12-15-2016

Data Assimilation for Spatial Temporal Simulations Using Localized Particle Filtering

Yuan Long

DATA ASSIMILATION FOR SPATIAL TEMPORAL SIMULATIONS USING LOCALIZED PARTICLE FILTERING

by

YUAN LONG

Under the Direction of Xiaolin Hu, PhD

ABSTRACT

As sensor data becomes more and more available, there is an increasing interest in assimilating real time sensor data into spatial temporal simulations to achieve more accurate simulation or prediction results. Particle Filters (PFs), also known as Sequential Monte Carlo methods, hold great promise in this area as they use Bayesian inference and stochastic sampling techniques to recursively estimate the states of dynamic systems from some given observations. However, PFs face major challenges to work effectively for complex spatial temporal simulations due to the high dimensional state space of the simulation models, which typically cover large areas and have a large number of spatially dependent state variables. As the state space dimension increases, the number of particles must increase exponentially in order to

converge to the true system state. The purpose of this dissertation work is to develop localized particle filtering to support PFs-based data assimilation for large-scale spatial temporal simulations. We develop a spatially dependent particle-filtering framework that breaks the system state and observation data into sub-regions and then carries out localized particle filtering based on these spatial regions. The developed framework exploits the spatial locality property of system state and observation data, and employs the divide-and-conquer principle to reduce state dimension and data complexity. Within this framework, we propose a two-level automated spatial partitioning method to provide optimized and balanced spatial partitions with less boundary sensors. We also consider different types of data to effectively support data assimilation for spatial temporal simulations. These data include both hard data, which are measurements from physical devices, and soft data, which are information from messages, reports, and social network. The developed framework and methods are applied to large-scale wildfire spread simulations and achieved improved results. Furthermore, we compare the proposed framework to existing particle filtering based data assimilation frameworks and evaluate the performance for each of them.

INDEX WORDS: Data assimilation, Particle filtering, Hard/Soft data, Spatial temporal simulations, Wildfire simulation.

DATA ASSIMILATION FOR SPATIAL TEMPORAL SIMULATIONS USING LOCALIZED PARTICLE FILTERING

by

YUAN LONG

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

2016

DATA ASSIMILATION FOR SPATIAL TEMPORAL SIMULATIONS USING LOCALIZED PARTICLE FILTERING

by

YUAN LONG

Committee Chair: Xiaolin Hu

Committee: Saied Belkasim

Wenzhan Song

Yichuan Zhao

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

December 2016

DEDICATION

I would like to dedicate this dissertation to my parents Shuping Long and Hongli Tan, who love me, support me, inspire me and encourage me through years of my studying. Also to my younger brother, Kang Long, who encourages me to go through any difficulties.

ACKNOWLEDGEMENTS

First of all, I wish to thank my supervisor, Dr. Xiaolin Hu, for his insightful guidance and dedicated assistance. Without his help and patient guide, I would hardly finish this dissertation and finish my Ph.D. study.

Secondly, I wish to thank my Committee members, Dr. Saied Belkasim, Dr. Wenzhan Song and Dr. Yichuan Zhao. Not only their informative have enriched my knowledge in my study field, but also their constructive suggestions and encouragement have inspired me and enlighten me in the research.

Also, I wish to thank my fellow labmates,,Sanish Rai, Peisheng Wu, Nicholas Keller, Fan Bai, Haidong Xue, Minghao Wang and Song Guo. Thank you for your valuable suggestions to my research. And thank you for being with me in ups and downs during the last five years.

Besides, I want to thank all the faculties and staffs in Computer Science Department at Georgia State University who encourage me, help me and contribute to my growth as a researcher.

At last, I wish to thank all my lovely family and all friends. Thank you for being my biggest supporters!

.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Background

Spatial-temporal simulations are more and more widely used for studying complex spatial-temporal systems. They are found in a variety of applications, such as wildfire simulation [1], real time traffic simulation[2,3] and pedestrian crowd simulation [4,5]. Traditionally these simulations are used as offline tools without assimilating real time sensor data from the systems under study. However, with sensor data become more and more available, there is a need to assimilate real time sensor data into spatial temporal simulations for more accurate simulation or prediction results.

Data assimilation is a process that iteratively corrects simulation result through feedback from physical environment. In traditional simulations, variants of factors may lead to incorrect prediction result. Two main factors are the "initial condition" and "model errors" [6]. The "initial condition" refers to input data for simulation; "model errors" represents the discrepancy between simulation model and physical model. For example, in wildfire simulation, the erroneous weather input data or geography information bring uncertainty to simulations and thus impact the final prediction accuracy; similarly, the simulation model itself is usually not a perfect model (i.e. the same as a physical model), so the simulation results will have errors compared with the physical system. We are not applying data assimilation to correct input data nor correct the imperfect model. However, through available observations from a physical system, and based on current estimated system state, a better prediction can be achieved by an indirect "synchronization" between current estimated system state and physical system state [7]. Since data assimilation process itself needs observation data, it can also be considered as dynamic data driven by a real or physical system (DDDS).

In previous work, we have applied Particle Filters (PFs) [8], also known as Sequential Monte Carlo methods, to data assimilation for spatial-temporal simulations1. Compared with other notable data assimilation methods, such as Kalman filters [8], PFs are non-parametric methods and work well with systems that have non-linear and non-Gaussian behavior. This makes them a desirable data assimilation method for complex spatial-temporal simulations.

PFs are sample-based methods, which represents the prediction of state by a posterior distribution of samples (also called as particles) [7]. A standard PFs method contains three main steps: sampling, weight calculation and resampling. Sampling step evolves system state of each particle to next data assimilation time point based on simulation model; weight calculation step calculates the weight for each particle based on observation data from physical system; resampling step selects a new set of particles by duplicating the samples with large weights and eliminate samples with negligible weights.

1.2 Challenges for PFs based data assimilation in spatial temporal system

In a spatial temporal system, the samples are drawn from space of unknowns and filtered by observations. However, applying PFs into a complex spatial temporal system has two main challenges. A major challenge is due to a large number of state variables. As the state space for spatial system is usually very large, a small number of particles is difficult to achieve satisfactory results by having "correct combination" of all state variables. Another challenge is from observation data. In traditional particle filter, the importance weights of different particles for any chosen state variable are influenced by all observation data, even if those observation data are nearly independent of the particular state variable [9,10].

Also from observation data aspect, observations mainly from sensors have the limitations on amount and diversity of measurement. Since sensor can only sense information in a local area,

a small number of sensors can only provide feedbacks for partial physical space and is hard to reflect the global system state of a large physical spatial system. So the amount of sensor distributed over physical space has a great impact on prediction accuracy. Besides, most of the sensors are designed and produced for specified purposes. For example, ground temperature sensor only provides readings for environment temperature in a local area; laser sensor only detects the distance to a target; Infrared sensor measures infrared light radiating from objects within the detectable local area. Therefore, to get sufficient observations for a complex spatial temporal system, usually number of sensors with different natures and purposes need to be distributed. However, in a real world, overcoming these two limitations will also increase economic cost.

1.3 Problem statement

For a high dimensional spatial temporal system whose states and observation data is spatially distributed and have finite correlation lengths (i.e. the observation area is limited in a local area), the PFs methods have underestimated the uncertainty of the posterior distribution due to limit number of particles and overestimated the information available in the observation data by calculating particle's weight based on all observations for a full state. Consider wildfire as an example, the observation data (e.g., ground temperature sensor data) from different regions of the fire typically reflect only the fire states in their corresponding regions, not others. However, in each particle the weight is calculated for a full state considering observations in all regions. And as a result, poor prediction performance may happen. This is because in the weight calculating step we are using all observations to weigh a particle, even part of which is independent of some observations. As a consequence, some particles with a low weight are removed even their sub-states are good. For example, in wildfire simulation, we have "particle 1"

generated in figure 1.1 left side. The sensors represented by black dots are distributed randomly over the whole area. The gray line shows a real fire front while the red line shows a predicted fire front by bootstrap PFs. Obviously, as the predicted fire front deviates greatly from the real fire, this particle is assigned a low weight provided by measurement function. We can also notice that a portion of the predicted fire front located in the orange circle is close to the real fire front. However, this good portion has a high chance to be discarded along the particle in resampling step because of an extremely small weight assigned to it. Also even for some "good" particles, we may neglect the "bad" sub-states. For example, in figure 1, the right side picture shows the system state in "particle 2". Although this particle is more likely to be maintained and duplicated in resampling because of a large weight assigned to it, the system state in an orange circle is even worse than that in "particle 1". Besides these examples, even in some situations where the proposal distribution is not perfect and, observations are not enough, all the particles may have extreme small weight, but the good portion is still maintained in a bad particle, which may lead to a failure of the algorithm.

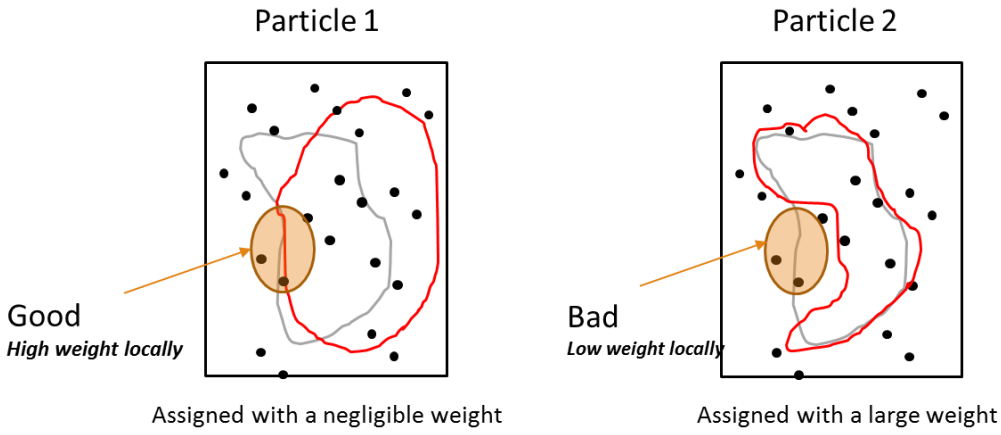


Figure 1.1 Example of ignorance of "spatial locality" nature in wildfire simulation.
Gray line shows the real fire front and red line shows the predicted fire front. Black dots are the location of sensors.

Furthermore, because of the limitations on observations, especially for the diversity of measurements, obtained observation data is not so sufficient that prediction accuracy needs to be

further improved. For example, in wildfire simulation, temperature sensors actually sense the closest ignition point in a circle area but cannot measure the direction to that point. Therefore, one particle with wrong ignition point but the same distance to sensor still can be assigned with a high weight and thus bring effect on the final prediction result.

In this dissertation, we will solve these two problems and preserve "good" local sub-state by exploiting two important natures in spatial temporal system, which have been ignored in most studies. The first one is the spatial dependency nature, which means that both the state and sensor are spatially dependent. The overall system state is composed of state variables of sub-areas. Similarly, typically sensors are located in different locations across the overall space. The second feature is the spatial locality feature, which means system state is correlated locally. And sensors can only sense the local information, which is limited in their own observation areas.

Motivated by the spatial locality property of both system state and observation data, we extend the standard bootstrap filter algorithm and propose a spatially dependent particle filtering framework. In this framework, we break state and observation into spatial regions and employ a divide and conquer strategy to reduce state dimension and data complexity. Also to improve the diversity of local observation data, there is a need to cooperate other kinds of data, e.g. soft sensor data that can be easily obtained from messages, reports, and social network and provide amounts of diverse information. Still, consider wildfire simulation, for example, soft data from the report can record fire spreading direction, fire speeding speed and even fire head location towards a landmark.

1.4 Organization

The rest of the work is organized as follows. Chapter 2 provides a literature review of related work. Chapter 3 exploits a spatially dependent particle filtering framework for data

assimilation in spatial temporal simulations by importing the two features. Chapter 4 studies automated partition methods for space partition based on the framework proposed in Chapter 3. Chapter 5 proposes PFs based data assimilation framework with hard/soft data. Chapter 6 compares the spatially dependent particle filtering framework to the existing particle filtering frameworks. After that, Chapter 7 concludes this work and points out the future research directions of this dissertation.

2 RELATED WORK

2.1 Overview of Data assimilation

2.1.1 Data assimilation applications and methods

Data assimilation, applied as the process for incorporating available observations to improve prediction results, has become more significant and popular in many spatial temporal systems, such as the fields in geoscience [11,12], ecosystems[13,14] and climate systems [15-17]. In these systems, the system dynamic models evolve both in time and space. So compared to the systems evolving only with time, the data assimilation for the spatial temporal system is even more complex. Recent studies have employed some stochastic methodologies in data assimilation, such as Kalman Filter, Extended Kalman filter(EKF) [18], Unscented Kalman filter(UKF) [19] and PFs. However, Kalman filter is constrained by an assumption of the linear system. Even for non-linear variants of Kalman filter --- EKF and UKF, the assumption is still limited to Gaussian distributed systems [19]. So, for a spatial temporal system, where the system model is always non-linear, non-Gaussian and unstable, PFs are preferred as there are no such assumptions on a system model.

2.1.2 Data assimilations with hard or soft data from local observations

Data plays an important role in DDDS system. The diversity of simulation application causes the variety of data, because the data type always relies on the application of simulation system. To simulate a transportation system, Hunter et al. imported variance of information to accurately determine the current traffic and predict the future traffic situation, such as intersection signal controllers, traffic flow volume, traffic flow density and traffic flow speed [20]. To track a target's position, the laser sensor is widely used to observe and infer the target's

relative position [21]. To simulate wildfire spread, temperature data from sensors deployed in a forest provides important information in a dynamic data driven wildfire system [1].

Importing soft data from human to provide real-time measurement of a system is a relatively new topic. The challenges come from the fact that soft data is fuzzy and usually hard to describe by a mathematic model, so the prediction of the system cannot depend only on soft data. Nevertheless, towards soft/hard data combination, some efforts have been made to improve the accuracy of prediction. Pravia et al. recently proposed a conceptual framework to assimilate both hard and soft data but did not provide realization of it [22]. Based on random finite set, Khaleghi et al. applied EKF (Evidential Kalman Filter) to incorporate both hard and soft data in target tracking application [21]. The proposed data fusion framework can only be applied to the linear system, and the experiments did not consider how to combine soft data and hard data that generated at the same time step. It also provided a soft data representation method for target tracking report, which is improved in our work to make it suitable for more general applications. To deal with a combination of hard and soft data, Gross et al. converted hard data into soft-compatible data, and used graph to associate all soft data [23]. Jenkins et al. employed fuzzy membership function to map qualitative estimations from human to quantitative values, which is used to score the similarity for data association and situation assessment [24].

2.2 Particle filters (PFs) and its applications

2.2.1 Overview of Particle Filters

PFs is a set of methodologies using Sequential Monte Carlo experiments to estimate the internal state of a dynamic system when given partial observations. A dynamic state space model is represented by two equations. One equation (1) is for state transition, showing how the system

state evolves x_{t-1} to next time step x_t; the another one (2) is for measurement, matching the observation y_t with predicted system state x_t. The two equations are formulated as follows:

$$x_t = f(x_{t-1}) + v_t \quad (1)$$

$$y_t = g(x_t) + \lambda_t \quad (2)$$

where f is the transition function, g represents the measurement function. v_t and λ_t are two independent random variables denoting added state noise and measurement noise.

The goal of PFs is to use a set of particles to approximate a posterior distribution $p(x_t|y_{1:t})$ of system state (also called conditional distribution) based on the observations in a stochastic process. Each particle consists of an index i and a system state $x_t^{(i)}$. Initially, the state of particles can be predefined in a proper way according to the needs. Then, the basic PFs follow a prediction-update methodology at each iteration. In the prediction part, we draw particles through a proposal density $q(x_t|x_{t-1}^{(i)}, y_t)$ (also called importance density) as described in equation (3). The proposal density $q(\cdot)$ could be system dynamics represented as $p(x_t|x_{t-1})$, or an optimal distribution $p(x_t|x_{t-1,}y_t)$ which also involves the latest observation y_t. After prediction, based on the latest observation, we assign each particle a weight $\omega_t^{(i)}$ calculated through the likelihood density $(y_t|x_t^{(i)})$. The weight is then updated by multiplying the previous weight $\omega_{t-1}^{(i)}$ as described in equation (4). After a number of recursions, the distribution of system state $p(x_t|y_{1:t})$ is estimated by these samples with probabilities proportional to the weights. The final distribution is approximated as in equation (5) where $\delta(x)$ is the Dirac delta function, and N is the number of particles[25,26]. If $N \to \infty$, the approximated distribution approaches the true posterior distribution.

$$x_t \sim q(x_t|x_{t-1}^{(i)}, y_t) \tag{3}$$

$$\omega_t^{(i)} \propto \omega_{t-1}^{(i)} \frac{p(y_t|x_t^{(i)})p(x_t|x_{t-1})}{q(x_t|x_{t-1}^{(i)}, y_t)} \tag{4}$$

$$p(x_t|y_{1:t}) \approx \sum_{i=1}^{N} \omega_t^{(i)} \delta(x_t - x_t^{(i)}) \tag{5}$$

There are varieties of PFs. The basic PFs Sequential Importance Sampling (SIS) described above is the simplest one containing only the sampling and weight calculation steps at each iteration [27]. However, a degeneracy problem may happen and let the prediction fail. The degeneracy means that, after a few of iterations, one particle will have a significant weight; the others will have a negligible weight and have almost zero contribution to the estimation of $p(x_t|y_{1:t})$. Thus, a lot of samples are wasted for computation, and the whole set cannot reflect the true posterior distribution correctly anymore. To deal with this problem, the sequential importance resampling (SIR) also called Bootstrap PFs is developed which uses importance resampling to generate a new set of particles with probabilities proportional to the weights. In the resampling step, we remove the particles with low weight and duplicate the particles with high weight. Then the newly propagated particles are assigned the same weight equals to $1/N$ for the iteration of next step. Furthermore, SIR chooses the system dynamic as the proposal distribution to obtain the new samples in the sampling step.

2.2.2 PFs based applications

Currently, numerous studies have been found using PFs for data assimilation work. Vermaak et al. [28] applied particle filter to approximate the clean speech and model parameters for the problem of speech enhancement. Towards the application of positioning, navigation and target tracking, Gustaffsson [29] presented a general framework implemented based on PFs. Nakamura et al. [30] assimilated real tide gauge data into the simulation by using PFs to correct

erroneous Tsnamic Simulation models. Mihaylova et al. [31] implemented PFs in freeway way traffic simulation which used a speed-extended cell transmission model for dynamics. Ruslan et al. [32] proposed PFs to predict flood water level for monitoring and tracking flood. But, most of them are for low-dimension space and the work implementing PFs in a large-scale spatial temporal system are quite few, because of a “curse of high dimensionality” [33].

2.3 Strategies towards complex spatial temporal simulations using PFs

Recently, several strategies are explored to overcome the high dimensionality problem existed in particle filter for a large-scale spatial temporal system. The first strategy is the local analysis which takes advantage the "spatial locality" feature in a spatial temporal system. The next strategy is to import the concept of distributed and parallel computing. Another strategy is to optimize the proposal distribution in sampling step. Besides those three strategies for a general high dimensional system, state partitioning is used as another methodology especially towards a spatial temporal system.

2.3.1 Local analysis

The feature “Spatial locality” has well been considered in EnKF [34-36] to reduce state dimension locally for geophysical systems. Accordingly, there are two common localization methods in EnKF: covariance localization and local analysis. The covariance localization [35-37] defined a distance-based correlation function for updating state error covariance. The local analysis applies localization in the local state variables (i.e. obtained by a sliding window) and updates states locally like LEKF[38,39] and LETKF [40]. Note that the local state vectors overlap when using a sliding window and thus leads to discontinuities across the system space. So Hunt et al. [40] proposed to use an error covariance matrix to reduce the impact from

boundary observations. Compared to covariance localization, the local analysis is more suitable for a large scale system since it is a scheme-independent method [41].

However, unlike EnKF, PFs do not reply on covariance matrix and may need resampling step to update ensemble (i.e. a set of particles). Hence different localization approaches are proposed in PFs. Lei and Bickel proposed a moment matching particle filter for a non-linear non-Gaussian system to enable localization [42] . This method can keep the spatial smooth by avoiding the resampling step. Similar to the idea of the distance-based correlation function in covariance location of EnKF, a recent study in [43] proposed a new localized particle filtering incorporating a localization function on the likelihood function to update the weights of particles. If the site of a single observation is far away from a state variable, the weight from that observation is almost 1; however, if it is close to the state variable, the weight become the same as before. Besides, the observations are assimilated sequentially, and for each observation, the weight updating and resampling step are performed based on the samples merged with prior particles. Soon after that, Poterjoy and Anderson [44] implemented this localized particle filtering into a high-dimensional geophysical system for the first time. Penny and Miyoushi [45] also presented a local particle filter for geophysical systems. Morzfeld et al. [46] pointed out that localized PFs work for the systems with small enough sub-problem dimension.

However, these local PFs require special operations on the system state and thus not applicable to a wildfire simulation model, such as DEVS-FIRE. In paper [42], state should be updated using a linear model based on EnKF by calculating the mean, which is also not applicable in DEVS-FIRE. As stated in previous response, either the localization function or merging operations in [43,44] is not suitable for wildfire simulation.

2.3.2 State partitioning

Recently, the strategy state partitioning is explored especially towards a spatial temporal system. State partitioning is a method that divides state into multiple partitions and then applies data assimilation. Generally, there are two state partition methods in general, depending on whether the system is geophysical based or not.

In a *non-geophysical* system, the system state is divided by exploring a complex hierarchical structure. Partitioned sampling proposed in [47] is a novel sampling method that using state partitioning concept in tracking multiple objects. In this method, each target was considered as a single partition. Then the system dynamics and observations were broken into multiple components. Based on the partition, each component applied dynamics and resampling sequentially in a hierarchical manner. Later, MacCormick and Michael [48] implemented this method in hand tracking and again proved the advantage of partitioned sampling. Also for the hand tracking application, Brandao et al. [49] presented a similar subspace hierarchical particle filter in hand tracking, which partitions the state space according to some implicit structure inferred from observation functions. Especially, the fingers in hands are broken into multiple groups (probability unbalanced) and execute PFs through a directed acyclic graph structure. Deutscher and Reid [50] introduced a crossover operator from Genetic Algorithms to populate new particles in proposed annealed particle filtering. Besides the object tracking in computer vision, the state partitioning is also used in other more applications, e.g. in multiple targets tracking, where the sensors are more diverse. Djuric´ et al. [51] partitioned the state dimensions into subspaces and applied particle filter in each partition for the target. Particularly, the state was decomposed according to the measurements by variance of sensors (e.g. signal strength, angle of signal arrive, direction of motion and absolute velocity of target).

In a geophysical system, the state is divided by partitioning the spatial space. This method is straightforward since the state variables always have the same properties and are correlated with their neighbors in a geography map. However, to the best of our knowledge, very few efforts focus on studying the impact from state partition on prediction accuracy in a geophysical system. In data assimilation of numerical weather forecast (NWF), a simple way to get a partitioned region (i.e. a sliding window) with lower dimension is to form a patch of grid points (e.g. rectangle shape) centered at a specified grid point [52][39]. Although Rebeschini et. al [53] analyzed the performance of local PF theoretically based on partitioned sub-states (i.e. blocks), they did not discuss on how to do state partition but suggest to select a typical small block size.

2.3.3 Optimization on proposal distribution

"Spatial locality" is also used in PFs by optimizing the proposal distribution in sampling step. The optimal proposal incorporates the current real observation to get more possible samples [54]. This method restricts the search space to possible space that can generate the current observation and therefore reduces the sampling spaces. In paper [33], Snyder reviewed the optimal proposal methods and showed the sample size is reduced dramatically compared with standard proposal. In paper [55], Xue et al. proposed an effective proposal distribution and applied this in wildfire simulation. The results showed the results have been improved, especially in the cases standard particle filter fails. Although the optimal proposal distribution is demonstrated as an efficient schema to reduce the sample size, and the design a proper optimal proposal distribution in a specified application is hard for complex system.

2.3.4 Distributed and parallel computing

Early studies on distributed PFs focus on minimizing the execution time by paralyzing the operations in the three main steps. The sampling and weight calculation step for each particle can be considered independent and simple to be parallized. But the resampling step is a centralized operation as it needs to gather and normalize all the weights from particles. So paralyzing resampling step is most critical. Miodrag [56] introduced novel resampling algorithms RNA (Resampling with proportional allocation) and RPA (Resampling with non-proportional allocation). It is implemented in a simple architecture, where several distributed processors (PEs) run sampling and compute weight for a sub-set of particles, and a central processor(CU) fulfills particle collection and particle scheduling work in resampling step. The resampling algorithms not only reduce the execution time and also the communications between PEs. Later, still based on this simple architecture, Balakumar [57] proposed a statically resampling method to mix the particles in each PEs to maintain the diversity of particle populations. Recently, Fan [58] found the unbalance problem among PEs after resampling. Therefore, with the purpose of reducing the total communication cost, he proposed multiple routing policies for selecting the surplus particles in a PE to another PE and proved the total transferring states has been minimized in the experiment of wildfire simulation. Also for the architecture, Shabany [59] proposed a new full parallel architecture for distributed resampling to fix the particle scheduling and thus make resampling step paralyzed completely. All the studies have proved that applying particle filter in a paralyzed manner efficiently reduce the computation time with a large sample size. But when state variables enlarge the sample size may exponentially increase also, we still need extremely huge set of samples to constitute the proposition of possible states and thus bring high overhead

in the processors. Also the hardware requirement constrains the implementation of distributed PFs.

2.4 Overview of the DEVS-FIRE Simulation Model

The DEVS-FIRE [60,61] model is a 2D cellular space model based on Discrete Event System Specification [62]. In DEVS-FIRE, the ground is modeled as a 2D cell space, which is divided into rectangle cells whose dimensions relying on the resolution of GIS fuel and terrain data. Each cell is a DEVS atomic model, in which fuel and terrain are assumed to be uniform assigned. Cells are coupled with their 8-neighbors. Then a weather model is coupled to all cells to receive dynamically changed weather data (wind speed and wind direction). Fire spread simulation in DEVS-FIRE is modeled as a propagation process as burning cells ignite their unburned neighbors. The rate of spread and spread directions for one ignited cell are calculated based on Rothermel's fire behavior model [63], depending on its fuel, slope, aspect, and weather data.

