12-18-2014

Distributed Particle Filters for Data Assimilation in Simulation of Large Scale Spatial Temporal Systems

Fan Bai

Distributed Particle Filters for Data Assimilation in Simulation of Large Scale Spatial Temporal Systems

by

FAN BAI

Under the Direction of Prof. Xiaolin Hu

ABSTRACT

Assimilating real time sensor into a running simulation model can improve simulation results for simulating large-scale spatial temporal systems such as wildfire, road traffic and flood. Particle filters are important methods to support data assimilation. While particle filters can work effectively with sophisticated simulation models, they have high computation cost due to the large number of particles needed in order to converge to the true system state. This is especially true for large-scale spatial temporal simulation systems that have high dimensional state space and high computation cost by themselves. To address the performance issue of particle filter-based data assimilation, this dissertation developed distributed particle filters and

applied them to large-scale spatial temporal systems. We first implemented a particle filter-based data assimilation framework and carried out data assimilation to estimate system state and model parameters based on an application of wildfire spread simulation. We then developed advanced particle routing methods in distributed particle filters to route particles among the Processing Units (PUs) after resampling in effective and efficient manners. In particular, for distributed particle filters with centralized resampling, we developed two routing policies named *minimal transfer particle routing policy* and *maximal balance particle routing* policy. For distributed PF with decentralized resampling, we developed a hybrid particle routing approach that combines the global routing with the local routing to take advantage of both. The developed routing policies are evaluated from the aspects of communication cost and data assimilation accuracy based on the application of data assimilation for large-scale wildfire spread simulations. Moreover, as cloud computing is gaining more and more popularity; we developed a parallel and distributed particle filter based on Hadoop & MapReduce to support large-scale data assimilation.

INDEX WORDS: Large-scale spatial temporal systems, Distributed particle filters, Routing and layout, Simulation performance, Hadoop & MapReduce.

DISTRIBUTED PARTICLE FILTERS FOR LARGE-SCALE SPATIAL TEMPORAL SYSTEMS

by

FAN BAI

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

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2014

DISTRIBUTED PARTICLE FILTERS FOR LARGE-SCALE SPATIAL TEMPORAL SYSTEMS

by

FAN BAI

Committee Chair: Xiaolin Hu

Committee: Anu Bourgeois

Yi Pan

Gengsheng Qin

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

December 2014

DEDICATION

This dissertation is dedicated to my parents Guofu Bai and Cuirong Yuan for their endless support, love and passion for my academics. I cannot finish my Ph.D. without your love and encouragement. Also dedicated to my wife Wei Sun and all my family members, I love all of you!

ACKNOWLEDGEMENTS

Firstly, I would like to express my deep and sincere gratitude to my research supervisor, Dr. Xiaolin Hu, for giving me the opportunity to do research and providing invaluable guidance throughout this research at Georgia State University. His dynamism, vision, sincerity and motivation have deeply inspired me. He has taught me the methodology to carry out the research and to present the research works as clearly as possible. It was a great privilege and honor to work and study under his guidance. I am extremely grateful for what he has offered me.

I would also like to thank my committee members, Dr. Anu Bourgeois, Dr. Yi Pan and Dr. Gengsheng (Jeff) Qin for serving as my committee members even at hardship. I also want to thank you for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you.

I would especially like to thank my colleagues from SIMS lab: Dr. Song Guo, Dr. Yi Sun, Dr. Feng Gu, Minghao Wang, Haidong Xue, Yuan Long, Sanish Rai, Nicholas Keller, Dan Jiang and Peisheng Wu for the useful discussion of research ideas.

Last but not least, my most sincere thanks as always, to my parents Guofu Bai and Cuirong Yuan for their strong encouragement, unconditional love and belief in me. Thank you for supporting me every step of the way and for constantly being there for me – I am greatly indebted to you. I could not thank you enough for all that you have done for me and continue to do. Special gratitude goes to my wife Wei Sun for the encouragement and support for my research, I love you!!!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Data assimilation for large scale spatial temporal simulations

Large-scale spatial temporal systems such as wildfire, road traffic and flood evolve system behavior in both space and time. Those systems have significant impact on both ecosystems and human society. Wildfires cause massive losses of natural forest resources, endangered species, properties, and even human lives. In the 2007 wildfire season, over 85,500 fires across the whole US burned more than 9.3 million acres of land. It cost 1.8 billion dollars in effort to fight wildfires and a potential 2.5 billion dollars in insured loss for California alone [1]. The Insurance Council of Texas estimates that 2011 was the costliest year for wildfires in Texas with insured losses estimated at more than $500 million. In addition, insured loss estimates from the Bastrop Complex Wildfire in the state have reached $325 million due to the destruction of more than 1,600 homes, becoming the costliest wildfire in the state's history [2]. In 2012, 20 large wildfires were burning in eight Western states, from Idaho and Wyoming to California and Arizona, according to reports from the U.S. Forest Service. Federal firefighting costs passed $1 billion for the first time in 2000 and have exceeded that mark every year but two. Together, the Forest Service and Interior Department have averaged $1.54 billion in fire suppression in the past decade. States pay another $1 billion to $2 billion annually, according to Headwaters Economics, a Bozeman, Montana-based research group. Fires affected about 7.3 million acres a year in the most recent decade, up 66 percent from the previous 10 years [3].

Road traffic has become a serious problem with rapid development of the economy. The increasing traffic flow is resulting in serious congestion of urban road networks, which can decrease flow rate, delay travel time, increase fuel consumption and travel costs, and create negative environmental effects [4]. The Texas Transportation Institute estimated that, in 2000,

the 75 largest metropolitan areas experienced 3.6 billion vehicle-hours of delay, resulting in 5.7 billion U.S. gallons (21.6 billion liters) in wasted fuel and $67.5 billion in lost productivity, or about 0.7% of the nation's GDP. It also estimated that the annual cost of congestion for each driver was approximately $1,000 in very large cities and $200 in small cities. Traffic congestion is increasing in major cities and delays are becoming more frequent in smaller cities and rural areas [5]. Floods are one of the few disasters that have the most extensive influence, the most frequent occurrence and the most severe losses [6]. The effects of floods include loss of life and damage to buildings and other structures, including bridges, sewerage systems, roadways, and canals. In order to effectively manage those systems, several major large-scale spatial temporal systems simulation research investigations have been performed. For example, several major models were developed for wildfire simulation, such as FARSITE [7], BehavePlus [8], and DEVS-FIRE [9] and Hfire [10]. For road traffic simulation, the work of [11] propose a set of methods aiming at extracting large scale features of road traffic, both spatial and temporal, based on local traffic indexes computed either from fixed sensors or floating car data and the work of [12] had shown the simulation of large spatial temporal system in flood risk estimation.

However, these systems are inherently difficult to study since the accuracy of large-scale spatial temporal systems simulations depends on many factors, such as GIS data, fuel data, weather data, and such. Moreover, due to their complex and dynamic behavior, it is very difficult to obtain all these data with no error. For example, the GIS data and fuel data which are used in simulation research contain discrepancies compared to the real data constrained by spatial resolution. This is the same situation for other data like weather data, which changes by second in the real world. Be that as it may, the weather data used in simulation models is typically obtained from local weather stations in a time-based manner such as every ten minutes to thirty

minutes. Thus, the weather is considered as unchanged in the simulation model until the next data arrives. With those kinds of errors, the predictions from the simulation model will be different from what occurs in real large-scale spatial temporal systems. Therefore, without assimilating data from the real large-scale spatial temporal systems and dynamically adjusting the simulation, the difference between the simulation and the real large-scale spatial temporal systems are likely to continue to grow.

Data assimilation is an analysis technique, in which the observed data is assimilated into the model to produce a time sequence of estimated system states [13]. Although data assimilation has been widely used in areas such as atmospheric, climate, and ocean modeling [14] [15], assimilating data in larger-scale spatial temporal systems simulation is still difficult to study because of the complexity of models. Additionally, the number of possible state variables and model parameters is extremely large, and many of them are spatially dependent. Moreover, another noteworthy complexity is associated with the nonlinear, non-Gaussian behavior of those models which makes it ineffective to use conventional inference techniques such as Kalman filter. Motivated by these problems, we select particle filter methods to support the data assimilation of large-scale spatial temporal systems. Particle filters (PFs) are a set of simulation based methods which provide a convenient and attractive approach to computing posterior distributions [16]. Particle filter estimation requires no assumptions about the state distribution or the state-space model components as nonlinear evolution and observation equations that have non-normal error distributions are allowed [17]. There are three major operations in PF processing: particle (or sample) generation, weight calculation, and resampling. Firstly, samples are generated from the space of unobserved states, and then probability masses associated with the particles are calculated. Finally, researchers undergo the process of removing particles with

small weights and replacing them with particles with large weight. Since PFs are very suitable for non-linear and/or non-Gaussian applications and also show great promise in addressing a wide variety of complex problems, they have already been widely used in many research areas such as wireless communications [18], robotics [19], navigation [20] and tracking systems [21] [22].

1.2 Challenges for data assimilation for large scale spatial temporal simulations

Particle filters provide a well-established methodology for generating samples from the prediction and filter distributions without requiring assumptions about the state space model or the state distributions. The evolution and observation equations can be nonlinear and the initial state and noise distributions can take any form required. However, particle filters do not perform very well when applied to high dimensional systems. Because weight disparity increases with increasing state and likelihood dimension, leading to severe weight collapse. Weight collapse can be mitigated by including a resampling step before weights become too uneven, but for high dimensional systems weight collapse can occur in a single time-step, rendering resampling completely ineffective [17]. When the observations are high dimension, the filter ensemble collapses to a small number of distinct points, providing very poor estimates. For example, geophysical systems such as the atmosphere or the oceans [23] are characterized by large state spaces which are nonlinear, especially in high resolution applications. It is shown that direct application of the basic particle filter, importance sampling using the former as the importance density, does not work in high-dimensional systems, but several variants are shown to have potential.

Moreover, the work of [24] also noticed that to avoid ensemble collapse, the particle population needs to increase exponentially with increasing state dimensions. Based on their

result, a nonlinear estimation problem with zero-mean unit-variance Gaussian noise, the 10^{11} particles are required for a 200-dimensional state-space. In the work of [17], Jonathan Briggs forced on the issue of high dimensional particle filtering in state-spaces where the noise distribution is meta-elliptical and proposed a location-domain particle filter which created a particle population for each component of the observation vector which greatly increased the space and time complexity of the algorithm. From the experiment, his proposed filter took 2100 seconds compared to the generic particle filter which took 0.034 seconds for an observation update on their test problem. When the number of observation vector components increased, the time taken by the algorithm for each observation update would increase too. This is a significant flaw since for their specific test problem with hundred observations a generic particle filter took approximately 4 seconds to run, while their proposed location-domain particle filter took approximately 60 hours [25]. This is also especially true for the wild fire simulation system where a large number of particles are needed in order for the data assimilation methods to converge to true system states. The state of wild fire simulation system is very expansive. This is because the state of wild fire simulation system may include many data such as fuel data, GIS data, weather data, etc. This will assuredly and significantly increase the computation costs and communication costs when applying PFs in wild fire simulation system.

Based on the problems we point our above, some major types of methods dealing with PF-based high dimensional data assimilation, like particle smoothers, have already discussed as well. Particle smoothers are similar to particle filters except that they use observations available before and after the current time point in making their state estimates. In the work of [17], the author uses a particle smoother defined on a sequence of locations (rather than the traditional sequence of time points) to carry out the Bayesian update. Considering only one location at a

time in the smoother reduces the dimensionality of the problem, avoiding filter ensemble collapse. An experiment showed a particle smoother update was applied to the same BATS model and observations as has been shown to lead to filter ensemble collapse (more details can be found in [17]). Another method named The Merging Particle Filter which was introduced by Nakano [26]; the main idea of this method is: linear combinations of the particles are taken at the measurement time to reduce the variance in the weights. The author compared the performance of the merging particle filter to the particle filter with resampling and to the EnKF for the Lorenz 63 and 96 models. They note that the EnKF works best with a low number of particles, but increasing the number of particles the Merging Particle Filter takes over and only with a very high number of particles is the particle filter with resampling superior [23]. Moreover, particle filter methods are very flexible, easy to implement, parallelizable and applicable in a variety of settings. Therefore, there are several distributed/parallel particle filters (DPFs) that have been developed [27] [28] [29] [30]. In these algorithms, the distributed nature is achieved by either transmitting local statistics of particles to a centralized unit or by using the message passing method.

1.3 Distributed particle filter and particle routing

According to the particle filter' processing, the first two parts, particle generation and weight computation are simple to parallel and distribute, since every particle can work independently. The bottleneck in real-time PF implementation is the resampling operation. That is because resampling cannot be computed unless data from all particles are available. On the other hand, the resampling step is very critical in every implementation of particle filtering because the variance of particle weights quickly increases without it. Therefore, the particles can be run independently on different working processor units (PUs) during the particle generation

step and weight computation step, but the PUs must be combined together in a central processor unit (CU) in order to perform the resampling step. That means resampling creates a significant amount of communication at every time step of filtering and prevents the particle filter from being parallelized efficiently. After the resampling step, another important step to do is particle routing. Particle routing is necessary because the numbers of particles on different PUs are unbalanced after resampling. Thus, PUs that have a surplus of particles need to route the extra particles to the PUs with a shortage of particles for the next iteration of computing. Particle routing deals with selecting particles on some PUs and routing them to other PUs across the network. In distributed PFs, routing particles among PUs can serve two different purposes: 1) to help the "good" particles, i.e., particles with high weights, to propagate among the PUs and thus potentially to lead to better estimation results; 2) to ensure that the different PUs have the same number of particles (i.e., load balance) after resampling.

The traditional method to handle the parallel and/or distributed computing method for particle filter required every detail, such as how to connect each PU and CU, what kind the communication method used inside each PU and CU, what is the computer network protocol will be used and etc. to be finished as well. Nowadays, a new technology can help us easily parallel/distributed the PF-based work. "Cloud Computing" is a technology that uses the internet and central remote servers to maintain data and applications. Cloud computing allows consumers and businesses to use applications without installation and access their personal files at any computer with internet access. "Cloud" refers to large Internet services running on tens of thousands of machines such as Amazon S3, Google AppEngine, Microsoft Windows Azure, etc. MapReduce is a software framework that allows developers to write programs that process massive amounts of unstructured data in parallel across a distributed cluster of processors or

stand-alone computers. Also, MapReduce is a programming model for processing huge data sets on certain kinds of distributable problems using a large number of computers (nodes), collectively referred to as a cluster [31] [32]. There are many different implementations of the MapReduce programming model, among which Apache's Hadoop is the most well-known one and it has been successfully applied for file based datasets.

1.4 Problem statement

The major difficulty of applying PFs to high dimensional data assimilation problems is its high computation cost due to the large number of particles used, where each particle represents a full-scale simulation to the next observation time. This is a huge problem especially for the centralized particle filter method, since all the particles run the simulation on a single computer, resulting in potential problems with CPU and memory costs. This is also especially true for large-scale spatial temporal systems where a large number of particles are needed in order for the data assimilation methods to converge to true system states. Also, the state of large-scale spatial temporal systems is very expansive. This is because the state of large-scale spatial temporal systems may include many data such as fuel data, GIS data, weather data, etc. This will assuredly and significantly increase the computation costs and communication costs when applying PFs in large-scale spatial temporal systems. In order to improve the performance of data assimilation, distributed/parallel particle filters are necessary. Since some research exists on improving resampling for parallel and distributed particle filter, we are authoring a literature review focusing on distributed particle filter algorithms while including a quick overview of data assimilation, particle filter and dynamic data driven application systems (DDDAS). Also, as we mentioned before, particle routing is necessary because the numbers of particles on different PUs are unbalanced after resampling. Thus, PUs that have a surplus of particles needs to route the

extra particles to the PUs with a shortage of particles for the next iteration of computing. As the number of PUs increases, the communication overhead rises. The unbalanced particles on PUs are caused by the fact that particles have different importance weights. As a result, PUs hosting high weight particles generates a lot more replicates in resampling and need to route a large number of particles to other PUs. The uneven distribution of particles' weights is common in data assimilation using PFs for spatial temporal simulations. Therefore, efficient particle routing is critical for reducing the communication costs in distributed PFs.

Figure 1.1 illustrates this situation based on a run of the bootstrap algorithm for large-scale wildfire spread simulation modeling. The figure shows the normalized weight of the three best particles out of 100 particles in each step of the data assimilation. Based on the figure, we can see the strong uneven distribution of particle's weight in almost every step. Except the first two or three steps, the three particles account for more than 80% of the overall weight of the 100 particles. (The reason why the first two or three steps did not have this same situation is because every particle's fire shape is very small in first two or three steps, so the weight of every particle is almost the same). This means in a distributed environment the PUs hosting these high weighted particles will generate a lot more replicates in resampling and need to route a large number of particles to others. According to this information, how to route particles among PUs after resampling in effective and efficient manners calls for more research.

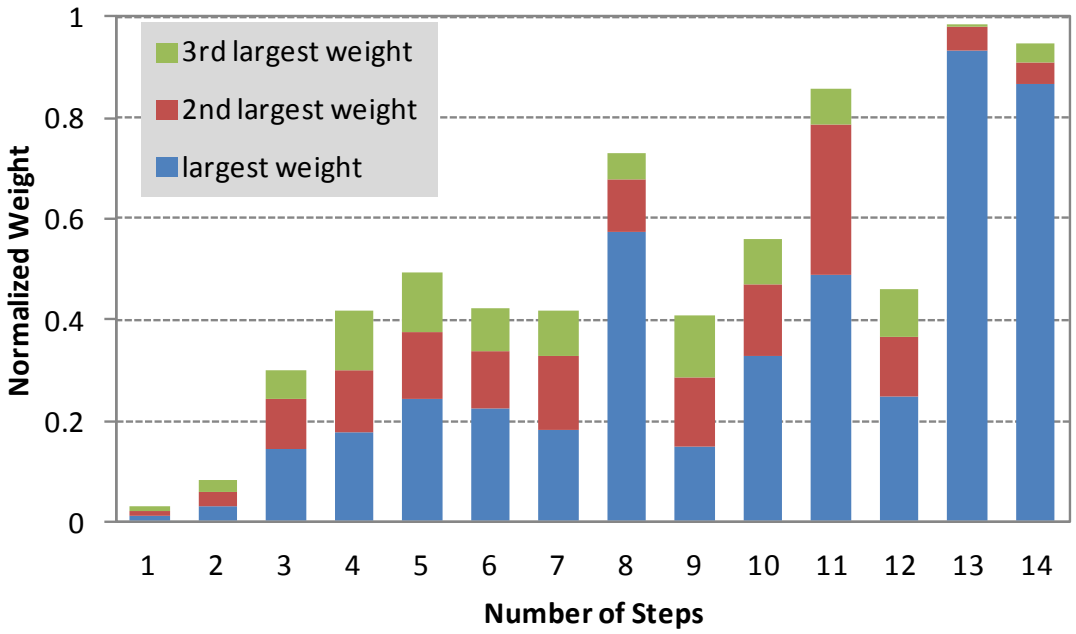


Figure 1.1 Normalized weights of the three best particles (out of 100)

On the other hand, for the centralized resampling method, we have to face some issues since we have the CU, since it still requires a complicated scheme for particle routing, and makes a complex PU design and area increase when more PU are involved. But for cloud MapReduce and Hadoop, we do not need configure every detail, such as how to connect each PU and CU, what kind of the communication methods should be used inside, what are rules between every machine, etc., because the Hadoop system can help us to do it. Our only work is finishing the PF-based application code following the MapReduce's *key/value* pairs rules. Moreover, the advantage of developing map-reduce PF is like using other "cloud" servers, for example: you do not need buy many machines yourself in advance; you just pay when you use the cloud's allocated machines. Also, you do not need upload your application work source (code) everywhere because you can just upload it to the cloud and use it when you want to do test.

1.5 My contributions

First of all, our work applied a particle filter-based data assimilation framework in wildfire spread simulation for state estimation and parameter estimation. Our work on data assimilation explored the possible applications of particle filters in wildfire spread simulation based on DEVS-FIRE model, and applied particles filters to assimilating temperature sensor data for estimating the dynamically evolving fire front of a spreading fire. On the other hand, the simulation models of large-scale spatial temporal systems rely on many parameters to model the structure and behavior of systems under study. To achieve accurate simulation results, a simulation model needs to use correct model parameters. However, it is common that during the modeling process the parameters are largely unknown, uncertain and/or vary with time or space. Therefore, we used the DEVS-FIRE wildfire spread simulation model to show the results of data assimilation based on PF for large-scale spatial temporal systems. We carried out experiments to estimate the fuel moisture content and fuel bed depth parameters used in the wildfire spread simulation.

Secondly, while several resampling algorithms [33] have been developed for distributed particle filters, less research has been conducted to investigate how to route particles among PUs after resampling in effective and efficient manners. In our work, we study the routing policies in distributed particle filters with both the centralized resampling schema and the distributed resampling schema. Based on the global information which the CU has full knowledge of the weight distribution of all particles on different PUs in the centralized resampling schema, we developed two efficient particle routing policies in distribution PF with centralized resampling, named ***minimal transfer particle routing policy*** and ***maximal balance particle routing policy***. On the distributed resampling side, communications are constrained between neighboring PUs.

This local communication schema supports a large degree of parallelism due to elimination of the centralized resampling step. However, it also results in slow propagation of high-weighted particles, and thus reduces the convergence rate of the particles. To address this issue, we propose a hybrid particle routing approach that combines global routing with local routing to take advantage of both. In this approach, we mainly use local routing to ensure scalability and low communication costs, and occasionally invoke global routing to support faster propagation of "good" particles. We evaluate and compare the different particle routing methods based on the application of data assimilation for large-scale wildfire spread simulations as well.

Moreover, using the technology of Cloud Computing, we developed a parallel and distributed computing method that uses Hadoop & MapReduce to handle the data assimilation in wildfire simulation based on particle filters. Our work build a foundation where future work can be carried out and the main experiment results showed the MapReduce-PF and Hadoop significantly increases the performance for data assimilation using large number particles.

1.6 Organization of the dissertation

Based on the structure of distributed PFs, the work will construct the entire system consisting of all the components, which will be explained later. Chapter 2 introduces the related work of data assimilation, sequential Monte Carlo methods (particle filters), dynamic data driven application systems (DDDAS), and several distributed particle filtering algorithms that have been developed in literature. Chapter 3 describes data assimilation based on PFs for large-scale spatial temporal systems, which includes an overview of DEVS-FIRE-based wildfire spread simulation and PF-based data assimilation framework. In Chapter 4 we detail the overall particle routing architecture and then describe three different particle routing policies. The experiment results and analysis for three different particle routing policies in distributed PFs with centralized

resampling will be discussed in this chapter as well. Chapter 5 will continue the introduction of particle routing in distributed PFs with decentralized resampling. Distributed resampling with local and global particle routing algorithms will be described in this chapter. In Chapter 6, based on MapReduce and Hadoop, we design a cloud MapReduce for data assimilation using sequential Monte Carlo methods in wildfire spread simulation. Finally, Chapter 7 contains conclusions and future research implications.

2 RELATED WORKS

2.1 Data assimilation

2.1.1 Overview of data assimilation

Data assimilation is the process by which observations are incorporated into a computer model of a real system [34]. The purpose of data assimilation is to use observation data to improve state estimation of the system. The data assimilation methods try to minimize the errors between the real system and the models. The data assimilation methods can be divided into three main classes [35]: 1) Empirical methods, which include Successive Correction Method (SCM), Nudging, Physical Initialization (PI) and Latent Heat Nudging (LHN). 2) Constant statistical methods, such as: Optimal interpolation (OI), 3-dimensional variational data assimilation (3DVar) and 4-dimensional variational data assimilation (4DVar). 3) Adaptive statistical methods which include Extended Kalman filter (EKF) and Ensemble Kalman filter (EnFK).

There are two main data assimilation algorithms: sequential based and cost function based. The sequential approaches are based on the Bayesian theories that combine the prior knowledge of the state vector and the measurement to obtain the posterior distribution of the state [36]. Some sequential data assimilation algorithms known as Kalman filters, Extended Kalman filter (EKF) and Ensemble Kalman filter (EnKF). A Kalman filter is an optimal estimator, which shares the static update with some of the variational techniques, but Kalman filter algorithms also explicitly compute the error covariance through an additional matrix equation that propagates error information from one update time to the next, subject to possibly uncertain model dynamics [37]. The EKF doesn't need the linear model operator and/or observation operator. The EnKF originated as a version of the Kalman filter for large problems and it is now an important data assimilation component of ensemble forecasting. Moreover, Particle filter is also called

sequential data assimilation filter that based on particle representations of probability densities, which can be applied to any state-space model and which generalize the traditional KF methods [38] [39]. For the cost function-based methods, the typical algorithms include the shuffled complex evolutionary (SCE) method [40] [41], a very fast simulated annealing (SA) algorithm [42] [43], the differential evolutionary (DE) method [44], and the genetic algorithm (GA) [45]. The common weakness of these methods is their slow computational speed, the more advantage and disadvantage can found in [36].

There are two basic approaches to data assimilation: sequential assimilation, that only considers observation made in the past until the time of analysis, which is the case of real-time assimilation systems, and non-sequential, or retrospective assimilation, where observation from the future can be used, for instance in a reanalysis exercise [46] . But it needs a statistical approach, because the Cressman analysis which is the one of simple analysis method has some disadvantages. In the statistical approach, we try to use all the useful information, but don't trust them at all. We can find a strategy to minimize the average of the differences between the analysis and the "truth" observation. In this sense, the analysis can be seen as the optimization problem. All the related errors are assumed to be unknown and have known statistical properties [47]. Note that the details of the most of algorithms above can be found in [46].

2.1.2 Application of data assimilation

Data assimilation has already widely used in many different fields, such as geosciences, weather forecasting, atmospheric, oceanic, hydrologic and other environmental systems. For example, data assimilation used for Global Positioning System (GPS) discussed as well in [48]. Hurricane initialization by data assimilation which used for National Centers for Environmental Prediction (NCEP) official hurricane track forecasts for seven Atlantic hurricanes [49]. The 3-

dimensional variational data assimilation (3DVar) and 4-dimensional variational data assimilation (4DVar) also widely used in data assimilation. The work of [50] gives some diagnosis statistical results of the assimilation surface observations with the regional GRAPES forecast and assimilation model. Chinese Meteorological Administration has developed a three dimension variational data assimilation system (Global/Regional Assimilation and Prediction System, shorten as Grapes 3Dvar), and with the ATOVS radiance data directly assimilated by RTTOV as observation operator [51]. Also, the work of [52] proposed a data assimilation system to improve ozone simulations in Mexico City basin using 3D-VAR that generated the optimal estimate of the true atmospheric state during the analysis time. In [53], the four-dimensional variational data assimilation technique (4D-VAR) is presented as a tool to forecast floods. The study is limited to purely hydrological flows and supposes that the weather, here a big rain, has been already forecasted by meteorological services. For adaptive statistical methods which include Extended Kalman filter (EKF) and Ensemble Kalman filter (EnFK) also used in some field such as data assimilation with an EKF for impact-produce shock-wave dynamics which present study represents the first attempt of applying the extended Kalman filter method of data assimilation to shock-wave dynamics induced by a high-speed impact [54].

The ensemble Kalman filter uses the nonlinear forecast model to transport the forecast-error covariance from one analysis time to the next. It therefore constitutes not only an approximation to, but also a nonlinear extension of, the standard Kalman filter. It represents a promising approach toward the goal of developing a Kalman filter–based algorithm for atmospheric data assimilation. However, for the technique to be feasible in an operational setting, a computationally efficient analysis algorithm is required [55]. The work of [56] discussed the application of the ensemble Kalman filter (EnFK) to hydrologic data assimilation

and in particular to the estimation of soil moisture from Lband microwave brightness temperature observations and their mainly results shown the EnKF significantly underestimates the forecast error variances for 100 ensemble members.

Data assimilation also widely used in the large-scale spatial temporal systems. For example in wildfire area, the work [57], the authors present an effective proposal distribution for SMC based wildfire data assimilation and in work [58] proposed an approach to estimate forest fires based on sequential Monte Carlo methods from video images. Moreover, in order to increase the accurate of the flood forecasting, [59] implement sequential data assimilation for short-term flood forecasting and parameter uncertainty assessment using grid-based spatially distributed hydrologic models. Data assimilation also used in agent based simulation, the work of [60] present a method that assimilates real time sensor data into an agent-based simulation model. The goal of data assimilation is to provide inference of people's occupancy information in the smart environment, and thus lead to more accurate simulation results. The author use particle filters to carry out the data assimilation and present some experiment results, and discuss how to extend this work for more advanced data assimilation in agent-based simulation of smart environment. The work of [61] presents a framework of behavior pattern informed data assimilation and describes the structure of this framework and focus on the task of behavior pattern detection using Hidden Markov Model. The author apply behavior pattern detection to a smart office case study example and discuss how the detected behavior pattern can inform the data assimilation in agent-based simulation of smart environments.

2.2 Sequential Monte Carlo methods (Particle Filters)

2.2.1 Overview of particle filters

Sequential Monte Carlo (SMC) methods, also called particle filters, are a set of simulation-based methods which provide a convenient and attractive approach to computing the posterior distributions [62]. There are three major operations in particle filters processing: particle (or sample) generation, weight calculation, and resampling. First of all, sampling from the space of unobserved states, then probability masses associated with the particles. Finally, process of removing particles with small weight and replace them with particles with large weights.

We can more detail the major steps of particle filters based on sampling importance resampling are described below:

Step 1: initialize N particles.

Step 2: calculate importance weights.

Step 3: normalize importance weights.

Step 4: resampling.

Step 5: predict new particles for future use.

Step 6: go to Step 2 to execute the next time step.

Based on the dynamic system, in the particle filter algorithm, step 1 initializes particles. With time advances, step 2 to step 5 are executed as shown in Figure 2.1.

