corresponding numeric results are shown in Figure 5.8. All of them reduced the error level to [0.05, 1] after 720 seconds that is 360 seconds earlier than the bootstrap filter. In Figure 5.9, Figure 5.10 and Figure 5.11, the errors are compared between the bootstrap filter and the SenSim method on particle numbers of 10, 40, and 70 respectively. It is obvious that the SenSim method

outperformed the bootstrap filter in each of the cases. All the numeric results are listed and compared in Figure 5.12.

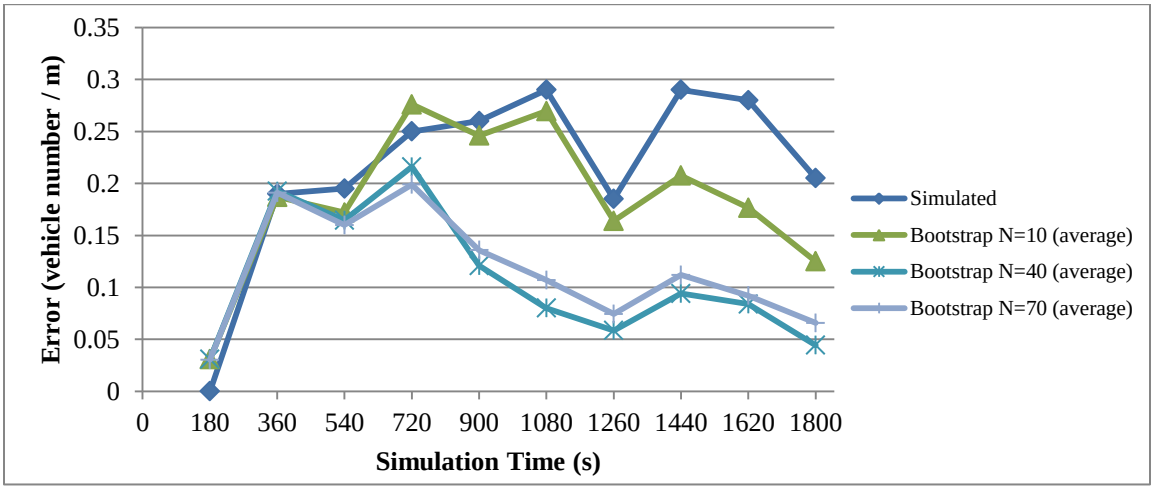


Figure 5.7 Results of the bootstrap filter with different particle numbers

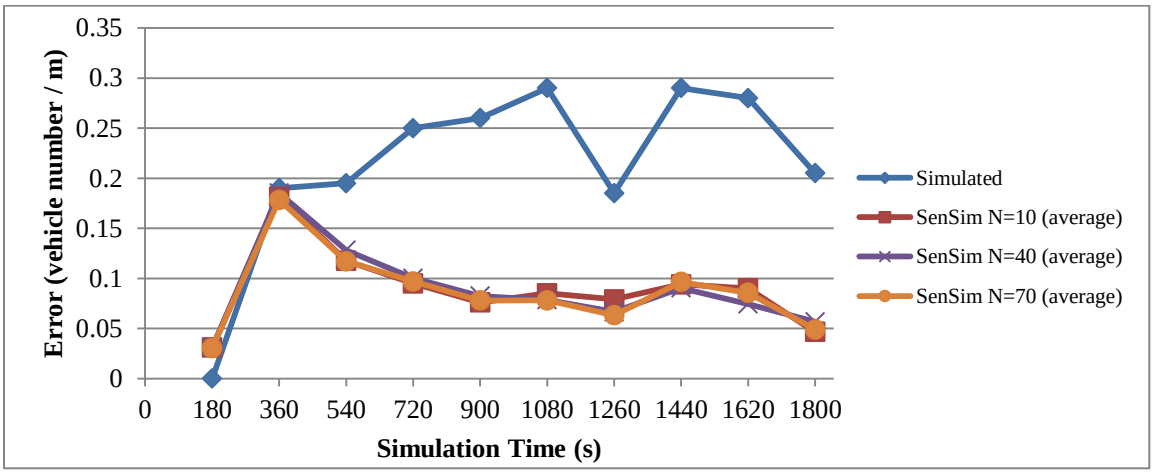


Figure 5.8 Results of SemSim with different particle numbers

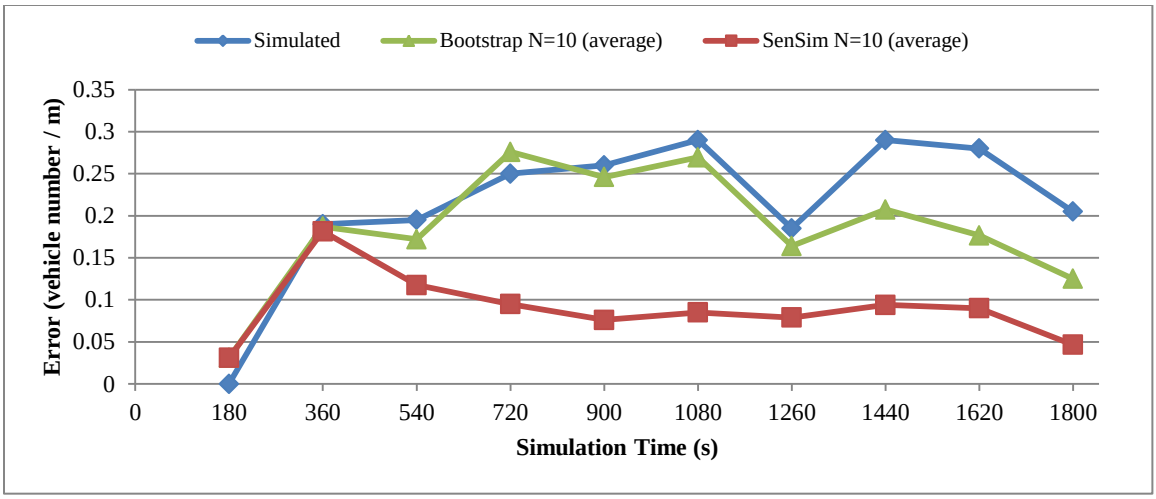


Figure 5.9 Comparisons between the bootstrap filter and SenSim (particle number = 10)

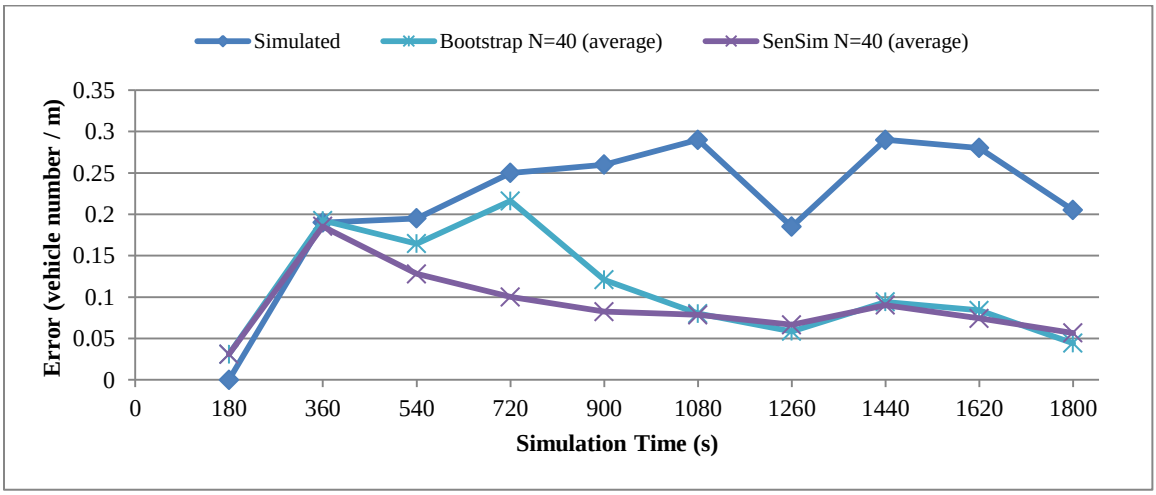


Figure 5.10 Comparisons between the bootstrap filter and SenSim (particle number = 40)