In DEVS-FIRE, initially we set all cells as unburned (passive) state. Once a cell ignited, it changes to burning state. After the maximum burn time expires (i.e. the length is computed by Rothermel's model and mainly depends on the size of cell and fuel type in it), a burning cell converts to the burned state. Figure 2.1 displays an example of simulation result with fuel type information in DEVS-FIRE. It comes from a portion of global map, which is composed of a set of colored cells. Red represents the cells in burning state; black denotes the cells in burned state. The other colors display the levels of fuel type in the cells, and also denote unburned cells marked transparent inside.

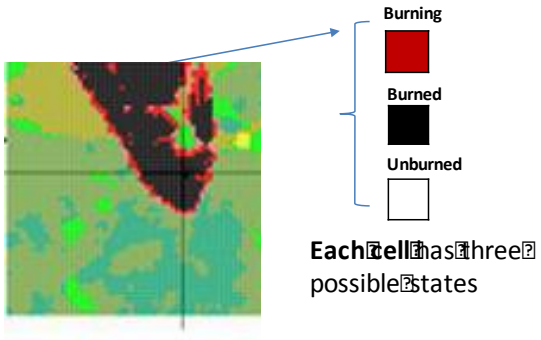


Figure 2.1 An example of wildfire spread simulation result.

The system state for DEVS-FIRE at time step t is defined as a vector of states for all cells, denoted by fire_t. Therefore, the system transition model for our wildfire simulation can be defined as

$$\text{fire}_t = \text{DEVSFIRE}(\text{fire}_{t-1}, \theta_t, \Delta t) + v_t \quad (6)$$

Where θ_t is a vector of model inputs(GIS, weather, and so on), Δt is the time duration.

3 SPATIALLY DEPENDENT PARTICLE FILTERING FRAMEWORK

3.1 Introduction

A standard PFs algorithm contains three main steps at each iteration: sampling, weight calculation and resampling [64]. The sampling step is to evolve the system state of each particle to the next data assimilation time point; the weight calculation step is to compute the weights of particles based on observation data (i.e., sensor data); and the resampling step is to select a new set of particles based on particles' normalized weights. The standard PFs provides a general framework for carrying out data assimilation. However, it faces challenges to work effectively for complex spatial temporal systems that have a large number of state variables due to the large spatial areas of interest [65]. For these systems, a small number of particles are difficult to achieve satisfactory results by having "correct combination" of all state variables. Take a cellular space-based wildfire spread simulation as an example [60], in which the system state is composed from the state of each cell in the cell space. A $200 * 200$ cell space would have $40{,}000$ cells. In order to have accurate data assimilation results using particle filters, a large sample size (i.e. the number of particles) is needed in order to work well with large number state variables. However, increasing the sample size leads to higher computational cost. This is especially true for complex spatial temporal simulations because each particle involves a full-scale simulation to the next observation time point.

The goal of this work is to improve PFs-based data assimilation for spatial temporal simulations by exploiting important features of spatial temporal systems. In particular, we exploit two features that are common for spatial temporal systems. The first one is that both the state and sensor are spatially dependent. The overall system state is composed from state variables of sub-areas. Similarly, typically sensors are located in different locations across the overall space. The

second feature is the *spatial locality* feature. The system state is correlated locally. And sensors can only sense the local information, which is limited in their own observation areas. These features are seriously ignored in standard PFs implementation. For example, in standard PFs, the system state is treated as a whole when carrying sampling, weight, and resampling. Furthermore, the importance weights of different particles for any chosen state variable are influenced by all observation data, even if those observation data are nearly independent of the particular state variable [9,10]. For a high dimensional spatial temporal system whose states and observation data are spatially distributed and have finite correlation lengths, the standard particle filters thus overestimate the information available in the observation data and underestimate the uncertainty of the posterior distribution. Consider wildfire as an example, fire only spreads to the neighboring cells and the observation data (e.g., ground temperature sensor data) from different regions of the fire typically reflect only the fire states in their corresponding regions, not others. Currently the spatial locality feature has been taken advantage in large-scale geospatial systems to improve prediction accuracy, such as numerical weather forecast (NWF) system. Accordingly, several algorithms deriving from ensemble Kalman filter (EnKF) have been proposed to achieve prediction accuracy improvement at a modest computation cost via incorporating "spatial locality" feature [39]. Generally, in these algorithms, the global space is divided into sub-spaces, and then data assimilation is performed at each sub-space using state variables and observations locally. Similar idea can be found in [66], which provided mathematical proof on a local particle filtering – block particle filtering. Motivated by the spatial dependency and locality features of both system state and observation data, we extend the standard bootstrap filter algorithm and propose a new spatial partition-based particle filtering algorithm, especially for wildfire

simulation. This algorithm breaks state and observation into spatial regions and employs a divide and conquer strategy to reduce state dimension and data complexity.

In this dissertation, we develop a new spatial partition-based particle filter framework. This new framework incorporates both the state partition and "spatial locality" in PFs based assimilation for simulations. We divide system state into multiple sub-states and conquer the processed sub-states into a full state to accomplish the iterations. Similar to standard PFs, it includes the three main steps at each iteration. Sampling is still based on a full state because simulation model needs the whole state to correctly simulate the evolution of the system state. However, unlike the standard PFs, weight calculation is based on each sub-state and takes into consideration the sensors that have observations coverage over the area the sub-state belongs to. Besides, resampling is also performed on sub-state. And finally in order to cooperate with next iteration's sampling step, we need to reconstruct sub-states in particles to form a full state. To support this framework, there are several issues we need to deal with, including how to divide a system state, how to calculate weight for each sub-state taking into account the boundary sensors (i.e. the sensors in which the observation area covers more than one sub-states), and how to carry out resampling to reconstruct new particles from sub-states. We propose methods for each of them. To evaluate the proposed framework, we applied it to wildfire-spread simulation. It is important to note the framework is general and can be applied to other spatial temporal systems. The contribution of this section is 1) we propose a spatially dependent PF framework, especially for wildfire simulation; 2) we point out the boundary sensor problem after dividing sub-states; 3) we provide a two-level spatial partition method to break a spatial area; 4) An case study in wildfire simulation have been conducted.

3.2 Overall Framework

Before describing the overall framework, we need to clarify how the divide and conquer strategy is incorporated over system state. 1) In order to divide the system state we partition the simulation space. Let the entire space $r = r_1 \cup r_2 \cup ... \, r_j \, ... \cup r_m$, which is broken into n smaller regions r_j for $j = 1..m$. Then according to the divided regions, we can partition the full system state x and observation data y into subgroups because they are spatially dependent. In this proposal, we use sub-state to refer to a partial of system state, and sub-observation to represent a subgroup of observation data. Figure 3.1 shows one example of state partitioning by gridding space r. In figure 2, r_j is one region in whole space r, x_{r_j} is the sub-state located in r_j , y_{r_j} denotes the sub-observations related to r_j . Secondly, after performing some activities individually (to be described later), the sub-states are reconstructed to form a full state for future operations. An early version of the framework can be found in [67].

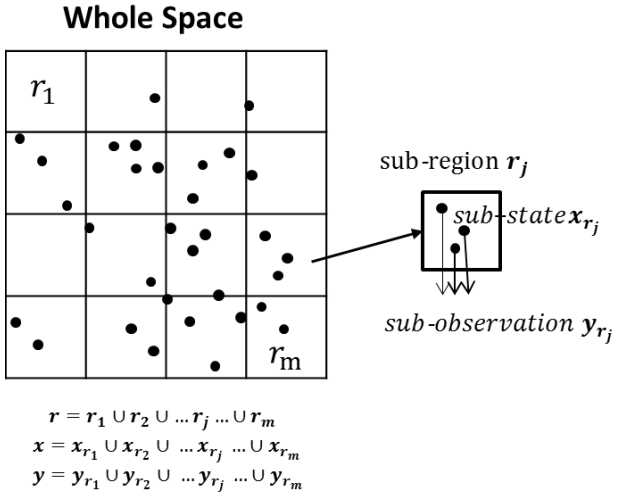


Figure 3.1 Example of state partitioning and generated sub-states and sub-observations.

The new spatial partition-based particle filter framework contains a similar three steps flow for each iteration as described in the bootstrap PFs. However, it defers from the bootstrap PFs greatly since the divide and conquer strategy requires some operations to be accomplished

on a sub-state instead of a full system state. A general flow of this new framework can be found in figure 3.2.

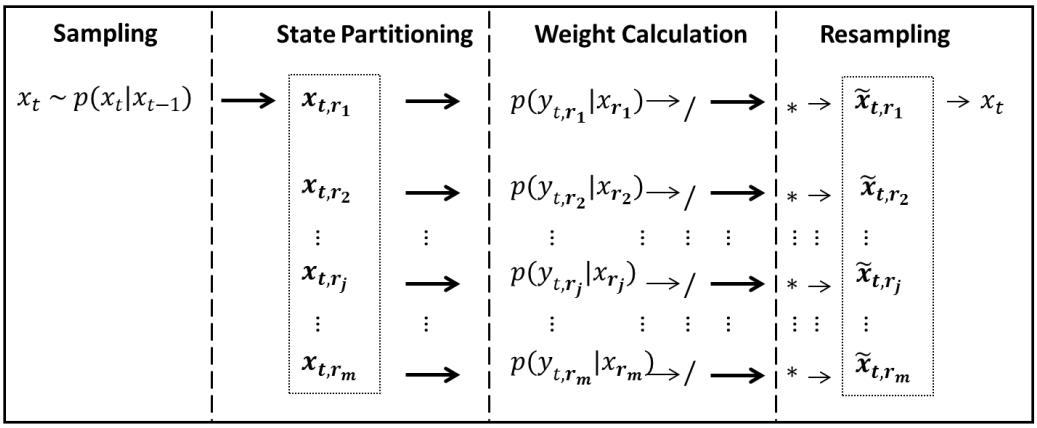


Figure 3.2 General flow of spatially dependent particle filtering method.
Weight normalization is denoted by /. Resampling is denoted by *.

As illustrated in figure 3.2, one more step for state partitioning is added, so there are about four steps totally at each iteration in SpSIR algorithm. 1) Sampling is still the first step. This sampling step shares the same sampling methods of Bootstrap filter, in which a transition equation is directly used to generate new full state for the samples given the last step's full state. 2) State partitioning is inserted as the second step. In order to partition a system state, we break a real system space into multiple regions while meeting some criteria. And then according to those regions, a full system state is divided into multiple sub-states. 3) Weight calculation becomes the third step. This step differs significantly form the bootstrap filter algorithm. We compute weight for each sub-state instead of a full system state. And the measurement equation for each sub-state becomes $p(y_{t,r_j}|x_{t,r_j})$, where x_{t,r_j} represents a sub-state belongs to region r_j and y_{t,r_j} denotes the observations associated to region r_j respectively at time step t. After that, weights are normalized for each region represented by / in figure 3.2. 4) Resampling (denoted by * in figure 3.2)

remains the last step. However, the particles are formed into multiple sets according to the number of regions m. Each particle set is for one region, while for each particle the system state is the sub-state belongs to the region and the weight is corresponding sub-state's weight. Then resampling is performed in each set. After that, we combine the sub-states from those sets and reconstruct a full system state x.

In order to implement this framework in a spatial temporal system, there are several problems we need to consider. These problems exist mainly in the last three steps of the flow. First of all, as state partitioning depends on space partitioning which has many variations, we need to find a proper way to design the partition. Another main problem called data association is found in weight calculation step. In this step, some boundary sensors located near the borders of regions may have observation coverage over several regions, but it is unknown that sub-state from which region impacts sensor and provides final real observation. So it is hard to give a correct association between the sub-state and observations. The last issue is how to apply resampling on each sub-state and reconstruct a new full system state. In the following sections, we will explain the solutions for each problem one by one.

3.3 State Partitioning

In a spatial temporal system, since system state evolves in a real space, state partitioning relies on a proper partition of space. Once a full space r is divided into multiple smaller non-overlapping regions denoted as r_1 to r_m, we can get corresponding sub-states x_{r_1} to x_{r_m}. Therefore, state partition indeed becomes space partitioning.

To divide a 2D space, there are varieties of methods. Basically, three approaches can be directly applied and easily implemented. They are arbitrary partitioning, geography partitioning

and sensor clustering partitioning methods. Figure 3.3 shows the generated sub-regions separated by black line after performing these three partitioning methods on the same space.

The first two methods---grid partitioning and geography partitioning are two straightforward space-partitioning approaches. Grid partitioning breaks a full space into multiple arbitrary grids (or other regular/irregular shapes). As it requires the least prior knowledge for partitioning, we will implement that in our experiment later. Geography partitioning process the geography information and then break the space according to location of city, forest, mountain, road and etc.

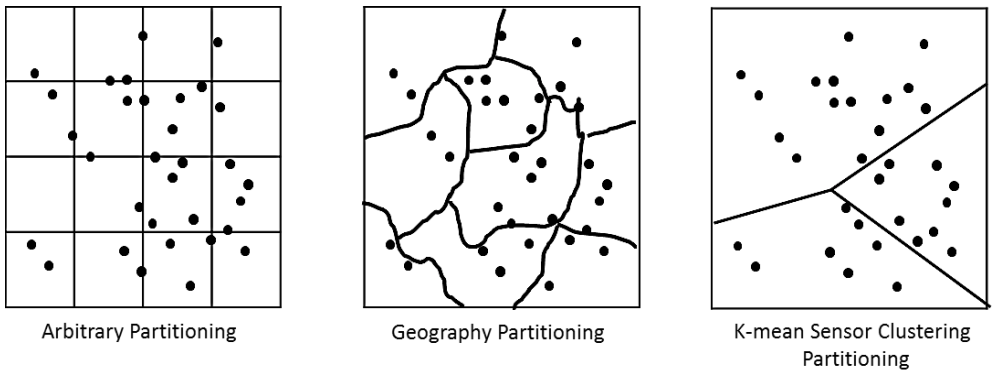


Figure 3.3 The results after performing three basic state partitioning methods on the same space. The generated regions are separated by black line. Black dots represent the sensor locations.

The last basic approach to divide a space depends on the distribution of sensors. Since in data assimilation method our system state is validated and updated by real observations from sensors at each iteration, appropriately employing sensor's observations is quite useful to improve the final prediction result. For example, if one region has no sensor located, the corresponding sub-state cannot be validated by real observation and thus is useless for obtaining a better forecast. K-mean sensor clustering partitioning is a classical sensor distribution based space partitioning schema. K equals to the provided number of sub-states m.

We consider sensor locations as 2D points in a plane space. In this partitioning method, by applying K-mean clustering on locations of sensors, the sensors are classified into K clusters, while each cluster has a center called centroid. Consider the space as a plane consisting of a finite set of points, for each centroid we can get a corresponding region in which the points are closer to the centroid than to any other. Then we build a voronoi graph in the whole space.

3.4 Weight calculation

Weight calculation step differs greatly from standard PFs. Since the full system state is broken into multiple sub-states after state partitioning, we assign a weight to each sub-state in every particle at this step. Traditionally, each particle is associated with a weight w. But in this framework, each particle is assigned a weight set $\{\omega_{x_{r_1}}, \omega_{x_{r_2}}, \dots, \omega_{x_{r_j}}, \dots, \omega_{x_{r_m}}\}$. The element $\omega_{x_{r_j}}$ in the weight set is for the weight of corresponding sub-state in r_j. To obtain the value of $\omega_{,x_{r_j}}$, we should use following equation

$$\omega_{x_{r_j}} = p(y_{r_j}|x_{r_j}) \tag{6}$$

In this equation, x_{r_j} represents the sub-state in region r_j and y_{r_j} denotes a set of observations impacted by system state in r_j. A standard PF takes accounts all the observations y to compute the weight for each particle. So in a standard PF, we can define a multivariate Gaussian distribution to calculate ω for the full system state in each sample. We assume the observations are independent, then a diagonal covariance matrix Σ is used in this multivariate Gaussian distribution as shown in equation (7) for n_{obs} observations. This equation can also be written as (8) by defining a density function $p(y_k|x) \sim N(y_k, {\delta_k}^2)$ for each single observation, and ${\delta_k}^2$ is the variance .

$$\omega = \frac{\exp(-\frac{1}{2}(g(x)-y)'\Sigma^{-1}(g(x)-y))}{(2\pi)^{n_{obs}/2}|\Sigma|^{1/2}} \quad (7)$$

$$\omega = \prod_{k=1}^{n_{obs}} f(y_k|x_k) \quad (8)$$

But here for each sub-state x_{r_j}, the weight calculation way is different as we only consider a subset of observations contributing to it. This owes to the spatial locality nature from sensor. It means that a widely distributed sensor can only provide information limited in its own observation area. As the observation area from one sensor may overlap with each other, some location can be observed by multiple sensors, but not all sensors distributed in space. Therefore, before calculating the weight for one sub-state, we need to assign observations to the possible sub-states. Since the assignment of sensor's observation depends on whether one sub-state impact it or not, we consider all regions covered in the observation area. If the observation area covers only one region, the observation from sensor is simply assigned to the sub-state in the covered region. However, some boundary sensors located nearing regions' border may cover two or more regions. Then these boundary sensors' can sense the sub-states from all covered regions. But it is unknown that which sub-states contribute to the final observation. So it is hard to assign a boundary sensor to the impacted sub-states correctly. Take the wildfire as an example, in which the ground temperature sensor is used to detect ignition point in a limited range. In figure 3.4, the state space is decomposed into 4 regions and red line shows the fire front from one particle. Each sensor has an observation range represented by gray circle. Sensor A's observation range limits in region 1 and only reflects the state in region 1; Similarly, sensor B's observation range limits in region 2 and only reflects the state in region 2. However, sensor's observation area can also across one or more sub-states like boundary sensor C and sensor D.

Sensor C locates at region 2 and covers both region 1 and region 2. It seems only reflect the fire information from region 1 in this particle. But we cannot simply assign it to region 1. Because the ignition points from real fire impacting sensor C may come from Region 2. We cannot ignore this case. Otherwise, there may be a wrong assignment for sensors to sub-states. Similarly, sensor D's observation ranges across both region 1 and region 3 and could reflect the information for both these two regions. But assigning Sensor D to which region or sub-state is also a problem.

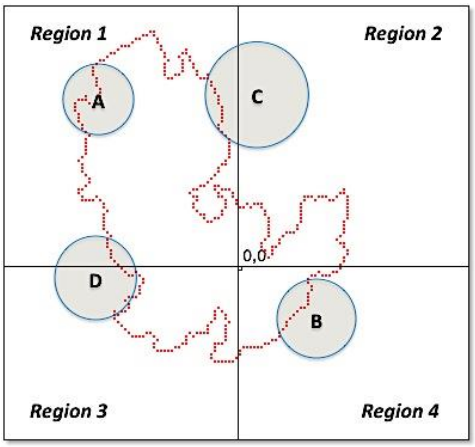


Figure 3.4 Example of sensor's locality nature and sensor assignment problem in wildfire simulation.

For boundary sensor, assigning to which sub-state becomes a classical data association problem. Currently, a series of algorithms have been developed for data association problem in target tracking, such as NNSF, PADF, JPDF[68], GNN[69,70]. However, in a general spatial temporal system, this problem could be more complex because of the diversity of sensors. The nature of sensor decides what process we should apply to associate observation with sub-state. A sensor's observation can actually derive from a particular point, like laser sensor in target tracking and ground temperature sensor sensing the nearest ignition point in wildfire; It can also be a synthetic value from a range, like density sensor measuring traffic flow of a segment of road

in traffic simulation. For the first kind of sensor, among all the covered sub-states, we need to find the most possible sub-state. For the second kind of sensor, we can either split the observation to all the covered regions according to some criteria (e.g. ratio of length or area among the sub-regions of observation) or directly build a likelihood function based on the real observation and simulated observation from sub-state. Because of the complex of data association problem, we will not discuss it in details.

In this paper, we only consider the ground temperature sensor for which the observation is impacted by the nearest ignition point from system state. Because in order to calculate simulated temperature, we must follow equation (9) to define measurement function g, where d is the closest distance between a sensor and ignition point[1].

$$\hat{y} = 376e^{-d^2/2\sigma^2} + 26 \tag{9}$$

In this case, that single ignition point may locate at all possible regions the observation covers in each particle. So we assume for each covered region, the probability containing that ignition point is the same. For each sub-state, to get the weight based on (6), we use the following equation to calculate the final weight.

$$\omega_{x_{r_j}} = p\left(\tilde{y}_{r_j} \middle| x_{r_j}, J_{r_j}\right) \tag{10}$$

where $\tilde{y}_{r_j}$ represents a set of observations which has a coverage area over region r_j and J_{r_j} denotes the associations that the real observation in $\tilde{y}_{r_j}$ is from the sub-state in region r_j (i.e. region r_j contains the particular point). x_{r_j} , J_{r_j} are jointed because only when data association is defined, then we can perform the measurement function. After normalizing the weights of all particles, the weight for particle i becomes

$$\bar{\omega}_{t,x_{r_j}}^{(i)} = \frac{p\left(\tilde{y}_{t,r_j}^{(i)} \middle| J_{t,r_j}, x_{t,r_j}^{(i)}\right)}{\sum_{i=1}^{n} p\left(\tilde{y}_{t,r_j}^{(i)} \middle| J_{t,r_j}, x_{t,r_j}^{(i)}\right)} \quad (11)$$

To calculate $p\left(\tilde{y}_{t,r_j}^{(i)} \middle| J_{t,r_j}, x_{t,r_j}^{(i)}\right)$, assuming observations are independent, we can use a distribution in equation (12). The difference for weight calculation between a full system state and sub-state is that a new function $h\left(J_{t,r_j,\tilde{y}_{t,r_j,k}}\right)$ is used to represent the data association probability in sub-state.

$$p\left(\tilde{y}_{t,r_j}^{(i)} \middle| J_{t,r_j}, x_{t,r_j}^{(i)}\right) = \prod_{k=1}^{m}\left[g\left(J_{t,r_j}\right) \cdot f\left(\tilde{y}_{t,r_j,k}^{(i)} \middle| x_{t,r_j}^{(i)}\right)\right] \quad (12)$$

where $g\left(J_{t,r_j}\right) = \frac{1}{c_{\tilde{y}_{r_j}}}$ and k represents the kth observation in vector $\tilde{y}_{i,t,r_j}$

$g\left(J_{t,r_j}\right)$ and $f(\tilde{y}_{k,i,t,r_j}|x_{t,r_j}^{(i)})$ are two density functions. $g(J_{t,r_j})$ describes the probability that real observation is reflected by sub-state $x_{t,r_j}^{(i)}$ in region r_j and $f(\tilde{y}_{t,r_j,k}^{(i)}|x_{t,r_j}^{(i)})$ shows the probability that sub-state $x_{t,r_j}^{(i)}$ is a real system state. $c_{\tilde{y}_{r_j}}$ is the number of regions observation y_{r_j} covering. So for example, if one observation covers 4 regions, then c equals to 4 and $g\left(J_{t,r_j}\right) = 1/4$ which means the chance the real observation impacted by this region is 1/4. The definition of $f(\tilde{y}_{t,r_j}^{(i)}|x_{t,r_j}^{(i)})$ depends on a specified application. Similar to that in standard PF, we use a Normal distribution for it as in equation (13).

$$p\left(\tilde{y}_{t,r_j,k} \middle| x_{t,r_j}^{(i)}\right) \sim N\left(\tilde{y}_{t,r_j,k}, {\delta_k}^2\right)$$

$$= \frac{1}{\delta_k\sqrt{2\pi}} e^{-\frac{(\tilde{y}_{t,r_j,k} - \tilde{y}_{t,r_j,k})^2}{2(\delta_k)^2}} \quad (13)$$

where δ_k is the variance used for the kth observation $\tilde{y}_{t,r_j,k}$ in vector $\tilde{\mathrm{y}}_{\mathrm{t,r_j}}^{(\mathrm{i})}$, $\hat{y}_{t,r_j,k}$ is obtained by equation (6) given sub-state $x_{t,r_j}^{(i)}$.

3.5 Resampling

Resampling step is applied to generate a new set of samples to overcome degeneracy problem and increase the diversity of samples. There are several resampling algorithms as described in paper [27]. Still based on Bootstrap PF, we use the method that draws samples through a probability proportional to normalized weight.

However, the resampling is for each sub-state separately instead of a full system state. After weight calculation, for each region $\mathrm{r_j}$, we obtain a set of normalized weight $\{\bar{\omega}_{\mathrm{t,x_{r_j}}}^{(1)}, \dots, \bar{\omega}_{\mathrm{t,x_{r_j}}}^{(\mathrm{N})}\}$ corresponding to the set of sub-states $\{\mathrm{x}_{\mathrm{t,r_j}}^{(1)}, \dots, \mathrm{x}_{\mathrm{t,r_j}}^{(\mathrm{N})}\}$. Then a new set of particles are resampled as $\{\tilde{\mathrm{x}}_{\mathrm{t,r_j}}^{(1)}, \dots, \tilde{\mathrm{x}}_{\mathrm{t,r_j}}^{(\mathrm{N})}\}$ for region $\mathrm{r_j}$.

Another task in resampling step is to reconstruct a full system state to fulfill the sampling step for next iteration. Since the space is divided into M regions, we simply group the sub-state in the same particle (i.e. with same particle index) to form a full state. So we group $\{\tilde{\mathrm{x}}_{\mathrm{t,r_1}}^{(\mathrm{i})}, \dots, \tilde{\mathrm{x}}_{\mathrm{t,r_m}}^{(\mathrm{i})}\}$ together to form a new full state, where $\tilde{\mathrm{x}}_{\mathrm{t,r_1}}^{(\mathrm{i})}$ is a sub-state of region $\mathrm{r_1}$ from original ith particle. For each new group of sub-state set, we assume the system state does not have a high dependency spatially so that sub-states can be combined directly and integrated into a full system state.