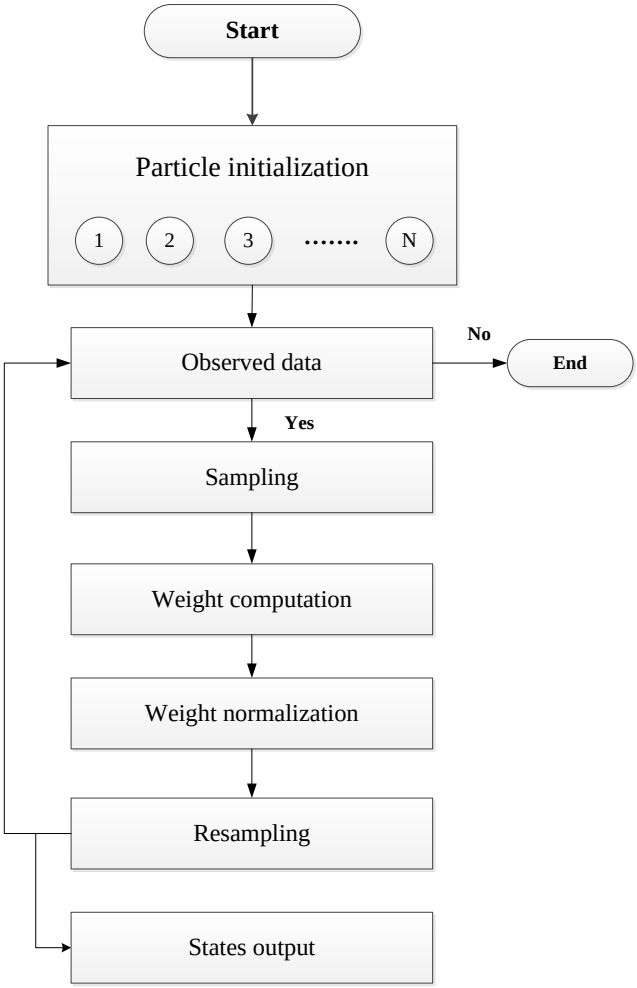


Figure 2.1 Particle filters algorithms of case study

For the general sequential importance sampling, we can formulate the sequential importance sampling method in terms of updates to the smoothing density. Based on the theory of particle filters, sample from a convenient proposal distribution $q\ (x_{1:t}\ |y_{1:t})$, then use importance sampling to modify weights [63]:

$$\int p\ (x_{1:t}\ |y_{1:t})\ f\ (x_{1:t})d_{x_{1:t}}\ = \int \frac{p\ (x_{1:t}\ |y_{1:t})}{q\ (x_{1:t}\ |y_{1:t})}\ q\ (x_{1:t}\ |y_{1:t})\ f\ (x_{1:t})d_{x_{1:t}}$$

$$\approx \sum_{i=1}^{N} w_t^i\ f(\ x_{1:t}^i)$$

where

$x_{1:t}^{i} \sim q\ (x_{1:t}\ |y_{1:t})$

and

$$w_t^i = \frac{p\ (x_{1:t}\ |y_{1:t})}{q\ (x_{1:t}\ |y_{1:t})}$$

So, we can define the un-normalized weight by:

$$\widetilde{w}_t^i = \frac{p\ (x_{1:t}\ , y_{1:t})}{q\ (x_{1:t}\ |y_{1:t})}$$

than calculate approximation to $p\ (y_{1:t})$ get:

$$p\ (y_{1:t})\ \approx \sum_{i=1}^{N} \widetilde{w}_t^i$$

Continue do the weight normalization step, the normalized weight is:

$$w_t^i = \frac{p\ (x_{1:t}\ |y_{1:t})}{q\ (x_{1:t}\ |y_{1:t})} = \frac{p\ (x_{1:t}\ ,y_{1:t})}{q\ (x_{1:t}\ |y_{1:t})}\ \frac{1}{p\ (y_{1:t})} = \frac{\widetilde{w}_t^i}{\sum_{i=1}^{N} \widetilde{w}_{t_t}^{i(i)}}$$

After we get the entire particle's weight, we can start the resampling work. Resampling is a critical operation in particle filters because with time, a small number of weights dominate the remaining weights, thereby leading to poor approximation of the posterior density and consequently to inferior estimates [64]. The idea of resampling is to remove the particle which have small weights and replace it by the particle which have big weights, so, after the resampling, the future particles are more concentrated in domains of higher posterior probability. Resampling was first introduced in [65], and later proposed for SIS [66] [67]. Resampling $j_t^{(n)}\ \sim\ a_t^{(n)}$, where $a_t^{(n)}$ is a suitable resampling function whose support is defined by the particle $x_t^{(n)}$ [68].

2.2.2 Application of particle filters

Since particle filters are very suitable for non-linear and/or non-Gaussian applications, and also show great promise in addressing a wide variety of complex problems, which have already widely used in many research areas such as, tracking application, wireless communications, robotics, mobile robot localization, computer vision and navigation. The book of [69] apply particle filters to tracking a ballistic object, detection and tracking of stealthy targets, tracking through the blind Doppler zone, bi-static radar tracking, passive ranging (bearings-only tracking) of maneuvering targets, range-only tracking, terrain-aided tracking of ground vehicles, and group and extended object tracking. The work [70] introduced a new method based on particle filters for multi-target tracking and data association in non-liner systems. This work firstly uses UKF to implement the single target tracking and then use particle filter for data association. The experiment result shown this method can reduce the algorithm execution, because the UKF need less particles compare to the traditional particle filter.

The work [71] apply particle filters for solving the problem of simultaneous localization of mobile nodes in wireless networks with correlated in time measurement noises. Several model particle filters are developed in this paper and they also evaluated performance of those model based on RSSIs by accounting for, but without considering the measurement noise time correlation. In the same research area, the work [72] introduced the results of simulations of their algorithm named ‘Monte Carlo Localization Boxed’. The paper use particle filters to improve the localization accuracy and efficiency by making better use of the information a senor node gathers which make Monte Carlo Localization more lightweight for use in wireless sensor networks. The work [19] presents two examples of used Sequential Monte Carlo methods, one in the domain of computer vision and the other in mobile robotics. The particle filter also used in image

processing area to improve the quality of the image [73]. In the biology research area, the work [74] provided and application of particle filters which populations of compact long chain polymers were created by the Monte Carlo methods to study the relationships between packing density and chain length.

2.3 Dynamic data driven application systems (DDDAS)

2.3.1 Overview of DDDAS

As we discussed in chapter 2.1, the data assimilation is the process used to incorporate observation into a simulation model of a real system. So, the DDDAS is an application or simulation that employs data assimilation that can effect and change which model or scale is used and in which the application can also steer how, when, and/or where data is collected [75]. There are three major components in a typical DDDAS: the model system, the measurement model and the data assimilation methods. We can viewed the DDDAS as a methodology to counterbalance incompleteness in model and capability to enhance the application models by imparting additional information into the model as at runtime additional data are used to selectively enhance or refine the original model. The DDDAS concept offers the promise of improving application models and methods and augmenting the analysis and prediction capabilities of application simulations and the effectiveness of measurement systems [76]. In DDDAS, the data from the sensors is fed into the simulation model to make prediction of the real systems which will treat as the measurement to evaluate the output and adjust states of the model. According to those measurements, we can evaluate, choose, or analyze the system states utilizing statistical tools, data processing, and numeric or non-numeric techniques to improve the simulation results [47].

2.3.2 Application of DDDAS

The National Science Foundation held the DDDAS workshop every year since 2000, which included numerous application scenarios which could advance both science and society by incorporating these ideas. Application areas described at the workshop include engineering (design and control), crisis management, medical, environment systems, manufacturing, business and finance [77].

The engineering part includes aircraft, oil exploration, semiconductor mfg and computing systems hardware and software design, etc. The crisis management includes transportation systems (planning, the accident response), the weather the hurricanes and floods, the wildfire and fire propagation. The medical includes customized radiation treatments, x-rays, NMR, surgery, etc. Moreover, the other part includes Supply Chain (Production Planning and Control) and Financial Trading (Stock Mkt, Portfolio Analysis). The work of [78] enable and promote active health monitoring, failure prediction, aging assessment, informed crisis management and decision support for complex and degrading structural engineering systems based on dynamic data driven methods. The work of [79] applies the DDDAS to monitor and manage surface transportation systems which composed of a heterogeneous collection of in vehicle, roadside and traditional computation and sensor node that mush analyze current system states, predict future states and rapidly adapt to unexpected disruptive event on short time scales. In work [80], a full DDDAS is proposed for dynamically estimation a concentration plume and planning optimal paths for unmanned aerial vehicles (UAVs) equipped with environmental sensors. The proposed DDDAS framework also creates solutions for efficient data collection and real-time vehicle control.

In work of [81] describes a DDDAS for coastal and environmental applications and the author coupled the real time sensor information with the water circulation model to forecast the emergency event of hurricane. The work [82] develop a dynamic data-driven planning and control system for laser treatment of cancer which include a general mathematical framework and a family of mathematical and computational models of bio-heat transfer, tissue damage, and tumor viability. The methodologies to be implemented involve uncertainty quantification methods designed to provide an innovative, data-driven, patient-specific approach to effective cancer treatment. An application of DDDAS also applied for wildfire simulation in [83] which the authors incorporate the real time data into the wildfire simulation model in order to improve the simulation results. The DDDAS also can applied in business area, in work of [84] introduced DDDAS concept to construct the model by the input data from the company which will give the employees the multiple choose to make their decisions.

2.4 Distributed particle filters

2.4.1 The centralized distributed particle filters

Figure 2.2 shows the architecture of distributed particle filter. There are multiple working processors units (PUs) and one central processor unit (CU) in distributed particle filter architecture. In distributed particle filter, the first two parts: particle generation and weight computation are simple to parallel and distribute, since every particle can work independently. In this most of basic way, it used centralized resampling which particles generate and calculate the weight in each PU then each PU have to send all the particle state and the weight to the CU in order to do the resampling. Finally, CU need send back all the state to PU again to finish after the resampling. So, this method will cause the huge communication cost between the PU and CU since the entire particle's state have to travel once between PU and CU in every step. Even in the

fully connected network, the scalability of the implementation is significantly affected by the sequential resampling and particle routing. One version of centralized resampling which is implemented on a network of personal computers is described in [33].

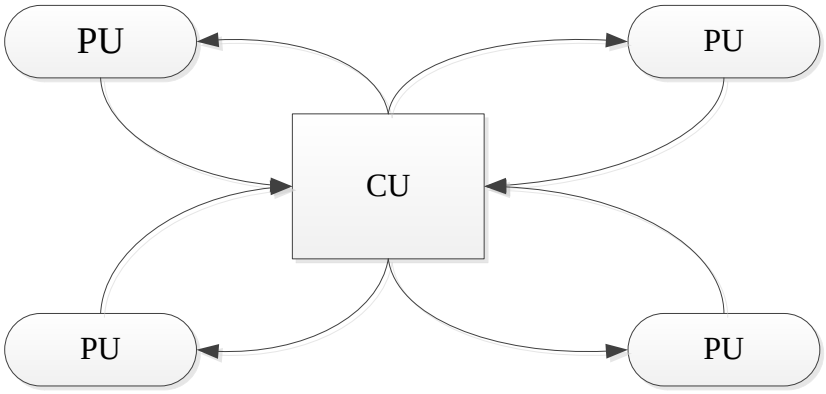


Figure 2.2 Architecture of the distributed particle filter

This centralized resampling method also will cause a big problem for large-scale spatial temporal systems. One feature of large-scale spatial temporal systems is that the state of these systems is large and has high dimension. Thus the particles representing system state have large size, and involve high communication cost when passed from one processing node to another. So, the basic centralized distributed particle filter method will cause the huge communication cost for the large-scale spatial temporal systems compare to the other systems. Therefore, to achieve the minimum execution time, some works show that resampling can be distributed to PU and that CU is then responsible only for a small portion of resampling [33][85][86][87][88]. In this chapter, we will mainly review two of the important method: distributed resampling with proportional allocation (RPA) and distributed resampling with now-proportional allocation (RNA).

2.4.2 Distributed resampling with proportional allocation

Compare to the centralized distributed particle filter, the RPA [33] method divided the sampling space to K disjoint areas at first, and each area corresponds to a PU. Since the proportional allocation is used in each area, the more samples are drawn from the strata with larger weights. In the CU side, the RPA do an extra work named "inter-resampling" which means the number of particle replicates is computed use the method residual systematic resampling (RSR) [88] after the weights of area are known. Under the inter-resampling method, every PU can be treated as the single particles. Thus the input of CU in there is w_k which means the sum of weight for all the particles in each PU. This is the first different part compare to the centralized distributed particle filter, because centralized distributed particle filter sends all the particle's weight and state to the CU. On the other hand, the output of the CU after inter-resampling is the number of particles that each PU will produce after resampling. This is the second different part compare to the centralized distributed particle filter, because centralized distributed particle filter send all the state back to the PU. Since the RPA just send the w_k to CU side, thus this method has to do the resampling again in PU side named as "intra-resampling". The goal of intra-resampling is make the enough particle state according the number which the CU gives the PU. Figure 2.3 shows the mainly different between centralized resampling (a) and RPA (b). The abbreviations are: S-sampling, I-importance computation, R-resampling, PR-particle routing.

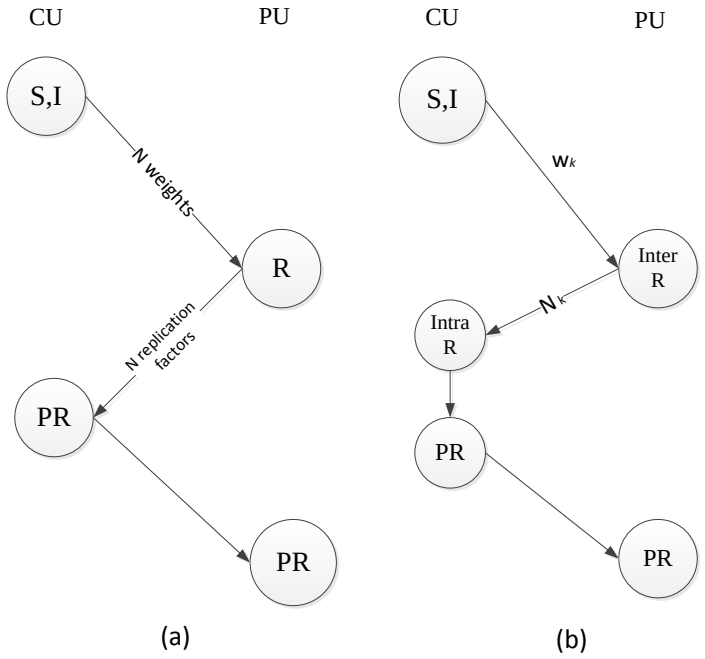


Figure 2.3 Sequence of operations performed for (a) centralized resampling and (b) RPA.

There is an example shown how the RPA method worked and the particle exchange in figure 2.4. According the RPA method, we divided the sampling space to four disjoint areas at first, which means we use 4 PUs in this example, and we assume each PU has 100 particles. After the sampling, the distribution of the normalized PU weights before resampling is shown in the figure2.4 (a). Continue do the inter-resampling, based on the weight of each PU; the number of particles that each PU will produce is 120, 120, 70 and 90. Therefore, PU_01 and PU_02 have surpluses of particles. In this example, PU_02 sends 10 particles to both PU_03 and PU_04, and PU_01 sends 20 particles to PU_03. Of course, the method also has another choose: PU_01 sends 10 particles to both PU_03 and PU_04, and PU_02 sends 20 particles to PU_03. The way of sending the particle named “particle routing”, in this example, both ways are same idea and have same communication cost, but particle routing is another side can reduce the communication cost, we will discuss the particle routing work in chapter 5.

	Weight before resampling
PU_01	0.3
PU_02	0.3
PU_03	0.175
PU_04	0.225
SUM	1

(a)

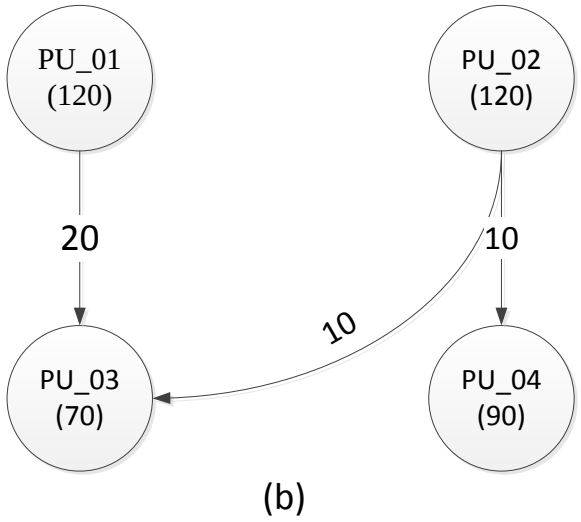


Figure 2.4 An example of particle exchange for the RPA algorithm

Simply summarize the RPA method, there are two mainly different between centralized resampling and RPA method, first of all, RPA only send the sum of the weight to CU and hold the state in each PU, but centralized resampling will send all the weight and state information to CU. So, RPA can significantly reduce the communication cost between the PU and CU because it avoids send the state. Another different is the centralized resampling only do the resampling once, but the RPA has to do twice resampling in order to make sure the number of particle's state is correct. The main advantage of distributed RPA over centralized resampling lies in reducing

the amount of deterministic communication and in the distributed resampling where the resampling is executed concurrently in the PUs instead in the CU.

2.4.3 Distributed resampling with non-proportional allocation

Although the distributed RPA method has already reduced the communication cost as well compare to the centralized distributed particle filter, it still required a complicated scheme for particle routing. Also it will make a complex PU design and area increase when more PU involve since it still needs the CU part. Moreover, the RPA is still a global pre-processing step because it still use inter-resampling in CU side, which may cause the extra delay because the CU have to get all the particle's weight in order to do the resampling. Due to those problems, the authors introduced the RNA algorithm to solve those problems. The main idea of RNA is use the term group instead of PU and no "CU" at all during the whole algorithm. The designer can deterministic and planned the particle routing, which means the term group where a group is formed from one or more PUs and no CU exist in this method. In RPA, the number of particles drawn is proportional to the weight of the stratum. On the other hand, in RNA the number of particles within a group after resampling is fixed and equal to the number of particles per group. So, full independent resampling is performed by each group. In this chapter, there are three RNA methods reviewed as well: regrouping, adaptive regrouping and local exchange [85] [86].

2.4.3.1 Distributed RNA with regrouping

The first RNA method named "distributed RNA with regrouping". First of all, the method signs all PUs to several groups, which a group is formed from one or more PUs. The first two steps particle generation and weight computation continue finished in each PU, and then resampling and particle routing are done in each group, which means the full independent resampling is performed by each group. At the next sampling instant, the PUs is rearranged so

that they form different groups. Therefore, after each time instant, regrouping is performed so that particles are exchanged among PUs and the variance is reduced. An example shown in figure 2.5, PU_01 and PU_02 form one group and PU_03 and PU_04 form another group at first step. In here, the RPA algorithm is applied to both groups in order to reduce the communication cost inside of each group, also, the particle routing just happened instead of each group. Next step, the method need rearrange the group in order to make all the "good" particle can travel to every PU. In this example, the new groups can be composed of PU_01 and PU_03in a group and PU_02 and PU_04 in a group.

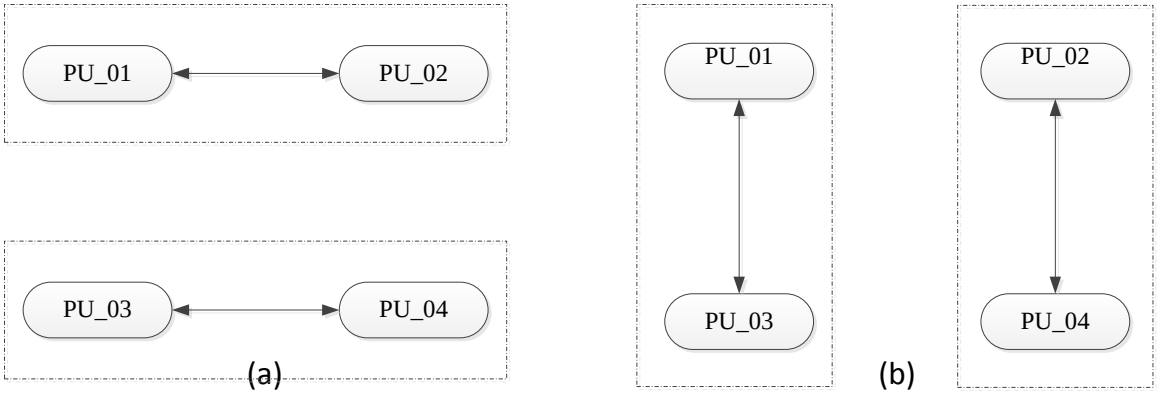


Figure 2.5 The example of RNA with regrouping

Compare to the RPA, distributed RNA with regrouping can reduce the communication cost since it does not need the CU at all, so the resampling and particle routing just need happened instead of the group. But it is not always efficacy in all situations, because it depends on the number of PUs. Because it is very difficult to decide how many PUs in a group and how many group created when the number of PUs increased. First of all, from the example above, we can easily create two groups and the two periods we can get the goal if we just have 4 PUs, since the local controllers are simple because there is only one PU with surplus and one with shortage of particles. But when we have more PUs, the situation becomes complicated, because if more than three PUs be a group, the inside of group still have the high communication cost between

each PU since we used the RPA inside of every group. Nevertheless, if we keep every group just maintain two or three PUs because we do not want to increase the communication cost inside of each group. But we still have much PUs; we have no choice but create more groups. Under this situation, we have to do more times regroup work in order to make sure the good particle can travel to each PU. Therefore, choosing so small value for the number of PU in a group could cause high distribution factor and large number of periods until full propagation of particles is achieved. In the extreme case, all the non-zero weights particles are in one PU and we still have much PUs, the distributed RNA with regrouping method will cost a long time to finish.

2.4.3.2 Distributed RNA with adaptive regrouping

Based on the distributed RNA with regrouping method, the author extend the method because they the distributed RNA with regrouping method uses the predefined fixed rules to form the groups. But omit a very important fact which we can get the distribution of the group weights. So, the method of distributed RNA with adaptive regrouping is forming the PU which has the largest weight and the PU which has the smallest weight to a group. Then the other group is formed from the remaining PUs. For each group, the RPA algorithm still applied in order to reduce the communication cost. Simply side, this method regroup the PU based every PU's weight, find the PU with the largest weight and PU with the smallest weight in every step and form these two PU to a group, then put rest of PUs to a group. For example, in Figure 2.6, we assume PU_01 and PU_04 have the largest and the smallest weights, so form that two to one group and the other group is formed from the remaining PUs (in this example is PU_02 and PU_03). The totally same rule applied in next around figure2.5 (b), we get the weight result after the first step and we know the PU_01 and PU_02 have the largest and the smallest weights, so form it to one group, and then form the rest of PU_03 and PU_04 to a group. The basic idea of

this method utilizes the Randez-Vouz load balancing algorithm [89], which is a simple greedy algorithm that associates the heavily and the lightly loaded groups. This method has the same disadvantage with distributed RNA with regrouping method, because if we face the much PUs, there are only two groups in this method: the first group just has 2 PUs which the PU with the largest weight and the PU with the smallest weight. But other group will have more than 3 PUs; it will face the same communication cost problem since we still use RPA inside of the group.

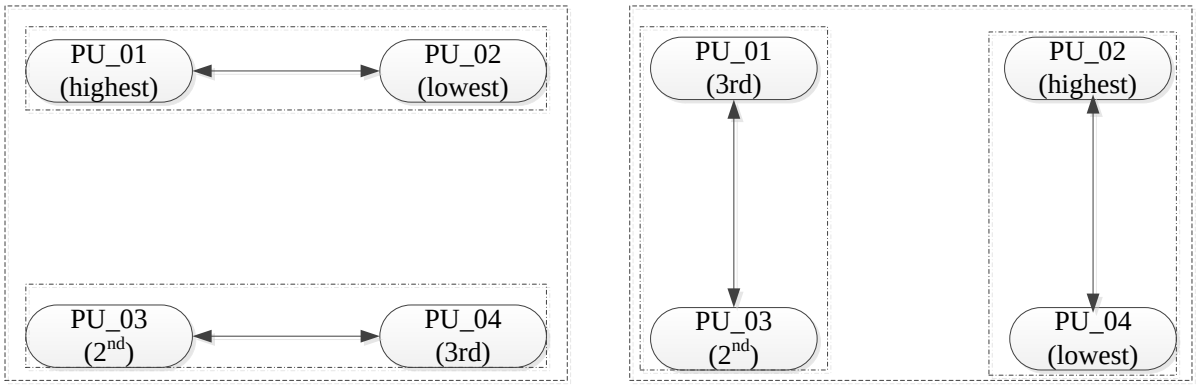


Figure 2.6 Example of RNA with adaptive regrouping

2.4.3.3 Distributed RNA with local exchange

Although the RNA with regrouping and RNA with adaptive regrouping methods did not have the CU in order to reduce the communication cost, it still apply the RPA algorithm worked inside of each group. As we discussed the limitation of those two methods, it still faces the same problem if more than 3PUs formed in one group. Moreover, those two methods also make the particle routing process is still random which cause very difficult for pipelining between the particle routing and sampling steps. So, based on those problems, the method of RNA with local exchange is introduced. In RNA with local exchange method, every PU equals a group and no RPA involved at all since only one PU in each group. So, the entire steps finished inside of the PU include the resampling part. And then the particles are exchanged in a deterministic way only among the neighboring PUs and the routing is done through local communication in every step.

Figure2.7 shows the example RNA with the local exchange. In this example, we assumed every PU has 100 particles and it is exchanges the 50 particles to their neighbor with an anti-clockwise direction after they did the resampling by themselves. The advantage of this method is the communication between the PUs is only local, so it does not need a complex PU design and the pipeline of particle routing is easy. But the local communication can increase the pass steps, the good particle have to travel every PU in order to get the full resampling in the extreme case. And this restricts the level of parallelism.

To sum of up, although the RNA solved the problem like make an easy CU design and no extra delay happened since no CU between each group, it also has two main problems: firstly, the efficacy of the RNA depend on the number of PUs. The efficacy will reduce with the more PUs involve because the RPA algorithm still applied inside of the group. On the other hand, from the simulation of result side, all the RNA method losing the accuracy, because those methods did not do the resampling together in every step.

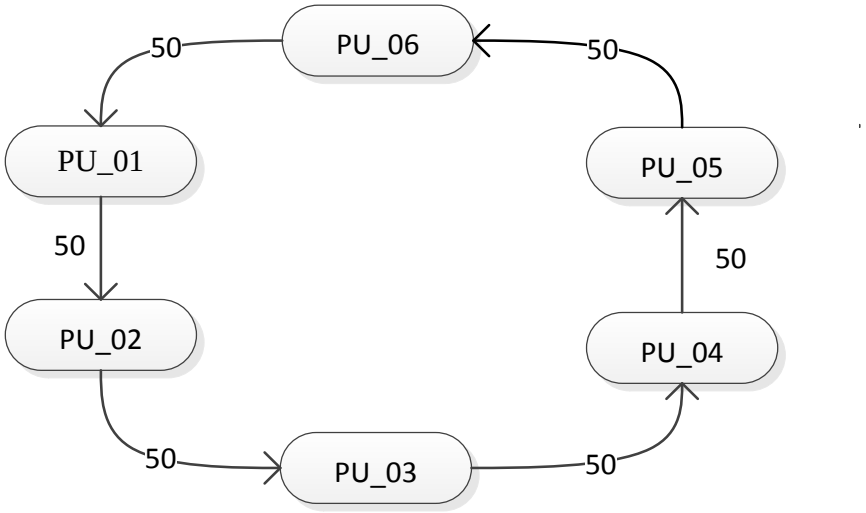


Figure 2.7 Example of RNA with local exchange

2.4.4 Compressed distributed particle filter

The work of [90] presents three schemes for distributing the computations of generic particle filter: global distributed PF (GDPF), local distributed PF (LDPF) and compressed distributed PF (CDPF). The GDPF basically same as the centralized distributed particle filter method we mentioned in 2.4.1, which sampling in every PU independently and the weight normalization and resampling performed at the CU. The LDPF has not formed to any group, the all the step include the resampling part finished in every PU. But different with the RNA with local exchange, the LDPF still has the CU exit, because each PU will sending only sufficient data to CU which is then responsible for providing the filter estimate. The main point of [90] is introducing the CDPF method. From the GDPF and LDPF method, the author found the GDPF will not loss the accurate at all but it causes the huge communication cost increase. Furthermore, the LDPF can reduce the communication cost since the resampling finished inside of every PU, but LDPF will loss the accurate because there is no centralized resampling at all. Based on those problems, the author introduced the CDPF method which can not only reduce the communication cost and also maintain a good simulation result.

For CDPF, the sampling part still finished in every PU independently and the weight normalization and resampling performed at the CU. However, the PU will not sending the entire select particle to CU, instead just send a representative distribution of reduced size. And the CU will do the resampling based on this representative distribution and send the information back to every PU. Actually, the CDPF method attempts to bypass the necessary of sending large amounts of repeated particles with representative datasets of non-repeated particles. It is based on the idea of the so called “fast bootstrap” proposed in [91], so, the resampling way is use the fast bootstrap method in CDPF. The CDPF method facilitates significantly less data exchange between the PU

and CU than direct parallelization, because this avoid to sending duplicate particles between the PU and CU. But since CDPF won't sending the whole information of select particle from PU to CU and just send a representative distribution of reduced size, so this method still was generally less accurate.

2.4.5 Distributed particle filter methods summary

Since the different distributed particle filter methods have the different properties such as the resampling step finished in PU side or CU side, the different particle routing way and performance is lose or not. So, we cannot say which method is always better than other one, because different method has its own merits and demerits. One method is better or not depends on several situations like the numbers of PU involved or what kind of application applied. Therefore, in order to choose the appropriate methods it is very necessary to summarize the key part properties of different distributed particle filter methods. In table 2.1 and table 2.2, we summarizes the 4 parts of distributed particle filter methods, such as the weight pass method from PU to CU, the resampling step finished in PU or CU (shown in table 2.1), the particle routing information and the performance situation (shown in table 2.2). The abbreviations shown in table are: CEDPF: centralized distributed particle filter; RNA-R: RNA with regroup; RNA-AR: RNA with adaptive regrouping; RNA-LE: RNA with local exchange; CODPF: compressed distributed particle filter; LDPF: local: distributed particle filter.