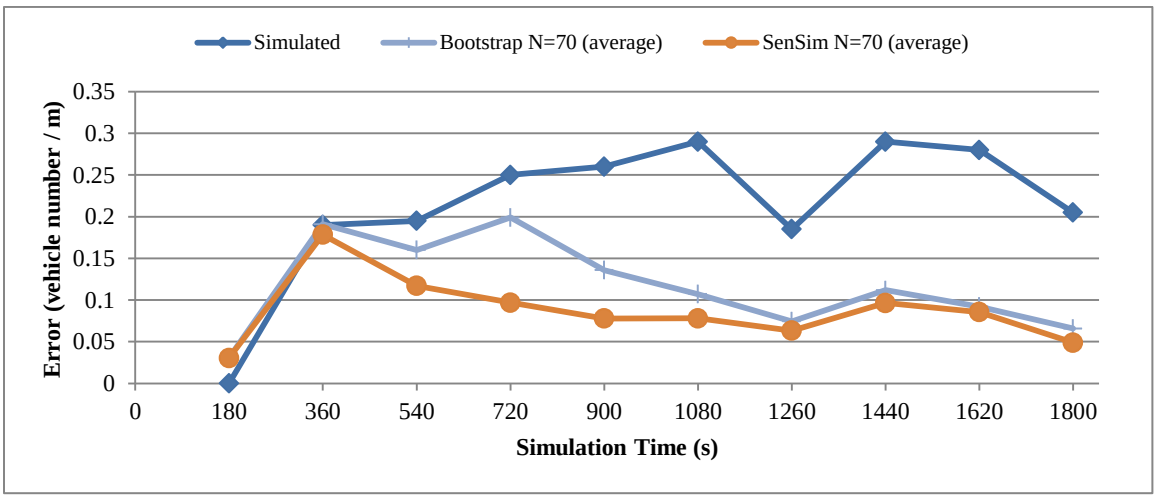


Figure 5.11 Comparisons between the bootstrap filter and SenSim (particle number = 70)

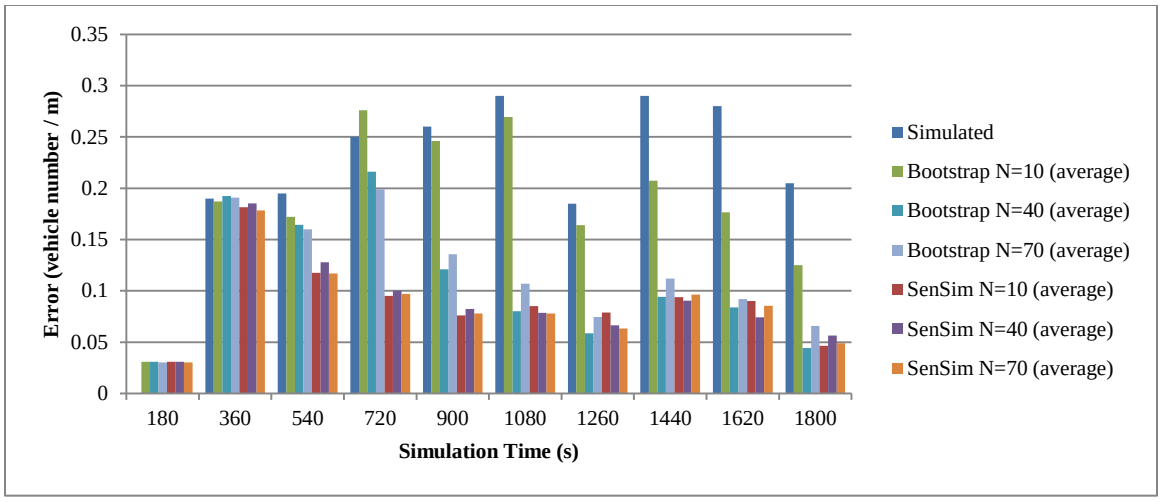


Figure 5.12 Numeric results of all data assimilation

*5.6.3 **Experiments on Accident Location Estimation***

With the micro-level traffic simulation model MovSim, it is possible to estimate the events happened on individual vehicles, and data assimilation methods can be applied to improve the estimation. This ability is tested by estimating accident locations. In each step of the data assimilation, a particle may have an estimated accident in its simulation. With the normalized importance weight on each particle, a probability is then associated with a location. Probability maps of accident locations are then formed. From the same experiments in section 5.6.2, the accident probability maps at 180, 720, 1260 and 1800 seconds are displayed in Figure 5.12 for the bootstrap framework and Figure 5.13 for the SenSim framework, where the blue circle is the real location of the accident, and red circles indicate estimated accident locations. The diameter of a red circle is proportional to the normalized importance weight.