After resampling step, not only the local good particle is remained but also the diversity of particles is increased. This is because that the sub-state with high weight is duplicated in resampling step even if the full system state it belongs has a relatively low weight. Also, each

sub-state is like a piece in gene, after resampling, the sub-states from different particle are reconstructed to form a new generation. This process is similar to crossover and thus increases the diversity of samples.

However, there is a concern about the continuity of system state. This resampling step simply constructs a full system without considering the continuity of system state. So our algorithm may not fit well in other systems, e.g. numerical system models, where the state variables are some numerical values and need specific process to form a continuous full state (i.e. the neighboring state variables should have similar values). However, it is not a major issue in wildfire simulation. On the one hand, the fire spread does not depend on a continuous model for simulation. On the other hand, the burning cells in fire front still get a continuous flame front because the burning cells spread to their neighbor cells after some time and eventually a continuous fire front is formed.

3.6 Experiment

3.6.1 Experiment settings

Since it is hard to get the real fire front and obtain sensor information in physical environment, we chose to use identical twin experiment to evaluate the prediction accuracy. First, we run a pure DEVS-FIRE simulation with no employment of data assimilation methods. And consider all those input data as the correct data, including weather data. Also, the corresponding output is considered as "true" result and generated sensor data are recorded as real observation data. Then we conduct another simulation applying data assimilation methods in condition of erroneous weather data and using ground temperature sensor data generated from the first run. For "correct" weather information, the wind speed 24 (m/s) was and wind direction was 30 degrees with random variances added every 30 minutes. Similarly, for "erroneous"

weather information, the wind speed was 22 (m/s) and wind direction was 60 degrees with random variances added every 30 minutes. The random variance follows a uniform distribution. For wind speed, it ranges from -2 to 2 (m/s); for wind direction, it ranges from -30 to 30 (degrees). Although the input data for weather differs in these two runs, the ignition points added for DEVS-FIRE model is the same. Initially, an ignition point was added at location (35,40). Later, at time step 4, another ignition point was added at location (45, -12). The settings for both weather data and ignition points can also be found in following table.

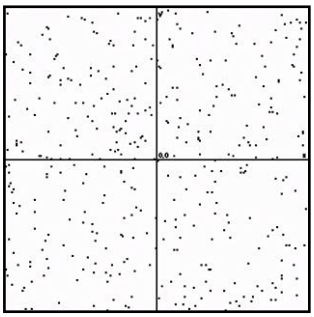
Table 3.1 Experiment setting for weather data and ignition points

	Weather data	Ignition Points
"Correct"	Speed: 24±2 m/s Direction: 30±30 degrees	Point 1: (35,40), added at T=0 Point 2: (45,-12),added at T=4
"Erroneous"	Speed: 22±2 m/s Direction: 60±30 degrees	Point 1: (35,40), added at T=0 Point 2: (45,-12), added at T=4

According to different settings of experiments, three terms are defined for the simulated results. They are "real" fire front, "simulated" fire front and "filtered" fire front. The results generated through pure simulation model in condition of "correct" and "erroneous" weather data are called "real" fire fronts and "simulated" fire fronts respectively. However, for the results coming from data assimilation method with "erroneous" weather data and ground temperature sensor data, they are called "filtered" fire fronts. In order to evaluate the proposed spatial partition-based PF method, for the "filtered" fire fronts we also use the standard bootstrap PF to get results and then compare our results with that from the standard PF. More details are described later when we present the results.

In simulation model, cell space is set as large as 200*200 where the side length for each cell is 30 (m). We use 12 time steps in the simulation, where for each time step the duration is 20 minutes. Four–hundred sensors are distributed over the whole space. The observation area for each sensor is limited in a circle area with radius equal to 150 (m). In this proposal, two

distributions for sensors are considered. The four hundred sensors are distributed randomly and uniformly over the space. The following figure displays sensors' location. The dots represent the locations of sensors.



Distribution-1:Uniform distribution

Figure 3.5 Sensor distribution for Distribution -1.

To test how the partition method impacts the prediction result, generally we used two partition strategies in this proposal. On the one hand, as grid partitioning is a straightforward division approach, it is employed to demonstrate the advantages on increasing partition number based on pre-knowledge of fire spreading information. Generally, six cases are designed for this partition strategy. For these six cases, we chose the partition number as 1,2,4,6,8,10 respectively. Note that, if partition number equals to 1, the system state indeed is not partitioned, so the prediction results based on this partition are the same as that from the standard PF based data assimilation method. Besides, for sensors' distribution, we use Distribution-1. 50 particles are used to obtain the filtered results.

The accuracy of prediction result is measured by mean square error (MSE):

$$MSE = \frac{1}{N_{cell}} \sum_{k=0}^{N_{cell}} (\widehat{x_{k,t}} - x_{k,t})^2$$

where N_{cell} denotes the number of cells (i.e. total state variables) over space, $\widehat{x_{k,t}}$ represents the estimated state of the kth cell from *filtered fire* at last time step, $x_{k,t}$ represents the kth cell state from *real fire* at last time step. Note that we chose the filtered fire front with the highest weight among all the particles. Since we cannot directly apply subtraction for the cell states, we define that if the estimated state differs from real state and one of them is "unburned", then $\widehat{x_{k,t}} - x_{k,t} = 1$, otherwise, $\widehat{x_{k,t}} - x_{k,t} = 0$. This definition means that we care more about the fire front, and the errors happen if the filtered fire front outside or inside of real fire front.

However, due to the discrete event nature in our wildfire simulation model DEVS-FIRE, it is difficult to directly apply other existing localized PFs or Monte Carlo methods into wildfire simulation as mentioned in Chapter 2 "Related Work". So to prove the improvement on prediction accuracy, we compare our method only to the standard PF (i.e. the cases when sub-state number equals to 1).

3.6.2 Experiment results and analysis

To compare the results on increasing partition number with pre-knowledge of fire spreading information, we used grid partition method to generate six cases for six kinds of partitions manually. The corresponding sub-state number was increased from one to ten, which is selected from set {1,2,4,6,8,10}. These partitions were designed under two general principles. 1) More partitions are made around the first initial point. This is because the fire fronts eventually become larger from the initial ignition points. 2) The areas with great prediction errors are more likely to be separated from others. This is based on that we already known how fire spreads generally.

After applying the spatial dependent PFs method, the filtered results of one run for all six cases were obtained as shown in figure 3.6. For each case in the figure, black line denotes the

"real" fire front, blue line represents the "simulated" fire front and red line indicates the "filtered" fire front. Green areas show the symmetric differences of cell status between "filtered" fire front and "real" fire front.

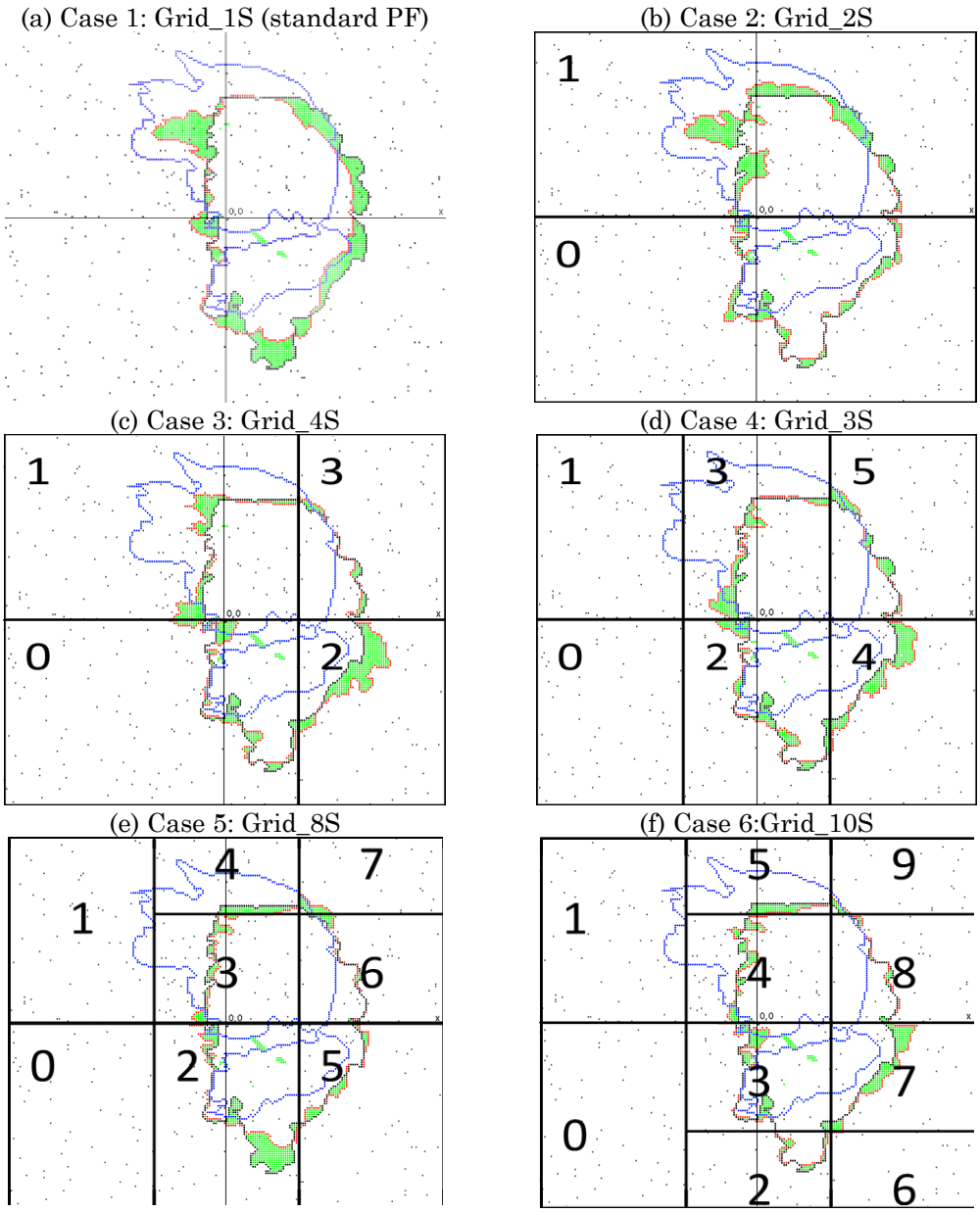


Figure 3.6 Experiment results for all six cases under grid partitioning.

From figure 3.6, we can find that "simulated" fire front deviated a lot from "real" fire front. Also, the filtered fire fronts differed among all six cases and it's shown that prediction

improved while the number of sub-states increased. We compare the "filtered" fire front to "real" fire front for each case one by one.

- In case 1, only a single sub-state is configured, so we can also consider this case as the standard PF. We can see that the "filtered" fire front was more close to the "real" one compared with "simulated" fire front as shown in figure 3.6(a). But due to the limit number of particles the predicted "filtered" fire front was still not precise (i.e. limitation in standard PF) since the symmetric difference of cells is as large as about one quarter of the "real" fire front area.

- In case 2, where the whole space was divided into 2 regions as shown in figure 3.6(b), the differences between "filtered" and "real" fire front was reduced significantly, especially in region "0". However, the "filtered" fire front in region "1" still spread "out" of the real one at most parts. Thus, more refinements are needed to reduce the impact from the northern part of the state to the whole state.

- In case 3, we further divided the whole space into 4 regions in case 3 as shown in figure 3.6(c). We can see that after refining, the northern "filtered" fire front became "better" and was more close to "real" fire front. But in region "2" of case 3, an unexpected fire head toward southeast was generated. This behavior owed to insufficient observations from sensors and data association problem from boundary sensors. 1) With insufficient observations, it is hard to provide a correct weight for a particle. Since region "2" was small, correspondingly the number of sensors distributed in this area was limited. Also, few sensors located nearing the "real" fire front in region "2" and fewer observations are obtained for this area. So, the calculated weight based on few observations cannot correctly represent if the sub-state of a particle is good or not, even if the weight is high.

2) Besides, boundary sensors also affect prediction accuracy when calculating weights for each sub-state. Overall, although some error exists in region "2", the total error was still less than that in case 2.

- Eventually, in case 4 and case 5, the whole space was divided into 6 and 8 regions separately. From figure 3.6(d) and 9(e), we can see that the differences in north part of fire front were decreased eventually. But at the southernmost area, the "filtered" front had a fire head inside of the "real" fire front in region "2" for both these two cases. It may happen because the results are only from one single run simulation. The random noise added in the fire front may not be the same as that in previous cases.

- In case 6, to further better predict that southernmost, the whole space was divided into 10 regions as shown in figure 3.6(f). As expected, the southernmost fire head in case 5 "disappeared" and prediction performance was improved as a result.

To show the quantitative results, we calculated MSEs for all six cases. Figure 10 displays average MSEs after 10 runs for each case, to reduce the impact of random noise added to the system. From figure 10, we can find at final time step T=12, the MSE for case 1 was the largest among all six cases. But this error decreased eventually when the number of divided regions increased, where our spatial partition-based PF was truly applied. At previous time steps, we also note that during the simulations, even if considering a larger number of sub-states, the MSE may be still higher than the case with less number of sub-state. This unexpected result was mainly from the boundary sensors. But once the fire crossed another region, the error was reduced. For example, at T=9, MSE in case 6 (Grid_10S) is higher compared to case 4

(Grid_8S), but at T=10 when the fire spread over the border between region “2” and region “3”, MSE decreased in case 6 and was smaller than that in case 4.

In summary, figure 3.6 and figure 3.7 show that the prediction accuracy has been increased significantly by increasing number of partitions. In figure 3.7, at final time step, about 45% errors (i.e. number of cells with different states compared to real fire front) can be reduced by spatially partitioned method when 10 sub-states are generated, compared to standard PF method when a single sub-state is considered (i.e. no partitions).

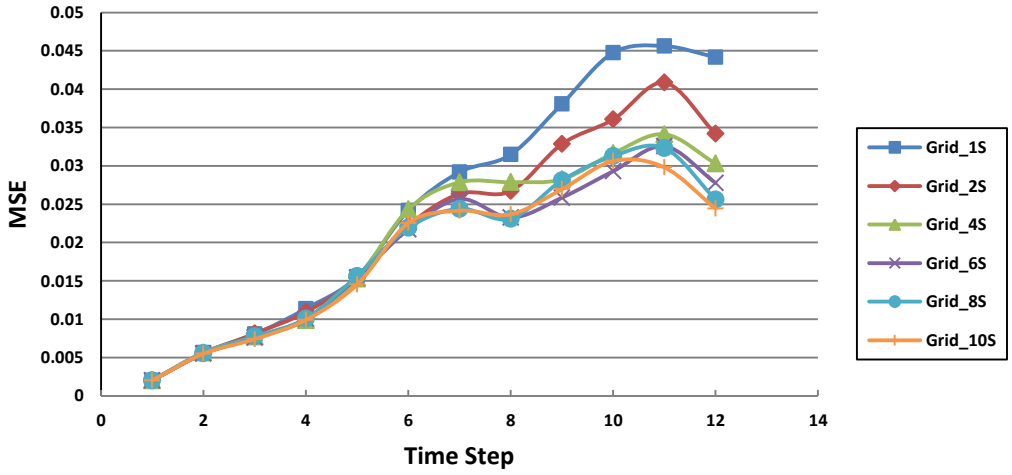


Figure 3.7 Comparisons of MSE among all six cases..

3.7 Conclusion

In this chapter, we propose a spatial partition-based particle filter framework for simulation and prediction in a spatial temporal system. This framework introduces the locality nature of system state. We divide the whole space into several smaller regions and the full state is broken into correspondingly sub-states. Unlike the tradition calculation on a whole system state, it calculates a set of local sub-state weights for each particle and performs resampling in a group of local sub-states. Experiments demonstrate the improvement on prediction accuracy

when space is divided reasonable with some prior knowledge and when partition number is increased in a proper range.

4 AUTOMATED SPATIAL PARTITIONING METHODS

To solve the boundary sensor problem existing in those basic approaches from Chapter 3, we propose a two-level automated partitioning method to provide an optimized balanced partition with less boundary sensors. The first level is a high level partitioning while the second level is a low level partitioning.

4.1 High level partitioning

The purpose of high-level partition is to find m-balanced regions, regarding some partition criteria. A general idea is that we break a full space into k atomic small blocks b_1 to b_k , then assign each block a score s through a score function defined according to those partition criteria, and finally partition the whole space into m balanced regions r_1 to r_m by combining the atomic blocks and maximize the total score.

To evaluate if a partition is good or not for scoring, we can consider several factors, such as sensor coverage rate $R_{coverage}$, boundary sensor number $Num_{bSensor}$ and total sensor number in a region r. The first factor $R_{coverage}$ shows how a region is covered by the observation area. The second factor $Num_{bSensor}$reprents how many boundary sensor exist in a region. The last factor total sensor number Num_{Sensor} tells the total number of sensors in a region including boundary sensors and non-boundary sensors. These three factors work together to estimate a partitioning. A partitioning is good if and only if $R_{coverage}$ is high and boundary sensor rate $R_{bSensor}$ is low which is calculated by a division of $Num_{bSensor}/Num_{Sensor}$. Then a score function Score() is designed as in equation (14). In this equation, c1 and c2 are two coefficients to adjust the importance of coverage rate and boundary sensor rate. However, in most cases where sensors are distributed uniformly in space, we can ignore the factors $R_{coverage}$ and $Num_{Sensor}$and only take care of factor $Num_{bSensor}$. Then another score function is designed as

in equation (15), where c3 is the average of number of boundary sensors calculated by $\mathrm{Num_{Sensor}}/\mathrm{m}$. In this proposal, we use the second score function for experiment and c3 equals to 100.

$$\mathrm{Score(r)} = \left(100 * \mathrm{R_{coverage}} + \mathrm{c1}\right) * \left(100 - 100 * \frac{\mathrm{Num_{bSensor}}}{\mathrm{Num_{Sensor}}} + \mathrm{c2}\right) \qquad (14)$$

$$\mathrm{Score(r)} = \mathrm{c3} - \mathrm{Num_{bSensor}} \qquad (15)$$

The high-level partitioning algorithm consists of two main steps. The first step is an initialization step for generalizing m balanced regions to obtain a coarse partition represented by $\mathrm{P_1} = \{\mathrm{r_1}, \ldots, \mathrm{r_j}, \ldots, \mathrm{r_m}\}$ where $\mathrm{r_j}$ is a set of blocks. And the area for each region approximates to $\mathrm{Area(Sp)}/\mathrm{k}$. The second step is a refinement step for refining the m balanced regions by moving blocks between regions recursively to maximize total score of all regions. The algorithms of these two main steps are described in Algorithm 1 and Algorithm 2 separately.

ALGORITHM 1. Initialization of Partitioning

Input: Two-dimensional area Sp for whole system space, number of total blocks k, sub-state number m, sensor locations, sensor observation range.

Output: A coarse partition represented by $\mathrm{P_1} = \{\mathrm{r_1}, \ldots, \mathrm{r_j}, \ldots, \mathrm{r_m}\}$ where $\mathrm{r_j}$ is a set of blocks

1. Divide the whole space Sp into k equal atomic blocks. SB denotes the current block set containing the blocks without assignment.

$$SB = \{b_j | \; j = 1..k \; and \; b_j \; is \; a \; block \; in \; space \; \}$$

2. Initialize the partitioning as $P_1 = \{r_j | j = 1 \ldots \; k \; and \; r_j \; is \; empty\}$

3. Randomly select m atomic blocks $b_j{}'$ from block set SB, where $j = 1 \ldots m$. Assign each selected atomic block $b_j{}'$ to region r_j.

$$SB = SB - \{b_j{}' | j = 1 \ldots k\}$$

4. Initialize index j as 0, j=0

5. **Repeat**

   5.1 From SB, find blocks neighboring to one of the blocks in the current region r_j. If nothing founded, randomly select one block b from SB and go to step 5.3. Otherwise, record each founded atomic block as $Neighb_n$, where n=*1..*s and s is the total number of founded neighboring atomic blocks.

5.2 For each Neighb_n , calculate the combination score with current region $CombineCurR_{n,j} = score(Neighb_n \cup r_j)$, and with other regions $CombineOther_{n,i} = score(Neighb_n \cup r_i)$ where $i \neq j$ and $i = 1..m$. Select the atomic block b which has minimum $CombineOther_{m,i}$ but then maximum $CombineCurR_{m,i}$.

5.3 $SB = SB - \{b\}$, $r_j = r_j \cup \{b\}$, ***j=(j+1)%m***.

5. Until SB is empty;

In initialization step, at beginning the regions in partition P_1 are empty. Next we randomly select k atomic blocks from the original block set and then assign each region in P_1 one selected atomic block. After that, from the remained block set excluded the k atomic blocks, we iteratively assign one block for each region. The selected block b has the worst combination score with other regions but good combination with current region. However, to maintain the continuous of the region, we start checking the blocks from neighbors. So for each region, we find the neighboring blocks in the remained block set SB and choose the block with minimum combination score with regions but then the maximum combination score with current region. However, if no neighboring blocks available, we randomly select one from set SB. Finally selected block is excluded from the remained block set and combined with current region r_j. This iteration stops when the remained block set is empty. After initialization step, an initial partition is built. But, this partition is not an optimized partition and the regions may be not continuous. So a refinement step is needed to maximize the total combination score and build continuous regions.

ALGORITHM 2. Refinement of Partitioning

Input: A coarse partition $P_1 = \{r_1, \dots, r_j, \dots, r_m\}$, sub-state number m, maximum iteration times T_{max} , maximum total score decreasing times T_{de_max}, sensor locations, sensor observation range.

Output: A refined partition $P_2 = \{r_1, \dots, r_j, \dots, r_m\}$

1. Current iteration times $T_{iter} = 0$,

2. Repeat *while* $T_{iter} < T_{max}$

2.1 Let $T_{iter} = T_{iter} + 1$

2.2 Find all the boundary blocks for each region in partitioning $P_1$and put them into a queue q.

2.3 **Repeat** *while q is not empty*

2.3.1 For each boundary block b_j in q, calculate the total score gain $gain_{i,k}$ by moving it from original region r_i to neighboring region r_k according to equation (16). Then form a pair $< b_j, gain_{i,k} >$ for each possible movement.

2.3.2 Filter the movements that can maintain the balance of partitioning and then among the results select the movement pair $< b_j, gain\_max_{i,k} >$ that has highest movement non-zero gain.

2.3.3 If a certain movement pair is selected, perform the movement for it and remove b_j from queue q. Otherwise, break;

2.3.4 Update each region and sensor information in P_1.

2.3.5 Calculate the total score of all regions. If total score decreases, $T_{decrease} + +$; Otherwise, $T_{decrease} = 0$

2.3.6 If $T_{decrease} = T_{de\_max}$, goto step 3.

2.3 **Done**

2. Done

3. **If** $\boldsymbol{T_{decrease} > 0}$ **,** trace back to previous partition by undoing last $\boldsymbol{T_{decrease}}$ movements.

In refinement step, partition is further refined by moving boundary blocks to neighboring regions. The algorithm contains at most a limited number of iterations, denoted by T_{max}. At each iteration, initially all the boundary atomic blocks are put into a queue. Then for each block in queue, we calculate the gain of total region score after moving the block from current region to neighboring region. Each movement forms a pair $< b_j, gain_{i,k} >$ which means the total score gain for moving block b_j from region original r_i to neighboring region r_k . And $gain_{i,k}$ is computed by following equation.

$$\mathrm{gain}_{i,k} = \left[\mathrm{Score}\left(r_i - \{b_j\}\right) + \mathrm{Score}\left(r_k + \{b_j\}\right)\right] + \left[\mathrm{Score}(r_i) + \mathrm{Score}(r_k)\right] \quad (16)$$

After that, we filter the movements which can still maintain a relative balanced partition and then choose the block b_j which has the highest movement gain $\mathrm{gain\_max}_{i,k}$. The relative balance partition means that after moving blocks the area of both two regions do not need to strictly the same as the average area of regions $\mathrm{Area(Sp)}/k$. This can make the movement more flexible and increase the possibility to find an optimal partition. However, the area should be still constrained in a range defined in the equation below

$$c1 * \left(\frac{\mathrm{Area(Sp)}}{m}\right) \leq \mathrm{Area}(r) \leq c2 * \left(\frac{\mathrm{Area(Sp)}}{m}\right) \quad (17)$$

where $c1$ and $c2$ are two coefficients predefined by user. Then for the selected block b_j, we move it to region r_k to achieve the max gain denoted by $\mathrm{gain\_max}_{i,k}$ and simultaneously remove it from queue q. Next, both the regions and sensor's belonging information will be updated while the total score will be re-calculated accordingly. After that, we count how many times the total score decreases continuously, if it reaches T_{de_max} we assume the total score cannot be increased anymore and the optimal partitioning has already been found. Otherwise, the next block will be chosen from updated queue q. Before the termination of algorithm once q is empty or no block b_j can be selected, q will be updated by rescanning the boundary blocks for all regions and a new iteration begins until iteration time reaches T_{max}. At the end of algorithm, the partition should be traced back to the status with maximal total score so far. Since this algorithm uses T_{de_max} to record the number of steps the total score reduces continuously at the last T_{de_max} movements, we need to undo these movements to recover the partition to that status. Later, in the experiment, c1 is set as 0.8, c2 is set as 1.2, T_{max} is configured as 10 and T_{de_max} is configured as 8.

4.2 Low level partitioning

The goal of low-level partitioning is to further reduce the boundary sensor rate by merging or splitting sensor's observation area based on a predefined partitioned space. The predefined partition can be generated either through those basic partitioning methods or the proposed high-level automated partition algorithm in previous section. From the previous section, we know that in order to decide whether a sensor is a boundary sensor or not we have to check if its observation area covers multiple regions. So through modifying one region's area we can convert a boundary sensor to a normal sensor and eventually decrease the boundary sensor rate of the region. The modification is based on the observation area. Two operations --- merging and excluding are considered for modification.