Table 2.1 Summarize first two parts of the different distributed PFs

	The weight pass method from PU to CU	Resampling finished in which side	
		PU	CU
CEDPF	Passed the weight of all particle		CU
RPA	Passed the sum of weight of all particle	Intra-resampling	Inter-resampling
RNA-R	Passed the weight in own group	PU, no CU	
RNA-AR	Passed the weight in selected group	PU, no CU	
RNA-LE	No weight passed	PU, no CU	
CODPF	Passed the sum of the weight	PU	
LDPF	No weight passed	PU	

Table 2.2 Summarize another two parts of the different distributed PFs

	Particle routing information	Performance loss
CEDPF	From PU to CU, then from CU back to PU	No
RPA	The PU with extra send the surplus particle to PU with shortage particles	No
RNA-R	Between their own group	Yes
RNA-AR	Between their own group	Yes
RNA-LE	Fixed with their neighbor	Yes
CODPF	From PU to CU, then from CU back to PU but without duplicate particles	Yes
LDPF	From PU to CU than from CU back to PU	Yes

3 PF-BASED DATA ASSIMILATION AND ITS APPLICATION TO WILDFIRE SPREAD SIMULATION

In this chapter, we evaluate the data assimilation based on PF for large-scale temporal systems as the wildfire spread simulation. Wildfire simulation is a very important research area in large-scale spatial temporal systems. Every year, wildfires incur sudden and rapid damages to and losses of natural forest resources, endangered species, human lives, and properties. Simulation of wildfire can provide an important tool for studying and predicting wildfire spread. Over the years, several major wildfire spread simulation models have been developed, such as FARSITE [7], BehavePlus [8], and DEVS-FIRE [9] and Hfire [10]. The used model in our work is DEVS-FIRE, an integrated wildfire spread and suppression simulation model built on Discrete Event System Specification (DEVS) formalism [92]. We will overview the DEVS-FIRE based wildfire spread simulation and go through the PF-based data assimilation framework. Then we will discuss how to apply the data assimilation for wildfire spread simulation, during which the detail steps like sampling, weight computation and resampling using DEVS-FIRE simulation are discussed and some main results also shown as well.

3.1 Overview of DEVS-FIRE-based wildfire spread simulation

DEVS-FIRE is a discrete event system specification (DEVS) model for simulating wildfire spread and suppression. Before we discuss the DEVS-FIRE model, we would better to take a look the DEVS formalism. There are two kinds of models in DEVS: atomic model and coupled model. The elements of an atomic model include: input events, output events, state variables and state transition functions, output function and time advance function. There are two different transitions in state transition functions: external transition, internal transition and confluent transition. The DEVS is a structure shown in equation (3.1):

$$M=<X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta> \quad (3.1)$$

which X is the set of input values and S is a set of states, Y is the set of output values, δ_{int}: S→S is the internal transition function. δ_{ext}: $Q \times X^b \rightarrow S$ is the external transition function, where $Q \in \{(s,e)|s \in S, 0 \leq e \leq ta(s)\}$ is the total state set, e is the time elapsed since last transition, X^b denotes the collection of bags over X. And δ_{con}: $S \times S^b \rightarrow S$ is the confluent transition function: S→ Y^b is the output function, ta:$S \rightarrow R^+_{0,\infty}$ is the time advance function.

The DEVS-FIRE model is based on the DEVS formalism and supports discrete event simulation of wildfire behavior and fire suppression tactics [9]. Figure 3.1 shows the architecture of the DEVS-FIRE.

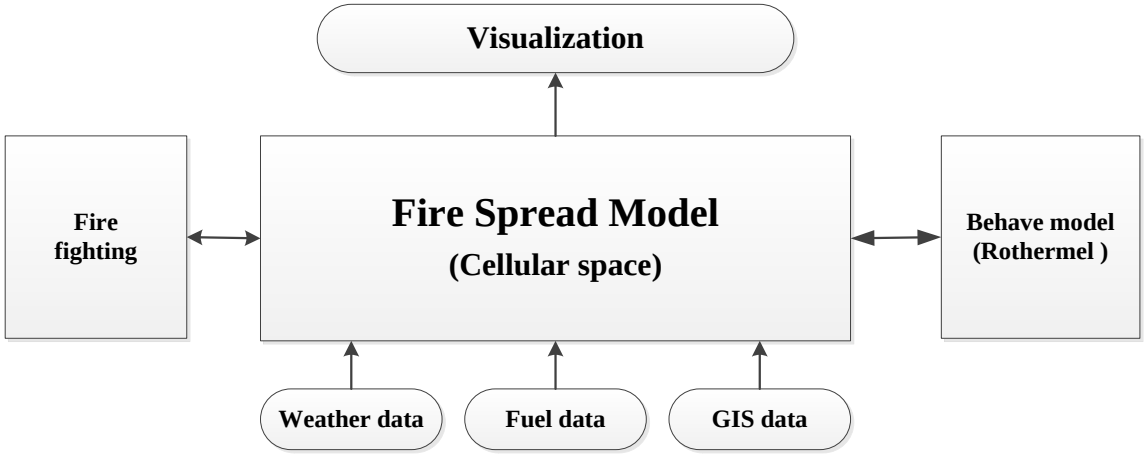


Figure 3.1 Structure of DEVS-FIRE model

Based on the structure of DEVS-FIRE model, we can see the core of DEVS-FIRE model is fire spread model which is modeled as a cellular space model containing individual cells coupled together. First of all, the cellular space model get the necessary information such as weather data, fuel data and GIS data (aspect data and slope data). When ignited, each cell uses the Rothermel model [93] to computer a one-dimension fire spread which include the fire speed and fire direction, then decomposed into two-dimensions based on an elliptical fire spread. The

visualization component displays the simulation results which changes the display color of a cell whenever the cell's state changes. From the left side of the structure, we can see the DEVS-FIRE also supports fire suppression simulation. This can be achieved by adding interactions between the fire spread model and the firefighting model. The work of [94] presents an integrated framework and demonstrates how fire spread simulation, firefighting resource optimization, and fire suppression simulation can effectively work together for wildfire containment, the more work about fire suppression were discuss very in [94] and omit in here.

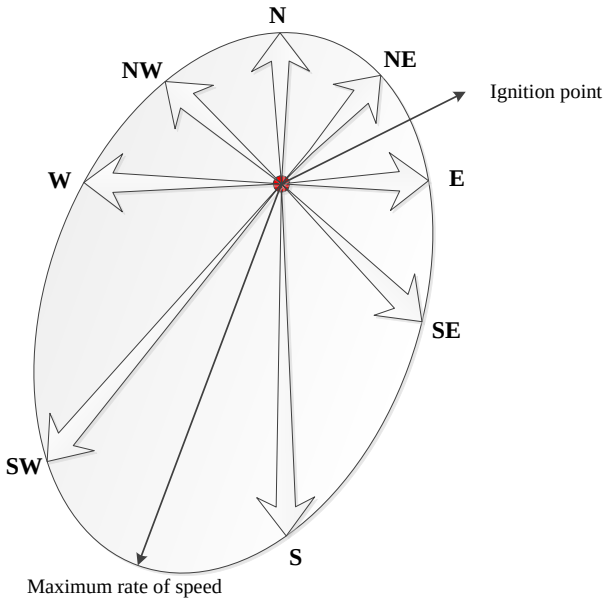


Figure 3.2 Fire spread decomposition of DEVS-FIRE

In the DEVS-FIRE model, the fuel data, GIS data and weather data within individual forest cells are assumed to be constants. Each cell has an ID(x, y) denote its location in the cell spaces, cell are coupled with their neighbors acceding to the Moore neighborhood, in which a central cell has eight surrounding cells, and its fire spread behavior is modeled by the Rothermel model. The entire cell space is a coupled model by connecting input ports and output ports between neighboring cells, so a cell can send messages to ignite its neighbor cells. Each cell is a

DEVS atomic model and transitions through different states like unburned, burning and burned. At first, the state of all the cells are unburned, once the cell receives and ignition message and also its fire line intensity is greater than the burning threshold, the state of this cell become to burning and the cell from burning to burned when its burn delay time has elapsed. In DEVS-FIRE, the rate of spread decomposed into eight directions including North, Northeast, East, Southeast, South, Southwest, West, and Northwest from the ignition point according to an elliptical shape as illustrated in figure 3.2, the method which calculate the spread rates can be find in[95].

3.2 Overview of PF-based data assimilation framework

In order to improve the simulation result, the real time data is assimilated into the simulation model. To apply particle filter methods for data assimilation, the system model and the measurement model need to be defined as well. Because particle filter methods are sample-based methods that use Bayesian inference and stochastic sampling techniques to recursively estimate the state of dynamic systems from some given observations [96] [47], a dynamic system is formulated as discrete dynamic state-space model, which is composed of the system model and the measurement model in equation (3.2) and (3.3) [97] as shown blow:

$$s_{t+1} = f(s_t, t) + v_t \tag{3.2}$$

$$mv_t = g(s_t, t) + w_t \tag{3.3}$$

In these equations, t is time step, s_t and m_t are the state variable and the measurement variable respectively, the functions of f and g define the evolutions of the state variable and the measurement variable. The v_t and w_t are two independent random variable to generate the state noise and the measurement noise. So, based on the simulation model, we formulate a non-linear state-space model as equation (3.4) and (3.5).

$$state_{t+1} = SF(state_t, t) + v_t \quad (3.4)$$

$$mv_t = MF(state_t, t) + w_t \quad (3.5)$$

which $state_t$ and $state_{t+1}$ are the system state variables of simulation state at time step t and t+1; SF is the system transition function, mv_t is the measurement variable, in our work we consider the measurement variables as the date obtained by the sensors deployed in the different filed. The v_t and w_t refer to the noises of the system state and those of the measurement data respectively. The system model and the measurement model are the essential components of the data assimilation system. The measurement model converts the output from the system model into the measurement data, which is used to compare with the real time data.

Because the sequential importance sampling has the limitation which the whole process relies on the initially generated sampling, so the particle filter methods used in here implement the sequential importance sampling with resampling (SISR) principle. The SISR forms the basic structure of particle filter methods which has been shown that a large number of particles are able to converge to the true posterior density even in non-Gaussian, non-linear dynamic systems [98]. For systems with strongly non-linear behaviors, particle filter methods are more effective than the widely used Kalman filter and its various extensions. More details about the algorithm can be found in [99]. A basic particle filter algorithm that implements the SISR procedure goes through multiple iterations. In each iteration, the algorithm receives a sample (particle) set $state_{t-1}$ and an observationmv_t. Each sample in $state_{t-1}$ is used to predict the next state in the importance sampling step. The importance weight of each particle is then updated and normalized. In the resampling step, N offspring samples are drawn with a probability proportional to the normalized sample weights. These samples represent the posterior belief of the system state and are used for the next iteration.

The figure 3.3 shows the structure of particle filter methods and the procedure of the data assimilation algorithm. In the figure, the rectangle boxes represent the major components and the circles and rounded rectangles represent the data or variables. The data assimilation algorithm runs in a stepwise fashion. The state from time step t-1 are fed into the system transition model, the result state set then denoted as $state_t'$. In order to computer the importance weight for each $state_t'$, the sensor data is computed according to the measurement function which denoted as M_t'. Then compare to the real observation data which collected from real time sensors, we can finish the weight calculation and weight normalization. After that, a resampling algorithm is applied to generate $state_t$ and it is the input for t+1 for next step. For the algorithm of particle filter methods, the set of states is represented by a set of particles.

The algorithm starts by initializing N particles representing the initial states and each particle's weight is initialized to 1/N. The algorithm goes through multiple iterations, every of which includes the sampling step, weight computation step (weight update) and resampling. In the sampling step, all the particles go through the system transition model to obtain their corresponding state of the next time step. In the weight update step, we can get the importance weights of the particle according to calculate the difference between the sensor data computed from the measurement model and the real sensor data. Finally, in resampling step, we select the particles based on their normalized weights to form a new set of particles. And we will use those particles and assign their new weight to 1/N to continue go the next iteration.

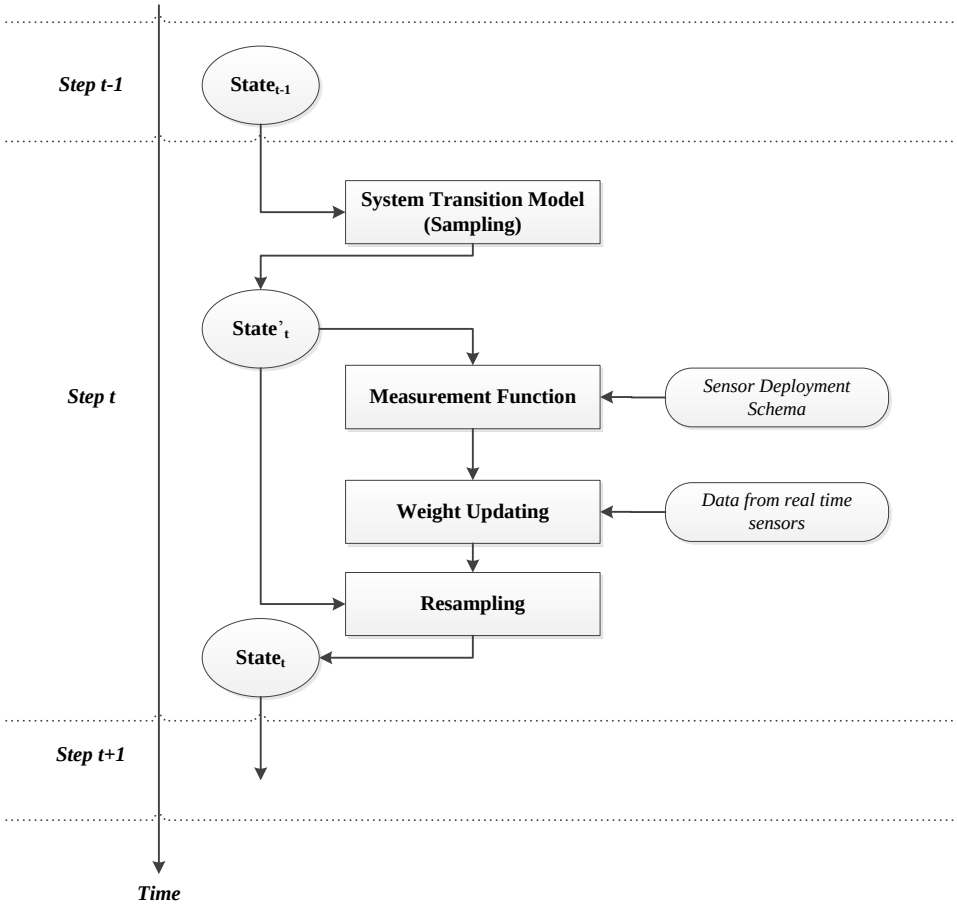


Figure 3.3 PF-based data assimilation

3.3 Data assimilation for wildfire spread simulation

Based on the information we shown in section 3.2, we can easily summarize the data assimilation for wildfire spread simulation work. Based on the DEVS-FIRE simulation model, we formulate a non-linear state-space model as show in equation (3.6).

$$\begin{cases} fire_{t+1} = DF(fire_t, t) + v_t \\ TM_t = MF(state_t, t) + w_t \end{cases} \tag{3.6}$$

which we define the state is the fire shape, because the fire front is the most important information in a wildfire spread simulation. However, the state in large-scale spatial temporal systems can include many parts such as the fuel data, slope data, aspect data and weather data,

etc. In equation (3.6), $fire_t$ and $fire_{t+1}$ are the fire shape in time step t and time step t+1. DF is the DEVS-FIRE simulation model; TM_t is the measurement variable, to apply the data assimilation in DEVS-FIRE simulation, they consider the measurement variables as the temperature data which collected from the ground temperature sensors deployed in the fire filed. MF is the measurement model that maps the fire front to the ground temperature sensors' temperature data.

The particle filter methods side, the algorithm is totally same as we shown in section 3.2. We have the fire front in time step t-1 and fed into the DEVS-FIRE simulation model, and then the output of the DEVS-FIRE simulation model gives the new fire front. In order to get the importance weights of these fire fronts, we calculate the difference between the temperature sensors data from those fire fronts and the real temperature sensors data. After the weight normalization, the weight will be used for resampling that draws a set of offspring samples which are used as the inputs for the next step t+1. In this section, we will mainly review the sampling algorithm, weight computation algorithm and resampling algorithm about PF-based data assimilation for DEVS-FIRE spread simulation, also some experiments result will discussed as well.

3.3.1 Sampling using DEVS-FIRE simulation

The goal of the sampling algorithm is to generate a fire state sample for the next time step give the current fire state based on the distribution $p(fire_t|fire_{t-1})$ [96]. First of all, we use DEVS-FIRE simulation starting from the fire state of each particle and run the simulation for one time step. The length of the time step is depending on how often the sensor data is collected. Moreover, let fire front $\overline{fire_t}$ denote the result of $DF(fire_t, t)$ and add the graph noise to it and

then reconstruct a new fire state from the noised fire front. In here, we let the reconstructed fire state as a sample of $p(fire_t|fire_{t-1})$.

The method of add the graph noise is: firstly, we divide the fire front into several segments and each segment consist the equal number of burning cell. Here, we use C_1 represent the number of segments. Secondly, we introduce a noise denoted as $d_i$that defines the change inside or outside a cell along the direction from the ignition point to this cell. So, the different segments get the different noise, but the cells in same segments get the same noise. After this, every burning cell for each segment moving to the new position based on d_i cells distains and the direction based on the ignition point to the cell. Finally, reconstructed all the cells one by one to a new fire front (before we divide the fire front, we record the cell location, after add the noise, we reconnect it according the location we recorded). We also need set all the cells which inside of the noised fire front to burned and obtain a noised fire state which considered as a sample of $p(fire_t|fire_{t-1})$. The algorithm 3.1 shows the system transition density sampling.

The sampling algorithm plays critical roles in the particle filter methods which modeled the simulation error and then generate a new and realistic noised fire front from the existing fire front. This can solve the problem which cause by imprecise fuel data, GIS data, weather condition, fire model error and other uncertain elements affecting fire spread.

Table 3.1 Algorithm 3.1 System Transition Density Sampling

Algorithm 3.1 System Transition Density Sampling

1. Let fire front $\overline{fire_t} = DF(fire_t, t)$
2. Divide the fire front into C_1 consecutive segments, represent as $SEG_1, SEG_2 \ldots SEG_i$.
3. Generate noise $d_1, d_2 \ldots d_i$ for each segment where

$$d_i \sim Gaussian\left(\frac{length\ of\ SEG_i}{C_2}, C_3\right)$$

4. Move every burning cell in each segment SEG_i based on the d_i and which direction according the ignition point to this cell.
5. Reconstruct each segment according the segment order before divide it to a closed fire front (noised fire front). And set all the cells on the noised fire front to burning status. Also set all the cell inside of noised fire front to burned status and all the cells outside of noised fire front to unburned;
6. Return the fire state.

3.3.2 Weight computation and resampling algorithm

Based on the figure 3.3, we have the fire state $fire_{t-1}$ in time step t-1 and fed it into the DEVS-FIRE simulation model. And then use algorithm 3.1 produce a sample for each particle in $fire_{t-1}$ based on $p(fire_t|fire_{t-1})$ and get the result fire state set $fire_t'$. Next step is the weight computation, which evaluate how good about a simulated particle compare to the real wildfire. The method is comparing the difference of temperature between the fire state and the real sensor temperature data to get the importance weight. The detail about the measurement density can be finding in [96]. Then after weight normalization, use resampling algorithm to generate $fire_t$ which is the input for the next step. The algorithm 3.2 shows the particle filter method in wildfire

simulation. The algorithm 3.3 shows the multinomial resampling which use for implement the resampling algorithm in this work.

Table 3.2 Algorithm 3.2 Particle filter method in wildfire simulation for one time step

Algorithm 3.2 Particle filter method in wildfire simulation for one time step

Input: The fire states and the corresponding importance weight at time step *t-1* $\left(\left\{fire_{t-1}^{(i)}\right\}_{i=1}^{N}, \left\{w_{t-1}^{(i)}\right\}_{i=1}^{N}\right)$, and the measurement at time step t (m_t).

Output: The fire states and the corresponding importance weight at time step *t* $\left(\left\{fire_{t}^{(i)}\right\}_{i=1}^{N}, \left\{w_{t}^{(i)}\right\}_{i=1}^{N}\right)$

1. Sampling

For each fire state in $\left\{fire_{t}^{(i)}\right\}_{i=1}^{N}$, draw a sample $fire'^{(i)}_{t}$ from $p\left(fire_{t}^{(i)}|fire_{t-1}^{(i)}\right)$ based on algorithm 3.1;

2. Weight computation and normalization
   (a. For each fire state in $\left\{fire'^{(i)}_{t}\right\}_{i=1}^{N}$, update the weight: $w'^{(i)}_{t} = w_{t-1}^{(i)} \times\ p\left(m_t|fire'^{(i)}_{t}\right)$
   (b. Calculate the normalized weight: $w''^{(i)}_{t} = \frac{w'^{(i)}_{t}}{\sum_{i=1}^{N} w'^{(i)}_{t}}$
3. Resampling
   (a. Draw N particles from $\left\{fire'^{(i)}_{t}\right\}_{i=1}^{N}$ and $\left\{w''^{(i)}_{t}\right\}_{i=1}^{N}$:
   $\left\{fire_{t}^{(i)}\right\}_{i=1}^{N}$ = Algorithm 3.3$\left(\left\{fire'^{(i)}_{t}\right\}_{i=1}^{N}, \left\{w''^{(i)}_{t}\right\}_{i=1}^{N}\right)$;

   (b. Set the weights: $w_{t}^{(i)}$ = *1/N, i=1,2, …,N*;

In algorithm 3.3, $w_t^{(i)}$ represents the normalized importance weight of the *i-th* particle at time step *t* and *N* is the total number of particles. At first, the cumulative sums of the normalized weight of N particles ($\tilde{q}_t^{(1)}, \tilde{q}_t^{(2)}, \dots, \tilde{q}_t^{(i)}, \dots, \tilde{q}_t^{(N)}$) are computed, where $\tilde{q}_t^{(i)} = \sum_{j=1}^{i} w_t^{(j)}$. Then we generate N random number between 0 to 1. Finally, we count the number of elements in $\{u_k\}_{k=1}^{N}$ that fall into the interval of $\tilde{q}_t^{(i-1)}$ and $\tilde{q}_t^{(i)}$. This number decides how many copies of the i-th particle will be selected.

Table 3.3 Algorithm 3.3: Multinomial resampling

Algorithm 3.3: Multinomial resampling

Input: The fire states and the corresponding importance weight at time step *t* $\left(\left\{fire_t^{(i)}\right\}_{i=1}^{N}, \left\{w_t^{(i)}\right\}_{i=1}^{N}\right)$

Output: Resampled fire states at time step t $\left\{fire'^{(i)}_t\right\}_{i=1}^{N}$

1. Compute the cumulative sums of the normalized weight of N particles ($\tilde{q}_t^{(1)}, \tilde{q}_t^{(2)}, \dots, \tilde{q}_t^{(i)}, \dots, \tilde{q}_t^{(N)}$), where $\tilde{q}_t^{(i)} = \sum_{j=1}^{i} w_t^{(j)}$;
2. Generate N ordered random numbers $\{u_k\}_{k=1}^{N}$, where $u_k \in (0,1]$
3. Generate n_i copies of $fire_t^{(i)}$, where n_i is the number of $u_k \in (\tilde{q}_t^{(i-1)}, \tilde{q}_t^{(i)}]$;
4. Return the new generated fire states as $\left\{fire'^{(i)}_t\right\}_{i=1}^{N}$

3.3.3 Experiments and analysis

First of all, we use the identical-twin experiment, which has been widely used in data assimilation research, to evaluate the data assimilation system of DEVS-FIRE. The purpose of identical-twin experiments is to study the assimilation in ideal situations and evaluate the proximity of the prediction to the true states in a controlled manner. There are three different type fire results in identical-twin experiment" "real fire", "simulated fire" and "filter fire". In the

identical-twin experiment, a simulation is first run and the corresponding data is recorded. These simulation results are considered as "real"; therefore, the observation data obtained here is regarded as the real observation data (because they come from the "real" model). Consequently, we estimate the system states from the observation data using particle filter methods, and then check whether these estimated results are close to the "real" simulation results. So, the "real fire" is the simulated fire spread from which the real observation data are obtained. The "simulated fire" is the simulation result based on some "erroneous" data; the "erroneous" data means some data such as fuel data, GIS data, weather data which are different from those used in the real fire. The "filter fire" is the data assimilation enhanced simulation result based on the same "erroneous" data as in the simulated fire. Our goal is to show that a "filtered fire" gives more accurate simulation results by assimilating observation data from the "real fire" even if it uses the "error" data as in the simulated fire.

The differences between a real fire and a simulated fire are due to the imprecise data such as wind speed, wind direction, GIS data, and fuel model, used in the simulation. In our experiments, we choose to use imprecise wind conditions (wind speed and wind direction) as the "erroneous" data. Table V shows the configurations of the set of experiments. The real wind speed and direction are 8 (mph) and 180 (degrees) with random variances added every 30 minutes. The variances for the wind speeds are in the range of –2 to 2 (mph) (denoted as 8 ± 2 in the table), and the variances for the wind direction are in the range of -20 to 20 (degrees) (denoted as 180 ± 20 in the table). Our experiment introduces errors to the wind speeds, which are randomly generated based on the wind speed of 6 (mph) with variances added in the range of –2 to 2 (mph) and also the wind direction of 130 (degrees) with added variances in the range of ± 20

(degrees) in the same time. For wind directions, the degrees indicate the angle between the north directions clockwise to the direction from where the wind comes.

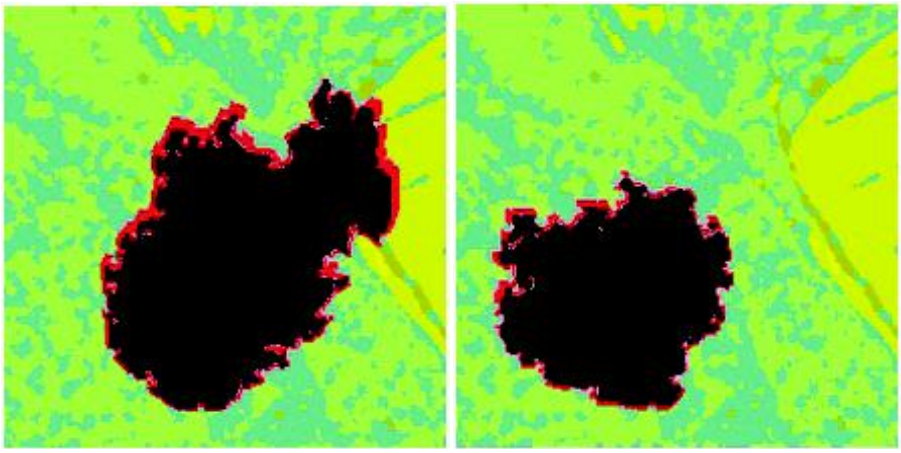
Table 3.4 Experiment set of the wind factor

"Error" data		Real data	
Speed (mph)	Direction(degrees)	Speed(mph)	Direction(degrees)
6±2	130±20	8±2	180±20

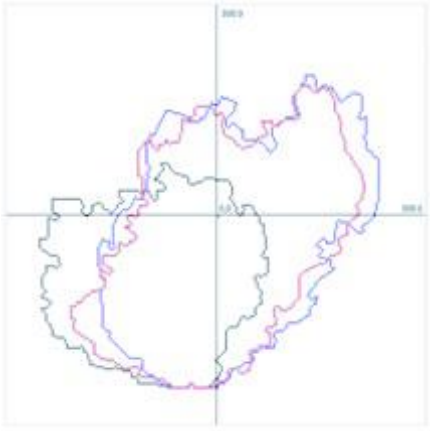
In the experiment, simulations use the real-world GIS data and fuel data. The cell space dimension is 200×200 and the cell size is 20 (m). The GIS data are airborne LiDAR (Light Detection and Ranging) [99] raster-based terrain data. The fuel data was obtained by classifying a multispectral QuickBird (DigitalGlobal) image [100]. Those data were acquired from Huntsville area, Texas, during the leaf-off season in March 2004 by M7 Visual Intelligence of Houston, Texas. The ignition point is set to the point (90, 55) of the cell space for all of the simulations. The observation data (ground temperature sensor data) from the real fire are collected every 30 minutes. We use 6 PUs (every PU has 50 particles, total 300 particles) to run 6 hours' simulation (12 steps and every step is 1,800 seconds) in all the experiments. Among these 6 PUs, one of them is functioned as a CU when completing the centralized resampling function for the global resampling step, otherwise a regular PU like others. All experiments are conducted under the supercomputer named Cheetah, which has 14 nodes, 160 computing cores, 32 CPUs, and 264 GB system memory.

Figure 3.4 shows the real fire spread with the real time data and the simulated fire with the imprecise wind data described in the above section. From the figure we can see the real fire (as shown in Figure 3. 4(a)) and the simulated fire (as shown in Figure 3.4(b)) are obviously different regarding the spread direction, and burned areas. After assimilating the real time data in the simulation, we expect the improved fire spread estimation. Figure 3.4(c) displays the real

fire, the simulated fire, and the filtered fire by assimilating the real data into wildfire spread simulation. In the figures, all the filtered fires (display in red) are close to the real fire (display in blue) although we run the data assimilation simulations with the error data. Figure 3.5 display the symmetric set differences for the simulated fire and the filter fire in every step. Compared to the simulated fires (display in black), all the simulation results are greatly improved.