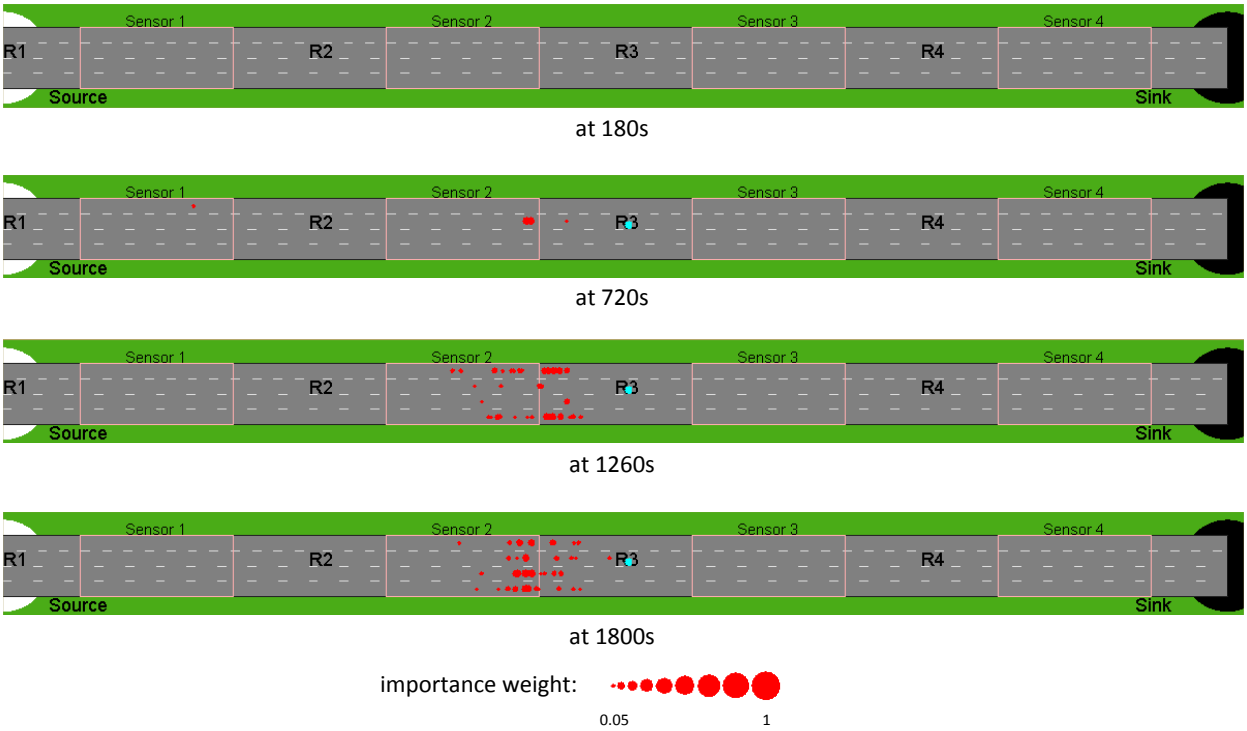


Figure 5.13 Bootstrap accident probability maps at 180s, 720s, 1260s and 1800s

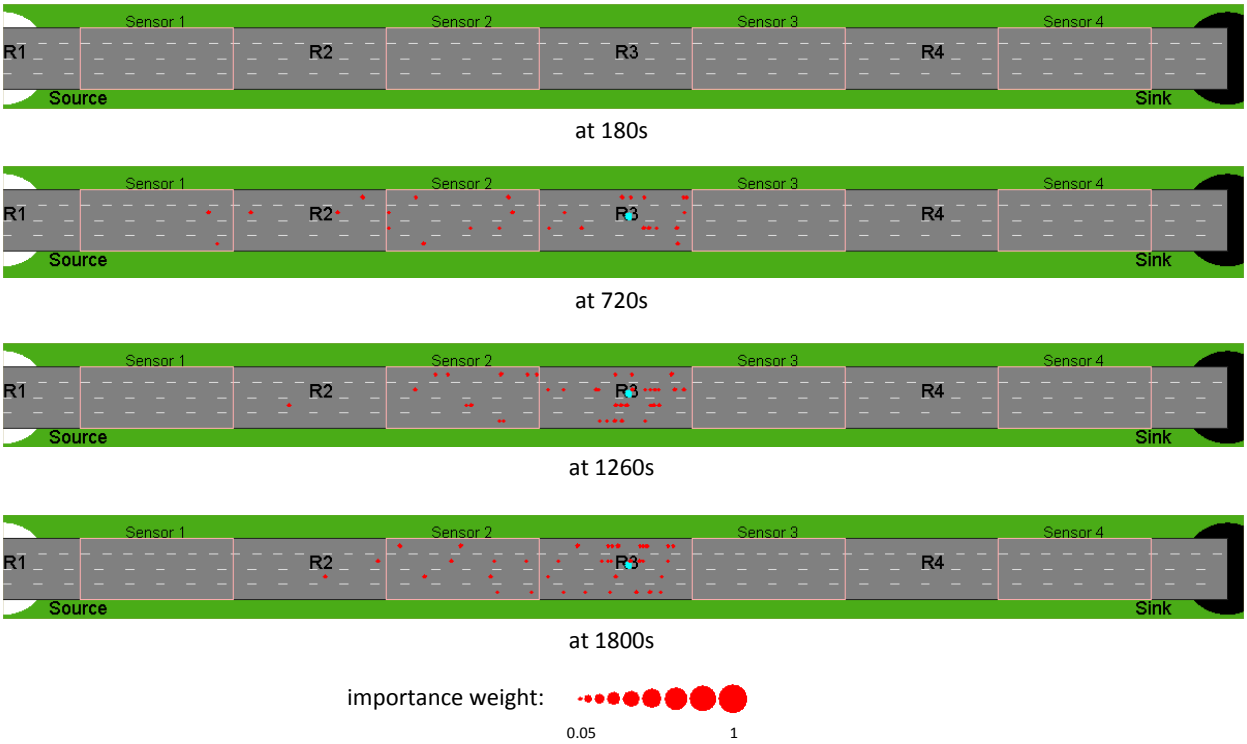


Figure 5.14 SenSim accident probability maps at 180s, 720s, 1260s and 1800s

The bootstrap filter simulates an accident only through the transition model, so accidents are added with a low probability; also, an accident is added randomly in the entire road, and it is then hard to be placed near the real accident location. As a result, at 720 seconds, only a few particles survive, expressing an estimated location about 100 meters away from the real location. At the same time, by the help of the *local sensor proposal*, more accidents are placed in the SenSim data assimilation, and it can be seen that many of them are around the true location. With more sensor data are assimilated, bootstrap particles slowly move to the real location, and in the last step, particles are closer to the real location but still away from it. On the other hand, SenSim particles show more accurate results, and they approach the real location from all directions. Most of the accident locations in particles are close to the real location in the last step.

6 MODELING SENSOR CORRELATION

6.1 Bias Incurred by Uneven Deployed Sensors

In a two dimensional area, close deployed sensors may have correlated sensor readings when their coverage overlaps. When sensors are unevenly deployed, ignoring the correlation may result into biased likelihood. Figure 6.1 shows an example. Each binary sensor covers a local area, reporting 1 when one or more targets are detected; otherwise, a sensor reports 0. In the real case, there is only one real target, and the real position of the target is in the middle, no sensor observes it, so sensor readings are $< 0, 0, 0, 0 >$ (from sensor 1 to sensor 4). In a simulation having a simulated target A, two sensors detect it, so the sensor readings are $< 1, 1, 0, 0 >$. In another simulation, there is a simulated target B, only sensor 4 convers it, so the sensor readings are $< 0, 0, 0, 1 >$. In Figure 6.1, the three targets are put in the same map, and it can be seen that they are on the same line, and the distance from target A to the real target is the same as the one from target B to the real target. In other words, the error level of target A is the same as the one of target B. However, they receive very different sensor readings (i.e., $< 1, 1, 0, 0 >$, two of them are consistent with the real sensor readings; $< 0, 0, 0, 1 >$, three of them are consistent with the real sensor readings) . If certain observation distribution that ignore the sensor correlation, for example, having a likelihood proportional to the sum of all sensor readings that are consistent with the real sensor readings, the likelihood of simulated target A is then smaller than the likelihood of the simulated target B. If a particle system uses this observation distribution to update importance weights, the particle with target B will have a larger chance to survive in a resampling step although they should have the same chance to survive, and a bias is then incurred. With the increase of deployment unevenness, the bias may become serious. In the same example, if another 3 sensors (e.g. sensor 5, 6 and 7) also cover target A but not the real target or

target B, target A will still have 2 consistent sensor readings (on sensor 3 and sensor 4), but target B will have 6 consistent sensor readings (sensor 1, 2, 3, 5, 6, and 7). The likelihood of target B is then as 3 times much as the one of target A. In section 8.4, more examples of sensor correlation incurred bias can be found in the wildfire data assimilation case study.

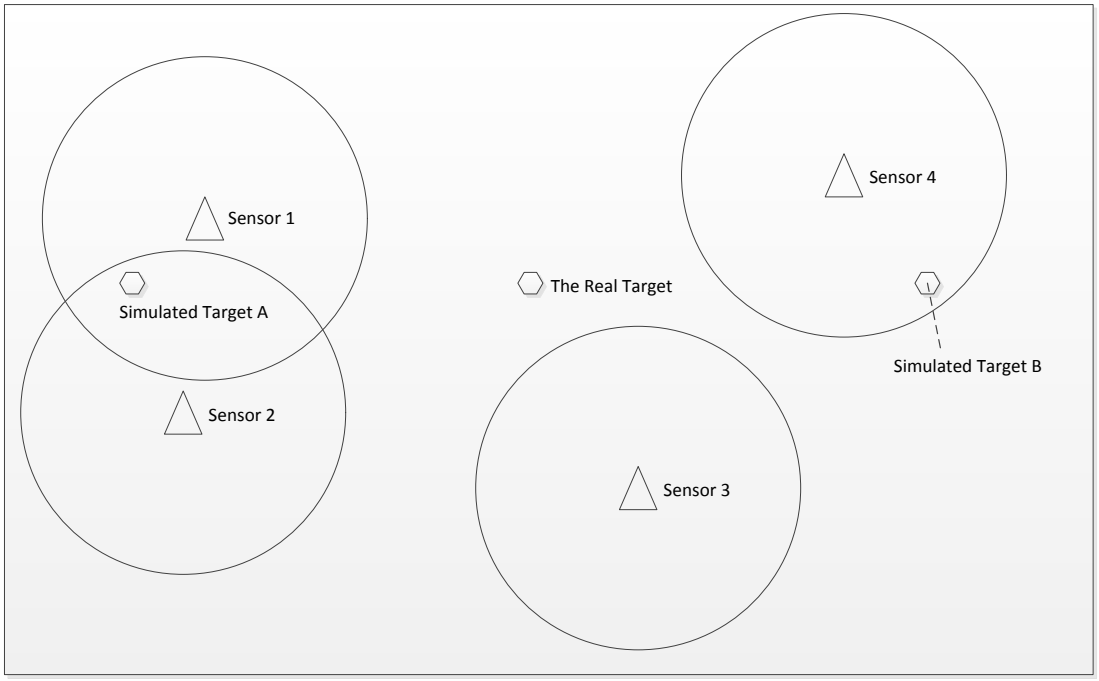


Figure 6.1 Biased likelihood in target tracking (Triangles are binary sensors; circles are the coverage sensors; hexagons are targets)

6.2 Correlation Estimation Model

Sensor readings are usually modeled as a multivariate random variable. Their correlation then can be expressed by a covariance matrix $[\rho_{ij}]_{i=1\ to\ k, j=1\ to\ k}$, where k is the number of sensors, and $\rho_{ij} = \sigma_i \sigma_j\ corr_{ij}$. $corr_{ij}$ is the correlation from the i-th sensor to the j-th sensor; σ_i and σ_j are the standard deviations of the i-th sensor and the j-th sensor. However, the true covariance matrix is often unknown. Each covariance is then estimated from a correlation model:

$$\rho_{ij} = \sigma_i \sigma_j\ Corr(d_{ij}), \tag{33}$$

where $Corr: \mathbb{R} \to [-1, 1]$ is a spatial correlation estimation function, d_{ij} is the Euclidean distance from the i-th sensor to the j-th sensor. $Corr$ returns 1 when $d_{ij} = 0$, and returns 0 when $d_{ij} \to \infty$. Four families of correlation estimation functions are widely used as summarized in (Berger, Oliveira et al. 2001):

Spherical functions:

$$Corr_{spherical}(d) = (1 - \frac{3d}{2\theta_1} + \frac{d^3}{2{\theta_1}^3})1_{\{\frac{d}{\theta_1} \leq 1\}}, \ \ where\ \theta_1 > 0, \tag{34}$$

They produce 0 correlation when the distance of two sensors is larger than certain predefined threshold.

Power Exponential functions:

$$Corr_{PowerExponential}(d) = \exp(-\left(\frac{d}{\theta_1}\right)^{\theta_2}), \ where\ \theta_1 > 0, \theta_2 \in (0, 2], \tag{35}$$

They can generate fast decreasing correlations.

Rational Quadratic functions:

$$Corr_{RationalQuadratic}(d) = (1 + (\frac{d}{\theta_1})^2)^{-\theta_2}, \quad where\ \theta_1 > 0, \theta_2 > 0, \tag{36}$$

Matern functions:

$$Corr_{Matern}(d) = \frac{1}{2^{\theta_2 - 1}\Gamma(\theta_2)} (\frac{d}{\theta_1})^{\theta_2} K_{\theta_2}\left(\frac{d}{\theta_1}\right), \quad where\ \theta_1 > 0, \theta_2 > 0, \tag{37}$$

K_{θ_2} is the modified Bessel function of order θ_2.

When performing data assimilation, a specific function is chosen based on the physical properties of sensor readings.

6.3 Model Sensor Reading Correlation in SenSim Framework

Sensor correlation information is integrated into SenSim framework through its observation model as described in section 5.2. The difference vector between real sensor readings and simulated sensor readings is modeled as a multivariate Gaussian variable with zero mean and a covariance matrix:

$$(m_t - [r_i]_{i=1}^{n_c}) \sim MN(0, \Sigma).$$

A correlation function from section 6.2 is then used to estimate the covariance matrix Σ. After the correlation is roughly estimated, the remaining bias may become much smaller than the bias when considering sensor readings as uncorrelated. Examples of how the bias is reduced can be found in section 8.4.