However, those two operations will change the boundary sensor rate in different situations. In general, four cases will happen as shown in figure 5 according to the relationship between a boundary sensor and non-boundary sensor. In figure 5, initially a rectangle space is divided into two equal regions R1 (left side) and R2 (right side). Sensor A, B and C are distributed over space. The circle denotes the observation area of sensor. Row (1) shows the coverage of sensors in four cases. Row (2) shows the corresponding final partition after performing merging or excluding operation using low-level partitioning algorithm for each case. And the red line denotes the border of two regions. In all four cases, sensor A is a boundary sensor located in region R1 initially. Our goal in the low level partitioning algorithm is to merge or exclude A's observation area for region R1 so that its boundary sensor number can be reduced. Thereby for each case, we have found a proper operation to achieve this goal. For case 1, initially sensor A has no overlapped observation area with non-boundary sensor. After merging operation, number of boundary sensor for region R1 is decreased by 1. For case 2, at

beginning, sensor A's has overlapped observation area with non-boundary sensor B from the same region R1. After merging operation, number of boundary sensor for region R1 can also be decreased by 1. For case 3, sensor A has overlapped observation area with non-boundary sensor B from the other region R2. After excluding operation, number of boundary sensor in region 1 is still decreased by 1. However, for case 4, sensor A has overlapped observation area with non-boundary sensors from both regions. After merging operation, although number of boundary sensor for region R1 has been decreased by 1, for region R2 it is also increased by 1 unlike the results in other cases where the number of boundary sensor in region R2 keeps unchanged.

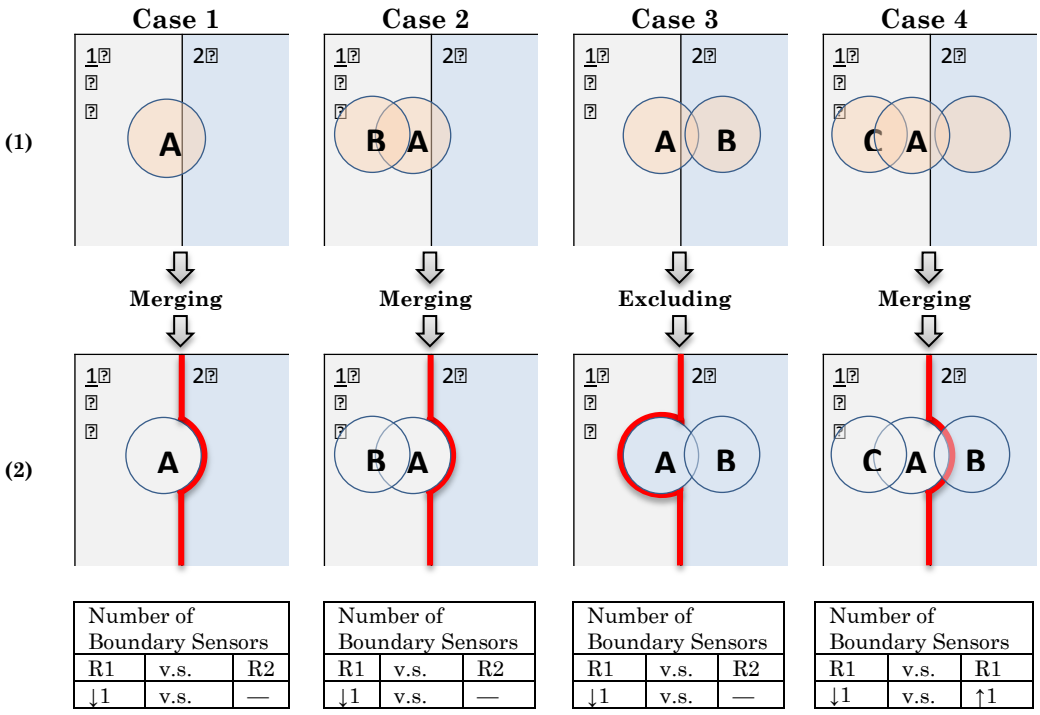


Figure 4.1 The results after operations for merging or excluding observation area in general four cases.

In figure 4.1, we only consider the observation area overlapping relationships between boundary sensors and non-boundary sensors. This is due to the reason that after modifications on regions the property of the non-boundary sensors with overlapped observation area may be

modified accordingly. Although after merging or excluding operations, the original boundary sensor will definitely be converted to a non-boundary sensor, simultaneously it also affects the non-boundary sensors' coverage situation and may lead to those non-boundary sensors change to boundary sensors. We do not expect this effect. So we have to find a proper operation in each case as shown in figure 4.1. However, if another sensor we considered is a boundary sensor, then no matter what operation is performed this boundary sensor with overlapped observation area cannot be converted to a non-boundary sensor in most cases unless multiple operations are carried out. So the result is the same to that in case 1 of figure above figures. And we could categorize this situation as case 1. But, in some special cases, the related boundary sensor can be converted to a non-boundary sensor only with single operation. For example, when original sensor's observation area overlaps with that of another boundary sensor symmetrically distributed in another region, both the boundary sensors can be converted to non-boundary by splitting or merging and thus the boundary sensor number of each covered region is reduced.

Based on the clarifying on operations done in different cases, an algorithm to adjust the boundary sensor rate for each region is designed as in Algorithm 3. In this algorithm, a threshold $\mathrm{Th}_{\mathrm{boundary}}$ for boundary sensor rate is predefined. However, this threshold cannot be set as 0. This is because of the overlapping of observation areas; so performing the two operations based on each sensor cannot guarantee all regions' boundary sensor rate decrease to 0 unless only one region remained in the space and no overlapped observation area. After configuration for boundary sensor rate threshold, for each region, the merging or excluding operation is done based on each boundary sensor iteratively until the boundary sensor rate reduces to the limit or boundary sensors are all checked. The decision to make merging or excluding operation depends on the total number of boundary sensor affected. One of the operations can be performed only

when the total number of boundary sensor is decreased as a result according to the first three cases in previous figure. Otherwise, as in case 4, the operation merging will be performed if the neighboring regions have a boundary sensor rate lower than threshold $\mathrm{Th}_{\mathrm{boundary}}$. In this proposal, we set $\mathrm{Th}_{\mathrm{boundary}}$ as 0.03 for experiments.

ALGORITHM 3. Low-level Partitioning

Input: A coarse partition $\mathrm{P} = \{\mathrm{r}_1, \ldots, \mathrm{r}_\mathrm{j}, \ldots, \mathrm{r}_\mathrm{m}\}$, sub-state number m, boundary sensor rate threshold $\mathrm{Th}_{\mathrm{boundary}}$, sensor locations, sensor observation range.

Output: A refined partition $P' = \{\mathrm{r}_1, \ldots, \mathrm{r}_\mathrm{j}, \ldots, \mathrm{r}_\mathrm{m}\}$

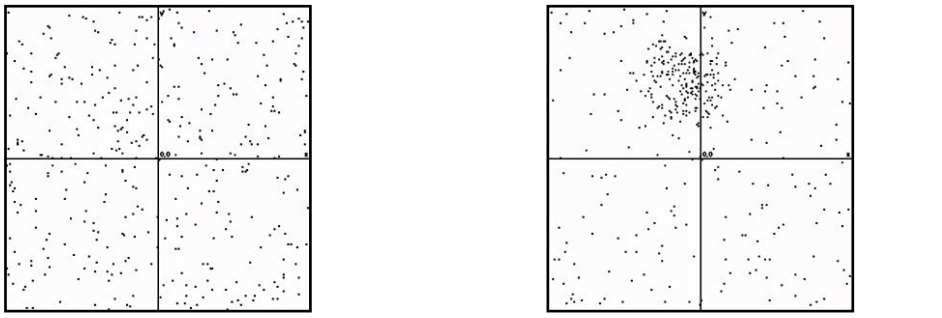
1. Initialization $j = 0$
2. **Repeat**
   - 2.1 **Repeat** while boundary sensor rate in region r_j $> \mathrm{Th}_{\mathrm{boundary}}$ or boundary sensors are all checked
     - 2.1.1 Select one new boundary sensor $\mathrm{Sensor}_\mathrm{b}$ located in region r_j
     - 2.2.2 Check $\mathrm{Sensor}_\mathrm{b}$'s overlapped observation area with other non-boundary sensors
     - 2.2.3 According to the first three cases in figure 4, exclude or merge the observation area of $\mathrm{Sensor}_\mathrm{b}$ if total number of boundary sensors in space is decreased and then go to step 2.2.5. Otherwise, continue on step 2.2.4.
     - 2.2.4 Merge the observation area of $\mathrm{Sensor}_\mathrm{b}$ if neighboring related region has a lower boundary sensor rate than threshold $\mathrm{Th}_{\mathrm{boundary}}$.
     - 2.2.5 Update the observation coverage status for all sensors in the space and also sensor's property information.
   - 2.1 **Done**
   - j=j+1;
- 1.2 **Done**

4.3 Experiment

4.3.1 Experiment settings

This Chapter shares almost the same experiment settings with Chapter 3. However, the difference is that, we consider two distributions for sensors. The first distribution Distribution-1 is also in Chapter 3, in which sensors are distributed uniformly. The second distribution Distribution-2 is a "mixed" distribution in which first two hundred sensors are distributed

randomly and uniformly over the space and the other two hundred sensors are distributed around point (-10,50) through a Gaussian distribution (0,15) on x and y separately.



Distribution-1
Uniform distribution

Distribution-2
"Mixed" distribution (Gaussian & Uniform)

Figure 4.2 Two distributions for sensors.

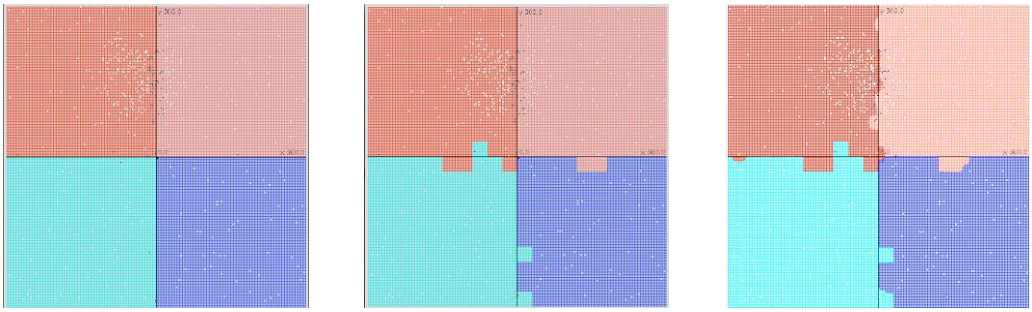
Automated partitioning methods are tested to explore the impact from boundary sensor and benefits on increasing partition number even without prior fire spreading information. In this strategy, we use Distribtuion-2 for testing boundary sensor problem and Distribution-1 for investigating the benefits by increasing number of sub-states. The unbalance distribution for sensors is employed to test the boundary sensor impact; the uniform distribution for sensors is set to explore the impact of partition number. Similar to Chapter 3, 50 particles are used to obtain the filtered results.

4.3.2 Experiment results and analysis

If without prior knowledge on how state evolves over space, automated partitioning methods can be applied to get an optimized division with less boundary sensors. In this section, we performed two tests by applying automated partition algorithm. Since the goal of automated partitioning is to reduce boundary sensor number, in first test, we compared the prediction result from equally grid partition (with large amount of boundary sensor) with that from automated partition. The automated partition skipped the high-level initialization algorithm and use original

grid partition as the initiation for high-level refinement and low-level partition algorithm. Then, in second test, multiple partitions with different numbers of sub-states were generated by a complement automated partitioning algorithm (including all three algorithms described in Chapter 4.1 and 4.2), and prediction results were compared among them.

In the first test, the initial partition is configured as a grid partition where whole space is divided into four balanced regions. This partition is named as "N4" shown in figure 4.3(a). Based on sensor distribution Distribution-2, auto partition algorithm then refined this initial partition into two partitions. We call the first partition as "R1" shown in figure 4.3(b), which was generated only by high-level refinement partition algorithm. The second partition was named as "R2" shown in figure 4.3(c), which was the result after applying low-level automated partition algorithm, based on R1. In each partition in figure, the whole space is divided into four regions denotes by four different colors. Dots represent the location of boundary sensors. Black dots denotes boundary sensor, while white dots denotes non-boundary sensors. In figure 4.3(b), since high-level refinement automated partition algorithm assigned some blocks to neighboring regions, the total boundary sensor number is reduced from 59 to 55. However, because densities of boundary sensors near border between red and pink region was high and similar in these two regions, high-level refinement automated partition algorithm failed to optimize the division around this area. Therefore, high-level automated partition did not make a great change on the total number of boundary sensor. Next after low-level partition algorithm, R1 is further optimized mainly by merging or splitting observation area nearing border between red and pink region. And we can find that in R2 boundary sensor number is dropped from 55 to 34, which outperformed high-level refinement automated partition algorithm.



(a) N4
Equally grid partition

(b) R1
High-level refinement automated partition

(c) R2
Low-level automated partition

Figure 4.3 Partitions after using three different partitioning methods.

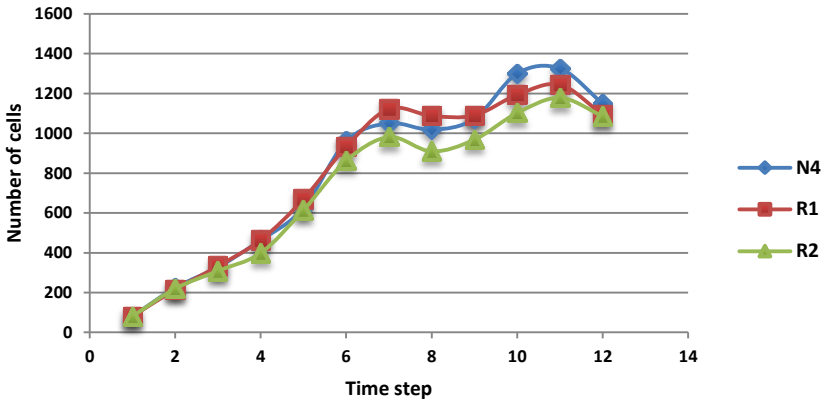


Figure 4.4 Comparisons of symmetric differences among three partitions in first test involving automated partition algorithm.

Figure 4.4 compares the results of symmetric differences of filtered fire to real fire front among the three partitions based on 10 runs. We can see that R2 has the lowest symmetric differences through all time steps because of smallest boundary sensor number. However, since a new partition may just modify the region area instead of make a significant change on boundary sensor number, the prediction result may generate unexpected behavior. Take R1 as an example, for pink region it only merged one block from blue region and did not reduce a lot of boundary sensors in it. So as shown in figure 4.4, when time step T equals to 7 and 8, since "real" fire did not across the blue region then, R1 even generated higher symmetric differences compared with grid partition. As fire eventually spread across the three borders and impact from boundary

sensors was decreased over time. So at final time step, although grid partition has the highest symmetric difference, the result was very close to that from R2 and R1.

In second test, we increased sub-state number from 2 to 121 and used the uniform distribution of sensors from Distribution-1. Since the block size was as large as 10*10 and cell space size is 200*200, there were totally 200 blocks in space. The maximum sub-state number was 200. When sub-state number equaled to 121, each sub-state may contain only 1 or 2 blocks. So there was no need to further increase sub-state number and make regions smaller than 1 block. The number of sub-states was selected from set {2, 4, 8, 9, 16, 25, 32, 49, 64, 81, 100, 121}. Each number was for one case. And for each case, we generate 20 partitions using only high-level automated partition algorithm and 20 partitions using a complete two-level partition algorithm. We should note that in this test high-level automated partition algorithm applied greedy algorithm to build an initial partition and then refined. Since the greedy algorithm needs randomly generated seeds, the 20 partitions in each case were different from each other.

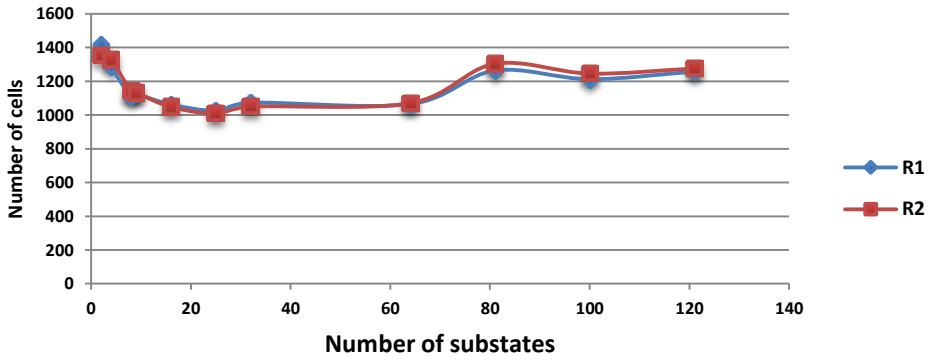


Figure 4.5 Comparisons of symmetric differences at last time step for different partition numbers in second test involving complete automated partition algorithm.

Figure 4.5 shows the average symmetric difference of 20 partitions for R1 and R2 at last time step when sub-state number is increasing from 2 to 121. 1) For both R1 and R2, they have

the same behavior of symmetric differences. We can see that initially when sub-state number was increasing, the symmetric difference deceased. But after a certain point, it stopped decreasing and became stable. This behavior maintained until sub-state number reached another point and then symmetric difference eventually arrived next summit. Finally, it changes within a small range. Although, the symmetric difference increased during some interval, the final symmetric difference is still less than that in the cases with few sub-state numbers. And we can still find a point with lowest symmetric difference, which is around sub-state number 25 in this test. Therefore, this figure demonstrates that prediction result improves when sub-state number increases, but will eventually become worse when sub-state number is over a certain value and stay relative stable finally. 2) Also because this test is based on uniform distributed sensors, there is no great difference on sensor densities across the whole space. Since the impact from boundary sensor is less, the symmetric differences between R1 and R2 is quite close.

4.4 Conclusion

In this part, automated partitioning methods are proposed to reduce the total boundary sensor number. Apparent prediction accuracy improvements are found in experiments after performing the automated partitioning method. However, data association problem may still exist even after performing automated partitioning method.

5 DATA ASSIMILATION WITH HARD/SOFT DATA

5.1 Introduction

In dynamic data driven simulation (DDDS), observations are assimilated into a real system or a computer model to provide better estimates than can be obtained by only the data or the model. The idea of assimilating observation data into running simulation models has found application in many problems, including ocean forecasting, weather forecast, oil well placement, transportation system, and wildfire simulation[71,75]. In previous work [1], we developed dynamic data driven simulation that assimilates ground temperature sensor data into a wildfire spread simulation model called DEVS-FIRE for improving simulation results. The ground temperature data assimilated in that work is an example of hard data, which refers to quantified observations and measurements from physics-based sources [76]. The physics-based sources mostly mean the physical-based sensor, such as radar, EO/IR (i.e., visible/thermal infrared) cameras, and ground temperature sensors.

Besides hard data, another type of data, referred to as soft data, can also provide valuable information for the system under study. Different from hard data, soft data refers to observation from human-based sources [76], such as human reports, intercepted text and audio communications and other open sources such as website, newspaper and TV broadcast. Compared with the quantified hard data, soft data are qualitative, fuzzy, unstructured, and often subject to interpretation. For example, to describe temperature in a room, a temperature sensor may provide a quantified data like 80F, while a person may report the temperature as “hot”, which could mean different temperatures according to the variant views by different persons. Another difference between hard data and soft data is on the representation format; the former is usually represented in numeric format (e.g., numbers), while the latter is usually represented in

linguistic format (e.g., natural language). Despite its differences from hard data, soft data can provide information complement to that from hard data. Considering the wildfire example, the observations in wildfire simulation is not just limited to the temperature data, soft data from human reports such as fire spread direction and speed can also be combined to improve the accuracy of simulation result. This information is difficult to be measured directly by hard data but can be obtained from human reports (if available).

In this chapter, we consider soft data in dynamic data driven simulation and develop a method that combines both soft data and hard data in data assimilation for improving simulation results. We apply the hard/soft data assimilation to the application of wildfire simulation. Note that even though we base our work on the wildfire application in this proposal, the data assimilation method can be adapted to other spatial temporal systems. The remainder of this chapter is organized as follows. Section 5.2 introduces the related work of soft/hard data assimilation both in the application and methodology. And then Section 5.2 presents the basic particle filter framework and the soft/hard data assimilation method. To validate the method, Section 5.4 shows the experimental studies. Finally, Section 5.4 draws the conclusion.

5.2 Soft/Hard Data Assimilation Using Particle Filter In Wildfire Simulation

5.2.1 PFs-based Soft/Hard data Assimilation Framework

In soft/hard data assimilation, the sources of real observations derive from hard or soft data. Considering the differences between these two types of data, the basic framework should be extended appropriately so that soft and hard data can be incorporated to improve prediction accuracy. The main extension is dividing measurement function into two parts according to the data type it measures. Figure 5.1 shows the extended framework based on the application of wildfire data assimilation. As shown in the figure, the samples are represented by Fire_{t-1}, which

are the fire front samples at last time step t-1. $Fire_{t-1}$ evolves to new fire front samples $Fire_t$ for the current time step through the system transition model (i.e. the DEVS-FIRE simulation model). Then $Fire_t$ are measured by the soft and data measurement functions respectively. After that, the simulated observations are compared with real observations, which include both hard data (such as temperature information form temperature sensors) and soft data (such as fire spread situation reported by humans). Then based on the divergence, the weight for each sample is updated. Finally, samples are processed in the resampling step to avoid degeneration which is the same as in the basic framework.

While the extended framework largely follows the same steps as in the basic framework, the soft observation data is significantly different from the typical hard data to be assimilated. As soft data is qualitative, the representation is more complex than that of hard data. Accordingly, the way to compute weight is also different. Below we present how soft data is represented in our framework and how it influences the weight computation [77].

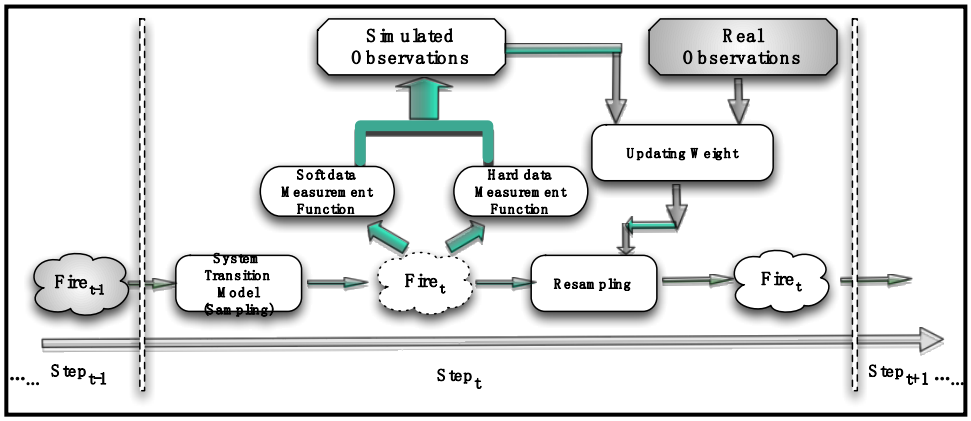


Figure 5.1 PF-based Soft/Hard data assimilation framework in wildfire simulation

5.2.2 Soft data representation

Soft data comes from reports provided by human observers. In the real world, the soft data may be newspaper, text message or professional reports that are written in natural language[78]. Before representing soft data in a structured format that can be assimilated, we need to extract important information (e.g., key words) from the soft data. Information extraction from unstructured natural language descriptions is not the focus of our work. In this proposal, we assume that information has already been extracted and focus on how to represent the soft data in a structured way for carrying out the data assimilation. In general, we propose that the soft data has a structured format as below:

Soft data = {Report}

Report = < Parameter, Time, Observer_location, Observer_range, Description>

Description = < Reference_location, Value, Qualifier>

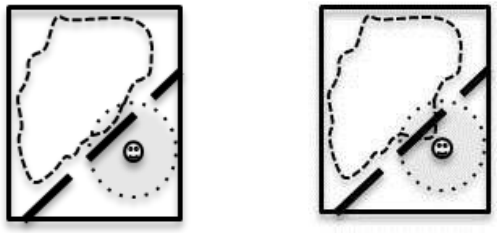
Soft data is a collection of "Report". Each "Report" contains five elements as shown above. The first element "Parameter" refers to the feature of the report that determines the "Report" type. The second element "Time" refers to the time when the soft data is reported. The "Observer_location" element is the human observer's geography location at the time of report, which can be measured by the observer's GPS device. "Observer_range" is the spatial range within which the reported data is meaningful. This is similar to the detection range of hard sensors such as ground temperature sensor. The last element "Description" is the most important element that refers to the observation description of specified "Parameter".

Among the five elements, “Observer_location” and “Observer_range” together specify a 2D observation area. The range of human observation is affected by many factors, such as geography, weather and individual vision. However, in this proposal, the range is simplified as a circle around “Observer_location”, where radius is the “Observer_range” in a report.

The two other elements “Parameter” and “Description” describe a specific observation within the corresponding observation area. The “Parameter” describes the type of the observation, such as temperature, distance or speed. Each “Parameter” corresponds to a formatted “Description” which consists of three elements: Reference_location, Value, and Qualifier. The Value provides the content information about the “Parameter”. Unlike hard data described by numbers, soft data are usually expressed with linguistic words. Therefore, the parameter values are words rather than numbers. In our work, we limit the word selection within a corresponding finite set (see examples later). The next element “qualifier” constrains the “value” and shows the incertitude degree of “value”. Same as the “value”, “qualifier” must be chosen from a finite set (e.g. {certainly, almost, slightly, perhaps} [79]). For the element “Reference_location” in the report, it is defaulted as the “observation_location” that is the location of the human observer. Nevertheless, reference location can be different from the observation location. To help limit the location of the observed object, we introduce the concept of landmarks.