(a) (b)



(c)

Figure 3.4 Real fire, simulated fire and filter fire for the experiment

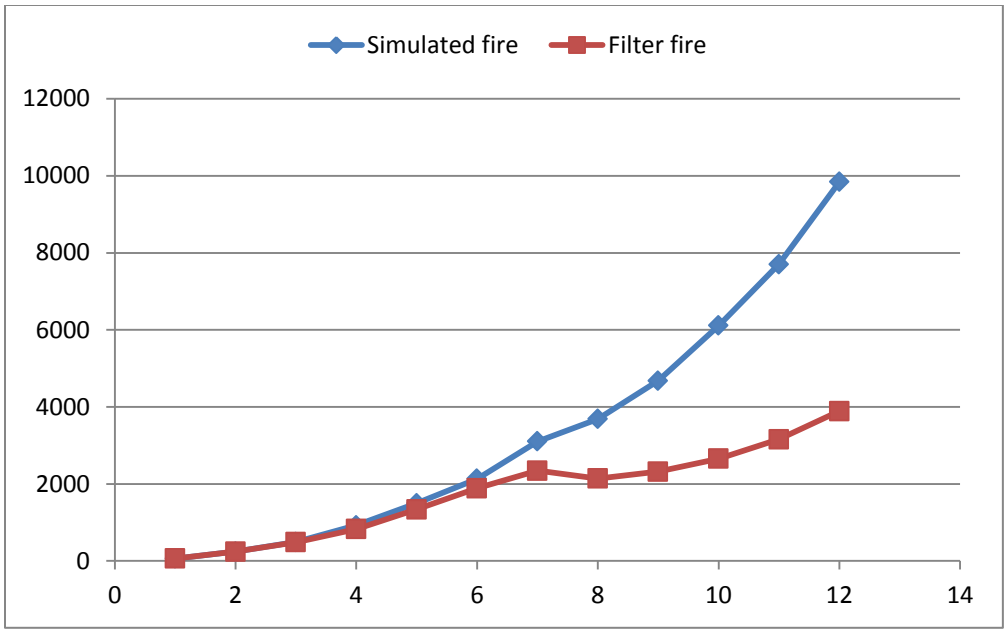


Figure 3.5 Symmetric set differences for simulated fire and filter fire

3.4 Data assimilation for parameter estimation

Wildfire simulation models also rely on many parameters to model the structure and behavior of systems under study. To achieve accurate simulation results, a simulation model needs to use correct model parameters. However, it is common that during the modeling process the parameters are largely unknown, uncertain and/or vary with time or space. Therefore, it is critical to develop methods to decide or estimate the correct set of model parameters in order to achieve reliable simulation results. In the section 3.3, our work on data assimilation explored the possible applications of particle filters in wildfire spread simulation based on DEVS-FIRE model, and applied particles filters to assimilating temperature sensor data for estimating the dynamically evolving fire front of a spreading fire. This section's work differs from the section 3.3's work by focusing on estimating the parameters of the wildfire spread simulation model in order to achieve better simulation results. In this section, we present a method to dynamically estimate model parameters by assimilating real time data using particle filter methods. We formulate the problem of single and multiple parameter estimations based on the context of

wildfire spread simulation. Preliminary results show that the developed method can be applied to parameter estimation in wildfire spread simulation to produce more accurate simulation results. The complexity and difficulties in multiple parameter estimation are discussed as well.

3.4.1 Particle filter based state estimation

Estimating model parameters of a simulation model is a challenging task. Different systems have different characteristics, which should be reflected in the chosen modeling parameters. For example, in a wildfire spread simulation, different study areas have different characteristics, such as different levels of fuel bed depth (FBD), which influence fire spread behavior. In practice, the values of these parameters are set based on experts' opinions and/or experimental testing data, which are not always readily available. More challengingly, some parameters are "dynamic" in nature as the values of these parameters dynamically changing over time as the simulation proceeds. Consider the wildfire spread simulation example again, an important parameter is the fuel moisture content of the vegetation in the fire area. In general, the fuel moisture content fluctuates during different time of a day: it is higher in the evening and morning and becomes lower as the sun rises. Thus depending on which time the fire spread simulation starts and how long the simulation lasts, the fuel moisture content parameter (and its dynamic change) needs to be different in order to achieve accurate results. Therefore, it is essential to develop methods to estimate the dynamically changing parameters in order to fit the daily variable environment.

Parameter estimation is widely used in many fields, such as image processing, chemical engineering, bio-molecular engineering, biochemical pathways and text analysis. In the fields of image processing, the authors [101] propose novel algorithms for total variation (TV) based image restoration and parameter estimation utilizing variational distribution approximations.

Within the hierarchical Bayesian formulation, the reconstructed image and the unknown hyper-parameters for the image prior and the noise are simultaneously estimated. Additionally, in the chemical engineering area, the parameter estimation problem for ordinary differential equations (ODE) is decomposed into two sub-problems [102]. And in the bio-molecular engineering area, one method is presented for deterministic global optimization in the estimation of parameters in models of dynamic systems [103]. The method can be implemented as a global algorithm, or, by use of the interval-Newton method, as an exact algorithm. A key feature of the method is the use of a new validated solver for parametric ODEs, which is used to produce guaranteed bounds on the solutions of dynamic systems with interval valued parameters, as well as on the first- and second-order sensitivities of the state variables with respect to the parameters. In the text analysis area, the authors [104] present parameter estimation methods common with discrete probability distributions, which is of particular interest in text modeling. Starting with maximum likelihood, a posteriori and Bayesian estimation, central concepts like conjugate distributions and Bayesian networks are reviewed.

Conventional methods for estimating parameters include statistical techniques, such as maximum likelihood technique [105], which rely on analyzing historic data. These conventional methods can neither automatically tune the parameters online for a specific study area, nor can it dynamically adjust the parameters as simulation proceeds. In here, we present a method to dynamically estimate model parameters by assimilating real time data collected from the system under study and we carry out this work within the context of wildfire spared simulations. We formulate the problem of parameter estimation based on particle filters for wildfire spread simulation using the DEVS-FIRE model [9] [106]. In our work, we consider two specific

parameters: fuel moisture content and fuel bed depth (FBD), in the wildfire spread model to demonstrate the developed method.

The DEVS-FIRE model used in this work is a discrete event model for wildfire spread simulations. DEVS-FIRE models the forest as a cellular space where fire spreading is simulated as a propagation process between neighbor forest cells, whereby burning cells ignite their unburned neighbor cells. Each cell has input and output ports through which couplings are made for exchanging messages, and it is coupled to the eight adjacent neighbor cells. When ignited, a cell uses the Rothermel model to compute a one-dimension fire spread rate, which is then decomposed into two-dimensions based on an elliptical fire spread. In Rothermel model, the basic fire spread is equation (3.7) and equation (3.8):

$$R = \frac{I_R * \xi * (1 + \Phi_w + \Phi_s)}{\rho_b * \varepsilon * Q_{ig}} \tag{3.7}$$

$$I_R = \acute{\Gamma} * w_n * h * \eta_M * \eta_s \tag{3.8}$$

In equation(3.7), where R is the Rate of spread, I_R is the reaction intensity, ξ is the propagating flux ratio, Φ_w is the wind coefficient, Φ_s the slope factor, ρ_b is the ovendry bulk density, ε is the effective heating number and Q_{ig} is the heat of pre-ignition. The reaction intensity I_R is computed according to equation (3.8), is the reaction intensity, where $\acute{\Gamma}$ is the optimum reaction velocity, w_n is the net fuel loading, *h* is the fuel particle low heat content, η_M is the moisture damping coefficient and η_s is the mineral damping coefficient. The above description shows that the fire spread behavior depends on many different factors. Among them *fuel moisture content* is an important fuel characteristic affecting fire behavior [107] [108]. The fuel moisture content determines how much fuel is available for burning, and ultimately, how much is consumed [109]. According to [110], the factors that regulate fuel moisture differ among live and dead fuels. The primary determinants of live fuel moisture content include factors such as internal

factors that regulate diurnal and seasonal changes, climate, site factors that affect the fuel environment. For the dead fuels, fuel moisture depend on factors such as particle size, short and long-term weather changes, topography, decay class, and fuel composition. In Rothermel's model, the fuel moisture content (denoted as Mf) is used to compute the moisture damping coefficient η_M (see Equation (3.8)) as shown in Equation (3.9):

$$\eta_M = 1 - 2.59 * \left(\frac{M_f}{M_x}\right) + 5.11 * \left(\frac{M_f}{M_x}\right)^2 - 3.52 * \left(\frac{M_f}{M_x}\right)^3 \quad (3.9)$$

where M_x is the moisture of extinction, which is the moisture content of the fuel at which the fire will not spread. The moisture of extinction is a property of the fuel type, and is considered as a constant for a given type of fuel.

Another important parameter influencing fire spread behavior is the *fuel bed depth* (FBD). The FBD is the accumulation of dead, woody residue on the forest floor. It begins at the top of the duff layer and above. It includes litter, dead limb wood and bole wood from tree species, as well as dead material from shrub, herbaceous, and grass species. In Rothermel model, the FBD related equations (3.10):

$$\rho_b = \frac{w_o}{\delta} \quad (3.10)$$

In equation (3.10), where δ is the FBD, w_o is ovendry fuel loading and ρ_b is ovendry bulk density.

$$\beta = \frac{\rho_b}{\rho_p} \quad (3.11)$$

$$\acute{\Gamma} = \acute{\Gamma}_{max} * \left(\frac{\beta}{\beta_{op}}\right)^A * \exp\left[A * \left(1 - \frac{\beta}{\beta_{op}}\right)\right] \quad (3.12)$$

$$\Phi_w = C * U^B * \left(\frac{\beta}{\beta_{op}}\right)^{-E} \quad (3.13)$$

In equation (3.11), (3.12) and (3.13), where β is the packing ratio, ρ_p is the ovendry particle density, $\acute{\Gamma}_{max}$ is maximum reaction velocity, β_{op} is the optimum packing ratio, C,U,B,E are experimentally defined constants.

For a given fire spared scenario, the FBD can be treated as a static parameter and does not change over time. However, the fuel moisture content dynamically change over time (e.g., due to changes of temperature during the day. More information about the change of dead fuel moisture can be found in [111]). This asks for the need of estimating parameters that are both static and dynamically changing over time. Also notice that both the fuel moisture content and the FBD impact the fire spread behavior. This means multiple combinations of these two parameters may leads to the same (or similar) fire spread behavior. This poses difficulties to estimate the precise values of the two parameters when both of them need to be estimated. This is shown in our experiment results in Section 3.4.3.

3.4.2 Problem formulation for parameter estimation

To extend previous work in section 3.2 and 3.3 for supporting online parameter estimation, we formulate the parameter estimation problem as a joint state-parameter estimation and uncertainty assessment problem, which treats model parameters as stochastic state variables that need to be estimated. To apply particle filter methods to this problem, in addition to sampling of the state variables, sampling in the parameter space is also needed using some kind of proposal density. Specifically, in this work we treat the parameters to be estimated as stochastic variables and perturb the parameters at each time step.

In order to estimate the parameters, we need to redefine the wildfire state by adding the parameters that need to be estimated as part of the state variables. Below we first consider the

case where there is only one parameter, the FBD, which needs to be estimated. With this parameter, we redefine the state and the state space model as follows equation (3.14):

$$\begin{cases} s_t = \begin{bmatrix} FS_t \\ FBD_t \end{bmatrix} \\ s_{t+1} = \begin{bmatrix} FS_{t+1} \\ FBD_{t+1} \end{bmatrix} = \begin{bmatrix} \mathrm{DF} & 0 \\ 0 & \mathrm{Gnoise} \end{bmatrix} \begin{bmatrix} FS_t \\ FBD_t \end{bmatrix} \\ DT_t = [SM, 0] \begin{bmatrix} FS_t \\ FBD_t \end{bmatrix} \end{cases} \tag{3.14}$$

where s_t and s_{t+1} is the new-defined state at time *t* and *t+1*. Note that this new-defined state includes the fire front *FS* and the parameter FBD. Same as before, *DF* is the DEVS-FIRE simulation model, *DT* is the measurement data, and *SM* is the measurement model that computes the sensor data based on the fire front. To perturb the FBD parameter, we add Gaussian noise to the FBD parameter in each step. This is represented by the *Gnoise* function in the above state space model.

If multiple parameters need to be estimated, we can formulate the state space model in a similar way as shown above. Below we consider two parameters: fuel moisture content (denoted as moisture) and FBD, and formulate the problem as below equation (3.15):

$$\begin{cases} s_t = \begin{bmatrix} FS_t \\ Moisture_t \\ FBD_t \end{bmatrix} \\ s_{t+1} = \begin{bmatrix} FS_{t+1} \\ Moisture_{t+1} \\ FBD_{t+1} \end{bmatrix} = \begin{bmatrix} \mathrm{DF} & 0 & 0 \\ 0 & \mathrm{Gnoise} & 0 \\ 0 & 0 & \mathrm{Gnoise} \end{bmatrix} \begin{bmatrix} FS_t \\ Moisture_t \\ FBD_t \end{bmatrix} \\ M_t = [SM, 0, 0] \begin{bmatrix} FS_t \\ Moisture_t \\ FBD_t \end{bmatrix} \end{cases} \tag{3.15}$$

As can be seen, we perturb both the FBD and fuel moisture content using a Gaussian noise function. This means during the sampling step of the particle filter method, we sample both the FBD and fuel moisture content parameters according to the Gaussian distribution. Note that in

our experiments presented below, the Gaussian distribution for the FBD has sigma 0.3 and the Gaussian distribution for the fuel moisture content has sigma 1.0. We chose a larger sigma for fuel moisture content because the fuel moisture content has more dynamics (e.g., due to temperature change during the day) compared to the FBD.

3.4.3 Experiments and analysis

Similar as the previous experiment work, we continue use identical-twin experiment in this section's experiment. We carry out the experiments in a stepwise fashion. In the first experiment, we consider the case where only one parameter (the fuel moisture content parameter) needs to be estimated. The to-be-estimated parameter is a static parameter, i.e., a constant that does not change. In the second experiment, we still consider only one parameter of fuel moisture content. However, this time the fuel moisture content dynamically changes over time. In the third experiment, we consider the case where both the fuel moisture content and FBD need to be estimated.

In all our experiments, we employed a 200 by 200 cell space. We used regular sensor deployment schema where the 100 sensors are regularly deployed and the observation data (ground temperature sensor data) from the real fire was collected every 30 minutes.

3.4.3.1 Estimating a single static parameter

In this experiment, we consider only one parameter, the fuel moisture content parameter, which needs to be estimated. We carried out two experiment cases based on different GIS data. Case 1 uses a uniform GIS data and case 2 uses a real GIS data for an area in eastern Taxes.

Case1 uses a uniform fuel model (fuel model 7) possessing zero slope and zero aspect, with simple wind flow along with a wind speed of 2 mph and a wind direction with a fixed value of 180 degrees, having the fuel moisture content initialized with a random number which the range

is 0% to 100%. The real fuel moisture content (the fuel moisture content used in the real simulation) keeps a constant value of 50%. Note that this is the data that need to be estimated by the particle filters. Based on the real fuel moisture content data, we use DEVS-FIRE to run the simulations to obtain the fire fronts and their corresponding temperature sensor data every 30 minutes. Our goal is to estimate the fuel moisture content data every 30 minutes by assimilating those sensor data using the developed method based on particle filters. The particle filters used 120 particles and the simulation time is 6 hours.

Case2 uses non-uniform GIS data, where cells have different fuel models, aspects, and slopes. The initial wind speed and wind direction are 1 mph and 180 degrees. And the real fuel moisture content also keeps a value of 50%. Figure 3.6 and Figure 3.7 display the values of fuel moisture content every 30 minutes for case 1 and case2. In the figures, '*initialized value'* means the value of the fuel moisture content which is random generated and 'particle filters' means the fuel moisture content value which is estimated by particle filters.

From figure 3.6 and figure 3.7, we can see that the estimated fuel moisture content have the same trend as those of the real fuel moisture content conditions. Therefore, in the practical applications, if we know the fuel moisture content condition every time period, we can estimate the fuel moisture content data each time slot between this time according to observed data by particle filters.

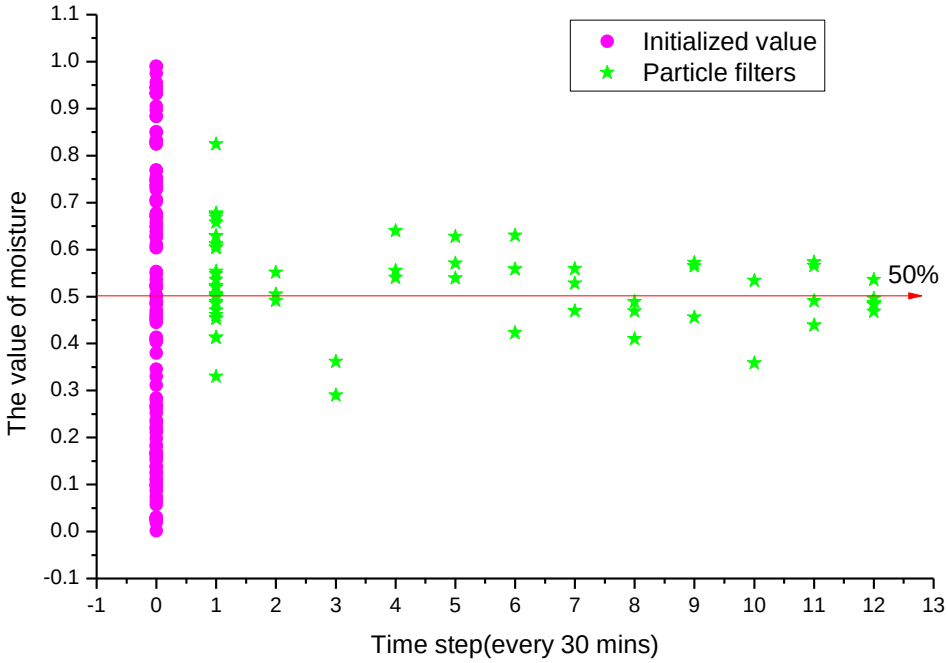


Figure 3.6 Fuel moisture content of case1

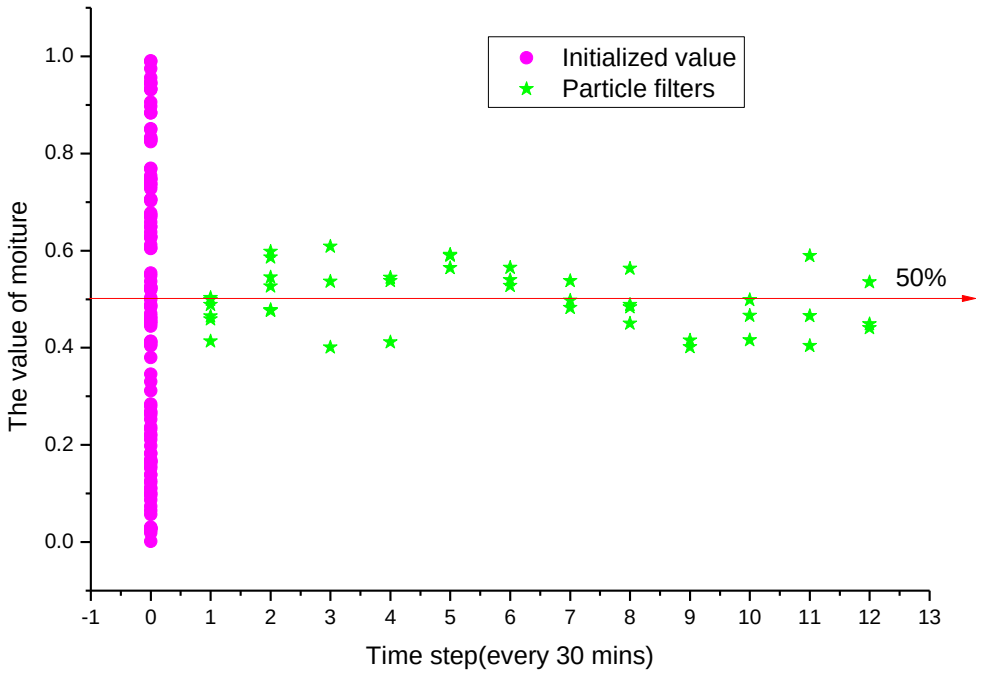


Figure 3.7 Fuel moisture content of case2

3.4.3.2 Estimating a single dynamic parameter

In this experiment, we also consider only one parameter, but the fuel moisture parameter dynamically varied with time, which needs to be estimated. Case3 uses the same conditions with case1 except we change the real fuel moisture content every two hours: fuel moisture content is 20% in first two hours, and then changed to 50% in next two hours and final change to 80% in last two hours. The particle filters still used 120 particles and the simulation time is 6 hours.

Figure 3.8 displays the values of fuel moisture content every 30 minutes for case 3. We can see that the estimated fuel moisture content have the almost same trend as those of the real fuel moisture content conditions, even those real fuel moisture content conditions varied with time.

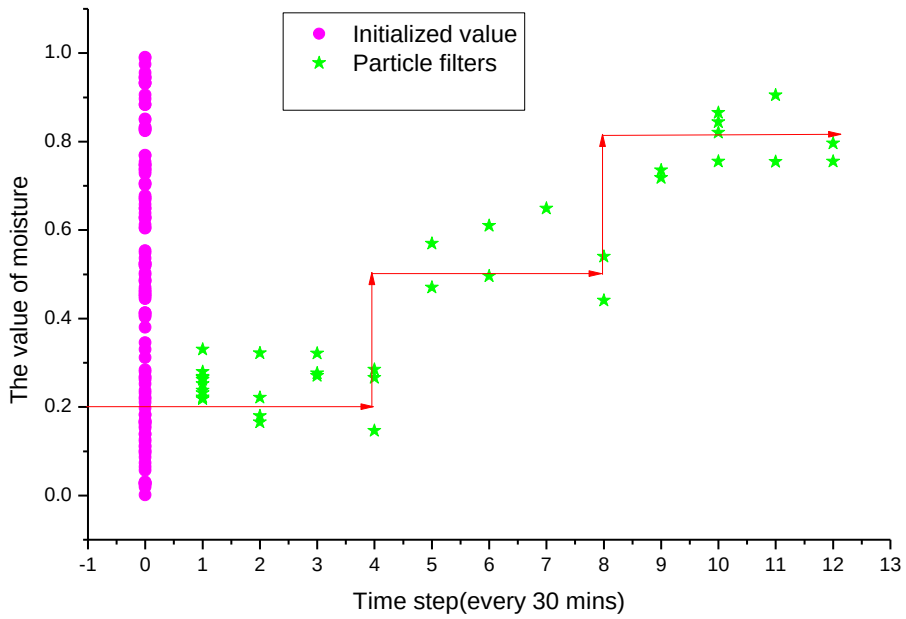


Figure 3.8 Fuel moisture content of case3

3.4.3.3 Estimating multiple parameters

In this experiment, we consider two parameters, the fuel moisture content and FBD, which are content parameters need to be estimated. Case4 uses almost the same experiment environment with case1 and the FBD initialized with a random number which the range is

between 0 to 3 meters. (The reason why we choose this range is the values of FBD normally are the static number, around 1.2m in real wildfire situation.) We also have the fuel moisture content initialized with a random number (0% ~ 100%). The real FBD keeps a value of 2.0 meters and the real fuel moisture content keeps a value of 50%. Note that this is the data that need to be estimated by the particle filters (both fuel moisture content and FBD). Our goal is to estimate both the FBD and fuel moisture content data every 30 minutes. The particle filters used 120 particles and the simulation time is 6 hours.

Figure 3.9 displays the values of FBD every 30 minutes for case 4. Figure3.10 displays the values of fuel moisture content every 30 minutes for case 4.

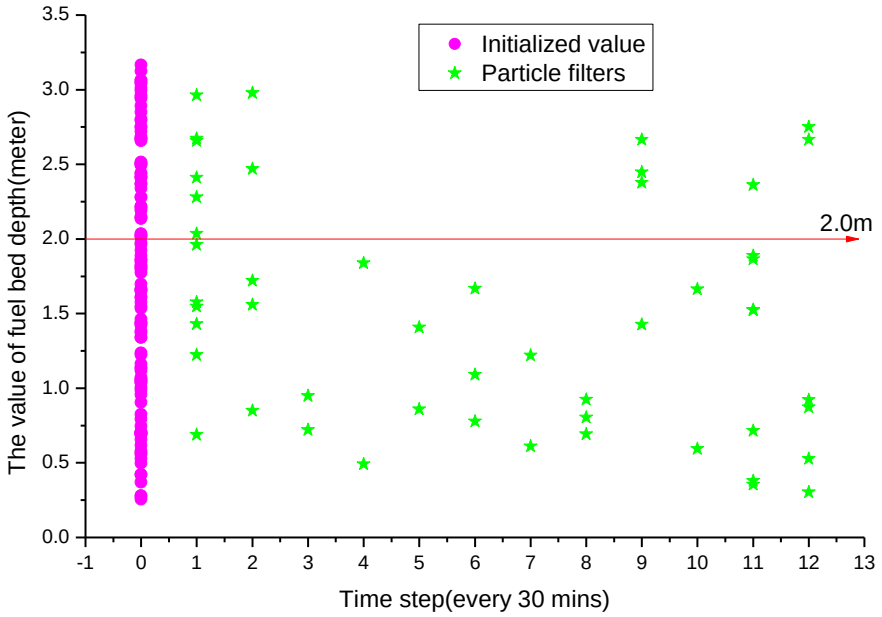


Figure 3.9 FBD value of case4

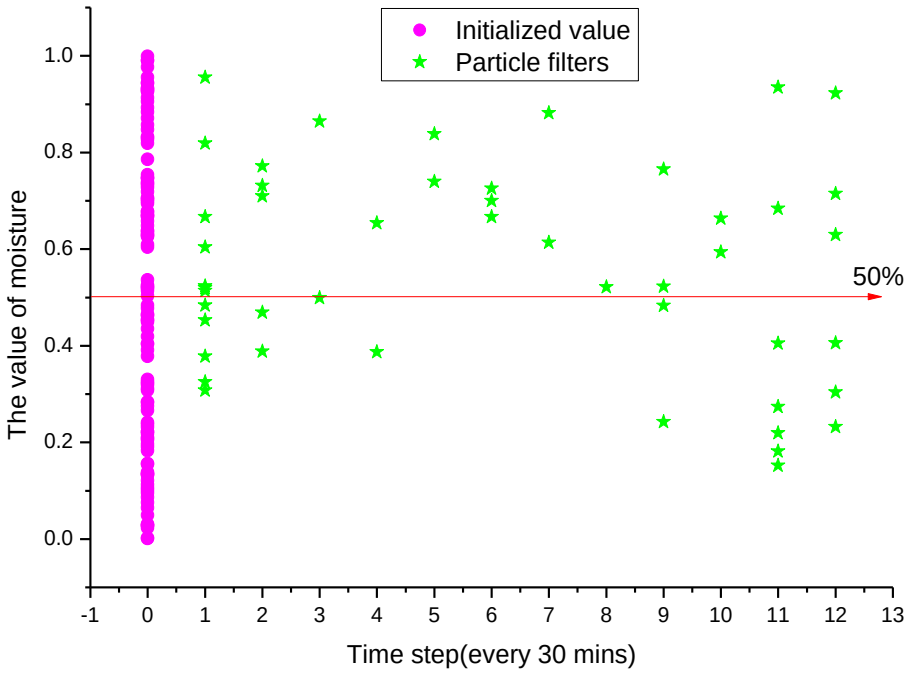


Figure 3.10 Fuel moisture content value of case4

From Figure 3.9 and Figure 3.10, we can see the PF cannot estimate parameter correctly when combining two parameters together. Because these two parameters have the same effect to the fire spread result. We can explain the reason is: according the equations (3.7) ~ (3.13) in section 3.4.1, we know both the fuel moisture content and FBD have the same effect to the final fire spread result and we cannot distinguish their exact values, because both of their effect in the same equations (equations (3.7) and (3.8) in section3.4.1). Therefore, there are many different combinations can get the same final fire spread result. From our experiment results, we found some fuel moisture content and FBD combinations can get the totally same result. For example: the FBD 2.0m with fuel moisture content 50% has the same result with FBD 1.2m with fuel moisture content 80%. In our experiments, we obtained the more precise value of FBD, and then we calculated the burned area, fire perimeter and then we compared the final fire shapes. In table 3.5, we show six different combinations using the DEVS-FIRE and we can get the same fire

spread results, which have the totally same burned area and fire perimeter. (Fuel Moisture Content is denoted as 'FMC' in Table 3.5)

Table 3.5 Different combinations

	FBD (m)	FMC (%)	Burned Area(ha)	Fire Perim.(km)
1	2.0	50	1057.77	1410.24
2	2.5401	40	1057.77	1410.24
3	1.6445	60	1057.77	1410.24
4	1.3938	70	1057.77	1410.24
5	1.2075	80	1057.77	1410.24
6	1.0643	90	1057.77	1410.24

In Figure 3.11, we show the final fire shapes with six different combinations (showed in table 3.5) running under the DEVS-FIRE for 6 hours.

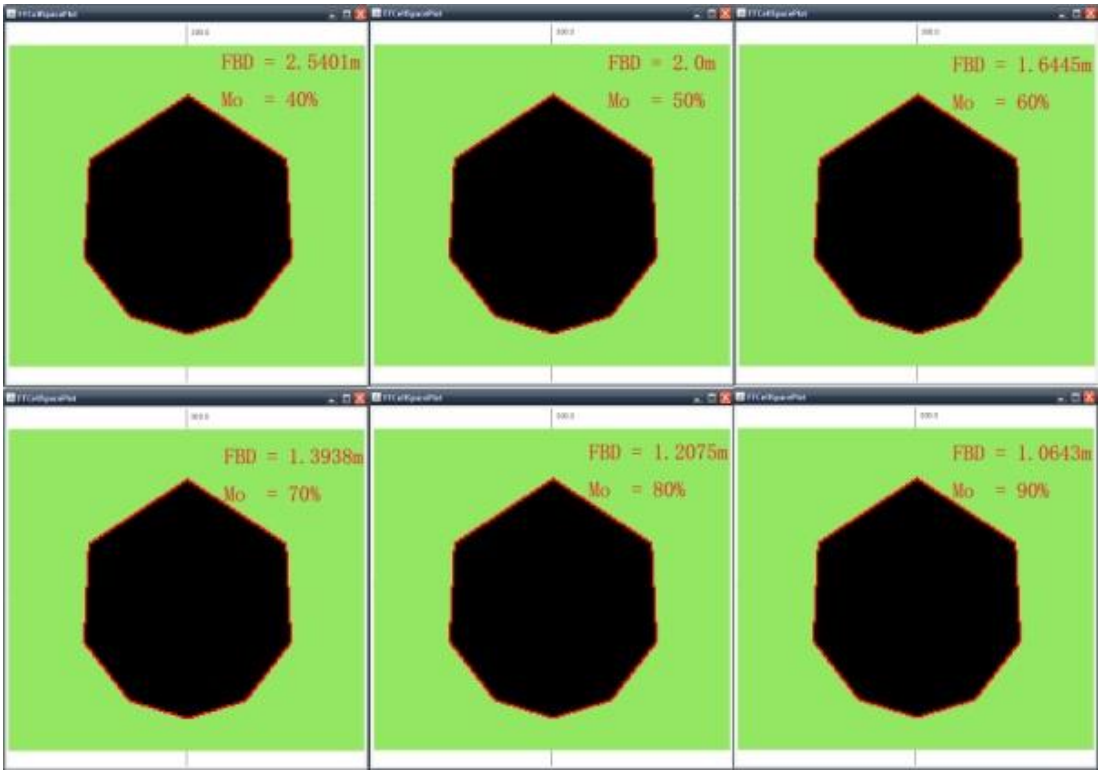


Figure 3.11 The final fire shape

3.4.4 Conclusions

In this section, we developed a method to dynamically estimate the parameters in DEVS-FIRE spread simulation model based on SMC methods. We carried out experiments to estimate the fuel moisture content and FBD parameters used in the wildfire spread simulation. Experiment results show that the developed method can be applied to parameter estimation in wildfire spread simulation to produce more accurate simulation results. However, there are complexities and difficulties associated with multiple parameter estimation.