6.4 Influence of Sensor Spatial Correlation on Wildfire Data Assimilation

To examine the influence for strong correlated sensor readings on data assimilation results, uneven sensor deployments are employed, and experiments are performed by the same SMC algorithm (the bootstrap framework), but with different covariance matrices. In each experiment set, one experiment uses a diagonal covariance, and the other uses the proposed correlation model estimated covariance matrix.

Fire state estimation experiments are firstly performed to show how the bias is incurred by the unevenly deployed sensors and how the correlation model helps to counter the bias. Another set of experiments estimate wind speeds and wind directions to directly compare the estimated posterior distributions with the true values. Identical twin experiment method is employed again. To more clearly show the incurred bias, the resampling steps are skipped.

6.4.1 ***Influence on fire state estimates***

In this group of experiment, the fire area is with the size $1000 \times 1000\ m^2$, divided into 200×200 cells, and each cell is a $5 \times 5\ m^2$ square. GIS and fuel inputs are the same one as in section 6.2. The fire is ignited at the middle point of this area.

In the real fire, the wind speed is in the range of 8±2 m/s; in all the simulated fires, the erroneous wind speed is in the range of 6±2 m/s. After 3000 seconds of simulation, the simulated fire is away from the real fire as in Figure 6.2. With the erroneous smaller wind speed the simulated fire is much smaller than the real fire. Starting from the results in Figure 6.2, data assimilation is performed to improve the simulated fire.

When sensors are highly unevenly deployed, SMC data assimilation may fail to improve the results. To test the hypothesis, two sensor deployments are employed. The first one is with regularly deployed 36 sensors (indicated by *Regu36*) as shown in Figure 6.3. The second one is with randomly deployed 50 sensors in a small area (indicated by Corr50) as shown in Figure 6.4.

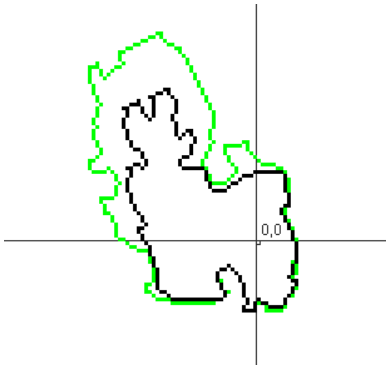


Figure 6.2 Real fire and simulated fire of spatial correlation experiment. The real fire front is shown in green, and the simulated fire front is shown in black.

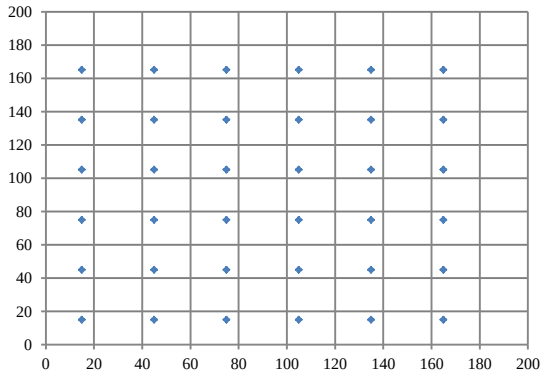


Figure 6.3 Regularly deployed 36 sensors (*Regu36*)

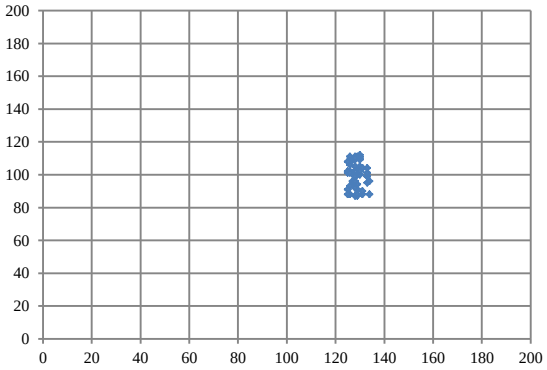
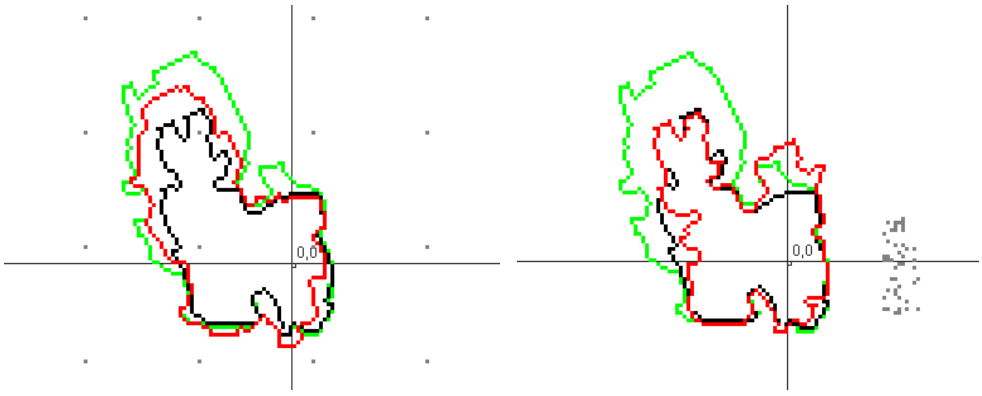


Figure 6.4 Random deployed 50 sensors in a small area (*Corr50*)

The bootstrap framework is then used to assimilate sensor readings from the real fire to DEVS-FIRE fire spread simulation model respectively with *Regu36* and *Corr50*, where a diagonal covariance matrix is employed. In both of them, the SMC algorithm assimilate sensor readings from the real fire every 500 seconds. Figure 6.5(a) shows a typical result of SMC with *Regu36*. At the same time, Figure 6.5(b) displays a biased final fire front produced by the PF with *Corr50*. In Figure 6.5(b), although on the right side of the final fire the error is very low, it has a much larger overall error compared with Figure 6.5(a). The biased observation of those 50 sensors is the reason of this result. They are capable to let the SMC method to choose the particles with small error on the right side, but lose much global information.



(a) (b)

Figure 6.5 Fire fronts after data assimilation of spatial correlation experiments: (a) The PF results using sensor deployment *Regu36*;(b) The PF results using sensor deployment *Corr50* (in each of them, the red line is the fire front produced by PF methods, the green line is the real fire front, the black line is the DEVS-FIRE simulated fire front with no data assimilation, gray points are sensors)

A sensor deployment is combined from *Regu36* and *Corr50* as shown in Figure 6.6 denoted as *Regu36Corr50*. Sensors in this deployment are capable to observe the global fire, but at the same time the 50 highly correlated sensors also incur strong biased likelihoods. Two data assimilation experiments are then performed, one of them considers all the sensors as independent (indicated as *Independent PF*) and the other uses the proposed correlation model to estimate the covariance matrix (indicated as *Correlated PF*).

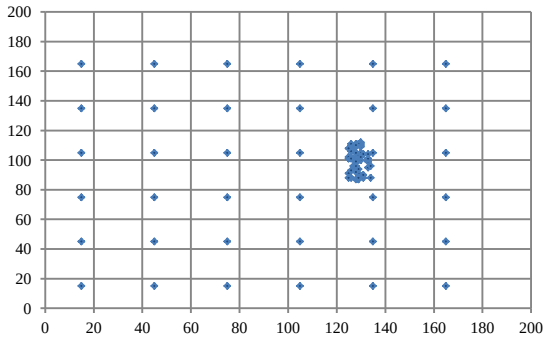


Figure 6.6 Regularly deployed 36 sensors and random deployed 50 sensors in a small area (*Regu36Corr50*)

With *Regu36Corr50*, both *Independent PF* and *Correlated PF* are run for 30 times. Figure 6.7 displays the results of one of the executions. Compared with Figure 6.2, it is clear that both *Correlated PF* and *Independent PF* have better final fire fronts than the result of simulated fire. Furthermore, *Independent PF* is harmed by the bias incurred by those 50 densely deployed sensors. As a result, compared with the fire front of *Correlated PF*, it has less error on the right side, but has more error nearly at everywhere else.

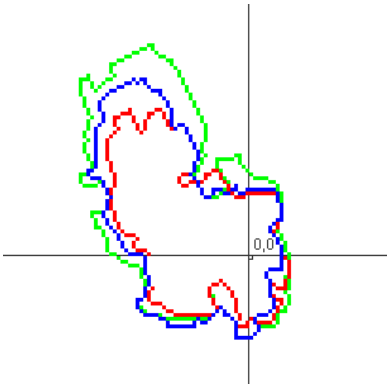


Figure 6.7 Estimated fire fronts with and without correlation model: real fire (as in green), *Independent PF* (as in red) and *Correlated PF* (as in blue)

Errors of a fire is calculated by counting how many cells of a simulated fire are with different states of the corresponding cells in the real fire, i.e. it is the number of incorrect cells of a simulated fire. The average errors of *Correlated PF*, *Independent PF* and the simulated fire without data assimilation for of 30 runs are shown in Figure 6.8.