The “Reference_location” could be one or more landmarks. Landmarks are similar to the real world landmarks, which generally represent interested or well-known places, such as a highway, a river, a building and so on. For instance, both cases (A) and (B) in figure 5.2 show “fire is at northwest side and close to the observer”, where the short dash line represents the fire front, the smile face is the observer and the dot line circle describes the observation area. The

long dash line denotes a highway near the observer, which can be used as a landmark in this situation. With this landmark, the soft data “fire is at the west of highway” distinguish case (A) from case (B) because in case (B) fire already passed highway leading to fire at both west and east sides of the highway.



(A) (B)

Figure 5.2 Observations with landmark

Following the above structured format, in this proposal we represent several types of soft data for data assimilation in wildfire simulation. We are interested in information about the spreading fire front, and thus divide the “Parameters” for fire front into two categories. One is fire spread information and the other is fire location information. Fire spread observation is about the moving direction and speed of the observed fire. It uses the observer’s location as the reference point. Fire location observation is about the location (location distance and location direction) of the fire. To locate a fire, it can refer to multiple reference locations, predefined by landmark or defaulted as the observer’s location.

More specifically, we have four types of soft data: fire spread speed, fire spread direction, fire location distance, and fire location direction. Each of them is described by a “Parameter”. The four finite sets designed for each parameter are as follows:

FSS (Fire Spread Speed) Set: {fast, slow, normal}

FSDR (Fire Spread Direction) Set: {E, N, W, S, NE, NW, SE, SW}

FLD (Fire Location Distance) Set: {far, close, covered}

FLDR (Fire Location Direction) Set: {E, N, W, S, NE, NW, SE, SW}

For FLD, a reference location may be covered by fire; therefore, we use "covered" to illustrate this situation.

Similarly, "Qualifier" is also selected from a finite set:

Qualifier Set: {very, seems, maybe}

Figure 5.3 shows an example of soft data reported by an observer. The reports are in the bottom box. The yellow smile figure represents the observer; the gray circle shows the observation area for this observer; the red line is the current fire front line while dash light red line is the fire front 30 minutes ago. In this example, the observer provided four formatted reports at 12:30 pm on 5/2/2013, which describes that the fire locates at the very northwest side and maybe far away, and the fire spreads very slowly and seems on the east side of Highway I-75.

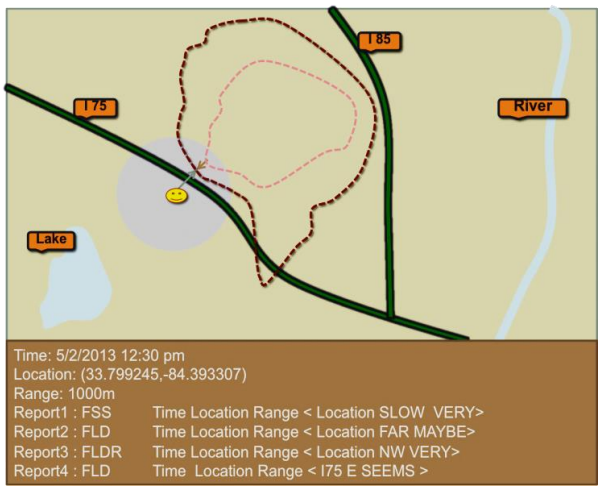


Figure 5.3 An example of soft data and corresponding fire shape

5.2.3 Measurement Function

We note that even though the real soft data is described in linguistic format, the measurement function for soft data still output quantitative values. Also note that when soft data include multiple reports, each of which corresponds to a particular parameter, for each report a measurement function needs to be defined.

For the wildfire application considered in this work, there are four parameters: Fire spread direction, fire spread speed, fire location direction, fire location distance. Then four measurement functions are defined respectively as $\mathrm{MF_{fsdr}}$,$\mathrm{MF_{fss}}$, $\mathrm{MF_{fldr}}$, and $\mathrm{MF_{fld}}$.

$\mathrm{MF_{fsdr}} = \mathrm{Direct}(\mathrm{CP_{Fire_t}}, \mathrm{CP_{Fire_{t+1}}})$ Where $\mathrm{Direct}\,(x, y)$ is a function calculating the direction from point x to point y. $\mathrm{CP_{Fire_t}}$, $\mathrm{CP_{Fire_{t+1}}}$ are the fire shape center points of fire fronts within observation range for current and next time step respectively.

$\mathrm{MF_{fss}} = \mathrm{Dist}(\mathrm{CP_{Fire_t}}, \mathrm{CP_{Fire_{t+1}}})/\mathrm{T}$ In which $\mathrm{Dist}\,(x, y)$ calculates the distance between point x to point y. Time unit T equals to one-time slot.

$\mathrm{MF_{fldr}} = \mathrm{LocDirect}\,(\mathrm{RefLoc}, \mathrm{Fire_t})$ In which LocDirect is a function calculating the direction from RefLoc to observed fire front at time step t. RefLoc denotes observer's location or the landmark composed by a series of points.

$\mathrm{MF_{fld}} = \mathrm{LocDist}\,(\mathrm{RefLoc}, \mathrm{Fire_t})$ In which LocDist calculate the shortest distance between RefLoc and observed $\mathrm{Fire_t}$.

5.2.4 Weight updating and data assimilation

At weight updating step, each sample is assigned a new weight according to the discrepancy between simulated observation and real observation. The process is modeled by a multivariate normal distribution:

$$\mathrm{P}\left(\mathrm{y}\middle|\mathrm{x_i(k)}\right) =$$

$$\frac{1}{(2\pi)^{\mathrm{n}/2}|\sum|^{1/2}} \exp[-\frac{1}{2}(\mathrm{MF}-\mathrm{Y})^{\mathrm{T}}\sum{}^{-1}(\mathrm{MF}-\mathrm{Y})] \quad (18)$$

Where $\mathrm{y} = [\mathrm{y_1}, \ldots, \mathrm{y_n}]^{\mathrm{T}}$ are real observations come from hard sensor readings and soft data reports, $\mathrm{MF} = [\mathrm{MF_1}, \ldots, \mathrm{MF_n}]^{\mathrm{T}}$, $\mathrm{Y} = [\mathrm{Y_1}, \ldots, \mathrm{Y_n}]^{\mathrm{T}}$. $\mathrm{Y_i}$ and $\mathrm{y_i}$ are real and simulated ith atomic observation respectively, $\mathrm{MF_i}$ is the measurement function to calculate $\mathrm{y_i}$. $\mathrm{Y_i}$ is a numeric number transformed from the real linguistic report, as each attribute value in the finite set should be mapped to a quantified number. $\sum$ is a covariance matrix. Assuming all the observations are independent $\sum$ is simplified to a diagonal matrix.

$$\sum = \begin{bmatrix} \delta_1{}^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \delta_n{}^2 \end{bmatrix} \quad (19)$$

The element δ_i^2 in the diagonal matrix shows the confidence of real atomic observation Y_i. If the qualifier shows higher certainty, δ_i^2 is higher because its confidence level is higher (i.e. qualifier has a lower uncertainty).

For soft data, each report is an atomic observation. For hard data, the atomic observation is the reading from one physical sensor. The value of sigma for hard data depends on the accuracy of the sensor. For a specific report in soft data, the qualifier's confidence of the report decides the value of δ_i^2 for the report. To calculate δ_i^2 in our work, assuming there are k qualifiers for a specific report and they are ordered decreasingly as $q_1, q_2, \ldots q_j, \ldots q_k$ according to their confidence level j, we assign δ_i^2 relates to the qualifier q_j as below:

$$\delta^2_{i,q_j} = [1 - (\frac{1}{2})^j]\frac{R_i}{n_i} \tag{20}$$

Where R_i is the range of element's quantified value in the finite set i, while n_i is the number of elements in the finite set i. For example, if three qualifiers are used to express the confidence for parameter' value, each of them should be assigned a sigma value as follows:

$$\delta^2_{i,very} = [1 - (\frac{1}{2})^1]\frac{R_i}{n_i} \tag{21}$$

$$\delta^2_{i,seems} = [1 - (\frac{1}{2})^2]\frac{R_i}{n_i} \tag{22}$$

$$\delta^2_{i,maybe} = [1 - (\frac{1}{2})^3]\frac{R_i}{n_i} \tag{23}$$

Figure5 shows ϭ varies according to different qualifier when constrains the direction, which ranges from -180 degrees to 180 degrees and has 8 elements in its finite set (i.e. R=360 and n=8). Compared with “Very”, “Maybe” has a wider range of x with a weight greater than 0.

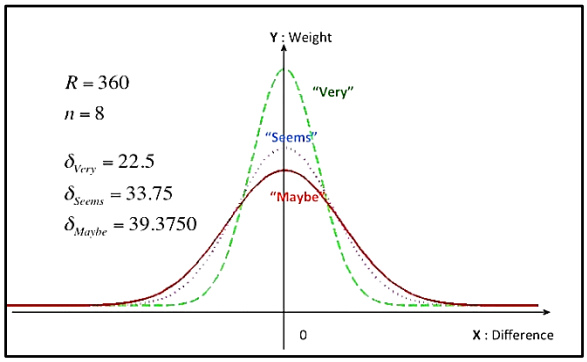


Figure 5.4 Normal distribution for the direction with different quantifiers

As can be seen, in our work each soft data report is treated an observation just as a hard sensor reading does and plays the same role in influencing the importance weights of particles. Nevertheless, the processing of soft data (e.g., data representation, measurement function, and sigma used in computing weight) is different from that of hard data as described above. By using this multivariate normal distribution, we can not only combine the parameters from different soft data reports, but also incorporate the observations for both soft and hard data in an effective manner.

5.3 Experiment

5.3.1 Experiment settings

As it is hard to get the observation and fire front from the real environment, we use identical-twin experiment, which is widely used in data assimilation research, to evaluate the soft/hard data assimilation method. Initially, a simulation (purely DEVS_FIRE) is run and the corresponding data are recorded. The corresponding data are considered as the real observation;

the corresponding simulation result is considered as the real fire. Then based on some "erroneous" input data, we can run another simulation (purely DEVS_FIRE) and the result represents the simulated fire. After that, experiments with enhanced data assimilation simulation in different cases will be executed, and the results are considered as filtered fire.

The simulation to generate real fire is based on the correct real weather information, in which the real wind speed and direction are 8(mph) and 180(degrees) with random variances added every 10 minutes. The simulation to generate simulated fire and filtered fire uses the error weather information, in which wind speed is randomly generated, based on 6(mph) with variances added in the range of -2 to 2 (mph), and wind direction has no errors.

To validate our method, four cases are designed to show the improvement by incorporating soft data. In case1, temperature sensor is deployed every 10 cells, and the number of temperature sensor is sufficient for the simulation to provide relatively good result. In case 2, several observers are added over the map and provide reports at some time. However, in case 3, only part of the map is distributed with sensors. In case 4, based on the partially distributed sensors in case 3, several observers are added. The simulated results for these four cases are all filtered fires. We use 50 particles in these four experiments. Each experiment has 8 time steps; the duration for each step is 20 minutes.

From the simulated "real fire", for experiment purpose we assume in each time step there are soft data available (besides the hard data) for data assimilation. These soft data are listed in Table 5.1 below：

Table 5.1: Real Observations of soft data from real fire

T	ObsLoc		R	Parameter	RefLoc	Value	Qualifier
1	5	20	10	FLDR	null	SW	SEEMS
1	5	20	10	FLD	null	FAR	VERY
2	5	20	10	FLDR	null	S	VERY
2	5	20	10	FLD	null	CLOSE	SEEMS
2	5	20	10	FLDR	null	NE	VERY
2	5	20	10	FSS	null	SLOW	VERY
3	-30	15	10	FLD	null	FAR	VERY
3	-30	15	10	FLDR	A	E	SEEMS
3	-30	15	10	FLD	A	FAR	VERY
3	5	20	10	FLD	null	CLOSE	VERY
3	5	20	10	FLDR	null	E	VERY
3	5	20	10	FSS	null	SLOW	VERY
3	15	5	10	FLDR	null	W	VERY
3	15	5	10	FLD	null	FAR	SEEMS
4	-30	15	10	FLDR	A	E	SEEMS
4	-30	15	10	FLD	A	CLOSE	SEEMS
4	-30	15	10	FLDR	null	W	VERY
4	-30	15	10	FSS	null	SLOW	VERY
5	5	20	10	FLDR	null	SW	MAYBE
5	5	20	10	FLD	null	CLOSE	VERY
6	-10	30	10	FLD	null	FAR	MAYBE
6	-15	40	10	FLD	null	FAR	SEEMS
6	-30	15	10	FLDR	A	NE	SEEMS
6	-30	15	10	FLD	A	COVERE-D	VERY
6	15	5	10	FLDR	null	W	VERY
6	15	5	10	FLD	null	FAR	SEEMS
6	15	5	10	FLDR	null	E	VERY
6	15	5	10	FSS	null	SLOW	VERY
7	-10	30	10	FLDR	null	W	VERY
7	-10	30	10	FSS	null	NORM-AL	SEEMS
7	-15	40	10	FLDR	null	NE	VERY
7	-15	40	10	FSS	null	FAST	VERY
7	5	20	10	FLD	null	CLOSE	VERY

In Table5.1, at each time step, there are multiple reports related to different “Parameters”. The ranges “R” for all observations are set as 10. And only one landmark named as “A” exists in the map. The “null” in column “RefLoc” means that the observation description considers the observation location as the reference location. For instance, when “T” equals to 6, a person locates at coordinate (-30, 15) provides a report for fire direction with a description: “Fire locates at seems NE side of reference location landmark A”.

5.3.2 Experiment result and analysis

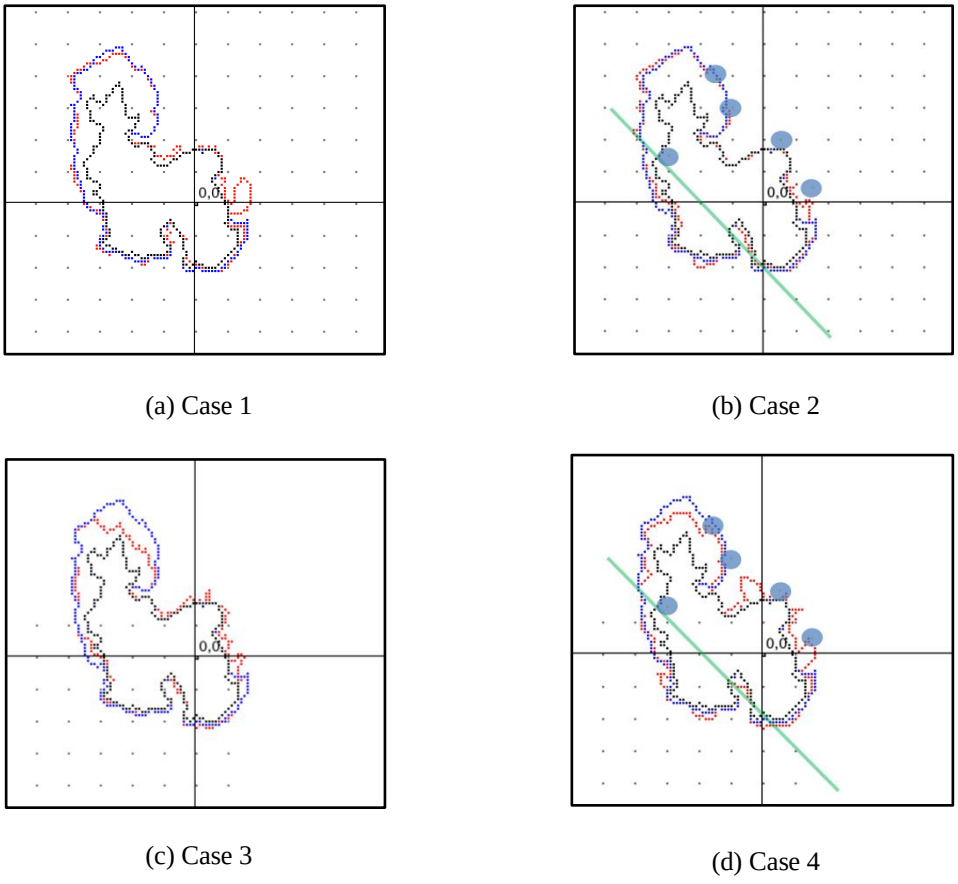
After incorporating fire reports into the four cases, we compare the results shown from Figure6 to Figure9. Each figure shows a part of the map, while the complete map has a size of 200*200 with origin located at the center. Gray dot is the location of temperature sensor; Blue circle is the location of human observer. Green straight line is the landmark "A". Blue line represents real fire front, black line shows the simulated fire front with imprecise weather data, and red line represents filtered fire front in current case.

As shown in these four figures, in all four cases the filtered fire front is much closer to the real fire front than the simulated one, no matter whether soft data is assimilated or not. This indicates that our fundamental PF-based Data Assimilation framework has improved the prediction accuracy.

In order to study the influence of soft data on prediction accuracy, we first compare the results in case 1 and case 2 in which hard data are distributed in whole map as shown in figure 5.5 (a) and (b) . In case 1, the filtered fire front has slightly difference from the real fire front, but around the center of the map, there is an obvious fire head out of the real front bound. However, in case 2, this error has been reduced because of the useful reports about how fire spreads and the location of fire from nearby observer's. For example, a report at time step 6 contains information that "fire spread very slowly to very east side", showing a correct behavior of real fire. Therefore, at final time step the sample fire front satisfying this information is assigned a higher weight and ultimately the fire front that moves slowly to the very east is chosen as final filtered fire front.

Moreover, we compare case 3 and case 4 where hard data does not cover the whole map as shown in figure 5.5 (c) and (d). Case 3 shows that at the northwest part of this map where

hard data are not distributed, there is a great gap between the real fire front and filtered fire front. But in case 4, the reports from northwest part provide useful information and improve the fire front prediction accuracy at that part. However, at the northeast part which is another observation blind area of hard data, we can still find some unexpected fire heads. This means soft data can improve fire front accuracy to some extent, but as it is still fuzzy and might not provide relative enough information, the prediction result may be not as perfect as expected. So, to ensure the prediction accuracy our framework should also incorporate sufficient quantified hard data. Or we need to design more parameters in the report to constrain a fire front.



(a) Case 1 (b) Case 2

(c) Case 3 (d) Case 4

Figure 5.5 Simulation results for 4 cases.
(Blue line represents real fire front; black line represents simulated fire front; red line represents filtered fire front.)

Therefore, the fire fronts for case 1 to case 4 explicitly shows that it is useful for incorporating soft data in the dynamic system. The next step is to use some quantitative data to analyze our result.

First, we calculate areas for each filtered fire front in all cases. After that, those areas are compared with that of real fire front to get the area differences. Figure 5.6 displays the final result of area differences for all 4 cases in each time step. As shown in figure 5.6, case 2 and case 4 has less area differences than the other two cases at the final time step.

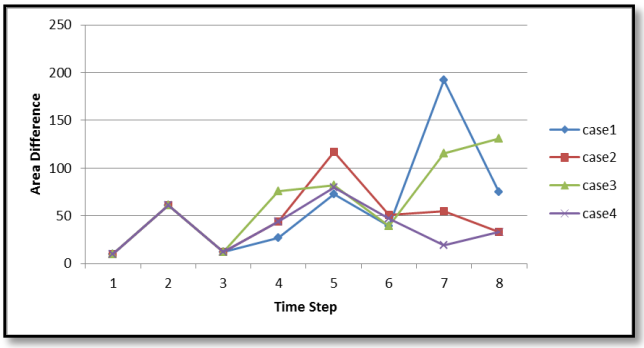


Figure 5.6 Comparison of area difference with real fire front for four cases

Second, we compute the number of cells with different state from real fire front. Figure 5.7 shows the number of different cells for all cases in each time step. During all time steps, case 2 has the minimal number of different cells. Case 3 has the most number of different cells because of the insufficiency of real observations. Case 4 shows higher number of different cells than case 1 because the quantity of soft data report is relatively not enough.

It can be concluded from figure 5.6 and figure 5.7 that case 2 has the best result which again proves our extended PFs based Data Assimilation Framework improves the prediction accuracy.

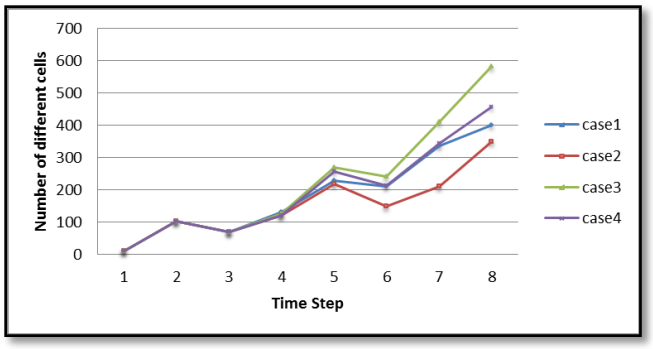


Figure 5.7 Comparison of number of different cells

5.4 Conclusion

In this chapter, a new PF-based data assimilation framework has been proposed to incorporate both soft and hard data in dynamic data driven system. For the framework, as soft data are qualitative and fuzzy compared to hard data, a general format of soft data report is designed to represent observations from humans. The experiment result shows that introducing soft data significantly improves the prediction accuracy. Although the proposal distribution applied in this paper is the system transition model, but it could also be replaced by more effective proposal distribution when importing real observations. One alternative method for improving the proposal distribution is to sample based on both hard and soft data. However, recently most research focuses on hard data, ignoring the soft data. So another future research problem can be how to sample from fuzzy soft data to generate optimal proposal distribution.

6 PERFORMANCE EVALUATION AND COMPARISON ON VARIANTS OF PFS FRAMEWORK FOR DATA ASSIMILATION IN SPATIAL-TEMPORAL SIMULATIONS

6.1 Introduction

Recently, variations of PFs based data assimilation frameworks have been proposed to incorporate observations to improve prediction accuracy in multiple spatial temporal simulations, such as wildfire simulation, traffic simulation and pedestrian simulation. They are designed based on different requirements and towards different applications. Traditionally, researchers employed standard PFs (also named as *whole state PF* in this dissertation) to make predictions. However, due to the high dimensional problem existing in the applications of large spatial temporal system, derivations of PFs based data assimilation framework have been studied. For example, *sub-state PF* reduce the state pace by dividing the system state into multiple sub-states and run single standard PFs in each sub-state. *Component set PF [80]* aim to increase the diversity of samples during resampling step by introducing crossover concept *[81]* from genetic algorithm. Spatially dependent PF calculate weight for each sub-state and do resampling based on groups of sub-states which are more suitable to applications where system dynamic model cannot be divided. However, to the best of our knowledge, there are few studies on the evaluation and comparisons of PFs based framework for data assimilation [82,83].

In this chapter, we investigate the fundamental methodologies of existing PFs based frameworks for data assimilation. In general, mainly according to the differences in sampling, weight calculation and resampling steps, there are four kinds of PFs based data assimilation frameworks as described above. They are whole state particle filtering, sub-state particle filtering, component set particle filtering and spatially dependent particle filtering. We first compare the

general frameworks of these four particle filtering methods. And then we summarize the potential issues for each them. At last we conduct experiments and simulations to evaluate the prediction accuracies of these four particle filtering methods.

6.2 Whole state PF

The whole state PF derives from the standard particle filtering. They are used widely as a conventional particle filtering method in data assimilation frameworks. Notice that during each iteration of whole state PF all the three basic steps (i.e. sampling, weight updating and resampling steps) are based on a whole and complete system state. The procedure for whole state PF is listed in Algorithm 4 below.

ALGORITHM 4. Whole state PF

---

1. **Initialization**

Draw N samples $\boldsymbol{X}_0^{(i)}$ from the prior

$$x_0^{(i)} \sim p(x_0), \qquad i = 1, \dots, N$$

and set

$$w_0^{(i)} = 1/N$$

2. **At each time step, repeat**

2.1. Sampling

Draw N samples $\boldsymbol{X}_t^{(i)}$ from the dynamic model:

$$x_t^{(i)} \sim p\left(x_t \middle| x_{t-1}^{(i)}\right)$$

2.2. Weight Updating

2.2.1 Calculate new weight for each particle based on all observations.

$$w_t^{(i)} = p\left(y_t \middle| x_t^{(i)}\right)$$

2.2.2 Normalize weights by

$$\widetilde{w}_t^{(i)} = \frac{w_t^{(i)}}{\sum_{i=1}^{N} w_t^{(i)}}$$

2.3 Resampling

Draw N new samples from the set $\boldsymbol{X}_t^{(i)}$ by sampling the indices 1…N, in proportional to the importance weights $\widetilde{w}_t^{(i)}$

As described in algorithm 4, initially all particles are randomly distributed and with equal weights. Then system states $x_t^{(i)}$ and weights $w_t^{(i)}$ are updated at each time step when observations are available. During the sampling step, the system state in each particle is evolved to next time step by state transition model $p\left(x_t \middle| x_{t-1}^{(i)}\right)$ (i.e. system dynamical model). After that, based on $p\left(y_t \middle| x_t^{(i)}\right)$, each particle is assigned a new weight by comparing the all "simulated" observations calculated from measurement function to real observations collected from sensors. In order to solve the sample degeneracy problem, in resampling step the weights of all particles are normalized so that N new samples are drawn from the set $\boldsymbol{X}_t^{(i)}$ with probabilities proportional to the weights $\widetilde{w}_t^{(i)}$.

Assume we use a sample set of three particles to predict the system state x, Figure 6.1 shows how the sample set is updated at a certain time step t. Initially, each particle has a full system state associated with a weight from last time step. For instance, the system state $x_{t-1}^{(1)}$ in particle 1 is assigned a weight w_{t-1}^1. Then in the sampling step, the full system state $x_{t-1}^{(1)}$ is evolved to the system state $x_t^{(1)}$ by the dynamic model. Next the weight associated to $x_{t-1}^{(1)}$ also is updated based on measurement model and all obtained real observations. Particle 2 and 3 perform the same steps to get new pairs $< x_t^{(2)}, w_t^{(2)} >$ and $< x_t^{(3)}, w_t^{(3)} >$ of system state and associated weight respectively. To draw new samples, the weights from each particle are summed and normalized so that $\widetilde{w}_t^{(1)} + \widetilde{w}_t^{(2)} + \widetilde{w}_t^{(3)} = 1$. Assume $\widetilde{w}_t^{(1)} = 0.6, \widetilde{w}_t^{(3)} = 0.3$, $\widetilde{w}_t^{(2)} = 0.1$, since $\widetilde{w}_t^{(1)}$ has a higher weight, particle 1 has higher probability to be duplicated, while $\widetilde{w}_t^{(2)} = 0.1$ has a much smaller weight, particle 2 is more likely to be eliminated. So in

resampling, particle 1 is duplicated, particle 2 is eliminated and particle 3 is maintained. Then this new set of particles are sent to next time step with new indices.