4 PARTICLE ROUTING IN DISTRIBUTED PARTICLE FILTERS WITH CENTRALIZED RESAMPLING

A vital component of distributed particle filters is particle routing because non-optimized routing may lead to high communication overheads. This is especially true for high dimensional systems due to the complex state represented by particles. Unfortunately, while different distributed particle filters have been developed, less research is devoted to the particle routing itself. Therefore, in this chapter, we propose detailed particle routing policies for both the centralized resampling and the distributed resampling and evaluate their impacts on data assimilation for large-scale spatial temporal systems. We study the routing policies in distributed particle filters with both the centralized resampling schema and the distributed resampling schema. In the centralized resampling schema [85], the central unit (CU) has the full knowledge of the weight distribution of all particles on different PUs.

Based on this global information we propose two efficient routing policies named as minimal transfer policy and maximal balance policy in section 4.2.3 and section 4.2.4. Moreover, in the distributed resampling schema (more specifically, the distributed RNA with Local Exchange in section 2.4.3.3), communications are constrained between neighboring PUs. This local communication schema supports a large degree of parallelism due to elimination of the centralized resampling step. However, it also results in slow propagation of high-weighted particles, and thus reduces the convergence rate of the particles. To address this issue, we propose a hybrid particle routing approach that combines the global routing with the local routing to take advantage of both. In this approach, we mainly use the local routing to ensure scalability and low communication costs, and occasionally invoke the global routing to support faster propagation of "good" particles. We evaluate and compare the different particle routing

methods based on the application of data assimilation for large-scale wildfire spread simulations [112]. The rest of this chapter is organized as follow. Section 4.1 introduces the particle routing and related work. Section 4.2 presents the proposed routing policies in centralized resampling, including the minimal transfer policy and the maximal balance policy. Section 4.3 shows the full design of the example and all the example results include the analysis will discussed in Section 4.4.

4.1 Introduction of particle routing

To improve of the performance of data assimilation, distributed / parallel particle filters are need. In section 2.4 several distributed particle filtering algorithm have been discussed as well. These algorithms mainly differ in how the resampling is carried out. Nevertheless, they all involve using multiple processing units (PUs) to carry out sampling of particles, and after resampling routing particles among the PUs. Particle routing is necessary because the numbers of particles on different PUs are unbalanced after resampling. Thus PUs which have surplus of particles need to route the extra particles to the PUs with shortage of particles for the next iteration of computing. As the number of PUs increases, the communication overhead rises. The unbalanced particles on PUs are caused by the fact that particles have different importance weights. As a result, PUs hosting high weighted particles generates a lot more replicates in resampling and need to route a large number particles to others. The uneven distribution of particles' weights is common in data assimilation using PFs for spatial temporal simulations. Therefore, efficient particle routing is critical for reducing the communication cost in distributed PFs. This is especially true for high dimensional spatial temporal simulations because the size of each particle is large due to the high dimensional state it represents [113].

Particle routing deals with selecting particles on some PUs and routing them to other PUs across the network. In distributed PFs, routing particles among PUs can serve two different purposes: 1) to help the "good" particles, i.e., particles with high weights, to propagate among the PUs and thus potentially to lead to better estimation results; 2) to ensure that the different PUs have the same number of particles (i.e., load balance) after resampling. While several resampling algorithms such as the centralized resampling [86], the distributed RPA [86] and the distributed RNA [86] have been developed for distributed particle filters, less research has been conducted to investigate how to route particles among PUs after resampling in effective and efficient manners.

In chapter 2, we have already overview the application of data assimilation and application of particle filters. For both of them, particle filters are used in data assimilation of various high dimensional systems including ocean systems, land surface systems, object tracking, and atmospheric systems. The work in [114] applied particle filters to Agulhas Current to test the data assimilation methods because of the highly nonlinear dynamics and the availability of high quality satellite measurement data. The dimension of the state space in this application reaches about 200,000. Also, the work in [115] analyzed the performance of particle filters in a large-scale nonlinear land surface data assimilation example, in which a total of 684 states were considered. The work of [116] presented particle algorithms for filtering in group object tracking with up to 40 states and demonstrated its performance and the work [117] adapted particle filters to one of high dimensional chaotic systems, an atmospheric model that mimics mid-latitude atmospheric dynamics with microscopic convective processes, in which 360 dimensions were present. In spite of the aforementioned work in data assimilation, much less research has been done to use particle filters in high dimensional systems compared to low dimensional systems. In

work [118], the authors pointed out the obstacles of applying particle filters in high dimensional systems. Particle filters suffer from the "curse of dimensionality" due to collapse of particles' weights for high dimensional systems [119]. To avoid this large number of particles are needed, which leads to high computation cost.

To address the performance issue, different approaches have been proposed to solve problems in various applications including wireless sensor networks, traffic state tracking, robotic systems, signal processing, image processing, and target tracking. The work of [120] presented two parallelized particle filtering algorithms to estimate the state of the freeway traffic networks based on the topological partitioning of a traffic network into sub-networks, and compared the accuracy, the computational complexity, and the communication costs of the proposed algorithm and the centralized approach. The work of [121] proposed a strictly decentralized approach in which only nearby platforms exchange information to maximize the information flow and evaluated it in a robotic system for playing the game of laser tag. Their work illustrated the scaling capability to a large team of vehicles. Target tracking is one of the important applications for particle filters. The work of [122] developed a decentralized particle filtering algorithm for multiple targets tracking in wireless sensor networks, and compared their results to the optimal centralized solution. The work of [123] described two methodologies for distributed particle filters in wireless sensor networks, which are the parametric modeling approach and the adaptive encoding approach. From the mentioned work above, we can see that distributed particle filters are used in many applications, such as object tracking problems with wireless sensor networks. They have proliferated distributed sensor data as the measurement available. A lot of work mainly focused on the paradigm of the distributed particle filters and used the simple examples to evaluate the proposed methods. They used the centralized approach

as the base algorithm to compare with. We also notice that some distributed particle filters systems were partitioned into multiple subsystems and each subsystem used its local measurement data to estimate the partial state of the whole sys-tem state. Therefore, the state was decomposed and executed in parallel. In contrast, other systems used particle filters to predict the entire system state, in which different estimations were located on different processing units. In this case, the sample space was partitioned, and the resampling stage was the main focus since the global routing was needed. In our work, we focus on the sample partition and parallel execution.

4.2 Particle Routing in Centralized Resampling

4.2.1 Overall architecture

In the general PF algorithms, three main steps are involved including sampling, weight computation, and resampling. Among them, resampling needs information of all the particles, and thus affects the parallelization of PFs. In the centralized resampling, two types of nodes are defined, the processing unit (PU) and the central unit (CU). Sampling and weight computation are implemented on PUs since they are independent for different particles. Resampling is performed on the CU due to its sequential nature. To carry out particle routing, during the resampling stage the CU collects the weights of particles from all the PUs, performs the resampling, decides the routing information according to routing policies (described later), and then transfers particles between the CU and PUs according to the routing information. When transferring particles, PUs can send particles directly to each other. However, to simplify the overall system architecture, in our work we use CU as a hub to collect particles from source PUs and send the collected particles to destination PUs. Note that this design choice does not affect the different particle routing policies described in our work.

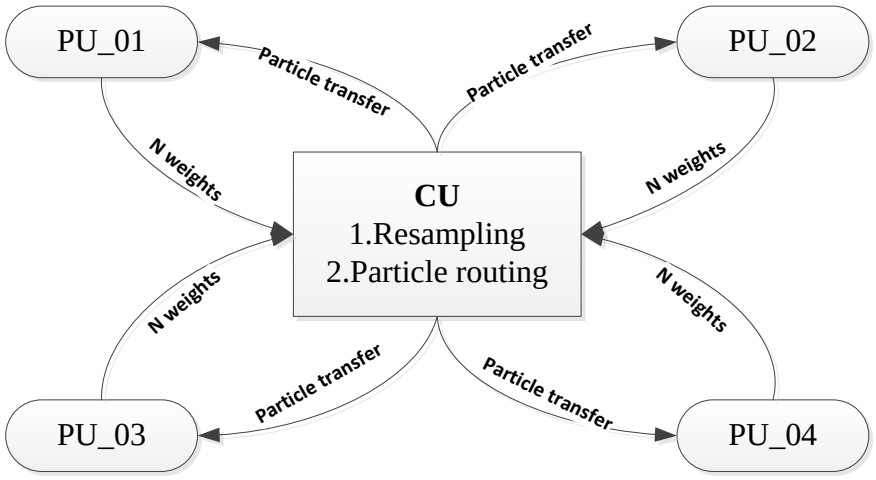


Figure 4.1 Overall architecture of particle routing in the centralized resampling

The overall system architecture is illustrated in Figure 4.1. In the figure, there are4 PUs (PU1, PU2, PU3, and PU4) and one CU and in each particle filtering iteration, a PU carries out sampling for its particles, computes particles' weights, and then sends the weights to the CU. After receiving all the weights, the CU normalizes the weights and performs the resampling algorithm. Consequently the CU carries out the routing procedure according to different policies (described below). According to the routing results, PUs with surplus of particles sends particles to the CU, and then the CU transfers them to the PUs with shortage of particles. After the routing completes, the system evolves to the next iteration.

Assume there are *n* PUs and *m* particles on each of them. A particle is denoted as $M_t^{(i,j)}$, $i \in \{1,\ldots,n\}$ and $j \in \{1,\ldots,m\}$, where *i* is the index for the particle's PU, and *j* is the particle's local index on that PU. After the resampling step, *m n* copies of particles are selected. We use the set shown in equation (4.1) to represent the resampling result.

$$S_t = \{ \left(M_t^{(i,j)}, N_t^{(i,j)}\right) : i \in \{1,\ldots,n\}, j \in \{1,\ldots,m\}, N_t^{(i,j)} \in \{1,\ldots,mn\} | \sum N_t^{(i,j)} \in mn \}$$

(4.1)

where *t* is the time step (or iteration step, used interchangeably in this paper), $M_t^{(i,j)}$ is a selected particle and $N_t^{(i,j)}$ is the associated number of copies for particle $M_t^{(i,j)}$. Note that particles that are eliminated, i.e., having zero copy after resampling, are not included in S_t. Given the above information, the particle routing answers the following two questions: (1) How to select particles on PU with surplus of particles? (2) How to choose the destination PU for a selected particle? A routing policy will provide a solution to the above questions. An important feature of particle routing is that if multiple copies of the same particle need to be transferred across the network, only one copy of the particle plus a number indicating the duplicate number are transferred. This removal of duplicated particles reduces communication costs because the destination PU can easily make multiple copies of the received particle locally.

The routing result can be defined by a set shown in equation (4.2):

$$R_t = \{ \left(M_t^{(i,j)}, P^k, N_t^{(i,j)}\right) : i,k \in \{1,\dots,n\}, j \in \{1,\dots,m\},\ SN_t^{(i,j,k)} \leq N_t^{(i,j)} |\ i \neq k \tag{4.2}$$

where P^k is a destination PU for $M_t^{(i,j)}$ and $SN_t^{(i,j,k)}$ is the number of copies for particle $M_t^{(i,j)}$ to be sent to P^k. Therefore, the routing problem can be defined as a function *f*: *S*→*R*, where *S* is the set containing the selected particles and their associated copies after resampling, and *R* is the set to store the routing result. Note that not all particles need to be routed to other PUs, thus the particles in *R* is a subset of the particles in *S*. The routing algorithm is composed of two main steps. (1) Particle selection: this step decides how to select particles on PUs with surplus of them. (2) Destination selection: for a selected particle, a destination PU is decided to route the particle to. The algorithm is executed in an iterative manner until all the particles that need to be routed out have a destination PU assigned. Afterwards, particles are transferred according to the routing result, with duplicated copies transferred only once. The following sections present three routing

algorithms based on three different particle routing policies, including the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy.

4.2.2 Random particle routing policy

In the random routing policy, we randomly choose a particle from a PU with surplus of particles, and then select any PU with shortage of particles. Although the random routing policy may lead to large communication costs, it is still presented in this paper due to its easy implementation. We use this policy as the base to compare with other policies. Table 4.1 shows the random routing algorithm. To start the process, we first need to calculate the total numbers of copies of selected particles on each PU, and use that information to decide if a PU has surplus of particles or shortage of particles. If the total number of copies of a PU is larger than *m*, we save its information of the selected particles and associated number of copies in a set $S1_t$ as shown in equation (3). If a PU has less than *m* total copies of particles, we save this PU and its needed number of particles into a set $S2_t$ as shown in equation (4). Obviously $S1_t$ is a subset of S_t.

$$S1_t = \{\left(M_t^{(i,j)}, N_t^{(i,j)}\right): i\{1,\dots,n\}, j\{1,\dots,m\}, N_t^{(i,j)}\{1,\dots,mn\} | \sum N_t^{(i,j)} \in m \text{ for PU with index } i\} \quad (4.3)$$

$$S2_t = \{\left(P_t^k, RN_t^k\right): k \in \{1,\dots,n\} | RN_t^k = m - \sum N_t^{(i,j)} > 0 \text{ for PU with index } k\} \quad (4.4)$$

We randomly choose a particle $M_t^{(i,j)}$ in $S1_t$ and a PU with index k in $S2_t$, add the route information ($M_t^{(i,j)}$, P_t^k, 1) to the set R_t, and then decrement $N_t^{(i,j)}$, RN_t^k, and $\sum N_t^{(i,j)}$. If $\sum N_t^{(i,j)}$ = m or RN_t^k = 0, we remove the information for the corresponding PU from $S1_t$ or $S2_t$.

Table 4.1 Random routing algorithm

Main steps at time step *t*

1. Calculate $\sum N_t^{(i,j)}$ for PU with index *i*, *i=1, 2,..., n*.
2. If $\sum N_t^{(i,j)}$ > m, save all the $\left(M_t^{(i,j)}, N_t^{(i,j)}\right)$ for PU with the index *i* to the set $S1_t$.
3. If $\sum N_t^{(i,j)}$ < m, save $\left(P_t^k, RN_t^k\right)$, where $RN_t^k = m - \sum N_t^{(i,j)}$ to the set $S2_t$.
4. Randomly select a particle $M_t^{(i,j)}$ in $S1_t$.
5. Randomly select a PU with index *k* in $S2_t$.
6. Append the route $\left(M_t^{(i,j)}, P_t^k, 1\right)$ to the set R_t. If

   $M_t^{(i,j)}$ and P_t^k already exist in R_t, increase the previous $SN_t^{(i,j,k)}$ by one but do not add $\left(M_t^{(i,j)}, P_t^k, 1\right)$.

   $N_t^{(i,j)}$- -, RN_t^k- -, and $\sum N_t^{(i,j)}$ - -;

   if $\sum N_t^{(i,j)}$= m

   Remove information of particles on PU with the index *i* from $S1_t$.

   if RN_t^k = 0

   Remove information of PU with the index *k* from $S2_t$.
7. Repeat Step 4 to 6 until both $S1_t$and $S2_t$are empty.

Figure 4.2 and figure 4.3 show an illustrative example of the random particle routing policy. Figure 4.2 shows the first part of random particle routing policy that how to select the particles in each PU which have the surplus particles, and it will be transfer to the PU with shortage of particles and figure 4.3 shows how to routing the particles. In figure 4.2, there are 4 PUs with the index P1, P2, P3 and P4 and each PU has 10 particles (*K*). Figure 4.2 (a) shows the selected particle index and its copy number. For example, for PU_01 has 2 copies of particle with the index 3 ($M^{(P1,3)} = (P1,3)\ and\ N^{(P1,3)} = 2$), and 4 copies of particle with the index

7 ($M^{(P1,7)} = (P1,7)\ and\ N^{(P1,7)} = 4$) and 10 copies of particle with the index 9 ($M^{(P1,9)} = (P1,9)\ and\ N^{(P1,9)} = 10$).

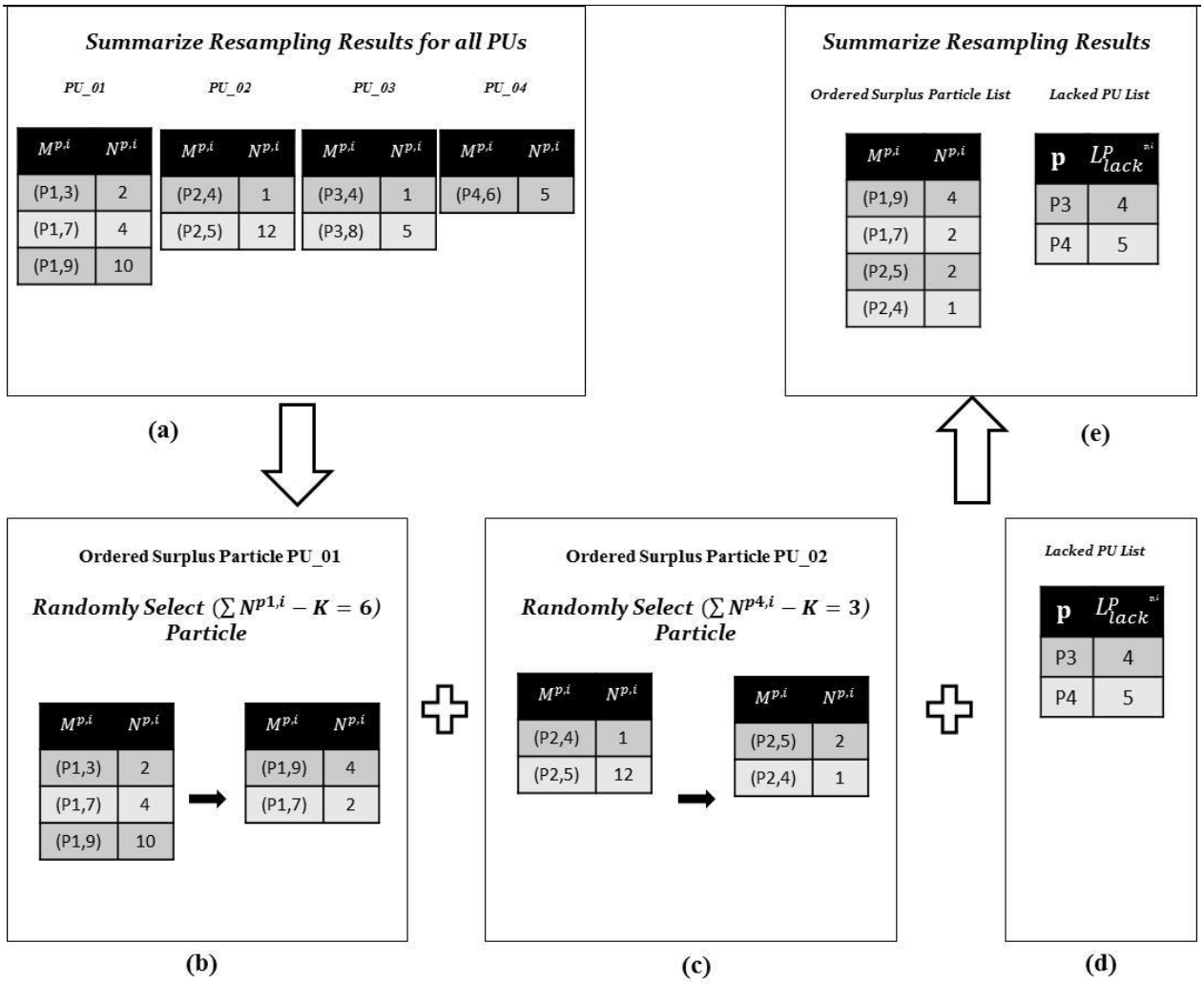


Figure 4.2 Example of random particle routing policy part 1

Based on the resampling result, we know the PU_01 and PU_02 have the number of surplus particle is 6 ($\sum N^{P1,i} - K = 16 - 10 = 6$) for PU_01 and 3 ($\sum N^{P2,i} - K = 13 - 10 = 3$) for PU_02. Figure 4.2 (b) and figure 4.2(c) show how to select the particle from the PU_01 and PU_04. Because it is the random routing policy, the entire select rule is totally random, that means we will random select six particles from the left side table in figure 4.2(b) and also random select ten particles from the left side table in figure 4.2(c). 4.2(d) shows the PUs with

shortage of particles and the number of particles it needs. Combine the (b), (c) and (d) we got the resampling results table in figure 4.2(e). The figure 4.3 shows the second part of random particle routing policy that how to routing the particle.

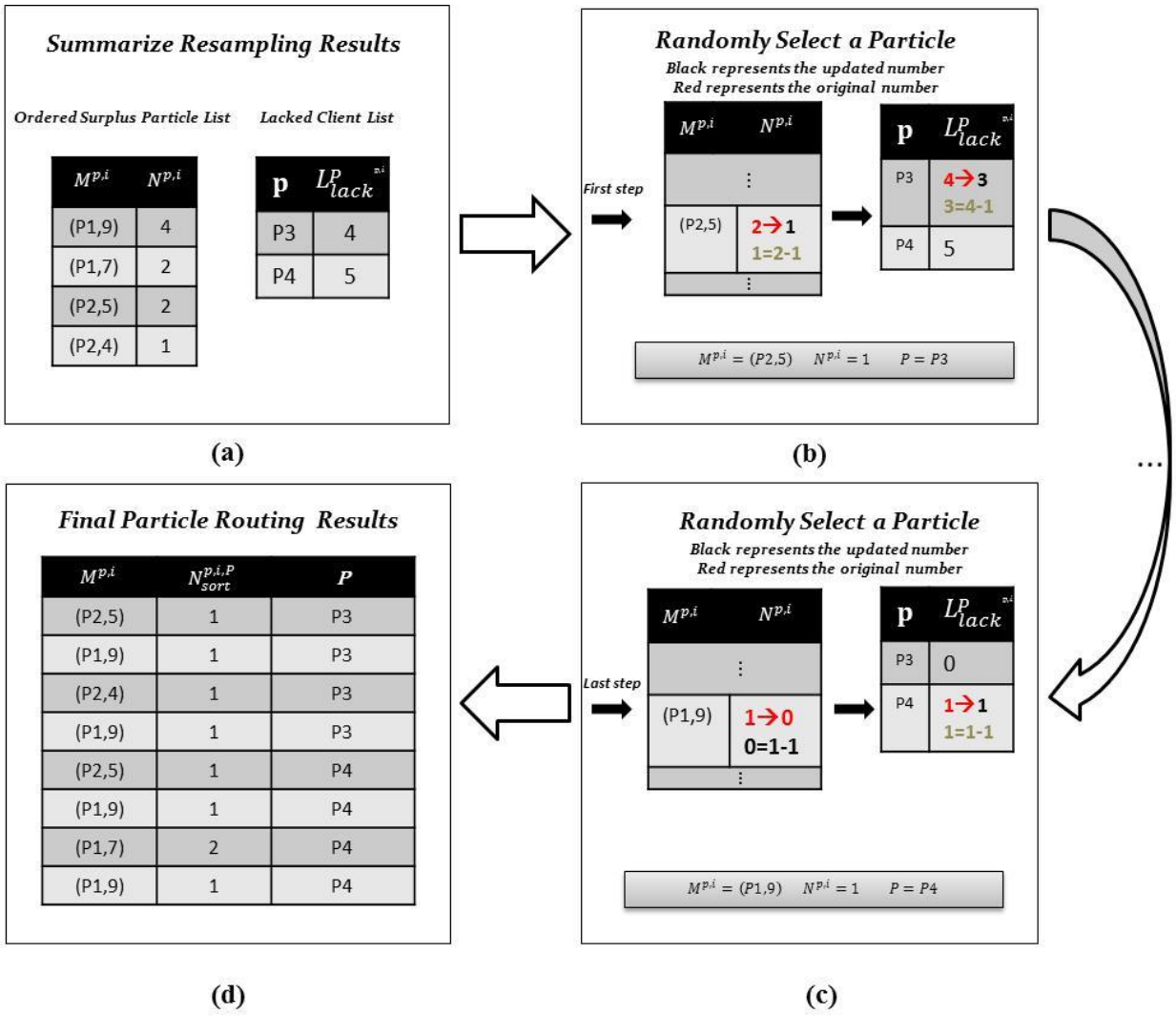


Figure 4.3 Example of random particle routing policy part 2

Continue from the last step of the part1 in figure 4.2 (e), in figure 4.3 (a) we need routing all nine particles in left side table to right side table. In random particle routing policy, the rule still is totally random. That means every time we random select one particle from the left side table for PU_03 or PU_04 until the particle in left side's copy number equals zero. The figure 4.3

(b) shows an example when the first step random select the (P2, 5) and how to update the both table, then it needs record the routing information in blow. Figure 4.3(c) shows the last step of the example, which every particle's copy number equals zero and the PU with shortage particle's lack number also equals zero. Finally, combine all the routing information which shown in figure 4.3 (d). From the example, we know the random particle routing policy used the random select method for every time's select. So, this method will cause the large communication cost, which is why we introduce the minimal transfer routing policy.

4.2.3 Minimal transfer particle routing policy

While the random routing policy is easy to implement, it does not exploit the global information of particles' distribution among PUs to reduce communication costs. The minimal transfer routing policy exploits that information and aims to achieve the minimal number of particles to be sent across the network, given that replicated particles need to be sent only once between two PUs. An intuitive way to implement this is to start from selecting the particle with the most number of copies from $S1_t$, and send them to the PU that needs the most number of particles in $S2_t$. This reduces the overall number of transfers because the duplicated particles are transferred only once between PUs. Compared to the random routing policy, we sort the obtained set $S1_t$and $S2_t$based on the number of copies $N_t^{(i,j)}$ and the needed number of particles RN_t^k respectively in the descending order. The sorted sets are denoted as $S1_t{}'$and $S2_t{}'$accordingly. In each iteration, we conduct the following three steps. 1) Select the first particle (denoted as $M_t^{(Fi,Fj)}$) in $S1_t'$ and its host (denoted as P_t^{Fi}) as the source PU. 2) The destination PU (denoted as P_t^{Fk}) is the first PU in $S2_t{}'$. 3) Rout a number of copies of particle $M_t^{(Fi,Fj)}$ to P_t^F. To calculate how many copies of $M_t^{(Fi,Fj)}$ to be routed to the destination PU P_t^{Fk}, we first

compare the number of copies $N_t^{(Fi,Fj)}$ of particle $M_t^{(Fi,Fj)}$ with the total extra number of particles $(\sum N_t^{(Fi,Fj)} - m)$ on the source PU P_t^{Fi}, and select the smaller number (denoted as *Q*) between the two. We then compare *Q* with the needed number of particles (RN_t^{Fk}) on the destination PU P_t^{Fk}. If the former is smaller than the latter, we transfer *Q* copies of the $M_t^{(Fi,Fj)}$ to P_t^{Fk}, and then remove $M_t^{(Fi,Fj)}$ from $S1_t'$ and update the needed number of particles of P_t^{Fk} by subtracting *Q*. If the former is greater than or equal to the latter, we transfer the needed number (RN_t^{Fk}) of particle $M_t^{(Fi,Fj)}$ to P_t^{Fk}, update the number of copies of $M_t^{(Fi,Fj)}$ by subtracting RN_t^{Fk}, and remove P_t^{Fk} from $S2_t'$. In both cases, the corresponding route info is added to R_t. Afterwards we resort the updated $S1_t'$ and $S2_t'$ and execute the same steps for the next iteration. This continues until $S2_t'$ is empty (which means all the PUs in $S2_t'$ have received the needed number of particles). The algorithm is described in Table 4.2.

Table 4.2 Minimal transfer routing algorithm

Main steps at time step *t*

1. Calculate $\sum N_t^{(i,j)}$ for PU with index *i*, *i=1, 2,..., n*.
2. If $\sum N_t^{(i,j)}$ > m, save all the $\left(M_t^{(i,j)}, N_t^{(i,j)}\right)$ for PU with the index *i* to the set $S1_t$.
3. If $\sum N_t^{(i,j)}$ < m, save $\left(P_t^k, RN_t^k\right)$, where $RN_t^k = m - \sum N_t^{(i,j)}$ to the set $S2_t$.
4. Sort the set $S1_t$ in the descending order by $N_t^{(i,j)}$ to $S1_t$.
5. Sort the set $S2_t$ in the descending order by RN_t^i to $S2_t$.
6. Select the first particle ($M_t^{(Fi,Fj)}$) in $S1_t{}'$ to be sent and its host PU as the source PU.
7. Select the first PU (P_t^{Fk}) in $S2_t{}'$ as the destination PU.
8. Compare $N_t^{(Fi,Fj)}$ with $(\sum N_t^{(Fi,Fj)} - m)$ and select the smaller number (denoted as *Q*) between the two.
9. Compare *Q* with RN_t^{Fk} :
   a. If $Q < RN_t^{Fk}$

      PassNum $SN_t^{(Fi,Fj,Fk)} = Q$,

      then $RN_t^{Fk} = RN_t^{Fk} - Q$,

      remove $M_t^{(Fi,Fj)}$ from $S1_t'$
   b. If $Q \geq RN_t^{Fk}$

      PassNum $SN_t^{(Fi,Fj,Fk)} = RN_t^{Fk}$,

      then $N_t^{(Fi,Fj)} = N_t^{(Fi,Fj)} - RN_t^{Fk}$

      remove P_t^{Fk} from $S2_t'$
10. Append the route info $(M_t^{(Fi,Fj)}, P_t^{Fk}, SN_t^{(Fi,Fj,Fk)})$ to the set R_t.
11. Sort the set $S1_t{}'$ in the descending order by $N_t^{(i,j)}$.
12. Sort the set $S2_t{}'$ in the descending order by RN_t^i.
13. Repeat step 6 to step 12 until $S2_t{}'$ is empty.