Both *Correlated PF* and *Independent PF* achieve better result than the simulated fire. Also, *Correlated PF* receives smaller errors than *Independent PF* in all the steps after step 2 since the bias incurred by the highly correlated 50 sensors is countered by the correlation model. It should be noted that *Independent PF* still has the ability to choose particles with lower errors than the one with no data assimilation, but with certain bias.

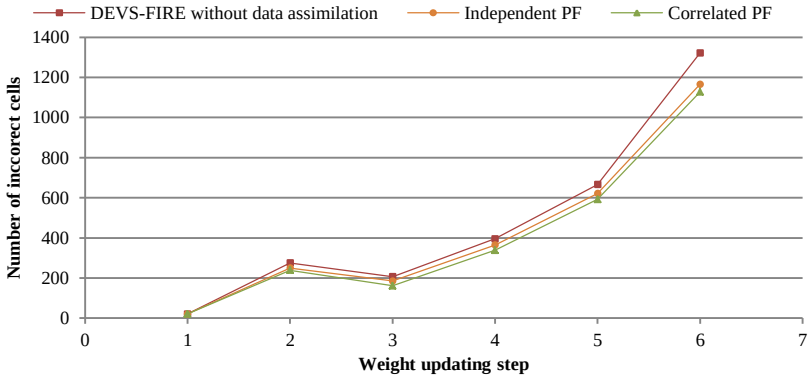


Figure 6.8 Error comparison among DEVS-FIRE, *Independent PF* and *Correlated PF*

6.4.2 ***Estimation of wind speed and wind direction***

To more clearly show how the *Correlated PF* benefits importance weights of particles, the experiment to estimate the wind speed and wind direction is performed. The assumption of this experiment is: there is no noise in the system transition distribution, so the values of wind speed and wind direction on each particle do not change during the whole data assimilation process. A sensor deployment is employed as shown in Figure 6.9, and experiments are carried out with both *Independent PF* and *Correlated PF*. In the real fire, wind speed is set as 5 m/s and wind direction is set as 270 degree. Each PF assimilates sensor data in every 1200 seconds, and the whole simulation is 9600 seconds. Also, to rule out the effect of random event, the same random seed is used for both *Independent PF* and *Correlated PF*.

In each experiment, the prior wind speed distribution is set as a uniform distribution from 4.5 m/s to 6.5 m/s, and the prior wind direction distribution is set as a uniform distribution from 265 degree to 275 degree. At the initial step, to obtain a particle, *Independent PF* and *Correlated PF* firstly draw samples from the prior wind speed and the wind direction distribution.

Having obtained 50 particles, they start the PF simulation and assimilate sensor data every 1200 seconds. After updating particle weights, the joint posterior distributions of the wind

speed and wind direction are changed in each time step. The corresponding marginal distributions represented by each of the 50 particles at step 5 are shown in Figure 6.10.

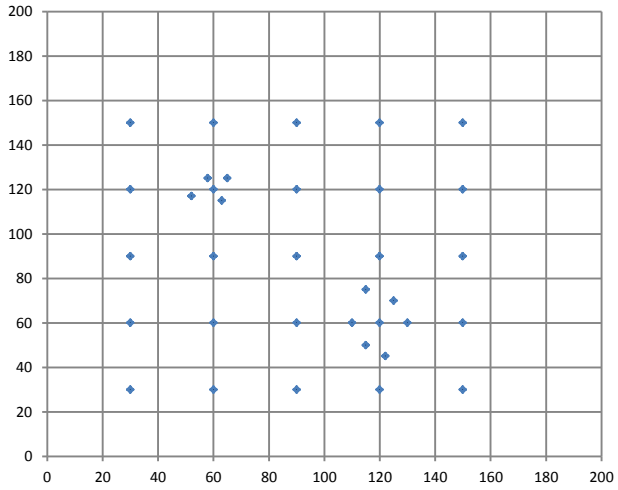


Figure 6.9 Sensor deployment for wind speed and wind direction estimation

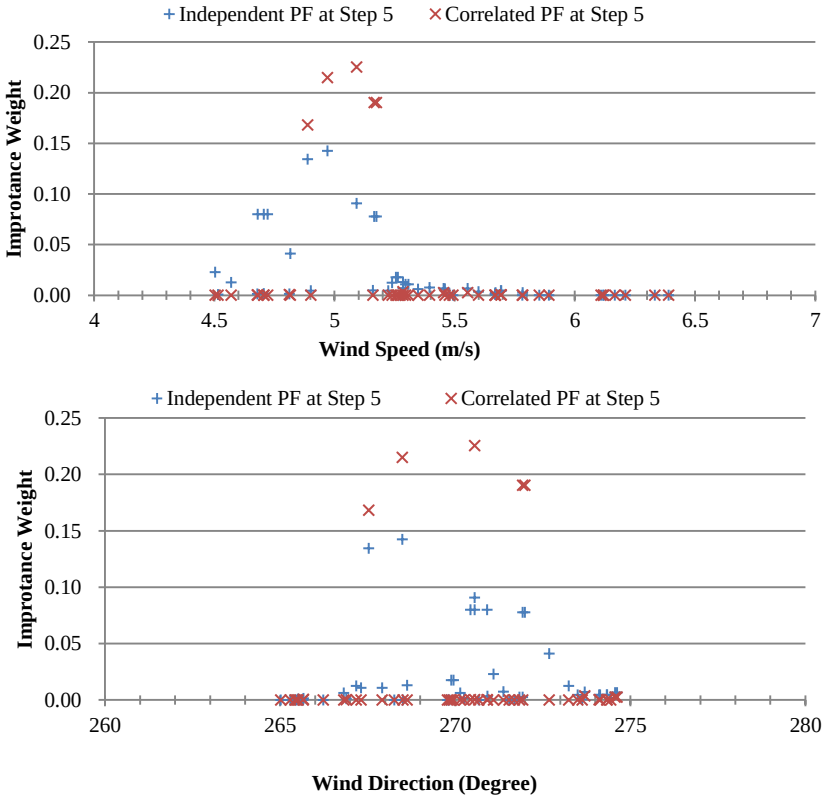


Figure 6.10 The particle represented posterior distribution of wind speed and wind direction at time step 5

From Figure 6.10, it can be seen that the posterior distributions of both the wind speed and wind direction have approached to the real distributions of wind speed (5 m/s) and wind direction (270 degree) after assimilating sensor data. Furthermore, for wind speed, it can be observed that non-zero weight particles of *Correlated PF* have a range about from 4.8 m/s to 5.2 m/s, and it is significantly smaller than the range of those of *Independent PF* that is about from 4.5 m/s to 5.4 m/s. Also, those particles in *Correlated PF* have larger weights than those of *Independent PF* when they are near the real value (5 m/s). Both the particle ranges and particle weights show that the posterior distribution produced by *Correlated PF* is closer to the real distribution than the one of *Independent PF*. Similar patterns were observed for the wind direction estimation results. *Correlated PF* has a better result than *Independent PF* because the bias incurred by unevenly deployed sensors is to some extent corrected by using sensor correlation information.

In PF data assimilation for wildfire simulation, sensor measurement could be highly spatially correlated, and they may in turn incur biased weights of particles and lower the correctness of the estimated distribution of system state. Experiment results demonstrate that after taking the sensor spatial correlation into consideration, the simulation error is further reduced.

7 DESIGN AND IMPLEMENTATION OF SMC DATA ASSIMILATION SOFTWARE PACKAGE

This section introduces how a SMC data assimilation code library is created. This library contains an extensible SMC data assimilation algorithm framework in which the implementation of each SMC component (e.g. the algorithms of sampling, weighing updating or resampling) can be easily upgraded and replaced (enabled by the *strategy design pattern*). It encapsulates SMC algorithms, and separates them from specific data assimilation applications that may largely reduce the efforts required of performing data assimilation for a given simulation model. The SMC library connects to a specific simulation model through the *adapter design pattern* and the *factory method design* pattern.

Identical twin experiments are widely used to test the effectiveness of data assimilation methods, and this library also contains a package supporting those experiments that operates simulation models and particle systems.

7.1 The “SMC” package

This package contains an abstract particle system including the following classes: Particle, *AbstractState*, *SamplingStrategy*, *WeightUpdatingStrategy*, *ResamplingStrategy*, *AbstractParticleSystem*.

A *Particle* object represents a particle in a SMC particle system. It contains an *AbstractState* object and a weight.

An *AbstractState* object represents a system state, and a sub-class extending *AbstractState* contains a simulation model. *AbstractState* and its sub-classes are then respectively the “adaptee” and “adaptors” of the *adapter design pattern*. *AbstractState* has the following abstract actions:

- Clone

Input: current system state

Output: a deep copy of the system state

Description: this function is frequently used by this package to create particle systems, and perform resampling.