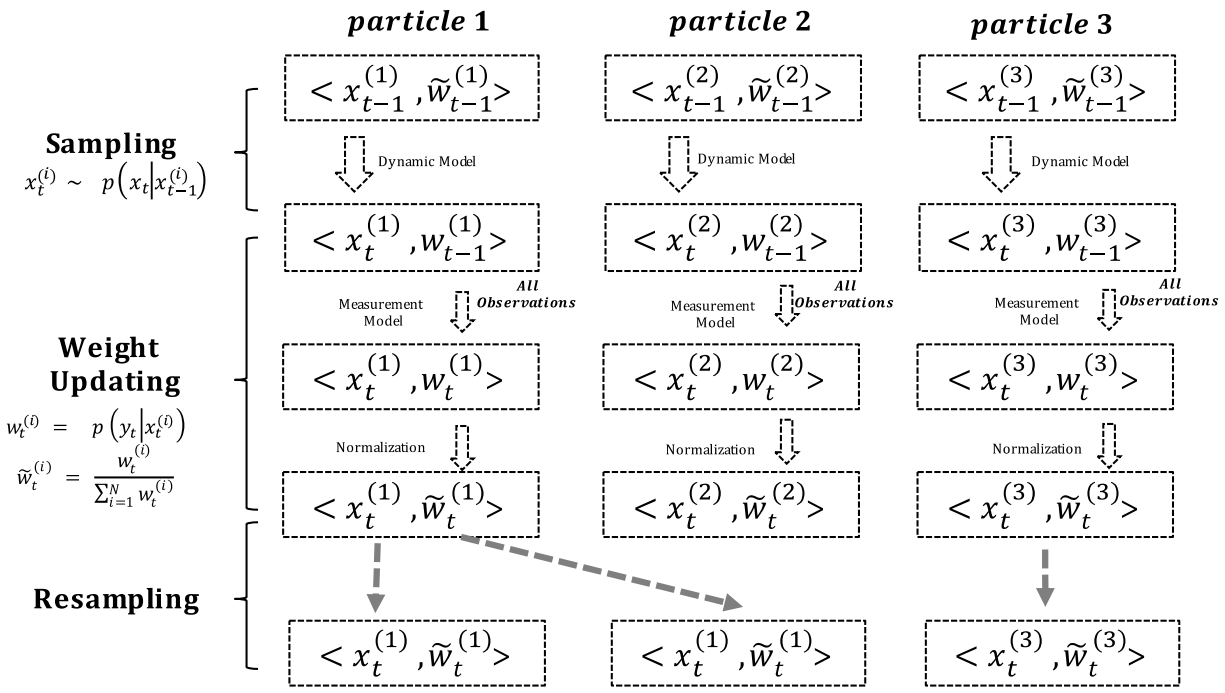


Figure 6.1 One example of whole state PFs.

6.3 Sub-state PF

Sub-state PF is a set of methods that decompose a system state into multiple sub-states and perform different particle filtering independently in each of the sub-states. This particle filtering method suits for a system in which the system state has a low dependency and system dynamic is dividable. This could be reasonable in a large spatial temporal system especially for a geophysical system where the system state crosses over a large space so that state variables far away from each other have no relations. Compared to whole state PFs, the sub-state PFs reduce searching spaces for the system state and thus provide one solution for the high dimensional issues in standard PFs. The procedure for sub-state PFs is listed in algorithm 5 below.

ALGORITHM 5. Sub-state PF

1. Divide a state space Sp into M sub-spaces (i.e. regions)

$$Sp = \{r_j | j = 1, .., M\}$$

2. For each sub-space r_j, run a whole state PFs independently

2.1. **Initialization**

Draw N samples $\boldsymbol{X}_{0,r_j}^{(i)}$ from the prior

$$x_{0,r_j}^{(i)} \sim p(x_{0,r_j}), \qquad i = 1, \dots, N$$

and set

$$w_{0,r_j}^{(i)} = 1/N$$

2.2. **At each time step, repeat**

2.2.1. Sampling

Draw N samples $\boldsymbol{X}_t^{(i)}$ from the "divided" dynamic model:

$$x_{t,r_j}^{(i)} \sim p\left(x_{t,r_j} \middle| x_{t-1,r_j}^{(i)}\right)$$

2.2.2. Weight Updating

2.2.2.1 Calculate new weight for each particle based on all observations.

$$w_{t,r_j}^{(i)} = p\left(y_{t,r_j} \middle| x_{t,r_j}^{(i)}\right)$$

2.2.2.2 Normalize weights by

$$\widetilde{w}_{t,r_j}^{(i)} = \frac{w_{t,r_j}^{(i)}}{\sum_{i=1}^{N} w_{t,r_j}^{(i)}}$$

2.3 Resampling

Draw N new samples from the set $\boldsymbol{X}_{t,r_j}^{(i)}$ by sampling the indices 1…N, in proportional to the importance weights $\widetilde{w}_{t,r_j}^{(i)}$

From Algorithm 5, we can see that sub-state PFs is similar to whole state PFs. However, before proceeding the three steps, there is a pre-process to each full system state, in which a full system state is divided into multiple sub-states $\{\boldsymbol{x}_{r_j} | i = 1 \dots M\}$ by partitioning the whole state space into smaller regions $\{r_j | j = 1, .., M\}$. Each sub-state $x_{r_j}^{(i)}$ then runs a single particle filtering independently. To implement sub-state PFs in a specific application, two issues should be taken into consideration. One issue is from the system dynamics. Sub-state PFs requires that the system dynamics is dividable so that a sub-state can evolve to the next iteration based on the dividable system dynamics. Another issue comes from the observations. In a spatial temporal

system, since both system states and observations are spatially distributed, for each sub-state only the sub-observations observing it should be calculated in likelihood function. But in the real world, most sensors provide measurements around a specific field crossing one or multiple sub-states, but not others. Then assigning observation to which sub-state becomes a problem.

Still based on the example in Figure 6.1 of Section 6.2, Figure 6.2 shows how the sub-state evolves during each iteration of sub-state particle filtering. Assume the state space is partitioned into two regions r_1 and r_2 , thus a full system state is consisted of two sub-states x_{r_1} and x_{r_2} accordingly. Then two groups of particles are running for each sub-state separately. One group is for sub-state in r_1, and the other is for sub-state r_2. Each group has three particles. For each group, of each particle the sub-state is evolved based on a "divided" dynamic model and weight is updated based on the sub-state and local observations through measurement model. So after weight updating step, the weights assignments vary in these two groups. Assume in the particle group for sub-state in r_1, the weights for particle 1 to 3 are assigned as $\widetilde{w}_{t,r_1}^{(1)} = 0.6, \widetilde{w}_{t,r_1}^{(3)} = 0.1,\ \widetilde{w}_{t,r_1}^{(2)} = 0.3$, so in resampling step particle 1 is duplicated, particle 2 is eliminated and particle 3 is maintained. Similarly, assume in the particle group for sub-state in r_2, if we have $\widetilde{w}_{t,r_2}^{(1)} = 0.2, \widetilde{w}_{t,r_1}^{(3)} = 0.1,\ \widetilde{w}_{t,r_1}^{(2)} = 0.7$, so in resampling step particle 1 is maintained, particle 2 is eliminated but particle 3 is duplicated.

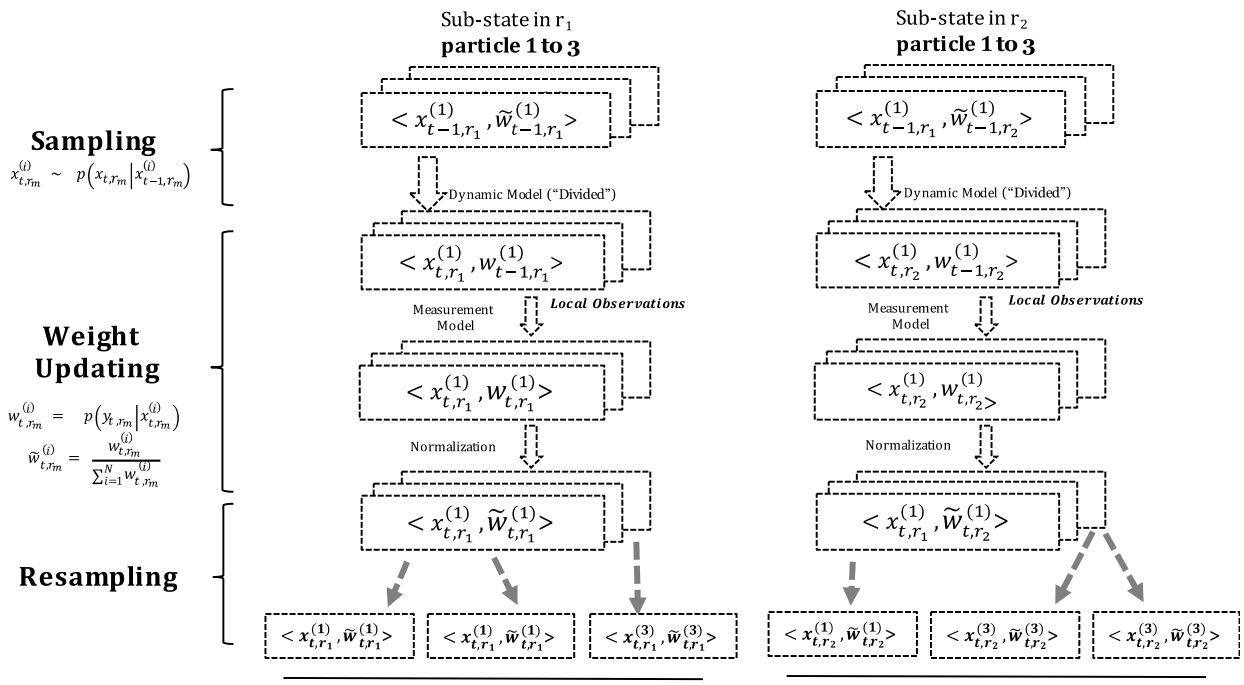


Figure 6.2 One example of sub-state PF.

6.4 Component set PF

Component-set PF is another set of particle filter methods based on component resampling methods Component resampling is inspired by genetic algorithm, in which crossover operator is used between particles to improve the diversity of samples in resampling step. One problem in whole-state PFs is that a "good" sub-state belonging to a "bad" particle has a high chance to be eliminated during resampling step. Crossover operator updates the population of samples and may let "good" sub-state combine with other "good" sub-state from other particles. Thus the diversity of samples is improved so that the distribution of samples is more "satisfactory".

Algorithm 6 illustrates the general steps fulfilled in component-set PF. Component set PF shares the same steps with whole state PF until the resampling step. In resampling step, before drawing new samples, we break the system state of each particle into several components (i.e.

sub-states) $x_t^{(i)} = \{x_{t,r_j}^{(i)} | j = 1, \dots, M\}$ by partitioning the state space into multiple smaller regions. Each component inherits the weight the same as the particle it belongs to. Among all the particles, the components from same space forms a group. Then we perform the resampling step in each of the component group. The next step is to obtain N new particles with a full state in each of them. At this step, a new full state is constructed by randomly selecting a component $x_{t,r_m}^{(q_m)}$ from each group.

ALGORITHM 6 Component Set PFs

---

1. **Initialization**

Draw N samples $\boldsymbol{X}_0^{(i)}$ from the prior

$$x_0^{(i)} \sim p(x_0), \qquad i = 1, \dots, N$$

and set

$$w_0^{(i)} = 1/N$$

2. **At each time step, repeat**

2.1. Sampling

Draw N samples $\boldsymbol{X}_t^{(i)}$ from the dynamic model:

$$x_t^{(i)} \sim p\left(x_t \middle| x_{t-1}^{(i)}\right)$$

2.2. Weight Updating

2.2.1 Calculate new weight for each particle based on all observations.

$$w_t^{(i)} = p\left(y_t \middle| x_t^{(i)}\right)$$

2.2.2 Normalize weights by

$$\widetilde{w}_t^{(i)} = \frac{w_t^{(i)}}{\sum_{i=1}^{N} w_t^{(i)}}$$

2.3 Resampling

2.3.1 Divide the system state of each particle into M sub-states by partitioning the whole state space into M regions $\{r_j | j = 1, .., M\}$

$$Sp = \{r_j | j = 1, .., M\}$$
$$x_t^{(i)} = \{x_{t,r_j}^{(i)} | j = 1, \dots, M\}$$

2.3.2 Form M component sets $\boldsymbol{X}_t^{(m)}$ of new particles by grouping the sub-states from same regions

$$\boldsymbol{X}_t^{(m)} = \{x_{t,r_m}^{(i)} | i = 1, .., N\}$$
$$\boldsymbol{X}_t = \{\boldsymbol{X}_t^{(m)} | m = 1, .., M\}$$

and associate each sub-state $x_{t,r_m}^{(i)}$ the weight $\widetilde{w}_{t,r_m}^{(i)}$ belonging to its full state $x_{t,r_m}^{(i)}$

$$\widetilde{w}_{t,r_m}^{(i)} = \widetilde{w}_t^{(i)}$$

2.3.3 Do resampling in each component set $\boldsymbol{X}_t^{(m)}$.

Draw N new samples from the set $\boldsymbol{X}_t^{(m)}$ by sampling the indices 1…N, in proportional to the importance weights $\widetilde{w}_{t,r_m}^{(i)}$.

2.3.4 Construct N new particles with a whole system state in it.

For each new particle, the system state is formed by randomly select a sub-state $x_{t,r_m}^{(q_m)}$ from each component set.

$$q_m \sim Uniform\ (M)$$
$$\boldsymbol{X}_t^{(i)} = \{x_{t,r_m}^{(q_m)} | m = 1 \dots M\}$$

Figure 6.3 provides an example of component-set PFs with three particles, the particle setting of which is identical to that of whole state PFs in Figure 6.1. However, after weight updating steps, two component sets are obtained for sub-state in r_1 and r_2 respectively. Different from sub-state PFs in which the sub-states from the same particle are assigned different weights, the sub-states from the same particle have same weights inherited from the particle. So assume $\widetilde{w}_t^{(1)} = 0.6, \widetilde{w}_t^{(3)} = 0.3,\ \widetilde{w}_t^{(2)} = 0.1$, after forming the component sets, since both $x_{t,r_1}^{(1)}$ and $x_{t,r_1}^{(1)}$ are from the same particle 1, $\widetilde{w}_{t,r_1}^{(1)} = \widetilde{w}_{t,r_2}^{(1)} = \widetilde{w}_t^{(1)}$ =0.6. Similarly, we have $\widetilde{w}_{t,r_1}^{(3)} = \widetilde{w}_{t,r_2}^{(3)} = \widetilde{w}_t^{(3)}$ =0.3 and $\widetilde{w}_{t,r_1}^{(2)} = \widetilde{w}_{t,r_2}^{(2)} = \widetilde{w}_t^{(2)}$ =0.1 . So in component set for sub-state in r_1, the set $\{x_{t,r_1}^{(1)}, x_{t,r_1}^{(2)}, x_{t,r_1}^{(3)}\}$ are resampled proportional to probability set $\{\widetilde{w}_{t,r_1}^{(1)}, \widetilde{w}_{t,r_1}^{(2)}, \widetilde{w}_{t,r_1}^{(3)}\}$={0.6,0.3,0.1}; in r_2, the set $\{x_{t,r_2}^{(1)}, x_{t,r_2}^{(2)}, x_{t,r_2}^{(3)}\}$ are resampled proportional to probability set $\{\widetilde{w}_{t,r_1}^{(1)}, \widetilde{w}_{t,r_1}^{(2)}, \widetilde{w}_{t,r_1}^{(3)}\}$={0.6,0.3,0.1}. With the same probability set, the same particle index sets are drawn from these two component sets. Thus, in both of two component sets, particle 1 duplicates, particle 3 remained. Finally, the sub-states from each group are randomly combined to form full states. On the one hand, the full state maybe combined by sub-states from same particles. For example, $x_{t,r_1}^{(1)}$ and $x_{t,r_2}^{(1)}$ from the same particle 1 form a new full system state. On the other hand,

the full state maybe combined by sub-states from different particles. For example, $x_{t,r_1}^{(1)}$ from particle 1 and $x_{t,r_2}^{(3)}$ from particle 2 construct a new full state $\{x_{t,r_1}^{(1)}, x_{t,r_2}^{(3)}\}$. Since new full states are generated by component set resampling, the diversity of particles has been improved. Compared to whole state PFs, the degeneracy problem of samples is further reduced.

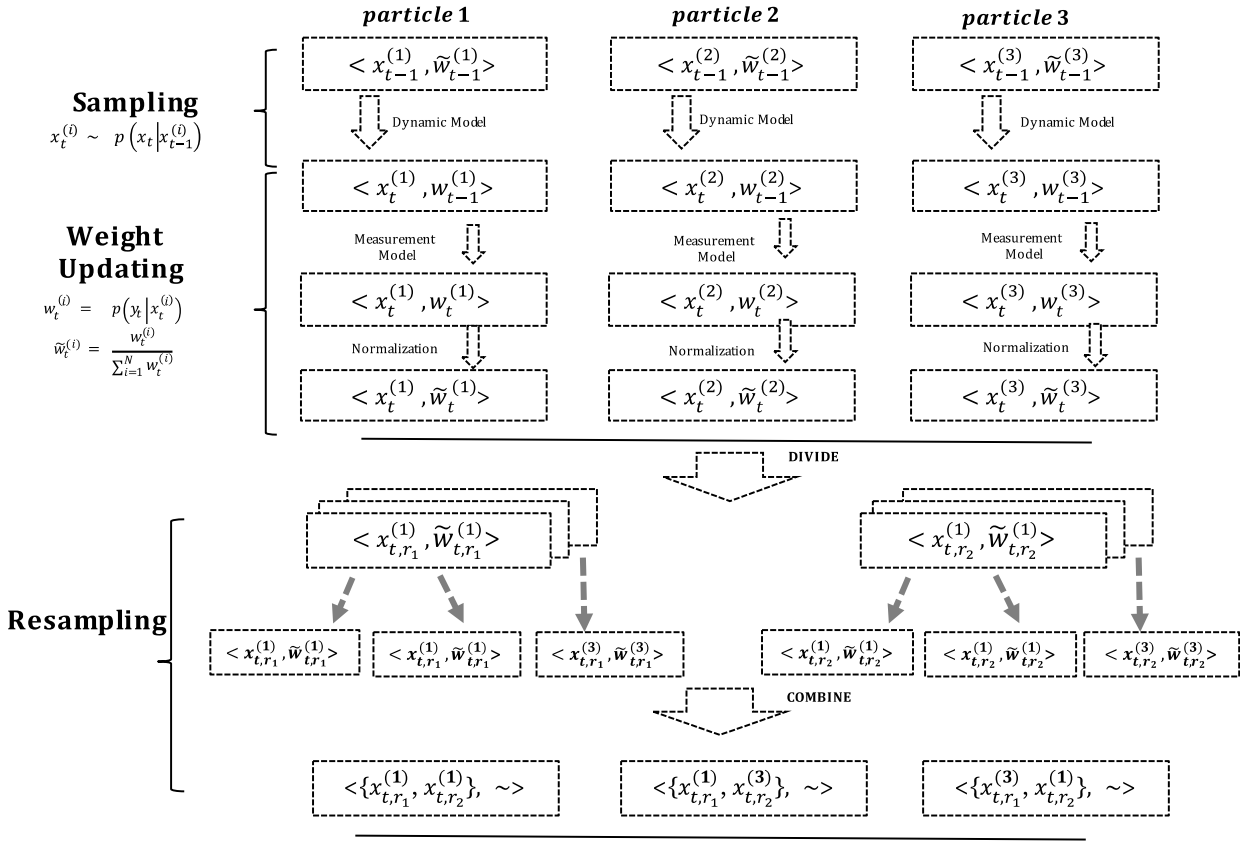


Figure 6.3 One example of component set PFs.

Compared to sub-state PFs, component-set PFs can be applied to more general cases, since the system dynamics does not have to be dividable. Besides, there is no need to worry about the data association problem when calculating weight for each particle. But, since resampling step is based on the weight for a full system state, a "good" sub-state still can be

hided with a very small weight before it combines with other sub-states from other particles. So spatially-dependent PFs is proposed to deal with this issue.

6.5 Spatially dependent PF

We know that since both the system state and observations are spatially distributed in a spatial temporal system, system state is spatially dependent. However, the PFs described in section 6.1, 6.2 and 6.5 have ignored important features in a spatial temporal system. On the one hand, a sub-state almost has no impact on the other sub-states which locates in a region far away. On the other hand, the observations can only reflect the status of system state in limited observation areas, but not whole system space. Therefore, spatially dependent PF is developed to take advantage of these important features. The main difference between spatially dependent PF and other PFs is that it calculates weights and resamples on a smaller local system state other than the large global system state. Details of the spatial dependent PFs can be found in algorithm 7 below.

As described in Algorithm 7, during each iteration, state partitioning is added as one important step between sampling and weight updating. The sampling step is still based on a full system state. So no modifications should be made to the original dynamic model. After state partitioning step, the weight for each sub-state $w_{t,r_m}^{(i)}$ is calculated based on local observations instead of inheriting the weight $w_t^{(i)}$ from a whole state based on all observations. So unlike component set PF, the weight for a sub-state $w_{t,r_m}^{(i)}$ may not be equal to the $w_t^{(i)}$. Then, the particles of different regions have distinctive weight distributions. As a result, among regions the particles are resampled independently. Therefore, the particles even with low weight of full state are duplicated because of a high weight of sub-state in it. Finally, since dynamic model is

based on a full state, the sub-states from different regions are grouped by randomly selecting an index q_m from the particles following by the uniform distribution.

ALGORITHM 7. Spatially-dependent PFs

---

1. **Initialization**

Draw N samples $\boldsymbol{X}_0^{(i)}$ from the prior

$$x_0^{(i)} \sim p(x_0), \qquad i = 1, \dots, N$$

and set

$$w_0^{(i)} = 1/N$$

2. **At each time step, repeat**

2.1. Sampling

Draw N samples $\boldsymbol{X}_t$ from the dynamic model:

$$x_t^{(i)} \sim p\left(x_t \middle| x_{t-1}^{(i)}\right)$$

2.2. State Partitioning

Divide the system state of each particle into M sub-states by partitioning the whole state space into M regions $\{r_j | j = 1, .., M\}$

$$Sp = \{r_j | j = 1, .., M\}$$
$$x_t^{(i)} = \{x_{t,r_j}^{(i)} | j = 1, \dots, M\}$$

2.3. Weight Updating

2.3.1 For each particle, calculate new weight for each sub-state based on associated local observations.

$$w_{t,r_m}^{(i)} = p\left(\boldsymbol{y_{t,r_m}} \middle| x_{t,r_m}^{(i)}\right)$$

2.3.2 Form M sets $\boldsymbol{X}_t^{(m)}$ of new particles by grouping the sub-states from same regions

$$\boldsymbol{X}_t^{(m)} = \{x_{t,r_m}^{(i)} | i = 1, .., N\}$$
$$\boldsymbol{X}_t = \{\boldsymbol{X}_t^{(m)} | m = 1, .., M\}$$

and associate each sub-state $x_{t,r_m}^{(i)}$ the weight $w_{t,r_m}^{(i)}$

2.2.2 In each set of new particles, normalize weights by

$$\widetilde{w}_{t,r_m}^{(i)} = \frac{w_{t,r_m}^{(i)}}{\sum_{i=1}^{N} w_{t,r_m}^{(i)}}$$

2.4 Resampling

2.4.1 Do resampling in each set $\boldsymbol{X}_t^{(m)}$.

Draw N new samples from the set $\boldsymbol{X}_t^{(m)}$ by sampling the indices 1…N, in proportional to the importance weights $\widetilde{w}_{t,r_m}^{(i)}$

2.4.2 Construct N new particles with a whole system state in it.

For each new particle, the system state is formed by randomly select a sub-state $x_{t,r_m}^{(q_m)}$ from each set $\boldsymbol{X}_t^{(m)}$

$$q_m \sim Uniform\ (M)$$
$$\boldsymbol{X}_t = \{x_{t,r_m}^{(q_m)} | m = 1 \dots M\}$$

As described in algorithm 7, during each iteration, state partitioning is added as one important step between sampling and weight updating. The sampling step is still based on a full system state. So no modifications should be made to the original dynamic model. After state partitioning step, the weight for each sub-state $w_{t,r_m}^{(i)}$ is calculated based on local observations instead of inheriting the weight $w_t^{(i)}$ from a whole state based on all observations. So unlike component set PF, the weight for a sub-state $w_{t,r_m}^{(i)}$ may not be equal to the $w_t^{(i)}$. Then, the particles of different regions have distinctive weight distributions. As a result, among regions the sets of resampled particles are different. Finally, since dynamic model is based on a full state, the sub-states from different regions are grouped by randomly selecting an index q_m from the particles following by the uniform distribution.

Figure 6.4 shows how a "good" sub-state in a "bad" full state particle is maintained in one iteration of spatially dependent PF. After sampling step, the full state in each particle is broken into two sub-states for r_1 and r_2 separately according to the space partitioning result. Then independent weight updating is performed for the sub-states in each region. Assume for region r_1 , the weights for all three sub-states from different particles are as $\widetilde{w}_{t,r_1}^{(1)} = 0.6, \widetilde{w}_{t,r_1}^{(3)} = 0.1$, $\widetilde{w}_{t,r_1}^{(2)} = 0.3$; for region r_2 the weights for all three sub-states from different particles are $\widetilde{w}_{t,r_2}^{(1)} = 0.2, \widetilde{w}_{t,r_2}^{(3)} = 0.1,\ \widetilde{w}_{t,r_2}^{(2)} = 0.7$. So in resampling step, in region r_1 particle 1 is duplicated, particle 2 is remained; in region r_2 , particle 1 is remained but particle 2 is duplicated. Although

originally particle 3 has a low weight assigned for its full state, after division, a good portion of state is discovered by the weight on a sub-state. Finally, this good portion forms another new system state with other sub-states. Thus, the good sub-state in a "bad" particle is not hidden anymore.