Figure 4.4 and figure 4.5 show an illustrative example of the minimal transfer particle routing policy. We use the sample example as shown in the example of random particle routing policy. Figure 4.4(b) shows the first step work is the descending sorting two PUs lists which have surplus of particles (PU_01 and PU_02).

According to the minimal transfer routing algorithm, first of all, $M^{(P1,9)}$will be selected 6 copies since it has the largest number of copies in PU_01 and $M^{(P2,5)}$ will be selected 3 copies in PU_02 with the same reason. Figure 4.4 (e) shows the PU with shortage of particles and the number of particles it needs. Combine the (e) and the right side table of (c) and (d), we got the resampling result for the start of part 2. The first step of part 2 also is descending sort the both lists with surplus of particles and shortage of particles. So, based on the figure 4.5 (b), $M^{(P1,9)}$ will be sent from PU_01 since it has the largest number of copies of 6 and its destination PU will be the PU_04 because it lock the largest number of particles. Then, we need update both table shows in figure 4.5 (c), because PU_04 just need 5 particles and (P1, 9) has 6 copies, so update the $N^{(P1,9)} = 1$, then remove the P4 in right table since the lock number of P4 already is zero and also record the route { (P1,9), 5, P4}. After updating the related information, we continue the process until we get the final routing set.

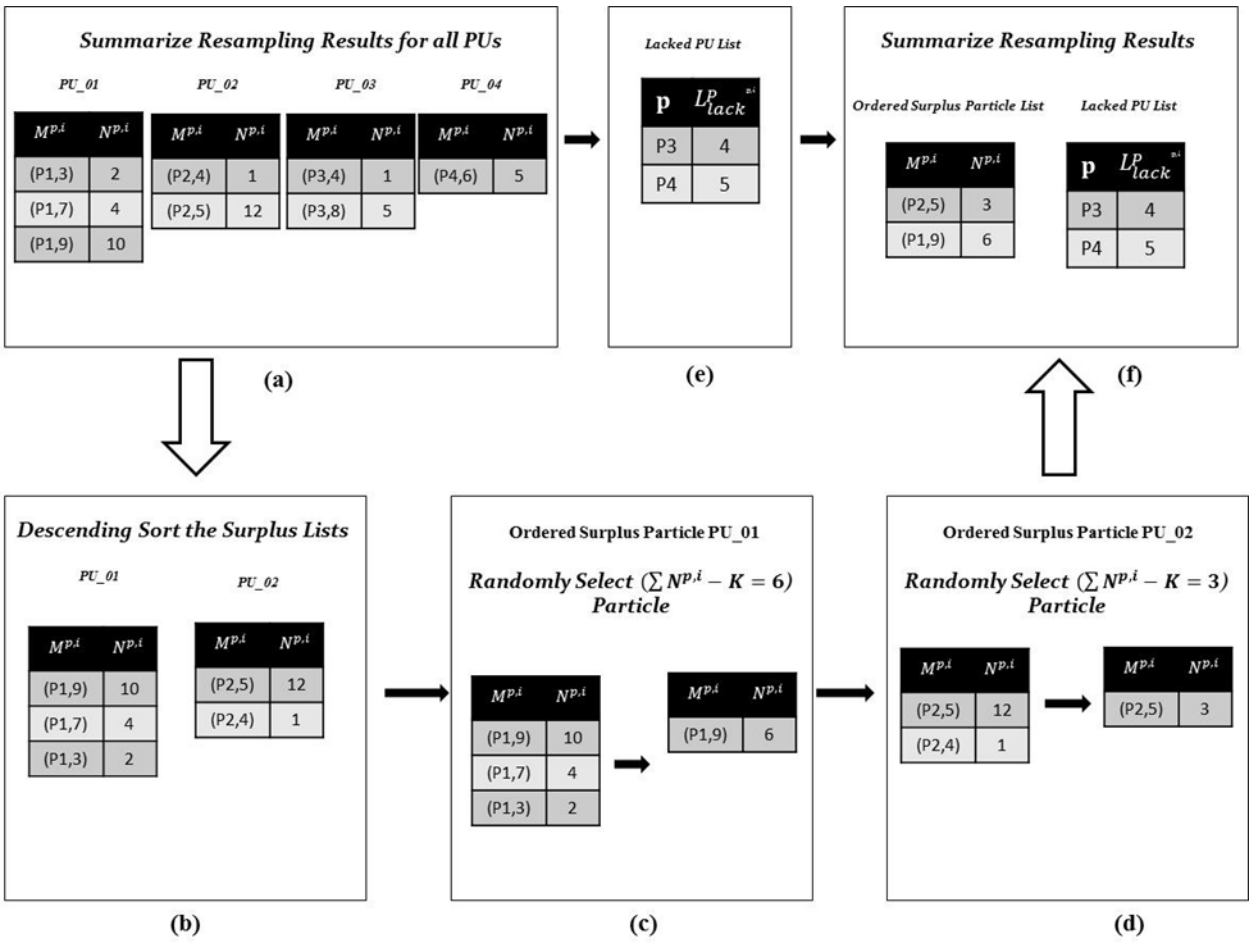


Figure 4.4 Example of minimal transfer particle routing policy part 1

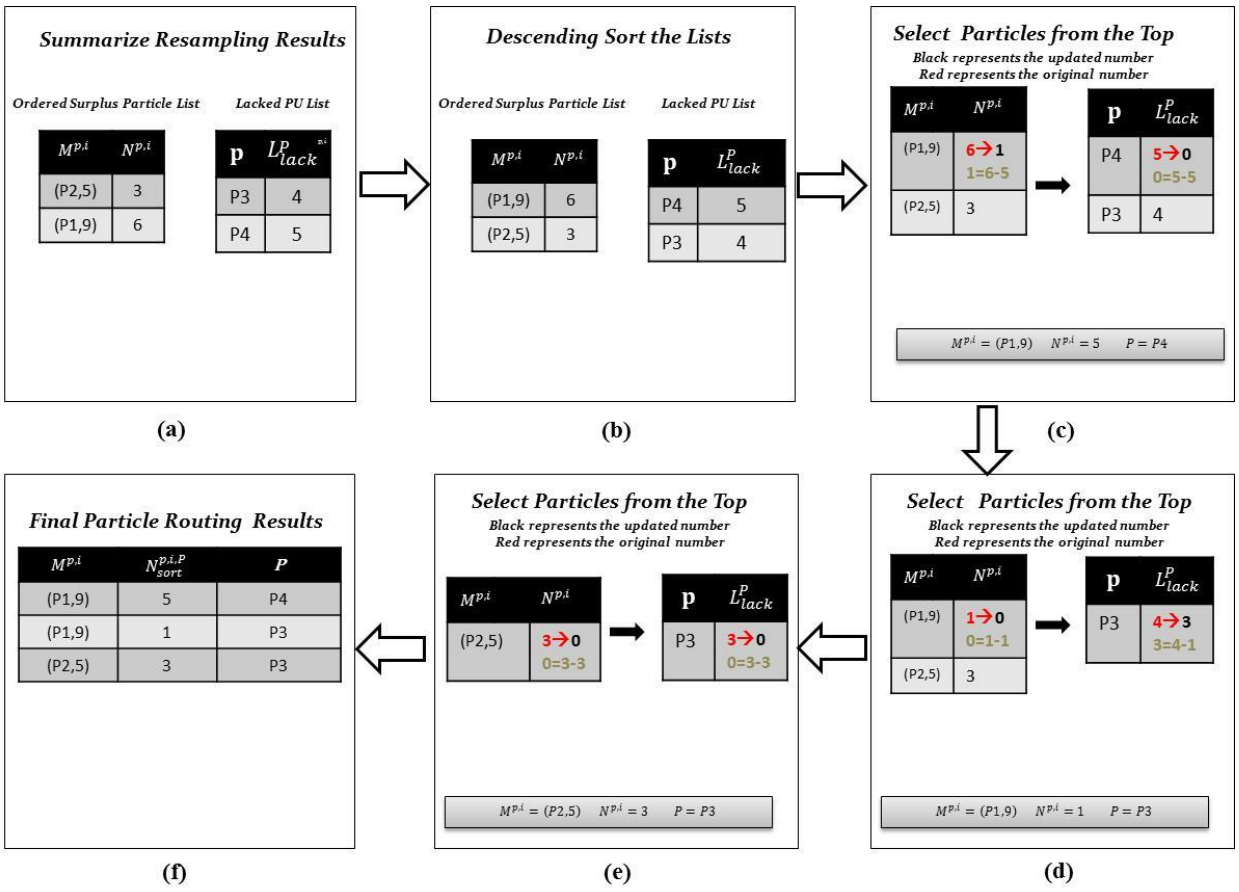


Figure 4.5 Example of minimal transfer particle routing policy part 2

One thing we want to mention in here is we will not descending the list again whatever for the surplus particle list or the lacked PU list. For example, in figure 4.5 (d), after update the information, the number of copies for (P1,9) is smaller than (P2,5), but we will continue follow this order to select the particle instead descend the list and then select the particle. Finally, after both $N^{(p,i)}$ and L^P_{lack} equals zero, we finish the routing part work and get the final particle routing result. In this example, based on the routing result overall three transfers occur because the duplicated particles only need to be transferred once.

4.2.4 Maximal balance particle routing policy

Different from the minimal transfer policy that is de-signed only from the communication cost point of view, the maximal balance routing policy aims to achieve the maximal balance of the particles with the high weights among all PUs after particle routing. This makes sense because the "good" particles with high weights are more likely to survive in future iterations. A balanced distribution of these particles among all PUs may reduce the need of particle routing in future iterations. The basic idea of the maximal balance policy is to select "good" particles and evenly distribute them to all PUs. Towards this goal, we need to define a criterion to decide which particles are "good". In our work, we set the criteria based on the number of copies of particles after resampling (which essentially reflect the weights of the particles). Specifically, a particle is “good” if its number of copies is greater than or equal to a threshold T.

Assuming the total number of “good” particles is G, each PU will receive no less than $\lfloor G/n \rfloor$ (the largest integer less than or equal to (*G/n*) and no more than $\lceil G/n \rceil$ (the smallest integer greater than or equal to G/n) “good” particles. To ensure that after receiving the “good” particles the total number of particles does not exceed m, each PU needs to first allocate “empty” spaces for receiving the “good” particles. To support this, the maximal balance particle routing algorithm includes two stages. The first stage involves only the “non-good” particles. In this stage, PUs transfer particles to each other using the minimal transfer policy described in the previous section (in this stage each PU uses $\lfloor mn - G/n \rfloor$ instead of m as the desired number of particles). After the first stage, all PUs have about the same number (with plus or minus *1* difference if cannot be evenly divided) of “non-good” particles. The second stage is to distribute the “good” particles to all PUs. Specifically, we sort all the “good” particles in descending order in a set $S1_t{}'$ and complete the following steps. 1) Choose the first particle in $S1_t{}'$. 2) The

destination PU will be selected from all the PUs with indexes from 1 to *n* in turn. Every step, one copy of the first particle will be distributed to a PU by turn until all its copies are distributed. We remove this particle's information from $S1_t'$ and execute the same procedure for the next particle until all the "good" are distributed.

Table 4.3 shows the maximal balance particle routing algorithm. We partition S_t into two sets $S1_t$ and $S2_t$. In $S1_t$ the weights of all the selected particles are larger than or equal to the threshold *T*, and they are smaller than *T* in $S2_t$. Firstly we apply the minimal transfer routing policy to $S2_t$, and obtain the routing set $S2_t$. For $S1_t$, we evenly distribute all the copies of the particles to all the PUs, and get the routing set $R1_t$. The final routing set $R_t = R1_t \cup R2_t$. If $S1_t$ is empty, which means no particles are considered as “good”, the maximal balance routing algorithm essentially gives the same result as the minimal transfer algorithm.

Table 4.3 Maximal balance routing algorithm

Main steps at time step *t*

1. Set a threshold *T*.
2. Partition S_t into two sets $S1_t$ (all the particles whose number of copies $\geq T$) and $S2_t$ (all the particles whose number of copies$< T$).
3. Apply the minimal transfer routing policy (table 4.2) to $S2_t$ to get the routing set $R2_t$. Note in this step, each PU uses $\lfloor mn - G/n \rfloor$ instead of m as the desired number of particles.
4. Sort $S1_t$ according to the number of copies of the particles in the descending order to ${S1_t}'$. If $S1_t$ is empty, use minimal transfer routing policy (table 4.2) to get the routing result.
5. Start with the first particle in ${S1_t}'$ and first PU and the following steps:
   a. Assign their copies to all the PUs only by one, and decrement its number of copies $N_t^{(i,j)}$.
   b. Append this route in $R2_t$.
   c. If $N_t^{(i,j)} = 0$, remove the particle from ${S1_t}'$.
   d. Repeat the e ~ c process until ${S1_t}'$ is empty.
6. The final routing $R_t = R1_t \cup R2_t$.

Figure 4.6 and figure 4.7 show the illustrative example of the maximal balance routing policy. We use the sample example as shown in the example of random particle routing policy and minimal transfer particle routing policy. First of all, different compare to the minimal transfer policy, we put all the resampling result together to a list at first (figure 4.6(b)), and then descending sort this list according to the value of $N^{(P,i)}$. In this example, the predefined threshold T is 8. Therefore, the particle $M^{(P2,5)}$ and $M^{(P1,9)}$ are "good" particles and will be evenly distributed to all the PUs.

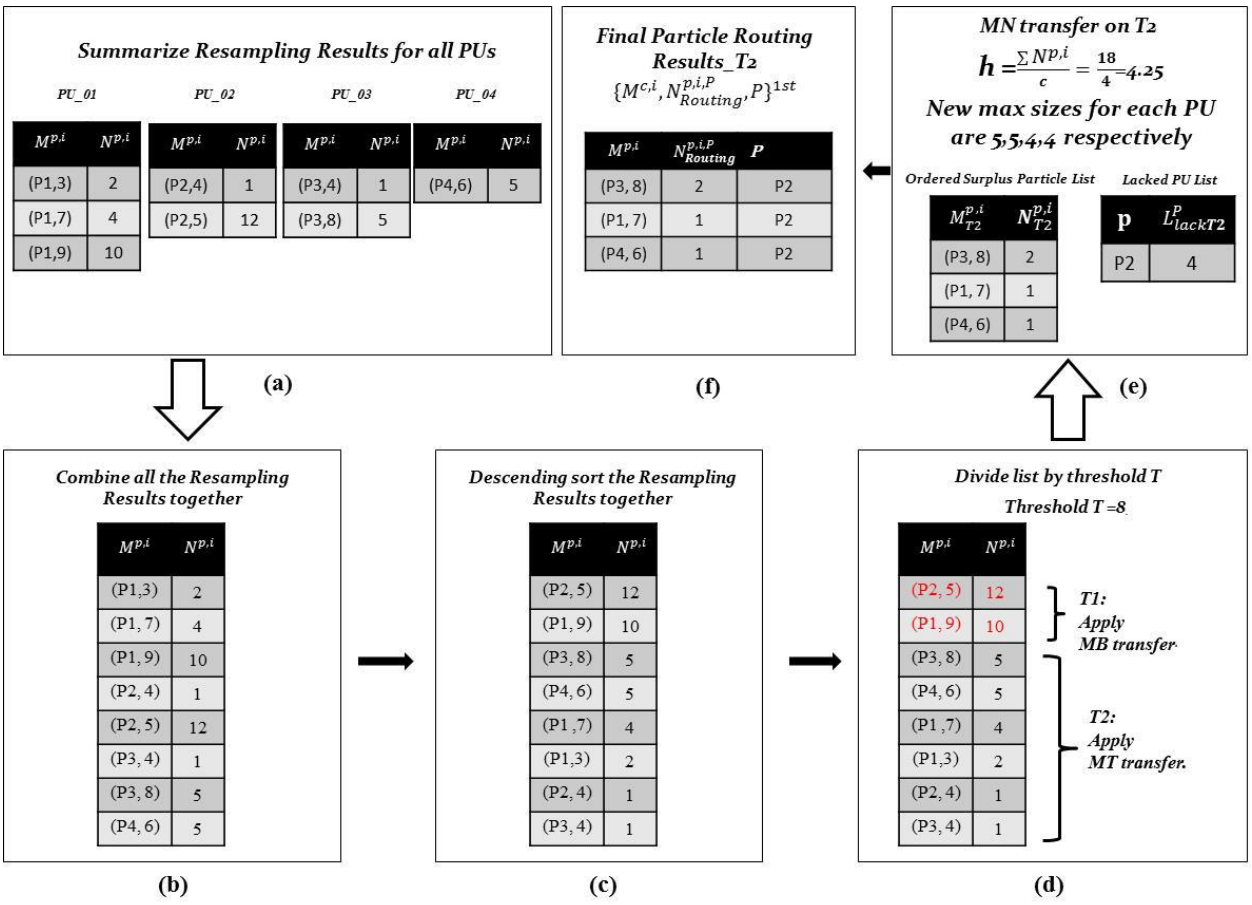


Figure 4.6 Example of maximal balance particle routing policy part 1

In figure 4.6(d), the list divide to two parts, the first part is the "good" particles which will be evenly distributed to all the PUs, we note it as T1. The remaining particles will be routed by the minimal transfer routing policy and we note it as T2. Firstly, we need handle the T2 part, because we need make every PU have the almost evenly space to get the "good" particle. This is because we need ensure every PU after receiving the "good" particles the total number of particles does not exceed *m*, each PU needs to first allocate "empty" spaces for receiving the "good" particles. For process the T2, we need calculate the particle routing result without the "good" particles. Sum the total $N^{(P,i)}$ but without the "good" particle, the $\sum N^{(P,i)\prime}$ =18 and divide it to 4 PUs. So, each PU the max size is 5,5,4,4 respectively. According to this, we get the

particle before routing result shown in figure 4.6 (e) and get the routing result in figure 4.6(f) by use minimal transfer policy.

After finished the T2, for the T1 part, 12 copies $M^{(P2,5)}$ and 10 copies of $M^{(P1,9)}$ will be transferred to all the PUs one by one. Because each PU the max size is 5,5,4,4 in T2, so each PU's lack number is 5,5,6,6 respectively. The figure 4.7 (b) and (c) show how to transfer the particle and update the information one by one, the rule is every time just only pass one particle in left side. Also, every time just sign the particle to one PU then change to the next PU one by one. Finally, combine the two routing results together which is the final routing result for the maximal balance particle routing policy. The maximal balance particle routing policy will transfer more particle compare to the minimal transfer particle routing policy (at least it is same when the entire particle' $N^{(P,i)}$ < threshold), but since the particles with the high weights are appeared in every PU with almost evenly, it will bring maybe two advantages: 1) may bring the more accurate simulation result because the "good" particles evenly in every PU which can generate more probability. 2) it may reduce communication cost in future iterations, which because the "good" particles already survived in every PU, so it will not get the big difference in future iteration, the routing will decreased.

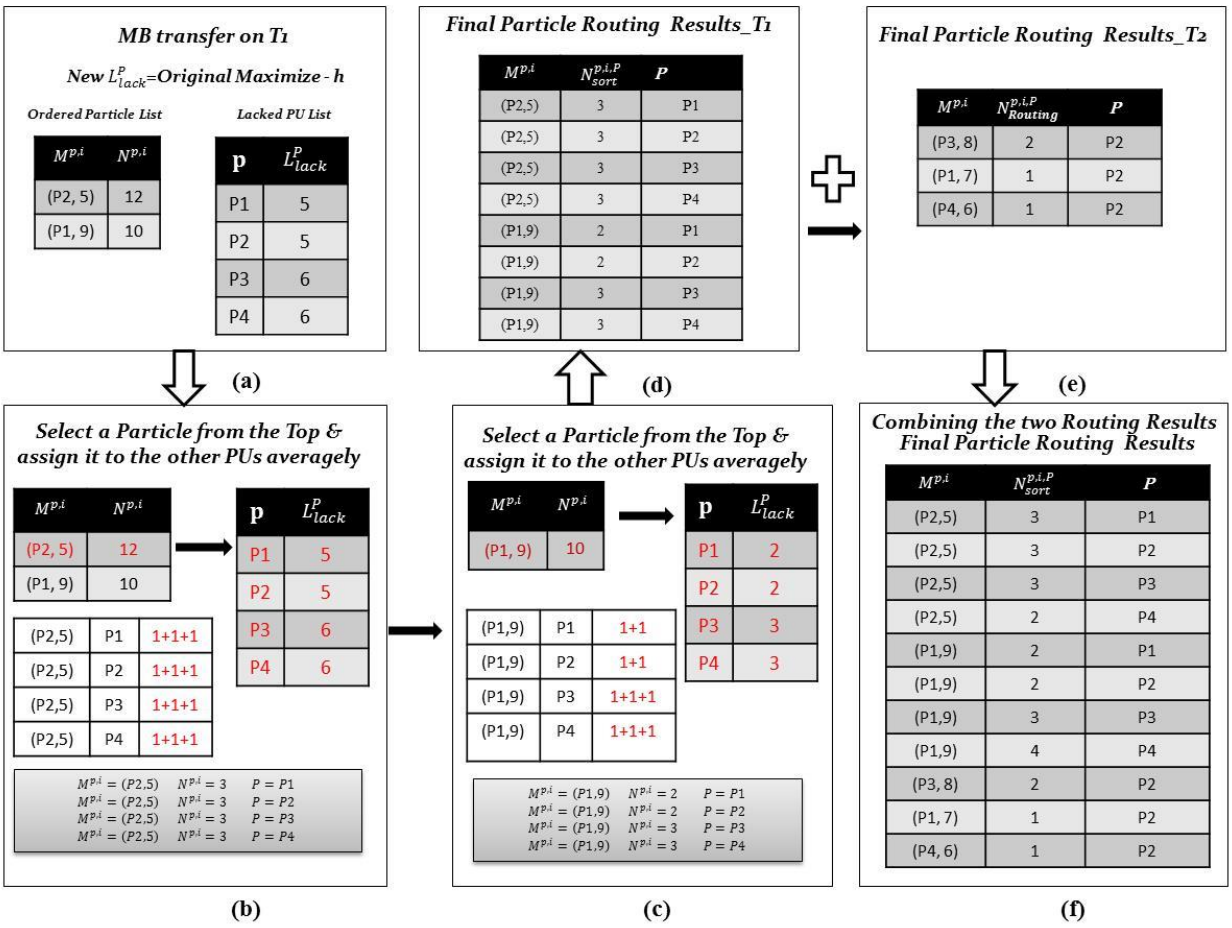


Figure 4.7 Example of maximal balance particle routing policy part 2

4.3 Experimental Designs

In our work, we evaluate the particle routing in distributed particle filter with centralized resampling based on the data assimilation system of large-scale wildfire spread simulation. The used model in this work is DEVS-FIRE which we have already introduced in section 3.1. For the experimental design we use the totally same experiment environment in section 3.3.3 which uses the real-world GIS data and fuel data. The cell space dimension is 200×200 and the cell size is 20 (m). The GIS data are airborne LiDAR (Light Detection and Ranging) raster-based terrain data. The fuel data was obtained by classifying a multispectral QuickBird (DigitalGlobal) image. Those data were acquired from Huntsville area, Texas, during the leaf-off season in March 2004

by M7 Visual Intelligence of Houston, Texas. The ignition point is set to the point (90, 55) of the cell space for all of the simulations. The observation data (ground temperature sensor data) from the real fire are collected every 30 minutes.

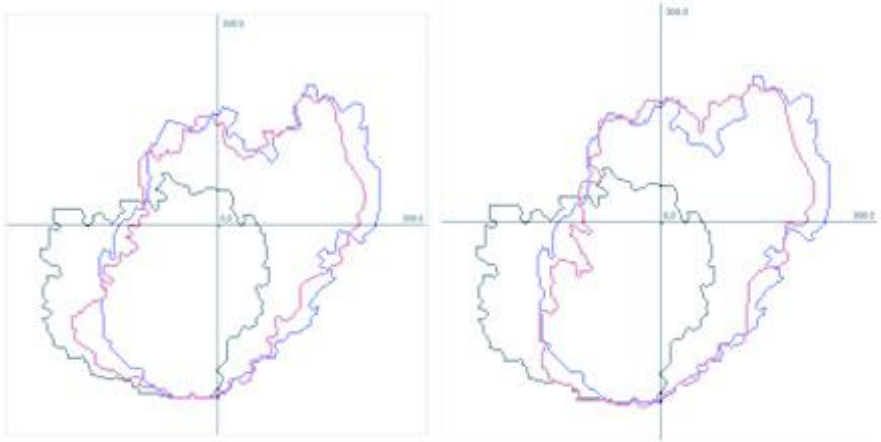
In our experiments, we still choose to use imprecise wind conditions which the wind speed and wind direction changed together. As we shown in Table 3.1, the real wind speed and direction are 8 (mph) and 180 (degrees) with random variances added every 30 minutes. The variances for the wind speeds are in the range of –2 to 2 (mph) (denoted as 8 ± 2 in the table), and the variances for the wind direction are in the range of -20 to 20 (degrees) (denoted as 180 ± 20 in the table). Our experiment introduces errors to the wind speeds, which are randomly generated based on the wind speed of 6 (mph) with variances added in the range of –2 to 2 (mph) and also the wind direction of 130 (degrees) with added variances in the range of ±20 (degrees) in the same time. For wind directions, the degrees indicate the angle between the north directions clockwise to the direction from where the wind comes.

All the experiments use 6 PUs (every PU has 50 particles, total 300 particles) to run 6 hours' simulation (12 steps and every step is 1,800 seconds) in all the experiments. Among these 6 PUs, one of them is functioned as a CU when completing the centralized resampling function for the global resampling step, otherwise a regular PU like others. All experiments are conducted under the supercomputer named Cheetah, which has 14 nodes, 160 computing cores, 32 CPUs, and 264 GB system memory.

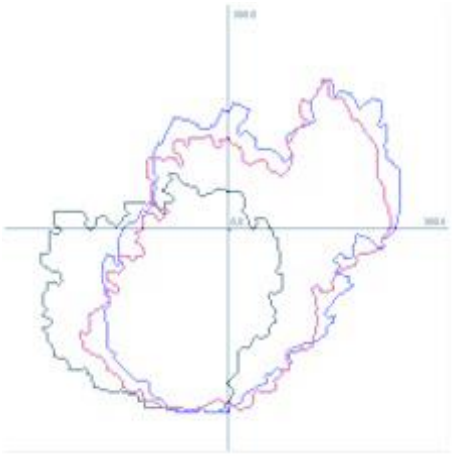
4.4 Experimental results and analysis

In this set of experiments, we conduct various experiments to show the simulation results using different routing policies including the random transfer policy, the minimal transfer routing policy, and the maximal balance routing policy. Figure 4.8 displays the real fire, the simulated

fire, and the filtered fire by assimilating the real data into wildfire spread simulation using the random routing policy, the minimal transfer routing policy, and the maximal balance routing transfer policy respectively. In the figures, all the filtered fires (display in red) are close to the real fire (display in blue) although we run the data assimilation simulations with the error data. Compared to the simulated fires (display in black), all the simulation results are greatly improved. To quantitatively examine the results, we choose the symmetric set difference as the metric to measure the similarity of the fires. In mathematics, the symmetric set difference of two sets is the set of elements in either set, but not in both. We use it to compare two fire fronts, which is the number of cells inside one of the fire front shapes, but not in both. The smaller the symmetric set difference, the more similar the two fire fronts are (the symmetric set difference of two same fire fronts is 0). Figure 4.9 shows the symmetric set differences of the simulated fire (compared to the real fire) and that of the filtered fire (compare to the real fire) using three routing policies including the random routing policy, the minimal transfer routing policy, and maximal balance routing policy. In the figure, the values of the filtered fires are the average of 6 independent runs. The horizontal axis represents the time step, and the vertical axis represents the symmetric set difference value in terms of the number of cells.



(a) (b)



(c)

Figure 4.8 Comparisons of real fire, simulated fire, and filtered fire using different routing policies. (a) Random routing policy. (b) Minimal transfer routing policy. (c) Maximal balance routing policy

From the figures, it can be seen that the symmetric set differences of the filtered fires are smaller than those of the simulated fires after step 5. With the increase of the time step, (i.e., when more sensor data are assimilated), the difference between the simulated fire and the filtered fire becomes more and more notable. At step 12, the symmetric set difference of the filtered fire is more than half of the symmetric set difference of the simulated fire. Also, the simulation results which use three different routing policies can get the similar accurate result. From Figure 4.8 and Figure 4.9 we conclude that the data assimilation using three different routing policies including the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy all significantly improve the simulation results. There is little difference between the three polices from the simulation results point of view. This is expected because in the centralized resampling all particles are resampled in each step. The three policies differ only in how particles are routed after resampling and thus only impact the communication cost but have little impact on the data assimilation results.

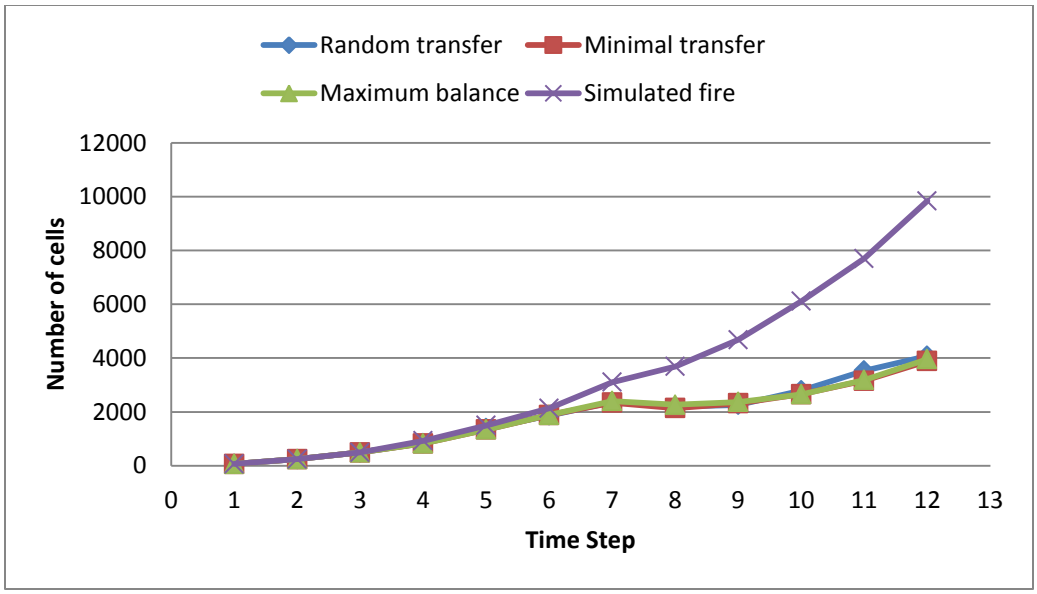


Figure 4.9 Symmetric set differences for simulated fire and filter fire with three different routing policies

To compare the communication cost of the three policies, Figure 4.10 shows the number of particles to be transferred in every step for the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy. Figure 4.11 shows the total number of particles to be transferred for the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy. From the figures we know that both the minimal transfer routing policy and the maximal balance routing policy significantly reduce the transfer number of the particle states and the minimal transfer routing policy has the lowest number of transfers.