- TransitionModel

Input: current system state, samples for all random components

Output: system state in the next step

Description: it is the *system transition function* described in section 5.1, and it is completed by executing the simulation model in subtypes.

- TransitionFunction

Input: current system state

Output: system state in the next step

Description: it is also the *system transition function*, but does not take updated samples for random components.

- MeasurementFunction

Input: current system state, sensor locations

Output: sensor readings

Description: It contains the measurement function described in section 5.1, and applies the function on each sensor.

- MeasurementPdf

Input: current system state, sensor readings

Output: likelihood

Description: it tells the probability density of observation distribution, and it can be implemented by the method described in section 5.2.

- Propose

Input: current system state, sensor readings

Output: a sample of the system state in the next time step

Description: it finishes the sampling of a SMC method, and can be implemented by the SenSim proposal described in section 5.3.

- DrawRandomComponentSample

Input: current system state

Output: samples for all random components

Description: based on the current system state, it draws samples of the random components.

- Distance

Input: current system state 1, current system state 2

Output: a distance between current system state 1 and current system state 2

Description: it tells the distance from two system states, and it is used to estimate density values in the weight updating step.

SamplingStrategy has a Sampling action; *ResamplingStrategy* has a Resampling action. Various sampling and resampling algorithms can be implemented and assembled into this framework as concrete strategies. In this library, a system transition sampling strategy, and a SenSim proposal based sampling strategy, and a systematic resampling strategy are implemented.

The *WeightUpdatingStrategy* class has already implemented the algorithm for normalizing importance weights given a collection of particle since it is the same for all strategies, and the action need to be implemented by sub-classes is:

- UpdateWeights

Input: a collection of particles, the current measurement, a SamplingStrategy object

Output: a collection of updated particles

Description: it finishes the weight updating step of a SMC method. In this library, a kernel method based strategy and a likelihood based strategy are implemented.

AbstractParticleSystem holds a collection of *Particle* objects, an object of *SamplingStrategy*, an object of *WeightUpdatingStrategy*, and an object of *ResamplingStrategy*. A action performing one step of SMC data assimilation is implemented:

- UpdateParticles

Input: the most recent sensor readings

Output: updated particles

Description: it uses the structure of Algorithm 1 described in section 3. With specific concrete strategies of sampling, weight updating and resampling, various types of SMC methods can be easily created, such as a bootstrap filter, or the proposed SenSim method.

The data structure of this “SMC” package is shown in Figure 4.1. Concrete strategies chosen for sampling, resampling and weight updating determines the algorithm used for SMC data assimilation. When observation data arrived, they update the collection of particles using the actions defined in *AbstractState*. Most of them are implemented by the simulation model associated with the “Adaptor”.

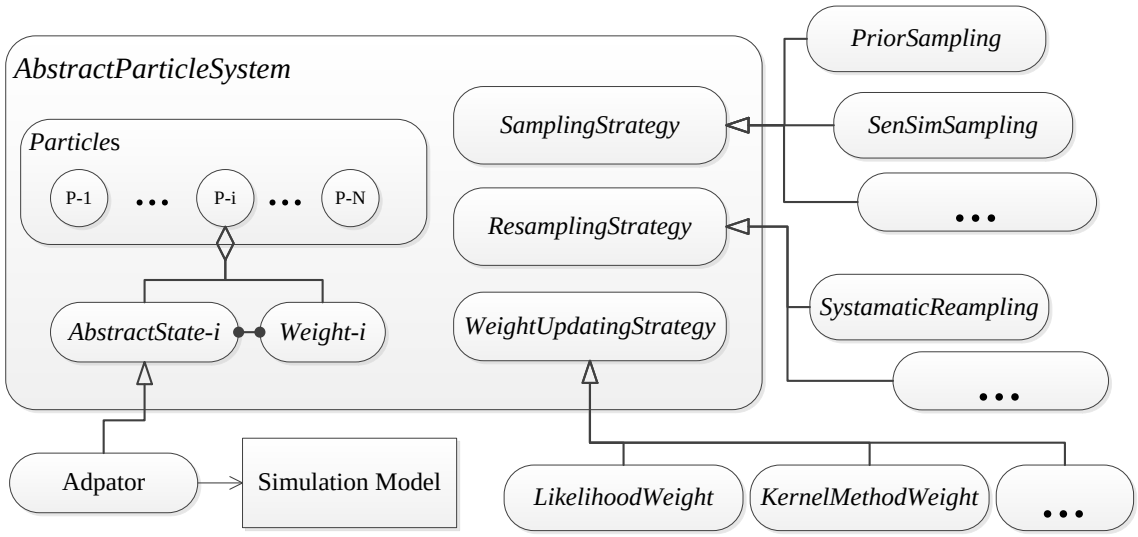


Figure 7.1 Architecture of the “SMC” package

7.2 The “Identical Twin Experiments” Package

Test data assimilation methods with real systems could be difficult. First, the event in interest may be very expensive to create, for example, a forest fire. The infrastructure bringing in real time sensor data could be also costly. If the real system is in a remote area, large amount of wireless sensors may need to be deployed. Identical twin experiments, as a much more affordable method, then are widely used to test the effectiveness of data assimilation methods.

In the paradigm of identical twin experiment, a simulation is treated as the real system. Real time sensor data are then generated by this simulation. Artificial errors are then added to another simulation, for instance, it may use wrong initial inputs, model parameters, or different model structures. With the increase of error level, the results of the second simulation (i.e. the

"simulation model" in a data assimilation application) could be far from the ones of the first simulation (i.e. the "real system" in a data assimilation application). Since the "real system" here is a simulation, "real time" sensor data can be easily simulated. To test data assimilation methods, one then takes the simulated sensor data from the first simulation (the "real system"), and let the second simulation assimilate those data to observe if the discrepancy to the "real system" is reduced.

Since identical twin experiments are important and widely used, a package (i.e. the "Identical Twin Experiments" package) is created to support them. Using this package, a user only needs to specify how the real system and simulated system are created. The package automatically finishes the rest of an identical twin experiment, such as collecting sensor data from real system, creating particle systems, and calculating errors.

The *AbstractIdenticalTwinExperiment* class contains all the components of an identical twin experiment: a real system and a simulated system (both of them are in the type of *AbstractState*), and an *AbstractParticleSystem* object representing the posterior distribution of the system state after data assimilation.

After the real system and the simulated system are created, the particle system is constructed by cloning the simulated system N times (the number of particles), and this operation is supported by the "Clone" of *AbstractState* as described in section 7.1. Users run the experiment with step length, and step number. In each step, the package then according calculates observation data using "MeasurementFunction" of *AbstractState*, performing data assimilation using "UpdateParticles" of *AbstractParticleSystem*, and calculating and compare errors from simulated system and data assimilated system to the real system using "Distance" of *AbstractState*.

However, as a generic identical twin experiment package, it does not have any knowledge of how to create real systems and the simulated systems, or the specific type of particle systems. The *factory method design* pattern is then applied to solve this problem. That is, CreateRealSystem, CreateRealSystem and CreateParticleSystem are defined in *AbstractIdenticalTwinExperiment* without implementations. A concrete sub class of *AbstractIdenticalTwinExperiment* overrides them to create those systems. The architecture of *AbstractIdenticalTwinExperiment* is displayed in Figure 7.2.

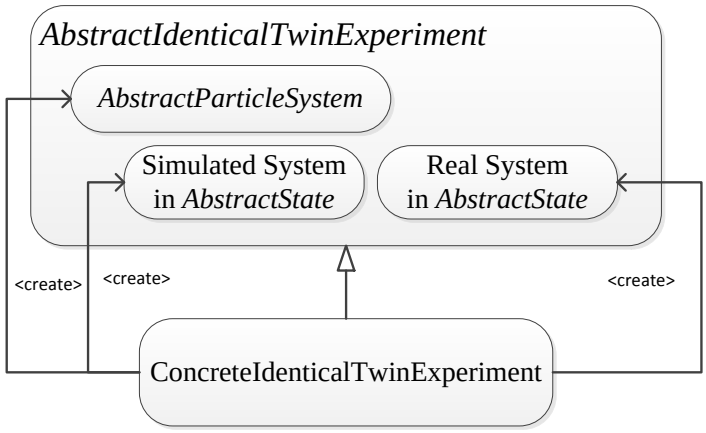


Figure 7.2 Architecture of *AbstractIdenticalTwinExperiment*

7.3 Use the Library

This code library hides most of implementations of SMC algorithm that may greatly reduce the work of applying SMC methods for given simulation models. For the users aiming at improving simulation results, given the simulation model, they need to

1. Extend *AbstractState* and implement all the functions listed in section 7.1 based on the simulation model.
2. Select concrete strategies for sampling, weight updating and resampling, and specify them when creating the particle system in CreateParticleSystem.

For data assimilation researchers, extended algorithms of each SMC component may be proposed. To integrate them into this library, they need to extend *SamplingStrategy*, *WeightUpdatingStrategy*, or *ResamplingStrategy*, and use them to create the particle system in CreateParticleSystem.