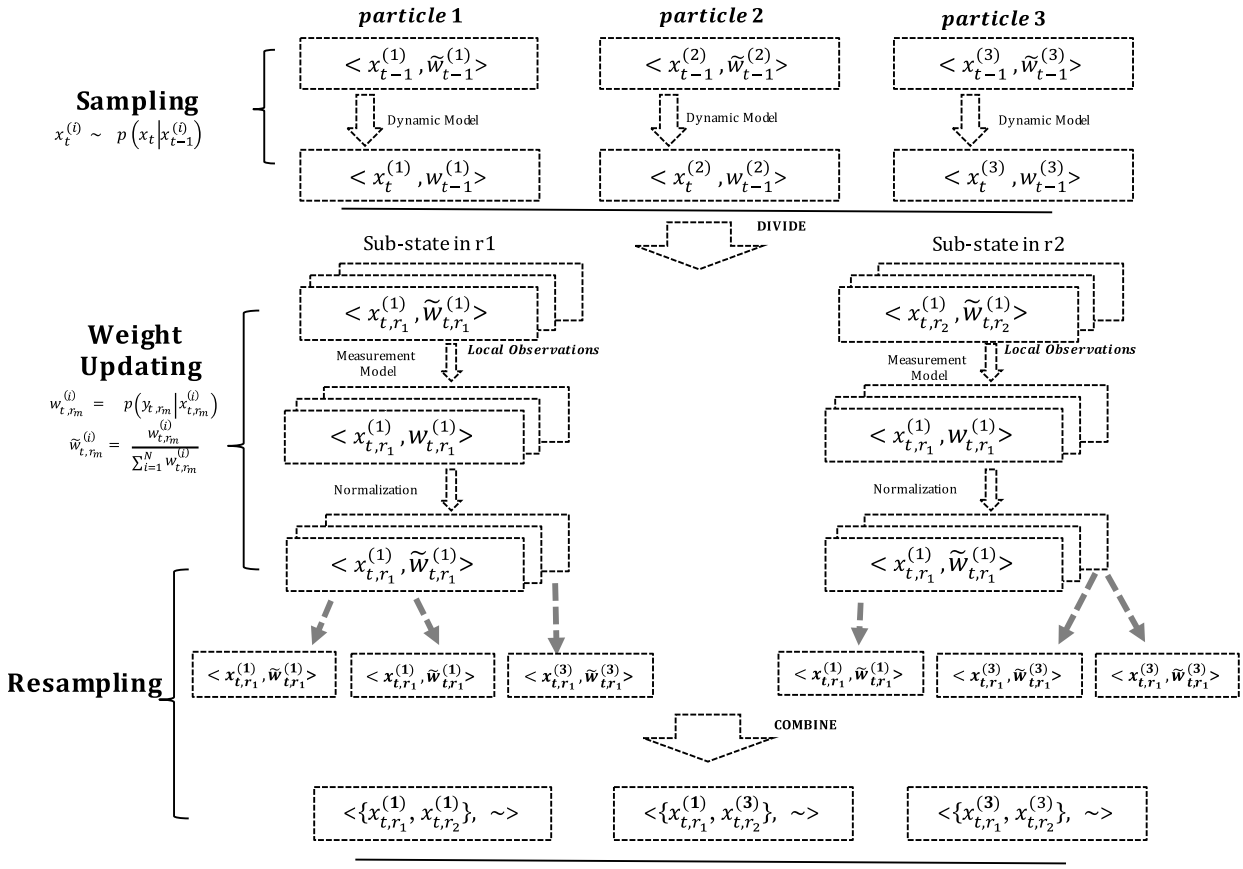


Figure 6.4 One example of spatially dependent PF.

6.6 Comparisons

Previous sections have detailed the procedure in four kinds of particle filtering methods for data assimilation in spatial temporal system. In this section, we first compare each other PF methods to the whole state PF and summarize the differences in the general three basic PF steps. Next, we compare the potential issues in these four PF methods.

1） Comparisons on the general PFs steps

Although each of the particle filtering methods described in this part follows the three main steps of stand PFs, the implementation in each step differs from each other according to the scale of state based on. Through Table 6.1, we can clearly find that only whole state PF uses a whole state for each particle to accomplish the three steps. However, the other three particle filtering methods also use sub-state in a smaller region to perform all or part of the three steps. Specifically, in sub-state PF, all the three steps are based on sub-states instead of full states. The component set PF only apply sub-states in resampling step to instance new set of full states by cross over operator. Besides resampling on sub-states, the spatially dependent PF also calculates weights for each sub-state in weight updating step.

Table 6.1: Comparisons on the general PFs steps among four kinds of particle filtering methods

General PFs steps	Whole state PFs	Sub-state PFs	Component Set PFs	Spatially Dependent PFs
Sampling	Whole state	Sub-state	Whole state	Whole state
Weight Updating	Whole state	Sub-state	Whole state	Sub-state
Resampling	Whole state	Sub-state	Sub-state	Sub-state

2） Comparisons on potential issues

Although we have discussed multiple particle filtering methods, some potential issues cannot be ignored. In general, there are four potential issues. They are concerned with data association, state partitioning, state dependency and applications as listed in Table 6.2.

- Data association

To calculate the weight for a sub-state instead of full-state, only the local observations should be considered. However, due to the diversity of sensors, how to

assign a boundary sensor to the real sub-state it observers is a problem. So before apply spatially-dependent particle filtering, there should be a solution to fix this problem. However, for the other PF methods, there is no need to consider the issue of data association.

- State partitioning

To obtain a sub-state, we should first partition the full state. However, the state partitioning result has effects on the final prediction result in all the particle filtering methods in which sub-states are related. For example, in component set PF, if a divided region is small and never touched (e.g. a fire never spread there), all the particles may have same sub-state in that region so that the diversity of samples is not improved even with crossover operator. Another example is spatially dependent PF, if a divided region has no sensors surrounded and no observation covered, the weights for all the sub-states are the same (i.e. 1/N) so that the goal to select good sub-state in a bad full-state particle cannot be achieved. Therefore, to apply the particle filtering methods encountered state portioning, we should be careful to choose a good state partitioning method.

- State dependency

State partitioning nowadays has been widely used in spatial temporal simulations to reduce space dimension and increase prediction accuracy with lower computational cost. Sub-state PF requires that the system state is lowly dependent, otherwise independent PF cannot be implemented for each region. However, if the system state is highly dependent and there is no way to divide it, only whole state PF can be used. When the system state

is not highly dependent, we can also consider both component set PF and spatially dependent PF.

- Applications

In these our algorithms, only whole state PFs can be applied to all applications. However, due to problems described above, the other three particle filter methods can only be applied to application of special filed. For example, we can use whole state PF, component set PF and spatially dependent PF in DEVS based wildfire simulation. But since each cell still correlates the neighboring cells and the system dynamic is not dividable, sub-state PFs cannot be applied in wildfire simulation.

Table 6.2: Comparisons on potential issues among four kinds of particle filtering methods

Potential Issues	Whole state PFs	Sub-state PFs	Component Set PFs	Spatially Dependent PFs
Data Association	NO	YES	NO	YES
State Partitioning	NO	YES	YES	YES
State Dependency	HIGH	LOW	?	?
Applications	ALL	PARTIAL	PARTIAL	PARTIAL

6.7 Experiment

To compare the prediction accuracy of these four PF methods, we conducted a case study in wildfire simulation. We designed three cases for experiments. The experiment settings for these three cases are as following:

(1) Case 1: A single fire with uniform distributed sensors

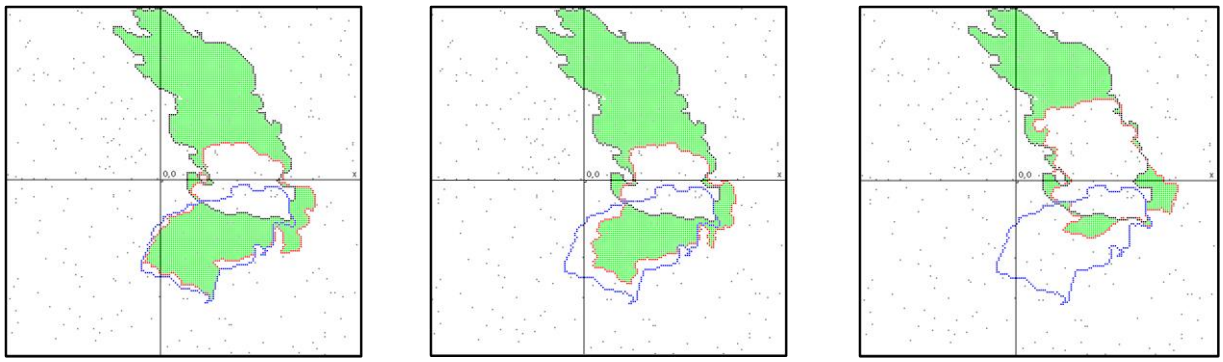
We started our experiments from a general case, where there was only a single fire and 400 sensors were uniformly distributed. For "correct" weather information, the wind speed varied between 20 and 24 (m/s) and wind direction ranges from 120 to 160 degrees. Similarly, for "erroneous" weather information, the wind speed ranged from 21 to 25 m/s and direction ranged from 30 to 70 degrees. There was only one ignition point at (55,-10), which was added at first time step. Details of experiment setting can be found at Table 6.3. To apply component PF and spatially dependent PF, we divided the system space into two equal regions. Region 1 was on top of the map, region 2 is on bottom of the map. And we used 50 particles for each particle filtering methods.

Table 6.3 Experiment setting for weather data and ignition points in case 1

	Weather data	Ignition Points
"Correct"	Speed: 22±2 m/s Direction: 140±20 degrees	(55,-10)
"Erroneous"	Speed: 23±2 m/s Direction: 50±20 degrees	(55,-10)

From figure 6.5 (a) we can find out that since the direction in erroneous weather information has great errors, the simulated fire front (in blue) diverged a lot from the real fire front. Although the filtered fire front in red from whole state PF results was better than the simulated fire front, the number of cells with different system states (in green) is still large. After applying component set PF, this difference is reduced as the filtered fire front at bottom shrinked. However, spatially dependent PF further reduced this symmetric difference by guiding fire spread to a correct direction. Figure 6.6 compares the number of cells with symmetric differences during all time steps among these three particle filtering methods. We can see that initially these three PF methods generate similar results, however after some time step, component set PF and spatially dependent PF generate better result with less number of cells with symmetric differences compared to whole state PF. Although the error at final step is still large, but

compared to traditional whole state PF, component set PF and spatially dependent PF both improved prediction accuracy, especially for spatially dependent PF.



(a) Whole state PF (b) Component Set PF (c) Spatially Dependent PF

Figure 6.5 Simulation results at last time step for three particle filtering methods in case 1

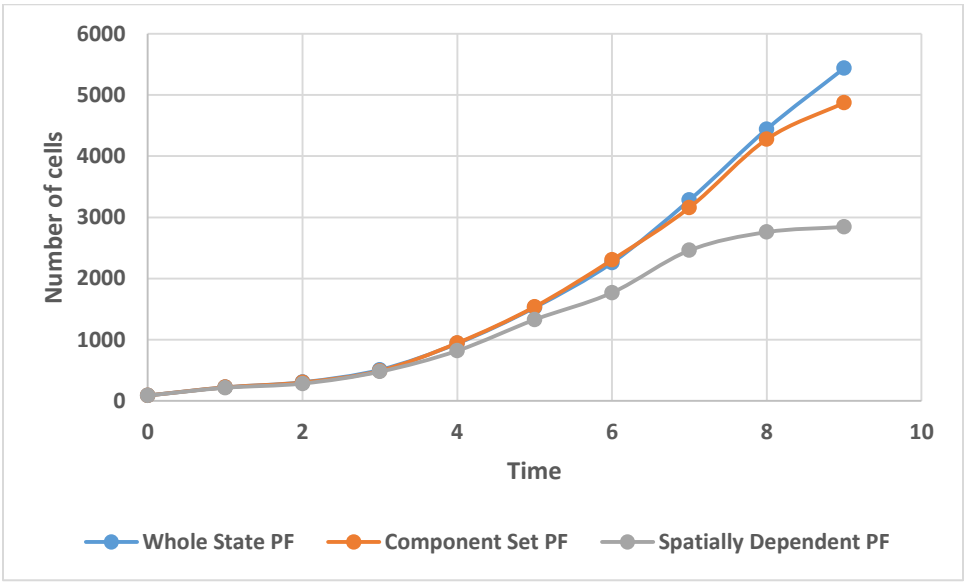


Figure 6.6 Comparison of symmetric differences among three particle filtering methods in case 1

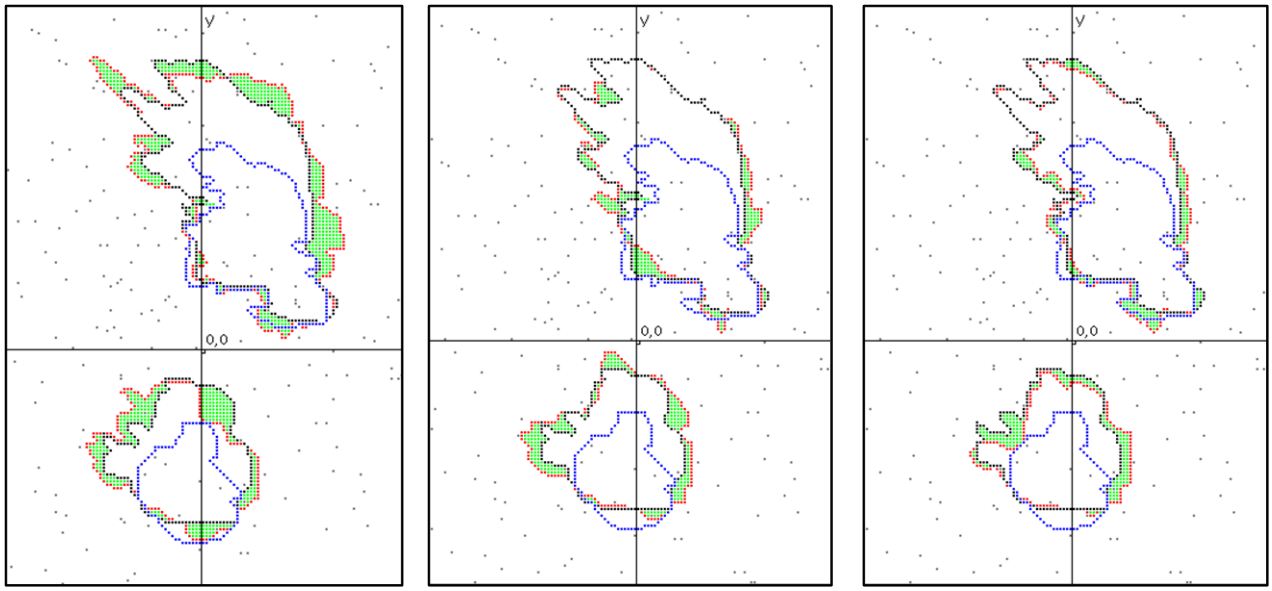
(2) Case 2: Two singles fires and never merge

Case 2 has the same sensor deployment as in case 1, however there are two single fires in the scenario. Since sub-state PF does not apply to the application of wildfire simulation, in which the dynamic model is not dividable, we design this special case and consider the two single fires are independent to each other so that the even for a full dynamic model these two fires evolves separately as they never merge. Then the sub-state PF can be converted spatially dependent PF in this case when using 2 sub-states, in two equally divided regions --- one on top and the other on the bottom. So in case 2, we can assume the spatially dependent PF has the same prediction result as in sub-state PF. More detailed setting of case 2 can be found in table 6.4.

Table 6.4 Experiment setting for weather data and ignition points in case 2

	Weather data	Ignition Points
"Correct"	Speed: 17±3 m/s Direction: 140±20 degrees	(25, 20) and (10, -50) added at initial time step
"Erroneous"	Speed: 15±3 m/s Direction: 120±20 degrees	(25, 20) and (10, -50) added at initial time step

Figure 6.7 shows the simulations results in case 2 under different kinds of particle filtering methods. In figure 6.7(a) for the results in whole state PF, the green area was larger compared to that in figure 6.7 (b) and 6.7(c). This means that both component set PF and spatially dependent PF has improved the prediction result, even for the sub-state PF. Figure 6.8 illustrates the change of number of cells with symmetric differences using ten time steps for all particle filtering methods. Although around time step 7, the number of cells with symmetric differences in component set PF is similar to to that in whole state PF. But it quickly dropped to a similar level as spatially dependent PF after time step 8. This happens because from time step 6 to time step 7, the fire spread fast and changed dramatically so that even for component PF it is hard to provide a good prediction result.



(a) Whole state PF (b) Component Set PF (c) Spatially Dependent PF(Sub-state PF)

Figure 6.7 Simulation results at last time step for three particle filtering methods in case 2

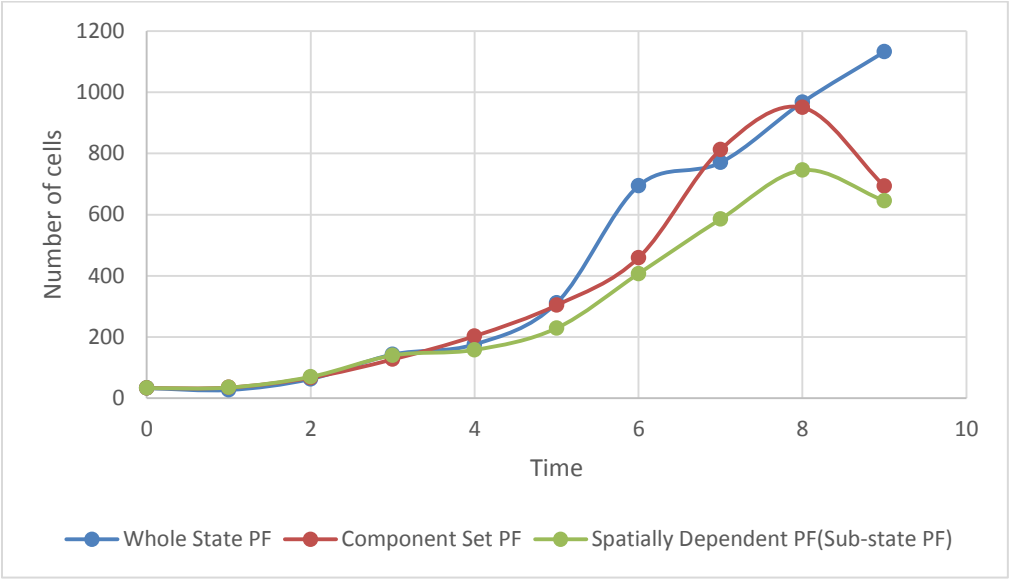


Figure 6.8 Simulation results at last time step for three particle filtering methods in case 2

(3) Case 3: A single fire with unevenly distributed sensors

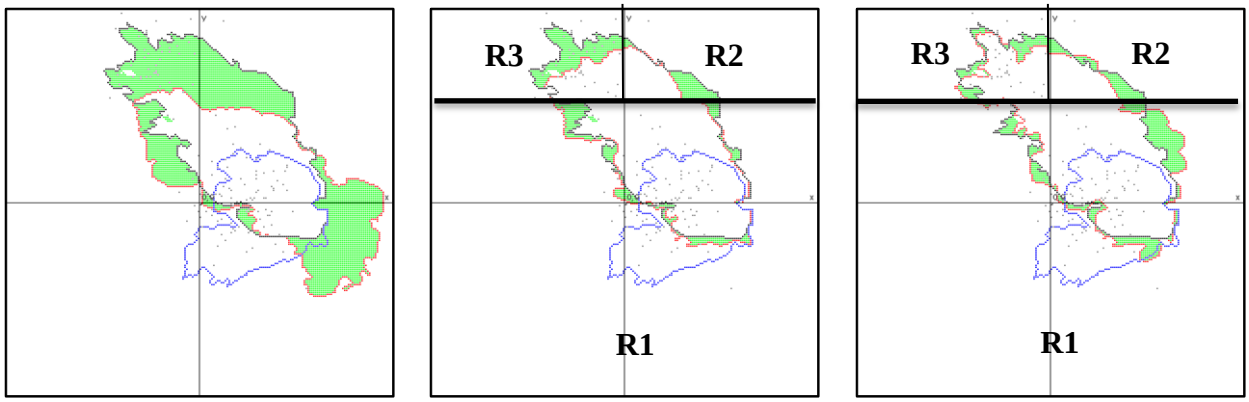
The goal of this case is to test the impact of sensor distribution on prediction accuracy of different PF methods. In this case, there were 150 sensors distributed unevenly over map. 100 sensors were distributed around (0, 20) through a Gaussian distribution with mean 15 and variance 0 on both x an y coordinate; another 50 sensors were deployed nearing (-20,75) through the same Gaussian distribution. 50 particles were used for each method. And system space was broken into three regions, one of them are on the top and the other is on bottom.

Table 6.5 Experiment setting for weather data and ignition points in case 2

	Weather data	Ignition Points
"Correct"	Speed: 22±2 m/s Direction: 120±20 degrees	(55,0)
"Erroneous"	Speed: 22±2 m/s Direction: 70±20 degrees	(55,0)

Figure 6.9 shows that since there were no enough sensors to observe the global fire spreading information, the whole state PF method provided the worst result with largest number of cells with symmetric differences (in green area). Normally, more particles should be added to cover more possible system states in unobserved areas. However, by using component PF the diversity of particles was increased, and good sub-states located at observed area have high chances to be combined. Similar to spatially dependent PF, the sub-states nearing sensor clusters can be weighted and thus generate more accurate result than whole state PF. Through figure 6.10, we note that although component set PF outperformed whole state PF at each time step, around time step 5, the spatially dependent PF even provided worse result than whole state PF. This due to the uneven distribution of sensors. Since at time step 5, the real fire started to spread to region 2 and top of region 1 where few sensors were distributed, the sub-state in region 2 was not observed by any other or had few sensor observations. The uncertainty of sub-state in region 2 leads to higher errors. However, when fire pass this region to region 3 with a high density of

sensors, the number of sensors began to drop. Note that even at last time step, in this case the results in component set PF and spatially dependent PF are close to each other. The component set PF even had higher prediction accuracy than spatially dependent PF.



(a) Whole state PF (b) Component Set PF (c) Spatially Dependent PF

Figure 6.9 Simulation results at last time step for three particle filtering methods in case 3

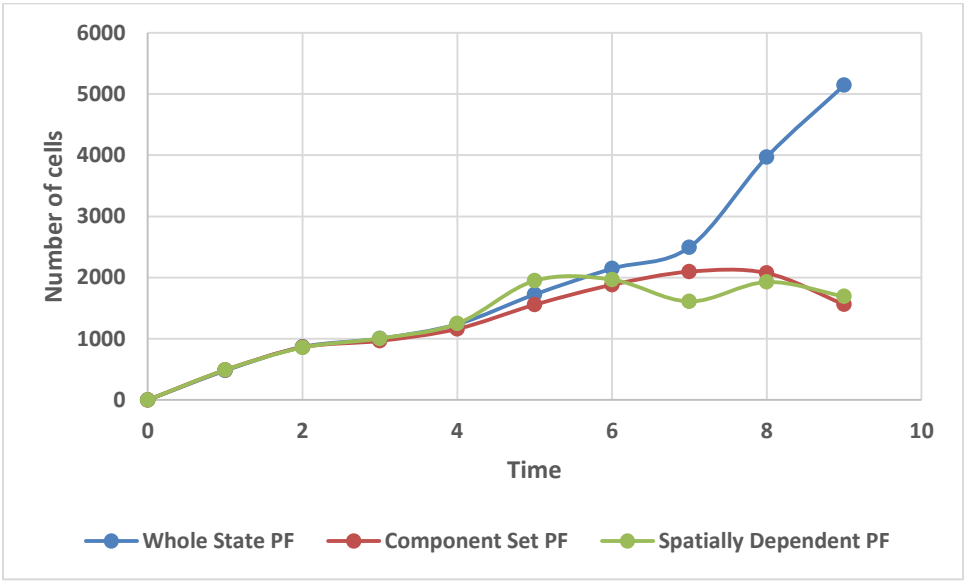


Figure 6.10 Comparison of symmetric differences among three particle filtering methods in case 3

6.8 Conclusion

In this part, we have compared four existing particle filtering methods applied in spatial temporal systems by analyzing their commons and differences in main steps of a standard PF. Also we have summarized the issues and limitations for each particle filtering methods. The general issues should be considered when choosing particle filtering methods are the data association problem, state partition way, state dependency and characteristics of applications. At last, the experiments show that the other three particle filtering methods outperforms the conventional whole state PF. Besides, the spatially dependent PF provides the best prediction results in most cases, where the sensors are not unevenly distributed over the space.

7 CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this work, we developed a localized particle filtering to support PFs-based data assimilation for large-scale spatial temporal simulations. We proposed a spatial partition-based particle filter framework for simulation and prediction in a spatial temporal system. This framework introduces the locality nature of system state. We divide the whole space into several smaller regions and the full state is broken into correspondingly sub-states. Unlike the tradition calculation on a whole system state, it calculates a set of local sub-state weights for each particle and performs resampling in a group of local sub-states. Experiments demonstrate the improvement on prediction accuracy when space is divided reasonable with some prior knowledge and when partition number is increased in a proper range. Furthermore, we also find out data association problem from boundary sensors. So within this framework, the automated partitioning method with two levels is proposed to reduce the total boundary sensor number. Apparent prediction accuracy improvements are found in experiments after performing the automated partitioning method. However, data association problem may still exist even after performing automated partitioning method.