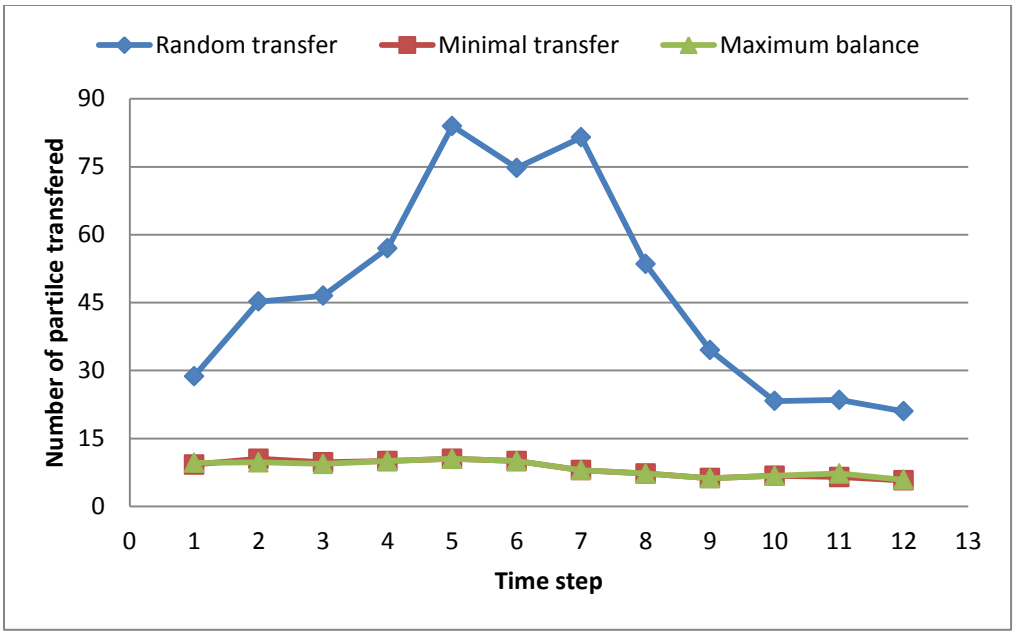


Figure 4.10 Number of particles to be transferred for the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy.

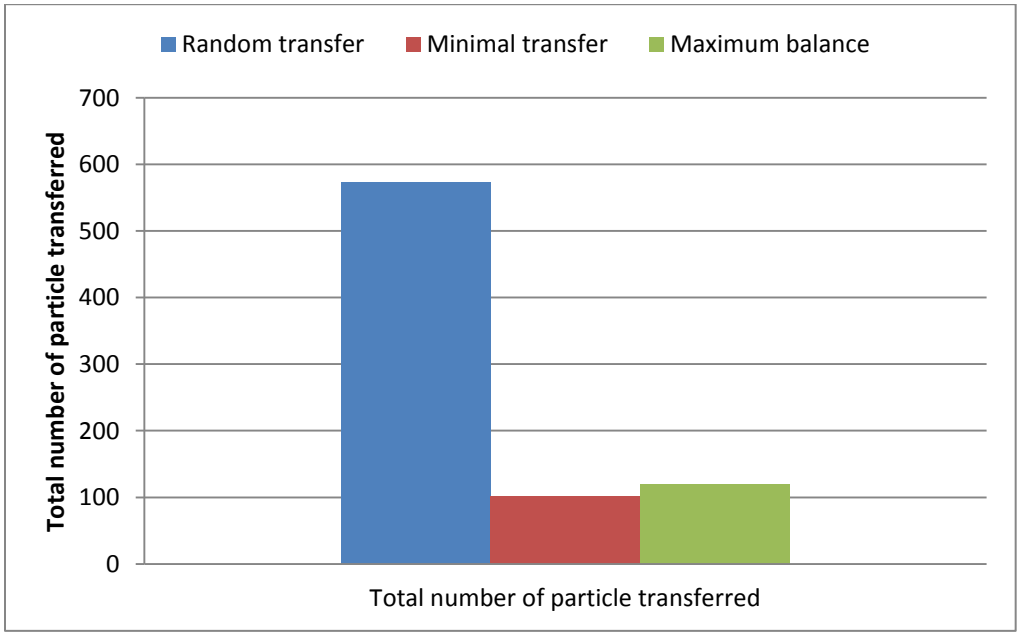


Figure 4.11 Total numbers of particles to be transferred for the random routing policy, the minimal transfer routing policy, and the maximal balance routing policy

For the high dimensional spatial temporal simulation because the size of each particle is large due to the high dimensional state it represents. So we intentionally increase the state size by

ourselves. We made one of the GIS data (aspect data used in this experiment) to be part of the particle state. So, the state of particle become to two parts: fire shape and aspect data. The size of the aspect data which we add to state is 15MB in this experiment. Figure 4.12 shows we get the totally same state transfer number after we increased the state compare to the state only just have fire shape. From the figure, we conclude increase the size of the state will not affect the number of state transfer in same experiment environment.

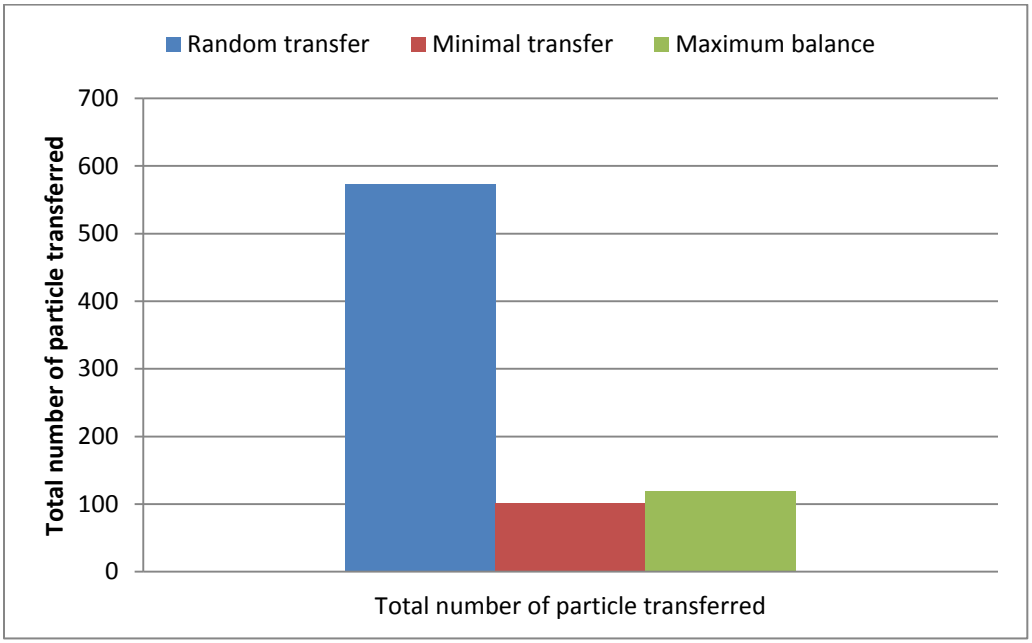


Figure 4.12 Total numbers of particles to be transferred for random transfer, minimal transfer and maximal balance after increase the state size

Figure 4.13 shows the execution time after we increase the state size. From the figure, we can see the total time is increased, but it did not increased too much, this is because we use the super computer (cheetah) is very fast and it is also share the memory and desk.

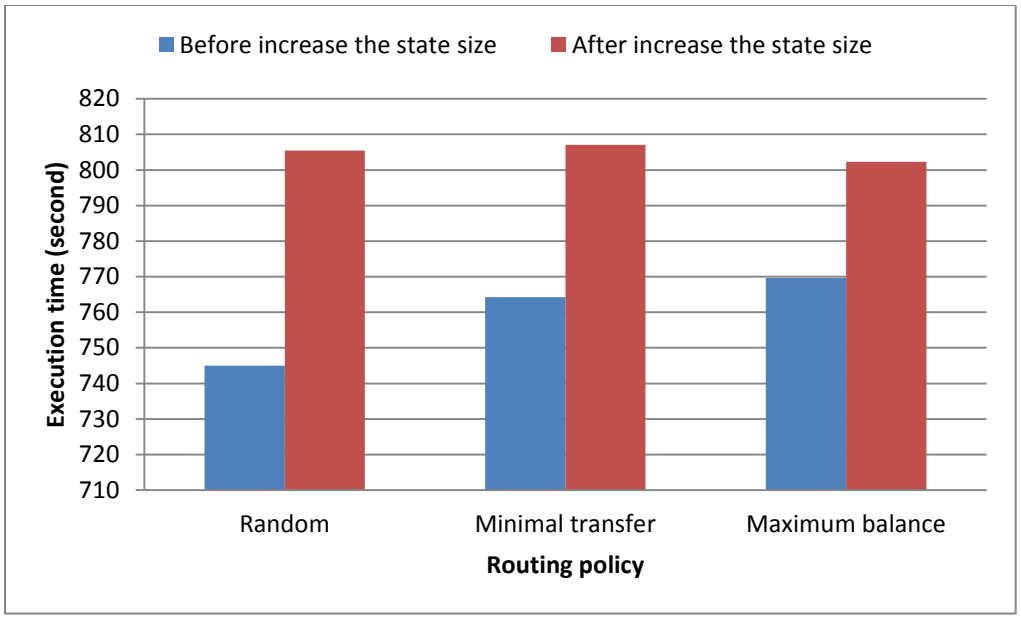


Figure 4.13 Compare to the total time cost between before add the state size and after add the state size for three different routing policies

4.5 Conclusions

In this chapter, we propose two centralized particle routing policies named as the minimal transfer routing policy and the maximal balance routing policy and show their impacts on distributed particle filters with centralized resampling. We evaluate the proposed methods based on data assimilation of a large-scale wildfire spread simulation. Experimental results show that the minimal transfer policy is the best choice for the centralized resampling because it can achieve the same data assimilation results with the lowest number of state transfers compared to the random routing policy and the maximal balance routing policy.

5 PARTICLE ROUTING IN DISTRIBUTED PARTICLE FILTERS WITH DECENTRALIZED RESAMPLING

5.1 Particle routing in decentralized resampling

The centralized resampling schema faithfully implements the particle filtering algorithm. Nevertheless, it suffers from scalability issues because it relies on a CU. To support scalable PF-based data assimilation, the distributed resampling is needed. Several distributed resampling schemas have been introduced, among which the distributed RNA uses a fully decentralized resampling schema. The main idea of the distributed RNA is no CU at all. A designer can define sub-groups among PUs and carry out full independent resampling only within the groups. The three distributed RNA methods distributed RNA with regrouping, distributed RNA with adaptive regrouping and distributed RNA with local exchange we have already discussed in section 2.4.3. Basically, the particle routing in decentralized resampling just happened between the two groups, expects there is three and more PUs in one group for distributed RNA with regrouping and distributed RNA with adaptive regrouping. That is because the RPA applied inside of group if three and more PUs in one group. So, we can continue apply the Minimal Transfer Particle Routing Policy and Maximal Balance Particle Routing Policy inside of the group when this group have three and more PUs. In this situation, the particle routing in decentralized resampling is totally same as the particle routing in centralized resampling since both of them have the CU and can apply the different routing policy. The distributed RNA with local exchange is a different method because every group only contains one PU. So the particle routing happened on a deterministic way only among the neighboring PUs and the routing is done through local communication in every step.

5.2 Distributed resampling with local and global particle routing

Based on the work which the distributed RNA with local exchange where PUs exchanges particles with local neighbors. Specifically, PUs are arranged in a ring topology and in each iteration each PU passes a subset of randomly selected particles to its neighbor in the anticlockwise order, and then carries out resampling locally. This local resampling schema supports a large degree of parallelism due to data parallelism and elimination of the centralized resampling step. However, it gives rise to a large number of iterations until full resampling is achieved. To overcome this problem, the strict local communication principle should be relaxed. Based on this idea, we propose using both local and global particle routing methods. The global particle routing is the same as in the centralized resampling algorithm described in Chapter 4, i.e., a CU is used to collect particles' weights and decide how to route the particles by using the two different particle routing method: Minimal Transfer Particle Routing Policy and Maximal Balance Particle Routing Policy. The goal of the global routing is to take advantage of the full knowledge of all particles' weights to quickly and efficiently route the "good" particles to all PUs. To avoid impairing the scalability of the distributed resampling, the global routing is invoked only occasionally, e.g., once in every K steps. Table 5.1 shows the algorithm of the distributed resampling with local and global particle routing on both the PU side and the CU side.

Table 5.1 Algorithm of distributed resampling with local and global particle routing

Main steps at time step *t*

PU side:

For all the PUs (in parallel)

1. Give a predefined integer *K*.
2. Run the sampling step.
3. Calculate the importance weights of particles.
4. If *t % K =0*, go to step5 (start the global resampling and routing procedure), otherwise go to step 10 (start the local routing and resampling procedure).
5. Send all weights to the CU.
6. Receive routing information from the CU.
7. If having surplus of particles, send the selected particles (based on the received routing information from CU) to the CU.
8. If having shortage of particles, receive particles from CU.
9. End.
10. Pass a subset of particles (and associated weights) to its neighbor.
11. Normalize and resampling locally.
12. End.

CU side:

1. Give the same predefined integer K as PUs.
2. If t % K =0, go to step3 (activate the global resampling and routing). Otherwise skip this iteration.
3. Receive particles' weights from all PUs.
4. Normalize and resampling.
5. Compute routing information by applying the minimal transfer routing policy or the maximal balance routing policy.
6. Send the routing information to PUs.
7. Receive particles from PUs that have surplus of particles.
8. Send particles according to the routing information to the PUs that has shortage of particles.

9. End.

Figure 5.1 shows the RNA (a) and distributed resampling with local and global particle routing method (b). The main different are we give a predefined number *K* at first. That means the global routing only happened every K step. Before we start every step, we calculate the time step t% K at first, if t% K =0, we do the global particle routing. In that time, one of the PU also plays the CU's role (for example, PU_01 also is a CU in figure 5.1) which receives the particle's weight from all other PUs and does the resampling, then compute routing information by applying the minimal transfer routing policy or the maximal balance routing policy inside.

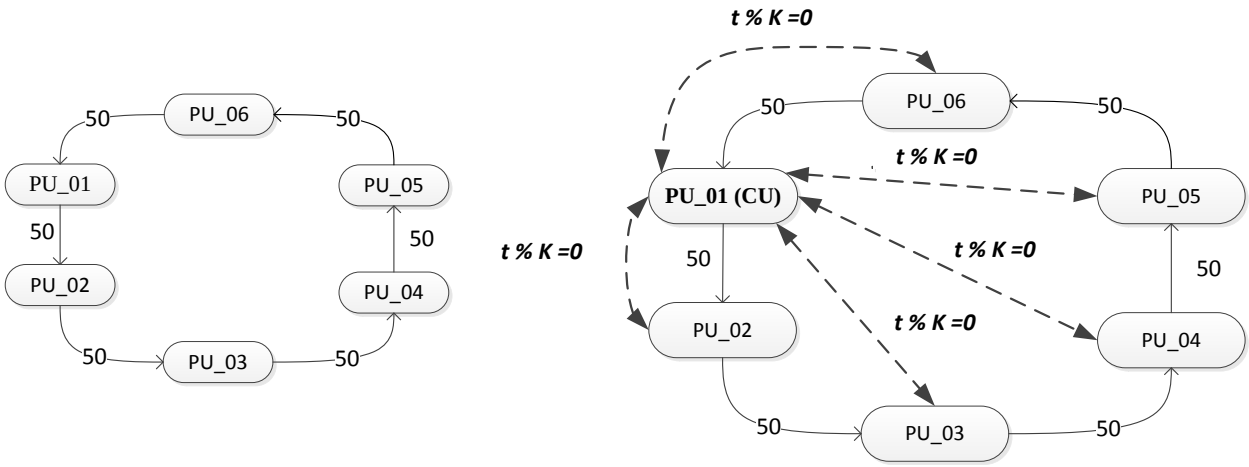


Figure 5.1 The different between RNA and distributed resampling with local and global particle routing method

If the t% K! =0, the rule is totally same as RNA method, every PU just pass a subset of particles and associated weights to its neighbor. Then normalize and resampling locally.

5.3 Experiment designs

We evaluate the particle routing in distributed particle filter with local and global particle routing method based on the data assimilation system of large-scale wildfire spread simulation. The used model in this work still is DEVS-FIRE which we have already introduced in section 3.1. For the experimental design we use the totally same experiment environment in section 3.3.3 and section 4.3 which uses the real-world GIS data and fuel data. For the weather data, we still choose to use imprecise wind conditions which the wind speed and wind direction changed together as we shown in Table 3.1. We do the same experiment using the distributed RNA, the distributed RNA with the minimal transfer routing policy, and the distributed RNA with the maximal balance routing policy. For the distributed RNA policy, each PU passes 10 particles to its neighbor in the anticlockwise order. For the latter two policies, we call the centralized resampling (the minimal transfer routing policy or the maximal balance routing policy) every 4 steps, and remove the duplicate particles. However, we don't do this in the distributed RNA policy, so its number of state transfer is 60 for 6 PUs in each step.

5.4 Experimental results and analysis

Figure 5.2 displays the number of transferred states of the three policies of the distributed RNA, the distributed RNA with the minimal transfer routing policy, and the distributed RNA with the maximal balance routing policy for step 4, step 8, and step 12 respectively. Figure 5.3 shows the total number of transferred states of the three policies of the distributed RNA, the distributed RNA with the minimal transfer routing policy, and the distributed RNA with the maximal balance routing policy. Note that all the values are the average of 6 independent runs. Note that the numbers of transferred particles are the same for all the steps except step 4, step 8, and step12. This is because we apply the minimal transfer routing policy or the maximal balance

routing policy every four steps, and the distributed RNA is used for other steps. For the steps where the minimal transfer routing policy or the maximal balance routing policy is applied, the numbers of the transferred states are greatly decreased, and they are reduced more by the distributed RNA with the minimal transfer routing policy. However, there is less obvious difference between the distributed RNA with the minimal transfer routing policy and the distributed RNA with the maximal balance routing policy regarding the total number of transferred states.

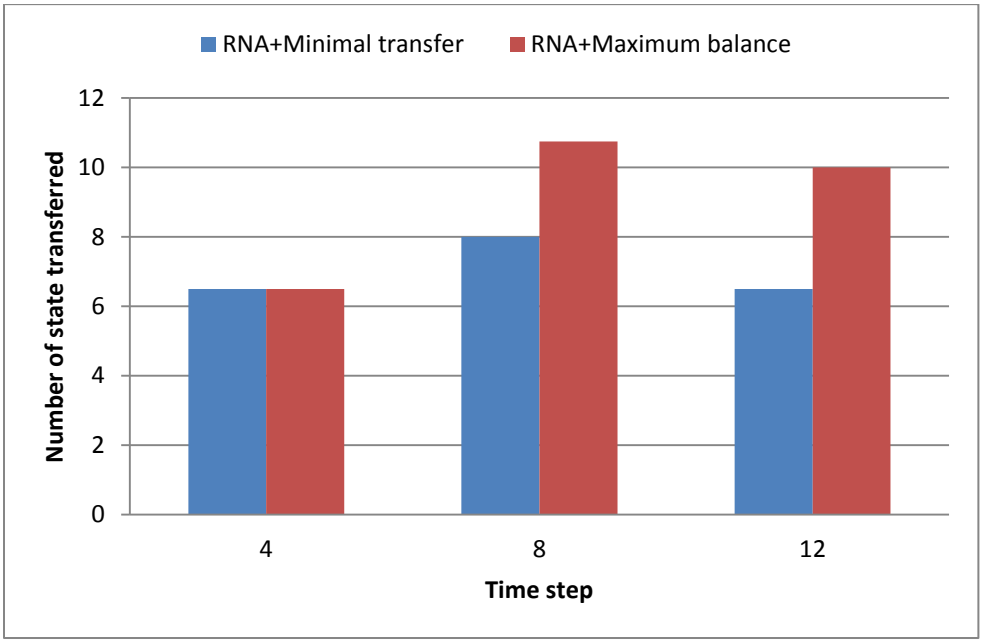


Figure 5.2 Number of transferred states for the distributed RNA with the minimal transfer routing policy and the distributed RNA with the maximal balance routing policy for step 4, 8, and 12

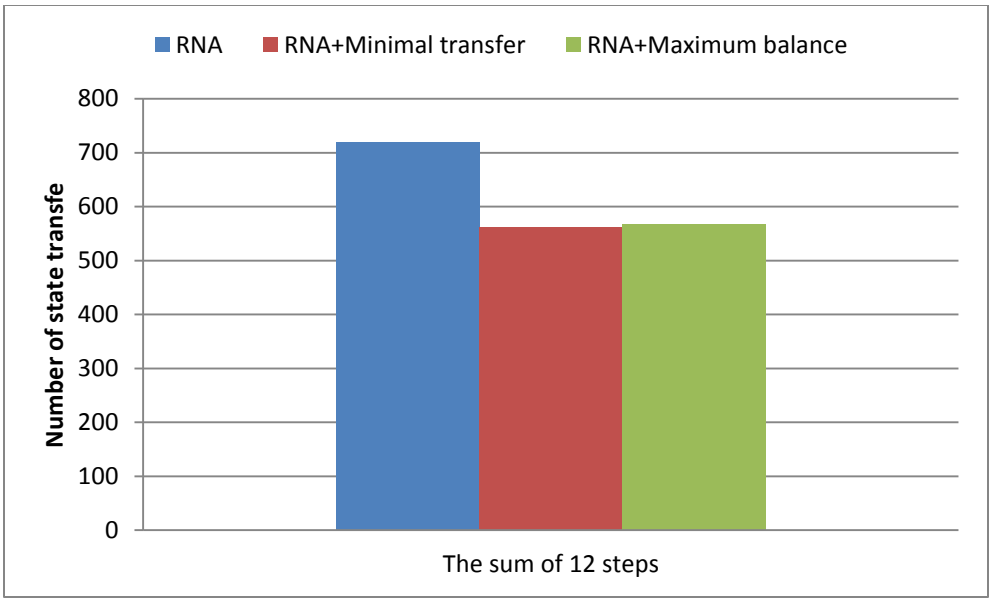


Figure 5.3 Total numbers of transferred states for the distributed RNA, the distributed RNA with the minimal transfer routing policy, and the distributed RNA with the maximal balance routing policy

Figure 5.4 display the real fire, the simulated fire, and the filtered fire by assimilating the real data into wildfire spread simulation using the RNA, centralized resampling, RNA and minimal transfer and RNA and maximal balance.

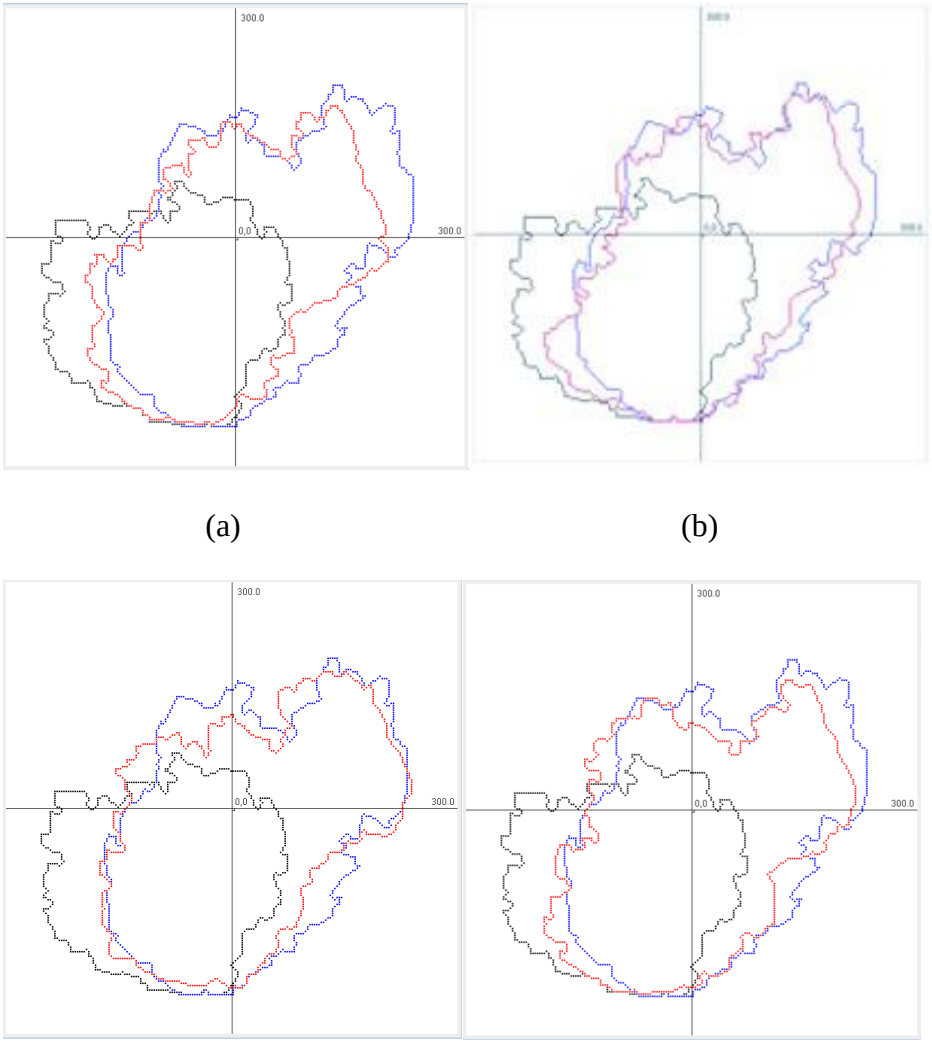


Figure 5.4 Comparisons of real fire, simulated fire, and filtered fire using different routing policies. (a) RNA (b) Centralized resampling. (c) RNA and Minimal transfer (d) RNA and Maximal balance

In figure 5.4, all the filtered fires (display in red) are close to the real fire (display in blue) although we run the data assimilation simulations with the error data. Compared to the simulated fires (display in black), all the simulation results are greatly improved. But, we can see the RNA get the worst filter result, but the result get better after we applied the minimal transfer and maximal balance with RNA together. For the accurate of simulation result, we continue use symmetric set differences to show it. Figure 5.5 shows the symmetric set difference of the

simulated fire, filter fire with centralized resampling with the minimal transfer routing policy and filter fire with RNA. We can see the result of RNA is worse than filter fire with centralized resampling.

Continue, the figure 5.6 shows the symmetric set differences between the real fire and the filtered fire using the centralized resampling with the minimal transfer routing policy, and the filtered fire using the distributed RNA respectively, the filtered fire using the distributed RNA with the minimal transfer routing policy, and the filtered fire using the distributed RNA with the maximal balance routing policy respectively for step 7 to 12. This is because the fire is small in the earlier steps and thus the difference is small too. We zoom in the later steps in order to better show the results. Although all of them have much less symmetric set differences than the simulated fire mentioned above, the distributed RNA has the worse results since it is a purely distributed resampling and suffers from the local resampling.