7.4 Address Small Weight Problem

When sensor numbers becomes high, a sensor reading difference vector based observation distribution easily has low probability density everywhere in the sample space. The same fact also holds true in transition distribution and proposal distribution. In the weight updating step, a particle then could receive a very small weight. It may not harm the theoretical SMC algorithm, but it could be fatal in implementation.

It is intuitive to use primitive floating-point variables to represent importance weights; however, they actually easily fail a SMC implementation. With the increase of system state dimensions or the number of sensors, the absolute values of true unnormalized importance weights could be extreme small; at the same time, a small weight could be still significant because it may still be a large one among all the particles. In other words, a particle with small unnormalized weight could have a large normalized weight. Unfortunately, when using floating-point variables, in most of programming languages, they become zero after they are smaller than the lower bounds. They then can never become significant even after a normalization step. When it happens, a SMC method usually fails to improve the simulation results.

To solve this problem, a data type supporting extremely small values need to be employed, such as the BigDecimal in Java, or arbitrary precision types, such as string represented numbers.

8 CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The developed SMC frameworks provide general and effective data assimilation solutions for complex simulation models that are strong non-Gaussian, nonlinear, with no analytical expression or with high computation load. It may greatly contribute to the data assimilation for agent-based simulation models where overall system behavior are seldom expressed in analytical forms, and may also help the data assimilation in the area of "simulation models as services" where model access is limited.

Using the develop algorithms and software package, performing data assimilation is largely simplified. Only a list of well-defined modules needs to be implemented. Instead of data assimilation focused, they are model-focused, such as the distance between two system states and the distance between two sensor readings, so they usually can be constructed in straightforward manners. At the same time, data assimilation algorithms, such as sampling, weight updating and resampling, are encapsulated and can be reused for difference applications. As a result, significant amount of efforts may be saved for users on creating the probability density functions when performing SMC data assimilation for specific applications.

The proposed observation model is easy to construct since it requires only a measurement function, a sensor reading distance function and a covariance matrix. It also works for sensor readings in complex data structures. Normalized distances can be calculated on all variables, and combined using the weighted average method as shown in the traffic data assimilation case study. Sensor correlation is also integrated in the SenSim framework through the covariance matrix. Using a well calibrated correlation estimation function, observation bias incurred by correlated sensor readings may be significantly reduced.

8.2 Future Work

Further research may apply the current methods to studying other interesting aspects. For example, with the ability to perform data assimilation on micro-level variables in the traffic simulation, dynamic observation from a single driver can be used as the observation data, and vastly improved system state predictions may then be generated inside an individual vehicle. If it became true, in turn, it may significantly affect the individual behavior and finally the later overall system behavior. All these new topics deserve more research efforts.

In the sampling step, sensor knowledge is introduced by the *local sensor proposal* distribution. In general, based on the local sensor readings and the simulated local system state, it provides estimates of a local system state with high likelihoods. When likelihood dominates the optimal proposal distribution (for example, when rare events or peaked likelihood happens), those local states help generate good particles since they drive particles to high likelihood area. However, in some cases, they may deteriorate the particles, that is, they may produce particles with high likelihoods but finally insignificant importance weights. When it happens on a large number of particles, many sensor-affected particles may be eliminated in resampling steps, and the data assimilation method then downgrades to a bootstrap filter but with much fewer particles. To solve this problem and construct more effective sampling methods, more research is needed.

The current measurement model is effective based on an assumption that, given a system state, a true sensor reading exists, and observed sensor readings are Gaussian distributed around the true reading. Although this assumption is sound in most of the applications, when it is violated, for instance, there are multiple true readings or observed sensor readings do not have a mean value at the true sensor readings, more advanced observation models need to be developed.

The developed methods may significantly reduce the number of particles required for effective data assimilation. However, when the computation load is extremely high, long computation time is still an issue. The proposed frameworks then need to be further extended to bring in computation parallelism. From the implementation aspect, the software package also needs to be upgraded to produce scalable solutions that automatically organize and utilize multi-threads, multi-cores, and distributed multi-machines to perform parallel data assimilation.

REFERENCES

Ahmed, N., M. Rutten, T. Bessell, S. S. Kanhere, N. Gordon and J. Sanjay (2010). "Detection and Tracking Using Particle-Filter-Based Wireless Sensor Networks." IEEE Transactions on Mobile Computing 9(9): 1332-1345.

Berger, J. O., V. D. Oliveira and B. Sanso (2001). "Objective bayesian analysis of spatially correlated data." Journal of the American Statistical Association 96(456): 1361-1374.

Bouttier, F. and P. Courtier (1999). Data assimilation concepts and methods. M. T. C. L. Series. Reading, England, European Centre for Medium-Range Weather Forecasts.

Buehner, M., P. L. Houtekamer, C. Charette, H. L. Mitchell and B. He (2009). "Intercomparison of Variational Data Assimilation and the Ensemble Kalman Filter for Global Deterministic NWP. Part II: One-Month Experiments with Real Observations." Monthly Weather Review 138(5): 1567-1586.

Byrne, A. S., S. United, S. Diaz and Associates (1982). Handbook of computer models for traffic operations analysis. Washington, D.C., U.S. Dept. of Transportation, Federal Highway Administration.

Cappe, O., S. J. Godsill and E. Moulines (2007). "An overview of existing methods and recent advances in sequential Monte Carlo." Proceedings of the IEEE 95(5): 899-924.

Cheng, C. and R. Ansari (2005). "Kernel particle filter for visual tracking." Signal Processing Letters, IEEE 12(3): 242-245.

Cohn, S. E., A. da Silva, J. Guo, M. Sienkiewicz and D. Lamich (1998). "Assessing the Effects of Data Selection with the DAO Physical-Space Statistical Analysis System." Monthly Weather Review 126(11): 2913-2926.

Cremer, M. (1979). Der Verkehrsfluss auf Schnellstrassen. Modelle, Uberwachung, Regelung. Berlin.

Crisan, D., P. D. Moral and T. J. Lyons (1999). "Non-linear filtering using branching and interacting particle systems." Markov Processes Related Fields 5(3): 293-319.

Darema, F. (2004). Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements. Computational Science - ICCS 2004. M. Bubak, G. Albada, P. A. Sloot and J. Dongarra, Springer Berlin Heidelberg. 3038: 662-669.

De Freitas, J. F. G., M. Niranjan, A. H. Gee and A. Doucet (2000). "Sequential Monte Carlo Methods to Train Neural Network Models." Neural Computation 12(4): 955-993.

Doucet, A. and A. M. Johansen (2011). A tutorial on particle filtering and smoothing: fifteen years later. The Oxford Handbook of Nonlinear Filtering. D. Crisan and B. Rozovski, Oxford University Press.

Evensen, G. (1994). "Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics." Journal of Geophysical Research: Oceans 99(C5): 10143-10162.

Fearnhead, P. (2008). "Computational methods for complex stochastic systems: a review of some alternatives to MCMC." Statistics and Computing 18(2): 151-171.

Fellendorf, M. and P. Vortisch (2001). Validation of the microscopic traffic flow model VISSIM in different real-world situations. Transportation Research Board 80th Annual Meeting.

Geman, S. and D. Geman (1984). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images." Pattern Analysis and Machine Intelligence, IEEE Transactions on PAMI-6(6): 721-741.

Godsill, S. and T. Clapp (2001). Improvement strategies for Monte Carlo particle filters. Sequential Monte Carlo Methods in Practice. A. Doucet, N. de Freitas and N. J. Gordon.

Gomes, G., A. May and R. Horowitz (2004). A microsimulation model of a congested freeway using VISSIM. Transportation Research Board Conference.

González, J., J. L. Blanco, C. Galindo, A. Ortiz-de-Galisteo, J. A. Fernández-Madrigal, F. A. Moreno and J. L. Martínez (2009). "Mobile robot localization based on Ultra-Wide-Band ranging: A particle filter approach." Robotics and Autonomous Systems 57(5): 496-507.

Gordon, N. J., D. J. Salmond and A. F. M. Smith (1993). "Novel approach to nonlinear/non-Gaussian Bayesian state estimation." IEE-Proceedings-F 140: 107-113.

Hu, X., Y. Sun and L. Ntaimo (2012). "DEVS-FIRE: design and application of formal discrete event wildfire spread and suppression models." Simulation 88(3): 259-279.

Hunter, M. P., R. M. Fujimoto, W. Suh and H. K. Kim (2006). An investigation of real-time dynamic data driven transportation simulation. Proceedings of the 38th conference on Winter simulation, Winter Simulation Conference.

Isard, M. and A. Blake (1998). "CONDENSATION - Conditional density propagation for visual tracking." International Journal of Computer Vision 29(1): 5-28.

Julier, S. J. and J. K. Uhlmann (2004). "Unscented filtering and nonlinear estimation." Proceedings of the IEEE 92(3): 401-422.

Kalman, R. E. (1960). "A New Approach to Linear Filtering and Prediction Problems." Transactions of the ASME–Journal of Basic Engineering 82(Series D): 35-45.

Kesting, A., M. Treiber and D. Helbing (2010). "Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity." Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 368(1928): 4585-4605.