Besides, we also considered different types of data to support data assimilation for spatial temporal simulations. We incorporated both hard data from physical devices and soft data from messages, report and social networks to data assimilation and achieved the improvement on the prediction result. Although soft data is fuzzy, it supplements the observation especially to the area with few sensors surrounded.

At last, we compared our spatially dependent particle filtering to the existing particle particle filtering for large spatial temporal simulations. Each particle filtering framework has its

own merits and limitations when applying to different applications. Although the whole state particle filtering can be applied to most applications, it has worst performance on the prediction accuracy. Sub-state particle filtering works only for the systems when the dynamic model is dividable. Component set particle filtering increases the diversity of samples but it can still hide the samples with good sub-states before resampling. Spatially dependent particle filtering is a localized particle filtering and achieves better prediction accuracy when state is partitioned in a proper way.

7.2 Future work

Although space division has been proved as an appropriate way to improve prediction performance, it also brings another problem from observation. This is a classical data association problem in multi-target tracking. Since sensor has observation area, when this area is broken into several part it is hard to decide which part the real observation reflects. In this paper, those observations are from the boundary sensors nearing borders of divided regions. From experiments, we found that boundary sensors have great impact on prediction accuracy. In this paper, for how to assign real boundary observation to possible sub-states or decide real sensor is stimulated by which region, we present a general solution, which treats each covered regions with the equal probability to trigger the sensor. However, we need to note that this solution may not be the best method to solve data association problem. It is possible that we can assign the probability with ratio of covered observation area in different region. Also, the data association may become complex due to the different nature of sensor observations and applications. Therefore, improving current solution for data association problem and developing alternative methods towards data association problem for complicated sensors in a spatial temporal system is also a future work for this dissertation.

The next direction for the future work is to improve the proposed spatial dependent particle filtering framework in other application, such as traffic simulation and pedestrian simulation. But more challenges emerge because of the characteristics of sensors in different applications. For example, there are two main challenges to be considered in traffic simulation when developing the spatially dependent particle filtering framework. One of the main challenges is from data association problem because of the specialty of observation data. In traffic simulation, usually sensor can reflect information for multiple regions. For example, density sensor reflects the density in a road segment. If the road segment falls in different regions, we need to find a solution to separate their impact to related regions.

Since this work is for the large-scale system, it exists one common problem, which is high computation cost. Therefore, another future work is to develop an effective data-driven framework in cloud computing to reduce time cost or to use parallel & distributed techniques to reduce workload in a single node. As my current work proposed to divide space into multiple regions and concentrate on local information instead of whole space in some steps, it is feasible to process local information in each node, and reduce both time cost and workload finally.

REFERENCES

[1] Haidong Xue, Feng Gu, and Xiaolin Hu. 2012. Data assimilation using sequential monte carlo methods in wildfire spread simulation. ACM Trans. Model. Comput. Simul. 22, 4, Article 23 (November 2012), 25 pages. DOI:http://dx.doi.org/10.1145/2379810.2379816

[2] Johannes Brügmann, Michael Schreckenberg, and Wolfram Luther. 2013. Real-Time Traffic Information System Using Microscopic Traffic Simulation. In *Proceedings of the 2013 8th EUROSIM Congress on Modelling and Simulation (EUROSIM '13)*. IEEE Computer Society, Washington, DC, USA, 448-453. DOI:http://dx.doi.org/10.1109/EUROSIM.2013.83

[3] Joaquin Maroto, Eduardo Delso, Jesús Félez, and Jose Ma Cabanellas. Real-time traffic simulation with a microscopic model. *IEEE Trans. Intelligent Transportation Systems*, 7, 4 (December 2006), 513-527. DOI:10.1109/TITS.2006.883937

[4] Suiping Zhou, Dan Chen, Wentong Cai, Linbo Luo, Malcolm Yoke Hean Low, Feng Tian, Victor Su-Han Tay, Darren Wee Sze Ong, and Benjamin D. Hamilton. 2010. Crowd modeling and simulation technologies. *ACM Trans. Model. Comput. Simul.* 20, 4, Article 20 (November 2010), 35 pages. DOI:http://dx.doi.org/10.1145/1842722.1842725

[5] Fasheng Qiu and Xiaolin Hu. 2013. Spatial activity-based modeling for pedestrian crowd simulation. *Simulation* 89, 4 (April 2013), 451-465. DOI:http://dx.doi.org/10.1177/0037549711435950

[6] Brian R. Hunt, Eric J. Kostelich, Istvan Szunyogh, Efficient data assimilation for spatiotemporal chaos: A local ensemble transform Kalman filter, Physica D: Nonlinear

Phenomena, Volume 230, Issues 1–2, June 2007, Pages 112-126, ISSN 0167-2789, http://dx.doi.org/10.1016/j.physd.2006.11.008

[7] Neil J. Gordon, J. Salmond David, and Adrian FM Smith. 1993. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *Proceedings F of IEE Conference on Radar and Signal Processing*. IEEE,vol.140, no.2, pp.107-113, April 1993. DOI:10.1049/ip-f-2.1993.0015

[8] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering* 82, 1 (1960), 35-45. DOI:10.1115/1.3662552

[9] Thomas Bengtsson, Chris Snyder, and Doug Nychka. 2003. Toward a nonlinear ensemble filter for high dimensional systems. *Journal of Geophysical Research: Atmospheres (1984--2012)*, 108, no. D24 (2003). DOI:10.1029/2002JD002900

[10] John Harlim, Brian R. Hunt. 2007. A non-Gaussian Ensemble Filter for Assimilating Infrequent Noisy Observations. *Tellus A*, [S.l.], 59, 2, (March 2007). DOI:http://dx.doi.org/10.3402/tellusa.v59i2.14935.

[11] Richard Swinbank, Victor Shutyaev, and William Albert Lahoz, eds. 2012. Data assimilation for the earth system. Vol. 26: Nato Science Series: IV. Springer Science & Business Media. DOI:10.1007/978-94-010-0029-1

[12] Eric Blayo. 2014. Advanced Data Assimilation for Geosciences: Lecture Notes of the Les Houches School of Physics: Special Issue June 2012. Oxford University Press.

[13] Yiqi Luo, Kiona Ogle, Colin Tucker, Shenfeng Fei, Chao Gao, Shannon LaDeau, James S. Clark, and David S. Schimel. 2011. Ecological forecasting and data assimilation in a data-rich era. Ecological Applications 21, 5 (July 2011), 1429-1442.

DOI:http://dx.doi.org/10.1890/09-1275.1

[14] Michael Dowd. 2007. Bayesian statistical data assimilation for ecosystem models using Markov Chain Monte Carlo. Journal of Marine Systems 68, 3 (December 2007), 439-456. DOI:10.1016/j.jmarsys.2007.01.007

[15] Ionel M Navon. 2009. Data assimilation for numerical weather prediction: a review. In Data assimilation for atmospheric, oceanic and hydrologic applications, pp. 21-65. Springer Berlin Heidelberg, DOI: 10.1007/978-3-540-71056-1_2

[16] Florence Rabier. 2005. Overview of global data assimilation developments in numerical weather-prediction centres. Q. J. R. Meteorol. Soc. 131, 613 (October 2005): 3215-3233. DOI:10.1256/qj.05.129

[17] Shaoqing Zhang, Guijun Han, Yuanfu Xie, and Juan Jose Ruiz. 2015. Data Assimilation in Numerical Weather and Climate Models. Advances in Meteorology 2015 (2015) 2 pages. DOI:http://dx.doi.org/10.1155/2015/626893

[18] Maria Isabel Ribeiro. 2004. Kalman and extended kalman filters: Concept, derivation and properties. Institute for Systems and Robotics, 43 (2004). DOI:10.1.1.2.5088

[19] Eric Wan, and Ronell Van Der Merwe. 2000. The unscented Kalman filter for nonlinear estimation. In Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000, pp. 153-158. DOI:10.1109/ASSPCC.2000.882463

[20] Hunter, M.P.; Fujimoto, R.M.; Wonho Suh; Hoe Kyoung Kim, “An Investigation of Real-Time Dynamic Data Driven Transportation Simulation,” Simulation Conference, 2006. WSC 06. Proceedings of the Winter , vol., no., pp.1414,1421, 3-6 Dec. 2006

[21] Khaleghi, B.; Khamis, Alaa; Karray, F., "Random finite set theoretic based soft/hard data fusion with application for target tracking," Multisensor Fusion and Integration for Intelligent Systems (MFI), 2010 IEEE Conference on , vol., no., pp.50,55, 5-7 Sept. 2010

[22] Pravia, M.A.; Prasanth, R. K.; Arambel, P.O.; Sidner, C.; Chee-Yee Chong, "Generation of a fundamental data set for hard/soft information fusion," Information Fusion, 2008 11th International Conference on , vol., no., pp.1,8, June 30 2008-July 3 2008

[23] Gross, G.A.; Nagi, R.; Sambhoos, K.; Schlegel, D.R.; Shapiro, S.C.; Tauer, G., "Towards hard+soft data fusion: Processing architecture and implementation for the joint fusion and analysis of hard and soft intelligence data," Information Fusion (FUSION), 2012 15th International Conference on , vol., no., pp.955,962, 9-12 July 2012

[24] Jenkins, M.P.; Gross, G.; Bisantz, A.M.; Nagi, R., "Towards context-aware hard/soft information fusion: Incorporating situationally qualified human observations into a fusion process for intelligence analysis," Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2011 IEEE First International Multi-Disciplinary Conference on , vol., no., pp.74,81, 22-24 Feb. 2011 doi: 10.1109/COGSIMA.2011.5753757

[25] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. 2002. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. Trans. Sig. Proc. 50, 2 (February 2002), 174-188. DOI:http://dx.doi.org/10.1109/78.978374

[26] David Salmond , Neil Gordon. An introduction to particle filters, September 2005 (incomplete draft).

[27] Jun S. Liu, and Rong Chen. 1998. Sequential Monte Carlo methods for dynamic systems. Journal of the American statistical association 93, 443 (September1998) 1032-1044. DOI:10.2307/2669847

[28] Jaco Vermaak, Christophe Andrieu, Arnaud Doucet, and Simon John Godsill. 2002. Particle methods for Bayesian modeling and enhancement of speech signals. IEEE Trans. Speech and Audio Processing, 10, 3 (March 2002): 173-185. DOI:10.1109/TSA.2002.1001982

[29] Fredrik Gustafsson, Fredrik Gunnarsson, Niclas Bergman, Urban Forssell, Jonas Jansson, Rickard Karlsson, and Per-Johan Nordlund. 2002. Particle filters for positioning, navigation, and tracking. IEEE Trans. Signal Processing, 50, 2 (February 2002), 425-437. DOI:10.1109/78.978396

[30] Kazuyuki Nakamura, Tomoyuki Higuchi, and Naoki Hirose. 2006. Application of particle filter to identification of tsunami simulation model. SCIS & ISIS, Japan Society for Fuzzy Theory and Intelligent Informatics, (2006), 1890-1895. DOI:10.14864/softscis.2006.0.1890.0

[31] Lyudmila Mihaylova, René Boel, and Andreas Hegyi. 2007. Brief paper: Freeway traffic estimation within particle filtering framework. Automatica 43, 2 (February 2007), 290-300. DOI:http://dx.doi.org/10.1016/j.automatica.2006.08.023

[32] Fazlina Ahmat Ruslan, Zainazlan Md Zain, Ramli Adnan, and Abd Manan Samad. Flood water level prediction and tracking using particle filter algorithm. 2012. In Proceedings of 8th IEEE International Colloquium Signal Processing and its Applications (CSPA), (March 2012), 431-435. IEEE Press. DOI:

10.1109/CSPA.2012.6194763

[33] Chris Synder, Thomas Bengtsson, Peter Bickel, and Jeff Anderson. 2008. Obstacles to high-dimensional particle filtering. Mon. Wea. Rev. 136, 12 (December 2008), 4629-4640. DOI:http://dx.doi.org/10.1175/2008MWR2529.1

[34] Geir Evensen. 1994. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. J. Geophys. Res. 99, C5 (1994), 10143. DOI:http://dx.doi.org/10.1029/94JC00572

[35] P.L. Houtekamer and H.L. Mitchell. 1998. Data assimilation using an ensemble Kalman filter technique. Mon. Weather Rev. 126, 3 (1998), 796–811. DOI:http://dx.doi.org/10.1175/1520-0493(1998)126<0796:DAUAEK>2.0.CO;2

[36] Peter Jan van Leeuwen. 1999. Comment on "Data Assimilation Using an Ensemble Kalman Filter Technique." Mon. Weather Rev. 127, 6 (1999), 1374–1377. DOI:http://dx.doi.org/10.1175/1520-0493(1999)127<1374:CODAUA>2.0.CO;2

[37] Thomas M. Hamill, Jeffrey S. Whitaker, and Chris Snyder. 2001. Background Error Covariance Estimates Simple Properties of Covariance Matrices From. , 303 (2001).

[38] D.J. Patil, Brian R. Hunt, Eugenia Kalnay, James a. Yorke, and Edward Ott. 2001. Local low dimensionality of atmospheric dynamics. Phys. Rev. Lett. 86, 26 I (2001), 5878–5881. DOI:http://dx.doi.org/10.1103/PhysRevLett.86.5878

[39] Edward Ott et al. 2004. A local ensemble Kalman filter for atmospheric data assimilation. *Tellus, Ser. A Dyn. Meteorol. Oceanogr.* 56, 5 (2004), 415–428. DOI:http://dx.doi.org/10.1111/j.1600-0870.2004.00076.x

[40] Brian R. Hunt, Eric J. Kostelich, and Istvan Szunyogh. 2007. Efficient data

assimilation for spatiotemporal chaos: A local ensemble transform Kalman filter. Phys. D Nonlinear Phenom. 230, 1-2 (2007), 112–126. DOI:http://dx.doi.org/10.1016/j.physd.2006.11.008

[41] Pavel Sakov and Laurent Bertino. 2011. Relation between two common localisation methods for the EnKF. Comput. Geosci. 15, 2 (2011), 225–237. DOI:http://dx.doi.org/10.1007/s10596-010-9202-6

[42] Jing Lei and Peter Bickel. 2011. A Moment Matching Particle Filter for Nonlinear Non-Gaussian Data Assimilation Jing Lei and Peter Bickel. Mon. Weather Rev. (2011), 0–37.

[43] Jonathan Poterjoy. 2015. A localized particle filter for high-dimensional nonlinear systems. Mon. Weather Rev. , 2001 (2015), 151020092806005. DOI:http://dx.doi.org/10.1175/MWR-D-15-0163.1

[44] Jonathan Poterjoy and Jeffrey L. Anderson. 2016. Efficient assimilation of simulated observations in a high-dimensional geophysical system using a localized particle filter. Mon. Weather Rev. , 2001 (2016), MWR–D–15–0322.1. DOI:http://dx.doi.org/10.1175/MWR-D-15-0322.1

[45] S.G. Penny and T. Miyoshi. 2015. A local particle filter for high dimensional geophysical systems. Nonlinear Process. Geophys. Discuss. 2, 6 (2015), 1631–1658. DOI:http://dx.doi.org/10.5194/npgd-2-1631-2015O

[46] Matthias Morzfeld, Daniel Hodyss, and Chris Snyder. 2016. What the collapse of the ensemble Kalman filter tells us about particle filters. *arXiv preprint* arXiv:1512.03720v2 [math.NA]

[47] John MacCormick and Andrew Blake. 2000. A Probabilistic Exclusion Principle for

Tracking Multiple Objects. Int. J. Comput. Vision 39, 1 (August 2000), 57-71. DOI:http://dx.doi.org/10.1023/A:1008122218374

[48] John MacCormick and Michael Isard. 2000. Partitioned Sampling, Articulated Objects, and Interface-Quality Hand Tracking. In Proceedings of the 6th European Conference on Computer Vision-Part II (ECCV '00), David Vernon (Ed.). Springer-Verlag, London, UK, UK, 3-19.

[49] Bruno Cedraz Brandao, Jacques Wainer, and Siome Klein Goldenstein. 2006. Subspace hierarchical particle filter. 2006. In Proceedings of the 19th Brazilian Symposium on Computer Graphics and Image Processing(SIBGRAPI'06), (Oct. 2006) 194-204. IEEE, Manaus. DOI:10.1109/SIBGRAPI.2006.42

[50] Jonathan Deutscher and Ian Reid. 2005. Articulated Body Motion Capture by Stochastic Search. Int. J. Comput. Vision 61, 2 (February 2005), 185-205. DOI:http://dx.doi.org/10.1023/B:VISI.0000043757.18370.9c

[51] Petar M. Djuric, Ting Lu, and Mónica F. Bugallo. 2007. Multiple particle filtering. In Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing,2007(ICASSP 2007). IEEE, Honolulu, pp.III-1181-III-1184. DOI:10.1109/ICASSP.2007.367053

[52] D.J. Patil, Brian R. Hunt, Eugenia Kalnay, James a. Yorke, and Edward Ott. 2001. Local low dimensionality of atmospheric dynamics. Phys. Rev. Lett. 86, 26 I (2001), 5878–5881. DOI:http://dx.doi.org/10.1103/PhysRevLett.86.5878

[53] Patrick Rebeschini and Ramon Van Handel. 2015. Can local particle filters beat the curse of dimensionality? *Ann. Appl. Probab.* 25, 5 (2015), 2809–2866.

DOI:http://dx.doi.org/10.1214/14-AAP1061

[54] Fasheng Wang, Zhao Qingjie, Yingbo Zhang, and Liqun Zhang. 2008. Particle Filtering with Multi Proposal Distributions. Int'l J. of Communications, Network and System Sciences 1, 1 (2008) 22-28. DOI:10.4236/ijcns.2008.11004

[55] H. Xue and X. Hu. An effective proposal distribution for sequential Monte Carlo methods-based wildfire data assimilation". *2013 Winter Simulations Conference (WSC)*, Washington, DC, 2013, pp. 1938-1949.DOI: 10.1109/WSC.2013.6721573

[56] Bolic, Miodrag; Djuric, P.M.; Sangjin Hong, "Resampling algorithms and architectures for distributed particle filters," in Signal Processing, IEEE Transactions on , vol.53, no.7, pp.2442-2450, July 2005. DOI: 10.1109/TSP.2005.849185

[57] Balasingam, Balakumar; Bolic, Miodrag; Djuric, P.M.; Miguez, Joaquin, "Efficient distributed resampling for particle filters," in Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on , vol., no., pp.3772-3775, 22-27 May 2011. doi: 10.1109/ICASSP.2011.5947172

[58] Fan Bai, Feng Gu, Xiaolin Hu, and Song Guo. 2014. Particle Routing in Distributed Particle Filters for Large-scale Spatial Temporal Systems. IEEE Trans. Parallel and Distributed Systems, PP, 99 (2014). DOI:10.1109/TPDS.2015.2405912

[59] Shabany, M.; Gulak, P.G., "An efficient architecture for distributed resampling for high-speed particle filtering," in Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on , vol., no., pp.4 pp.-3425, 21-24 May 2006. doi: 10.1109/ISCAS.2006.1693361

[60] Lewis Ntaimo, Xiaolin Hu, and Yi Sun. 2008. DEVS-FIRE: Towards an Integrated

Simulation Environment for Surface Wildfire Spread and Containment. *Simulation* 84, 4 (April 2008), 137-155. DOI:http://dx.doi.org/10.1177/0037549708094047

[61] Xiaolin Hu, Yi Sun, and Lewis Ntaimo. 2012. DEVS-FIRE: design and application of formal discrete event wildfire spread and suppression models. *Simulation* 88, 3 (March 2012), 259–279. DOI:http://dx.doi.org/10.1177/0037549711414592

[62] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. 2000. Theory of Modeling and Simulation, 2nd Edition. Academic Press

[63] Richard C. Rothermel. 1972. A mathematical model for predicting fire spread in wildland fuels. *USDA For. Serv. Res. Pap. INT USA* , INT-115 (1972), 40.

[64] Arnaud Doucet, Nando De Freitas, and Neil Gordon. 2001. An introduction to sequential Monte Carlo methods. Sequential Monte Carlo methods in practice. Springer, New York. DOI:10.1007/978-1-4757-3437-9_1

[65] Van Leeuwen, Peter Jan. 2009. Particle filtering in geophysical systems. Mon. Wea. Rev. 137, 12 (December 2009), 4089-4114. DOI:http://dx.doi.org/10.1175/2009MWR2835.1

[66] Patrick Rebeschini and Ramon Van Handel. 2015. Can local particle filters beat the curse of dimensionality? *Ann. Appl. Probab.* 25, 5 (2015), 2809–2866. DOI:http://dx.doi.org/10.1214/14-AAP1061

[67] Yuan Long, Xiaolin Hu. 2015. SpSIR: A Spatially-Dependent Sequential Importance Resampling For High Dimensional Spatial Temporal System Simulation. In Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative (DEVS '14). Society for Computer Simulation International, 7 pages.

[68] Yaakov Bar-Shalom and Thomas E Fortmann. 1988. Tracking and Data Association. Mathematics in Science and Engineering, vol. 179. Academic Press.

[69] Pavlina Konstantinova, Alexander Udvarev, and Tzvetan Semerdjiev. 2003. A study of a target tracking algorithm using global nearest neighbor approach. *Proc. 4th Int. Conf. Comput. Syst. Technol. E-Learning - CompSysTech '03* (2003), 290–295. DOI:http://dx.doi.org/10.1145/973620.973668

[70] Ingemar J. Cox and Sunita L. Hingorani. 1996. Efficient implementation of Reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE Trans. Pattern Anal. Mach. Intell.* 18, 2 (1996), 138–150. DOI:http://dx.doi.org/10.1109/34.481539

[71] N M Patrikalakis, J J McCarthy, A R Robinson, H Schmidt, C Evangelinos, P J Haley, S Lalis, P F J Lermusiaux, R Tian , "Towards a Dynamic Data Driven System for Rapid Adaptive Interdisciplinary Ocean Forecasting," F. Darema 780 (Ed.), Invited paper in Dynamic Data-Driven Application Systems.

[72] Beth Plale, Dennis Gannon, Dan Reed, Sara Graves, Kelvin Droegemeier, Bob Wilhelmson, and Mohan Ramamurthy. 2005. "Towards dynamically adaptive weather analysis and forecasting in LEAD." In Proceedings of the 5th international conference on Computational Science - Volume Part II (ICCS'05), Vaidy S. Sunderam, Geert Dick Albada, Peter A. Sloot, and Jack J. Dongarra (Eds.), Vol. Part II. Springer-Verlag, Berlin, Heidelberg, 624-631.

[73] Manish Parashar, Vincent Matossian, Wolfgang Bangerth, Hector Klie, Benjamin Rutt, Tahsin Kurc, Umit Catalyurek, Joel Saltz, and Mary F. Wheeler. 2005. "Towards

dynamic data-driven optimization of oil well placement." In Proceedings of the 5th international conference on Computational Science - Volume Part II (ICCS'05), Vaidy S. Sunderam, Geert Dick Albada, Peter A. Sloot, and Jack J. Dongarra (Eds.), Vol. Part II. Springer-Verlag, Berlin, Heidelberg, 656-663.

[74] R. M. Fujimoto, R. Guensler, M. Hunter, H.-K. Kim, J. Lee, J. Leonard III, M. Palekar, K. Schwan, and B. Seshasayee, "Dynamic Data Driven Application Simulation of Surface Transportation Systems," Workshop on Dynamic Data Driven Application Simulations, pp. 425-432, June 2006.

[75] Jan Mandel, Lynn S. Bennethum, Mingshi Chen, Janice L. Coen, Craig C. Douglas, Leopoldo P. Franca, Craig J. Johns, Minjeong Kim, Andrew V. Knyazev, Robert Kremens, Vaibhav Kulkarni, Guan Qin, Anthony Vodacek, Jianjia Wu, Wei Zhao, and Adam Zornes. 2005. "Towards a dynamic data driven application system for wildfire simulation." In Proceedings of the 5th international conference on Computational Science - Volume Part II (ICCS'05), Vaidy S. Sunderam, Geert Dick Albada, Peter A. Sloot, and Jack J. Dongarra

[76] Pravia, M.A.; Prasanth, R. K.; Arambel, P.O.; Sidner, C.; Chee-Yee Chong, "Generation of a fundamental data set for hard/soft information fusion," Information Fusion, 2008 11th International Conference on , vol., no., pp.1,8, June 30 2008-July 3 2008

[77] Yuan Long and Xiaolin Hu. 2014. Dynamic data driven simulation with soft data. In Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative (DEVS '14). Society for Computer Simulation International, San Diego, CA, USA, , Article 16 , 8 pages.

[78] M. A. Pravia, R. K. Prasanth, P. O. Arambel, C. Sidner and C. Y. Chong. Generation of a fundamental data set for hard/soft information fusion. 2008. 11th International Conference on Information Fusion, Cologne, 2008, pp. 1-8.

[79] B. Khaleghi and F. Karray, Random set theoretic soft/hard data fusion framework, in *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, no. 4, pp. 3068-3081, October 2014.doi: 10.1109/TAES.2014.120621

[80] Minghao Wang, Xiaolin Hu, Data assimilation in agent based simulation of smart environments using particle filters, Simulation Modelling Practice and Theory, Volume 56, August 2015, Pages 36-54, ISSN 1569-190X, http://dx.doi.org/10.1016/j.simpat.2015.05.001.

[81] Higuchi, Tomoyuki. Monte Carlo filter using the genetic algorithm operators. *Journal of Statistical Computation and Simulation* 59.1 (1997): 1-23.

[82] Tiancheng Li, Shudong Sun, Tariq Pervez Sattar, and Juan Manuel Corchado. 2014. Review: Fight sample degeneracy and impoverishment in particle filters: A review of intelligent approaches. Expert Syst. Appl. 41, 8 (June 2014), 3944-3954. DOI=http://dx.doi.org/10.1016/j.eswa.2013.12.031

[83] Johansen, Adam Michael. Some non-standard sequential Monte Carlo methods and their applications. Diss. University of Cambridge, 2006.