Kitagawa, G. (1996). "Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models." Journal of Computational and Graphical Statistics(1): 1.

Kyriakides, I., D. Morrell and A. Papandreou-Suppappola (2008). "Sequential Monte Carlo methods for tracking multiple targets with deterministic and stochastic constraints." IEEE Transactions on Signal Processing 56(3): 937-948.

Lao, Y. W., J. D. Zhu and Y. F. Zheng (2009). "Sequential Particle Generation for Visual Tracking." IEEE Transactions on Circuits And Systems For Video Technology 19(9): 1365-1378.

Lopez, P. (2011). "Direct 4D-Var Assimilation of NCEP Stage IV Radar and Gauge Precipitation Data at ECMWF." Monthly Weather Review 139(7): 2098-2116.

MacCormick, J. and A. Blake (2000). "A probabilistic exclusion principle for tracking multiple objects." International Journal of Computer Vision 39(1): 57-71.

Mandel, J., J. D. Beezley, J. L. Coen and K. Minjeong (2009). "Data assimilation for wildland fires." Control Systems, IEEE 29(3): 47-65.

Mandel, J., L. S. Bennethum, J. D. Beezley, J. L. Coen, C. C. Douglas, M. Kim and A. Vodacek (2008). "A wildland fire model with data assimilation." Math. Comput. Simul. 79(3): 584-606.

Merwe, R. v. d., A. Doucet, N. d. Freitas and E. Wan (2000). The Unscented Particle Filter.

Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller (1953). "Equation of State Calculations by Fast Computing Machines." The Journal of Chemical Physics 21(6): 1087-1092.

Mihaylova, L., R. Boel and A. Hegiy (2006). An unscented Kalman filter for freeway traffic estimation, IFAC.

Moral, P. D. (1997). "Nonlinear filtering: Interacting particle resolution." Comptes Rendus de l'Académie des Sciences - Series I - Mathematics 325(6): 653-658.

Musso, C., N. Oudjane and F. Gland (2001). Improving Regularised Particle Filters. Sequential Monte Carlo Methods in Practice. A. Doucet, N. Freitas and N. Gordon, Springer New York: 247-271.

Noh, S. J., Y. Tachikawa, M. Shiiba and S. Kim (2011). "Applying sequential Monte Carlo methods into a distributed hydrologic model: lagged particle filtering approach with regularization." Hydrol. Earth Syst. Sci. 15(10): 3237-3251.

Ntaimo, L., X. Hu and Y. Sun (2008). "DEVS-FIRE: Towards an Integrated Simulation Environment for Surface Wildfire Spread and Containment." Simulation 84(4): 137-155.

Park, B. B. and J. Schneeberger (2003). "Microscopic Simulation Model Calibration and validation: case study of VISSIM simulation model for a coordinated actuated Signal system." Transportation Research Record: Journal of the Transportation Research Board 1856(1): 185-192.

Payne, H. J. (1971). Models of freeway traffic and control. USA, Simulation Councils, Inc.

Reichle, R. H. (2008). "Data assimilation methods in the Earth sciences." Advances in Water Resources 31(11): 1411-1418.

Reichle, R. H., S. V. Kumar, S. P. P. Mahanama, R. D. Koster and Q. Liu (2010). "Assimilation of Satellite-Derived Skin Temperature Observations into Land Surface Models." Journal of Hydrometeorology 11(5): 1103-1122.

Rosenblatt, M. (1956). "Remarks on Some Nonparametric Estimates of a Density Function." Annals of Mathematical Statistics 27(3): 832-837.

Rubin, D. B. (1988). Using the SIR algorithm to simulate posterior distributions. Bayesian Statistics 3. J. M. Bernardo, M. DeGroot, D. Lindley and A. Smith, Oxford University Press: 395-402.

Saha, S. and F. Gustafsson (2012). "Particle Filtering With Dependent Noise Processes." IEEE Transactions on Signal Processing 60(9): 4497-4508.

Sakov, P., F. Counillon, L. Bertino, K. A. Lisæter, P. R. Oke and A. Korablev (2012). "TOPAZ4: an ocean-sea ice data assimilation system for the North Atlantic and Arctic." Ocean Sci. 8(4): 633-656.

Schreiter, T., C. Van Hinsbergen, F. Zuurbier, J. Van Lint and S. Hoogendoorn (2010). Data-model synchronization in extended Kalman filters for accurate online traffic state estimation. TFTC Summer Meeting 2010.

Shah, G. A. and M. Bozyigit (2007). Exploiting Energy-aware Spatial Correlation in Wireless Sensor Networks. Communication Systems Software and Middleware, 2nd International Conference on: 1-6.

SunHee, Y. and C. Shahabi (2005). Exploiting spatial correlation towards an energy efficient clustered aggregation technique (CAG). Communications, 2005 IEEE International Conference on. 5: 3307-3313 Vol. 3305.

Talagrand, O. and P. Courtier (1987). "Variational Assimilation of Meteorological Observations With the Adjoint Vorticity Equation. I: Theory." Quarterly Journal of the Royal Meteorological Society 113(478): 1311-1328.

Tampère, C. M. and L. Immers (2007). An extended Kalman filter application for traffic state estimation using CTM with implicit mode switching and dynamic parameters. Intelligent Transportation Systems Conference (TSC 2007), IEEE.

Treiber, M., A. Hennecke and D. Helbing (2000). "Congested traffic states in empirical observations and microscopic simulations." Physical Review E 62(2): 1805.

Treiber, M. and A. Kesting (2010). "An open-source microscopic traffic simulator." Intelligent Transportation Systems Magazine, IEEE 2(3): 6-13.

van Leeuwen, P. J. (2010). "Nonlinear data assimilation in geosciences: an extremely efficient particle filter." Quarterly Journal of The Royal Meteorological Society 136(653): 1991-1999.

van Leeuwen, P. J. and G. Evensen (1996). "Data Assimilation and Inverse Methods in Terms of a Probabilistic Formulation." Monthly Weather Review 124(12): 2898-2913.

Vuran, M. C., Ö. B. Akan and I. F. Akyildiz (2004). "Spatio-temporal correlation: theory and applications for wireless sensor networks." Computer Networks 45(3): 245-259.

Vuran, M. C. and I. F. Akyildiz (2006). "Spatial correlation-based collaborative medium access control in wireless sensor networks." Networking, IEEE/ACM Transactions on 14(2): 316-329.

Wagner, W., A. Ullrich, T. Melzer, C. Briese and K. Kraus (2004). "From single-pulse to full-waveform airborne laser scanners: potential and practical challenges." International Archives of the Photogrammetry, Remote Sensing, and Geoinformation Sciences, XXXV (B/3) 414-419.

Wang, Y. and M. Papageorgiou (2005). "Real-time freeway traffic state estimation based on extended Kalman filter: a general approach." Transportation Research Part B: Methodological 39(2): 141-167.

Wang, Y., M. Papageorgiou and A. Messmer (2007). "Real-time freeway traffic state estimation based on extended Kalman filter: A case study." Transportation Science 41(2): 167-181.

Wang, Z., X. Yang, Y. Xu and S. Yu (2009). "CamShift guided particle filter for visual tracking." Pattern Recognition Letters 30(4): 407-413.

Work, D. B., O.-P. Tossavainen, S. Blandin, A. M. Bayen, T. Iwuchukwu and K. Tracton (2008). An ensemble Kalman filtering approach to highway traffic estimation using GPS enabled mobile devices. Decision and Control, 47th IEEE Conference on, IEEE.

Wunderlich, K. E. (1995). "Macro-level traffic simulation and case study development for evaluation of IVHS system architecture." Proceedings of the annual meeting of ITS America 2: 875-886.

Xue, H., F. Gu and X. Hu (2012). "Data Assimilation Using Sequential Monte Carlo Methods in Wildfire Spread Simulation." ACM Transactions on Modeling And Computer Simulation 22(4): 1-25.

Xue, H. and X. Hu (2012). Exploiting Sensor Spatial Correlation for Dynamic Data Driven Simulation of Wildfire. 26th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS2012). Zhangjiajie, China.

Xue, H. and X. Hu (2013). An Effective Proposal Distribution for Sequential Monte Carlo Methods-Based Wildfire Data Assimilation. 2013 Winter Simulation Conference (WSC2013). Washington, DC, USA.

Yingqi, X. and W. C. Lee (2006). Exploring spatial correlation for link quality estimation in wireless sensor networks. Pervasive Computing and Communications, 4th Annual IEEE International Conference on: 10 pp.-211.

Zeigler, B. P., T. G. Kim and H. Praehofer (2000). Theory of Modeling and Simulation, Academic Press, Inc.

Zhai, Y., M. B. Yeary, S. Cheng and N. Kehtarnavaz (2009). "An Object-Tracking Algorithm Based on Multiple-Model Particle Filtering With State Partitioning." IEEE Transactions on Instrumentation and Measurement 58(5): 1797-1809.

Zoghi, M. R. and M. H. Kahaei (2009). Efficient sensor selection based on spatial correlation in wireless sensor networks. 14th International CSI Computer Conference: 627-632.