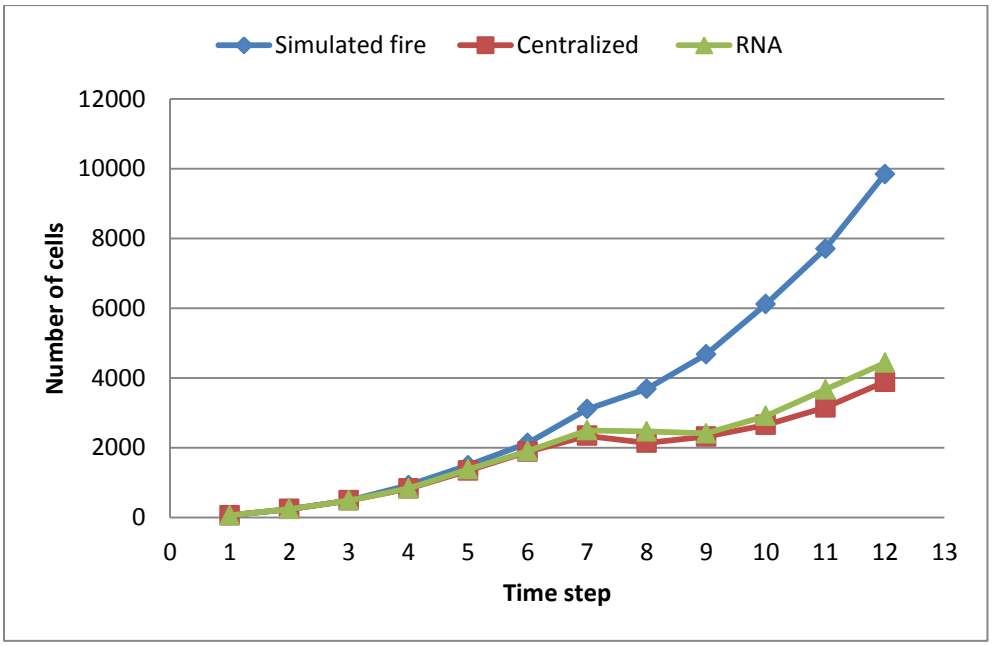


Figure 5.5 Symmetric set differences for simulated fire and filter fire with centralized resampling (minimal transfer) and filter fire with RNA

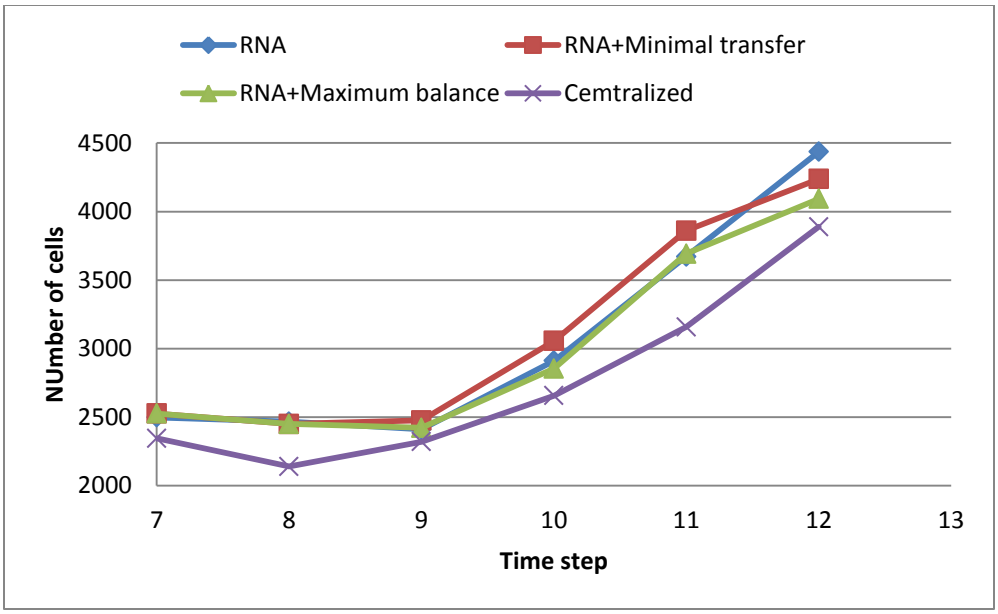


Figure 5.6 Symmetric set differences for the filtered fire with the centralized resampling using the minimal transfer routing policy, and the filtered fire with the distributed RNA, the filtered fire with the distributed RNA using the minimal transfer routing policy

Figure 5.7 displays the symmetric set differences for the distributed RNA, the distributed RNA with the minimal transfer routing policy, and the distributed RNA with the maximal balance routing policy at time step 8 and 12 respectively. At these two steps, both the distributed RNA with the minimal transfer routing policy and the distributed RNA with the maximal balance routing policy have smaller symmetric set differences than the distributed RNA, and the distributed RNA with the maximal balance routing policy has better results than the distributed RNA and the distributed RNA with the minimal transfer routing policy. This is because the maximal balance routing policy evenly distributes the copies of particles with high weights to all the PUs during the routing process. To summarize, the distributed RNA with the maximal balance routing policy has the best simulation results among all the routing policies above with

slightly more total number of transferred states compared to the distributed RNA with the minimal transfer routing policy.

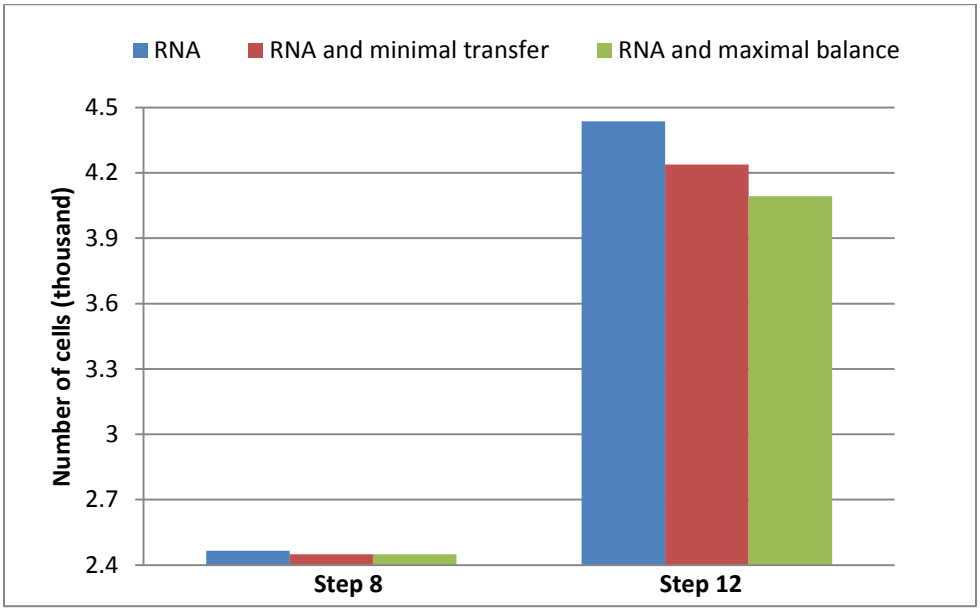


Figure 5.7 Symmetric set differences for filtered fire using the distributed RNA, filtered fire using the distributed RNA with the minimal transfer routing policy, and the filtered fire using the distributed RNA with the maximal balance routing policy at time step 8 and 12

The frequency of applying the centralized resampling to the decentralized sampling is an important factor. It is the tradeoff between the communication cost and the precision of the simulation results. We do the centralized resampling every 4 steps in all our experiments above, now we want to see the impact if we do the centralized resampling for shorter or longer step with different routing policy. Firstly, we change do the centralized resampling every 2 steps and every 5 steps. Figure 5.8 shows at time step 12, the symmetric set difference of the filtered fire using the distributed RNA, filtered fire using the distributed RNA with the minimal transfer routing policy for every 2 steps, 4 steps and 5 steps. From the figure, we can be seen: 1) the distributed RNA with the minimal transfer routing policy for every 2 steps get the best simulation result. This is because it did more centralized resampling compare with for the every 4 steps and every

5 steps. 2) The distributed RNA with the minimal transfer routing policy for every 5 steps get the worst result (even worse than RNA). This is not only because it just did twice centralized resampling, but also it did not finish the centralized resampling in last step.

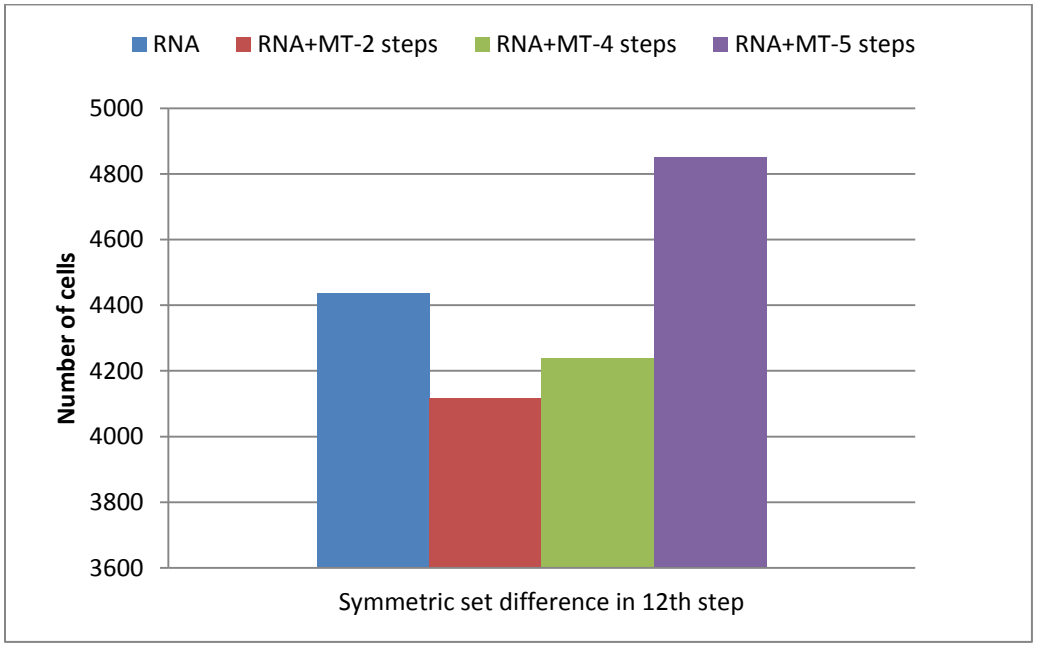


Figure 5.8 Symmetric set difference of the filtered fire using the distributed RNA and filtered fire using the distributed RNA with the minimal transfer routing policy every 2 steps, 4 steps and 5 steps at time step 12

Continue we apply the same shorter or longer step for the filtered fire using the distributed RNA with the maximal balance routing policy. Figure 5.9 displays the symmetric set differences of the filtered fire using the distributed RNA and the distributed RNA with the maximal balance routing policy for every 2 steps, 4 steps and 5 steps at time step 12. From the figure we know that the distributed RNA with the maximal balance routing policy every 2 steps has the smallest symmetric set difference and thus has the best simulation results. This is because the global routing is applied more often. Among the hybrid approaches with the global routing

and the local routing, the distributed RNA with the maximal balance routing policy every 5 steps even has the worse results since it only does the global routing twice and doesn't do it at this step.

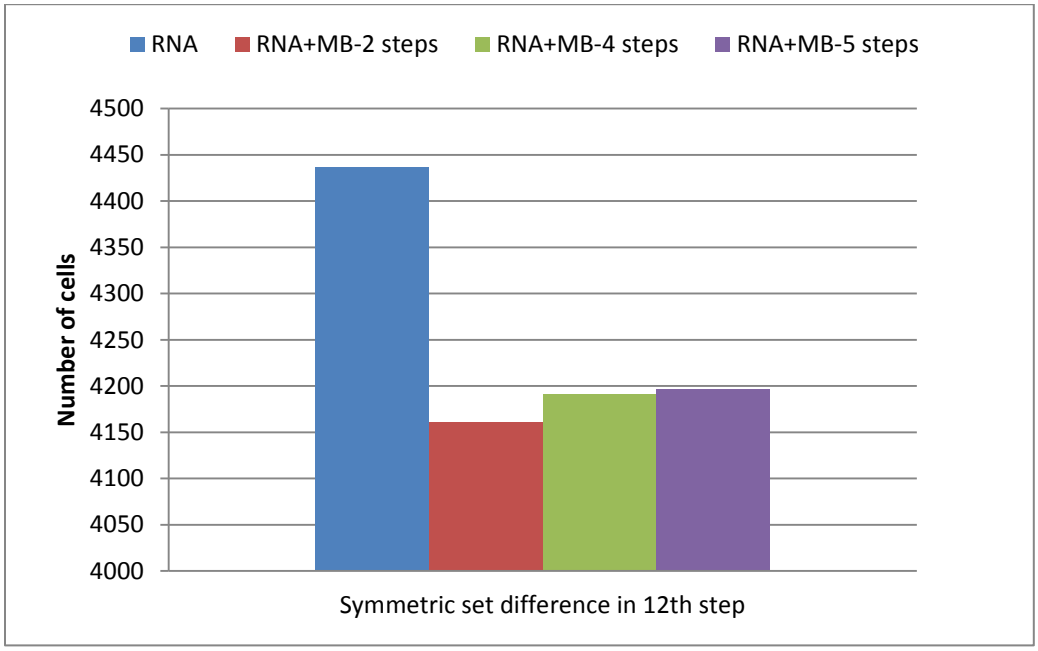


Figure 5.9 Symmetric set differences of the filtered fire with the distributed RNA, and the distributed RNA with the maximal balance routing policy every 2 steps, 4 steps and 5 steps at time step 12

Figure 5.10 displays the total number of transferred states for the distributed RNA with the minimal transfer routing policy and the distributed RNA with the maximal balance routing policy for every 2, 4, and 5 steps. From the figure we can see that if more centralized resampling steps are added, more numbers of state transfers are needed. However, for the same number of added centralized resampling steps, the distributed RNA with the minimal transfer routing policy has less number of state transfers. This is consistent with the previous observations.

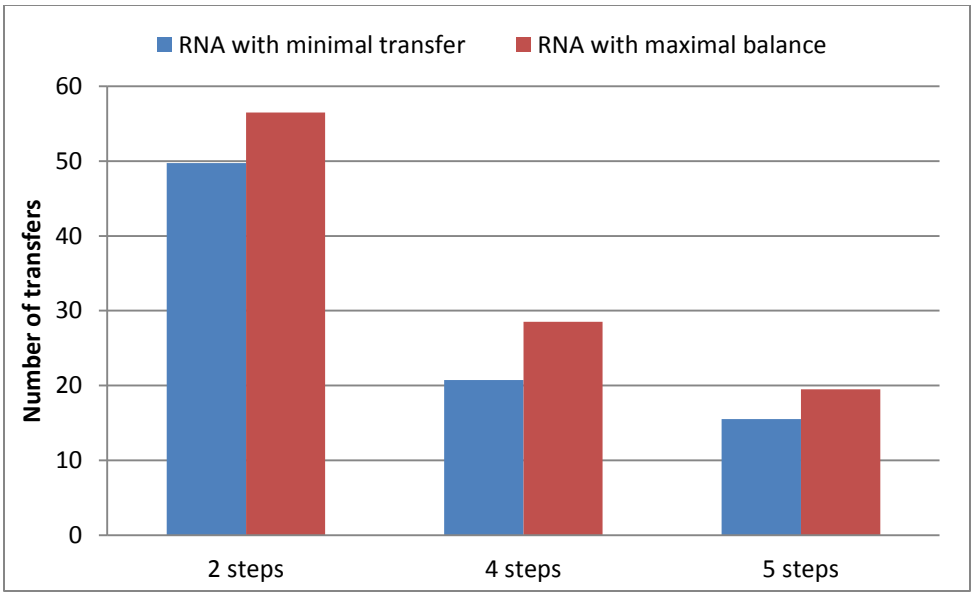


Figure 5.10 Total numbers of transferred states of the distributed RNA with the minimal transfer routing policy and the distributed RNA with the maximal balance routing policy for every 2, 4, and 5 steps at time step 12

5.5 Discussions and conclusions

In this chapter, we propose a hybrid approach that uses both local and global particle routing in the distributed resampling with non-proportional allocation (RNA). We show how the minimal transfer routing policy and the maximal balance routing policy can be used in the global routing step and their impacts on performance and accuracy of particle filtering. We evaluate the proposed methods based on data assimilation of a large-scale wildfire spread simulation. Experiment results shows, for the hybrid approach of particle routing in distributed resampling with RNA, the maximal balance routing policy is preferred in the global routing step because it can gain the best data assimilation results with slightly more number of state transfers compared to the minimal transfer routing policy.

6 CLOUD MAPREDUCE FOR DATA ASSIMILATION USING SEQUENTIAL MONTE CARLO METHODS IN WILDFIRE SPREAD SIMULATION

6.1 Motivation

Above chapters, we discussed develop a parallel and/or distributed computing method for particle filter-based data assimilation in DEVS-FIRE spread simulation for large-scale temporal systems with the tradition method. For the centralized resampling method, we have to face some issues since we have the CU exist, such as it still required a complicated scheme for particle routing, it make a complex PU design and area increase when more PU involve. Also, we need 100% know about our configuration such as how many machines we have, since we need coding them based on our design and put the simulation model in every PU.

"Cloud Computing" is a technology that uses the internet and central remote servers to maintain data and applications. Cloud computing allows consumers and businesses to use applications without installation and access their personal files at any computer with internet access. "Cloud" refers to large Internet services running on 10,000s of machines such as Amazon S3, Google AppEngine, Microsoft Windows Azure, etc. In cloud, the user do not need buy any machines, that means no upfront capital costs building data centers, buying servers, etc. Only do two things: 1) design the user own cloud application, 2) pay it when you use it. Therefore, if we use the cloud method for our data assimilation using sequential Monte Carlo Methods in wildfire spread simulation, we can get more accurate simulation result because we can use more machines but do not need buy more machines.

MapReduce is a software framework that allows developers to write programs that process massive amounts of unstructured data in parallel across a distributed cluster of

processors or stand-alone computers. Also MapReduce is a programming model for processing huge data sets on certain kinds of distributable problems using a large number of computers (nodes), collectively referred to as a cluster [31] [32]. In the MapReduce programming model, the computation takes a set of input *key/value* pairs, and produces a set of output key/value pairs. Users of the MapReduce library express their computation as two functions: *Map* and *Reduce*, which are then automatically executed in parallel by the underlying MapReduce framework. There are many different implementations of the MapReduce programming model, among which Apache's Hadoop is the most well-known one and it has been successfully applied for file based datasets. In this chapter, we propose a parallel and distributed computing method that uses Hadoop MapReduce to handle the data assimilation in wildfire simulation based on particle filters.

6.2 Overview of MapReduce and Hadoop

Followed by Google's work, many implementations of MapReduce emerged and lots of traditional methods combined with MapReduce have been presented until now [124].

• *Implementations of MapReduce*

Apache Hadoop is a software framework that helps constructing the reliable, scalable, distributed systems

[125]. Phoenix is a shared-memory implementation of Google's MapReduce model for data-intensive processing tasks [126]. Mars is a MapReduce framework on graphic processors (GPUs) [127]. Twister is a lightweight and Iterative MapReduce runtime system [128].

• *Traditional methods combined with MapReduce*

Apache Mahout can help to produce implementations of scalable machine-learning algorithms on Hadoop platform [129]. Menon et al. gave a rapid parallel genome indexing with MapReduce [130]. Blanas et al. proposed crucial implementation details of a number of well-known join strategies for log processing in MapReduce [131]. Ene et al. developed fast clustering algorithms using MapReduce with constant factor approximation guarantees [132]. Lin et al. presented three design patterns for efficient graph algorithms in MapReduce [133]. Moreover, MapReduce is rarely employed in the field of Systems Biology. In [134], the authors investigate whether a MapReduce approach utilizing on-demand resources from a Cloud is suitable to perform simulation tasks in the area of Metabolic Flux Analysis (MFA). Also, the authors introduced an implementation of a simple MapReduce method for performing fault-tolerant Mont Carlo computations in a massively-parallel cloud computing environment shown in [135].

The MapReduce architectural pattern has evolved as a generic, domain-independent processing method for large amounts of data. Two functions: map and reduce, are required to be implemented by the user with the following prototypes [32]:

map *(k1, v1) → list (k2, v2)*

reduce *(k2, list(v2)) → list (v2)*

Which list denotes a list of objects, k1 and k2 represent key types, v1 and v2 are value types. The input key/value pairs (k1, v1) are pairwise independent, thus, map can be invoked in parallel for all pairs, yielding an intermediate list of mapped (k2, v2) pairs. As an outstanding feature, MapReduce jobs may be defined by using native libraries such as C++ and Java. For our work, all the experiments use Java. More information about MapReduce can be found in [31] [32].

6.3 DEVS-FIRE & particle filter MapReduce approach

Based on the major step of particle filters shown in Figure 2.1 and the basic MapReduce prototypes, we introduce our new definition of MapReduce framework application of particle filters in DEVS-FIRE. The Algorithm 6.1 shows the **map** part, where key is the index of the particle, the value include all the necessary data, such as the GIS data, weather data (wind speed and wind direction), ignition points and sensor data. The Algorithm 2 shows the **reduce** part. In our method, we use the reduce part to do nothing, that means we parallelize the sampling and weight computation steps in map part, then do nothing in reduce part and put weight normalization and resampling parts. The reason why we cannot parallelize the weight normalization and resampling parts is the sampler requires information of all the particles for the systematic resampling. Also, the reduce part allows one to combine results produced in the map function based on the key. In our work, the key is particle's index, and all the information needed for each particle has already been produced in the map function. Thus there is no need to use the reduce function.

Table 6.1 Algorithm of DEVS-FIRE & particle filter MapReduce Approach

Algorithm6.1: Map (key, value)

Input:

//key: Particle index

//value: S= {GIS data, Weather data, Ignition Points, Senor Data}

Output:

//key': Particle index

//value': {Fire front, weight}

```
1 begin
2       let key' = key = Particle index
3       let value as the input of DEVS-FIRE
4    Sampling: run DEVSFIRESpread simulation and add the graph noise
5       Weight computation
6       value' = the fire sharp and the particle weight
7       output.collect (key' ,value');
1   end
```

Algorithm 6.2: Reduce (key, V)

Input: //key': Particle index

//value': {Fire front, weight}

Output: //key': Particle index

//value': {Fire front, weight}

Do nothing

Figure 6.1 shown the MapReduce particle filter (MapReducePF) algorithm: run the DEVSFIRE spread simulation in different node (computer), also sampling and weight computation in same node, all those parts are parallel worked. Then as we mentioned before,

since the resampling part have to get the information about all the particles, we put the weights normalization and resampling parts in one single node.

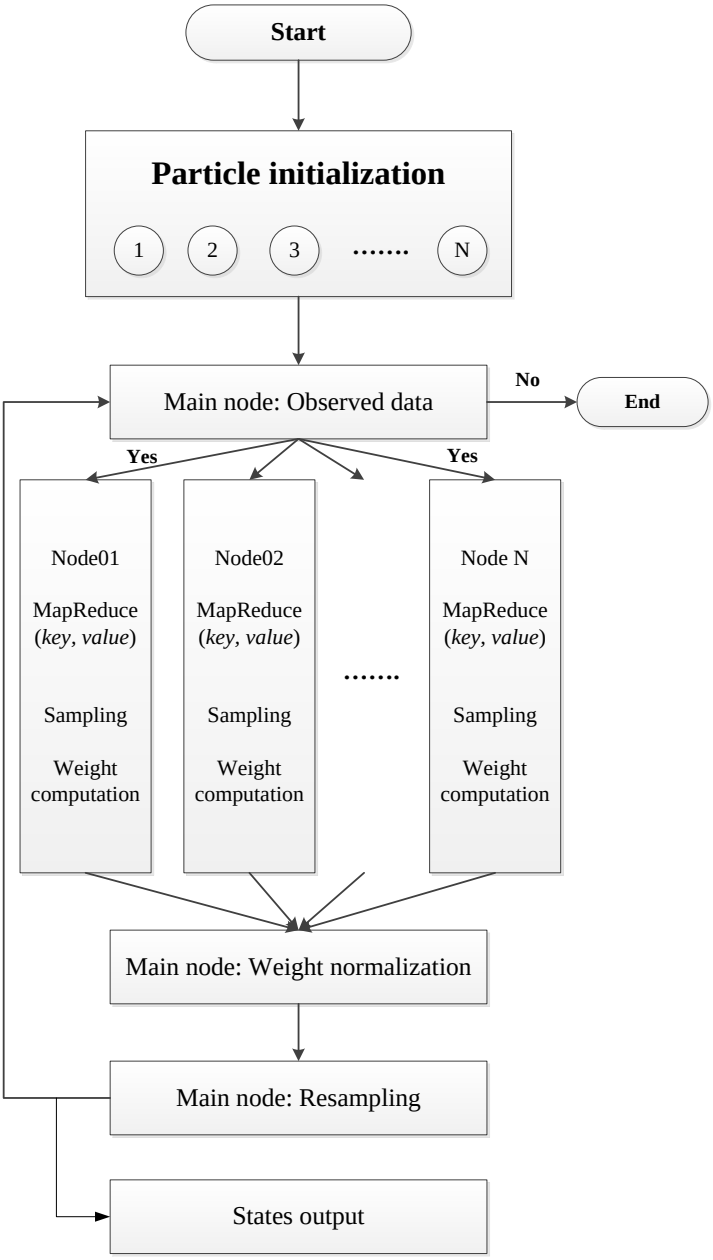


Figure 6.1 MapReducePF algorithms of case study

6.4 Experiments and analysis

We continue used the identical-twin experiment to evaluate the data assimilation system of DEVS-FIRE. In our experiments, we intended to show a filtered fire gave more accurate simulation results by assimilating observation data from the real fire even it still used the "error" data. In this experiment, we chose to use the "error" wind conditions as the "error" data. The real wind speed and direction are 8 (m/s) and 180 degrees (from south to north) with random variances added every 10 minutes. The variances for the wind speeds are in the range of –2 to 2(m/s) and the wind direction to be exactly the same as the real wind direction (Table 6.2). For the sensor deployment, we employed a regular sensor deployment schema and design our experiment as follow, use a uniform fuel model (fuel model 7) and zero slope and aspect. The simulations are run for 5 steps (hours), the weather changed every 30 minutes.

Table 6.2 Experiment sets of wind factor

	"Error" data		Real data	
	Speed	Direction	Speed	Direction
case	6 ±2	No error	8 ±2	180 ±20

Secondly, all experiments run under the super computer named Cheetah, which has 14 nodes, 160 computing cores, 32 CPUs and 264 GB system memory. 7 nodes equipped with NVIDIA GTX 285, 485, or Tesla c2075 Graphic processing units for CUDA development 6TB disk storage [136]. The software package which we use is Apache Hadoop Cloud Computing Software. Hadoop version 1.0.1 and Java 1.6.0.12 are used as MapReduce system. Finally, In order to test the performance, we use four nodes for MapReducePF and one of those four nodes for CentralizedPF, we use the particle number is: 50 particles, 100 particles and 200 particles.

The Figure 6.2 display the filtered fires (displayed in yellow) after 5 steps of simulation, compared with the real fire (displayed in red), and the simulated fires (displayed in blue). The particle number used for this experiment is 100 particles.

Figure 6.3 display the result performance for the single step (1 hour) DEVS-FIRE spread simulation based on SMC method use different particle numbers: 50 particles, 100 particles, 200 particles, 500 particles and 800 particles. We can see the simulation time almost same when we just use 50 particles (CentralizedPF: 120 seconds and MapReducePF 122 seconds), but with increase the number of particle, the simulation time of MapReducePF getting better and better: the simulation time for MapReducePF are less than half of the simulation time for CentralizedPF when using 200 particles (CentralizedPF: 1002 seconds and MapReducePF 436 seconds). And in our single node, the machine will appear “out of memory” problem when we run more than 250 particles. But for the MapReducePF, we can continue running the particle number to 800 particles (even more), and the simulation time for MapReducePF using 800 particle are less than double of the simulation time for CentralizedPF using 200 particles (CentralizedPF: 1002 seconds and MapReducePF 1960seconds).

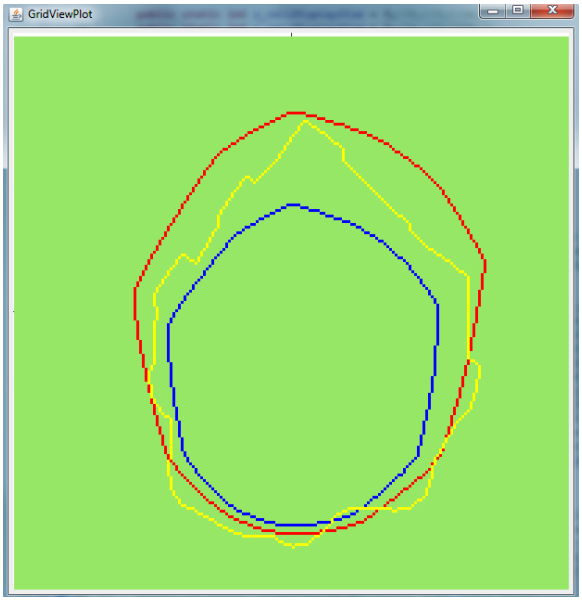


Figure 6.2 Comparisons of real fire, simulated fires, and filtered fires

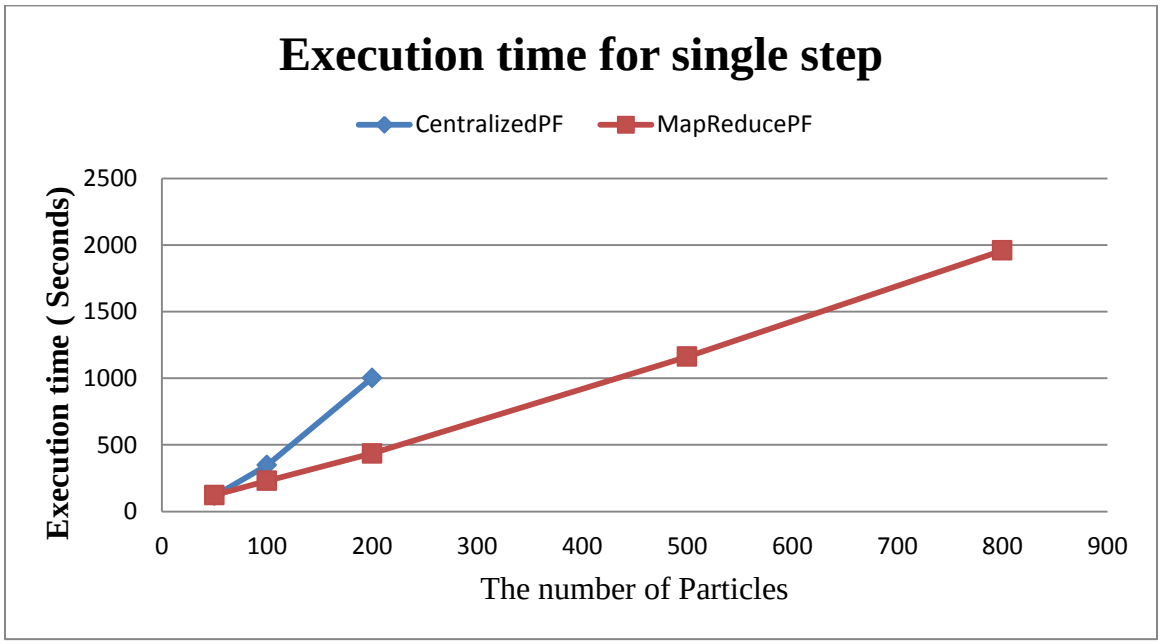


Figure 6.3 Execution time for single step

Figure 6.4 display the result performance for the five steps (5 hour) DEVS-FIRE spread simulation based on SMC method use different particle numbers: 50 particles, 100 particles, 200 particles, 500 particles and 800 particles. Start from using 50 particles, the simulation time of MapReducePF are less than half of the simulation time for CentralizedPF using 50 particles

(CentralizedPF: 942 seconds and MapReducePF 2065 seconds). And the simulation time for CentralizedPF are great more than four times compare to the simulation time for MapReducePF when using 200 particles (CentralizedPF: 9162 seconds and MapReducePF 2631 seconds). The machine will still appear "out of memory" problem when we run more than 250 particles on the single node. In MapReducePF, the simulation time for MapReducePF when using 800 particle is just a litter bit longer than the simulation time for CentralizedPF using 200 particles (CentralizedPF: 9162 seconds and MapReducePF 9984 seconds).

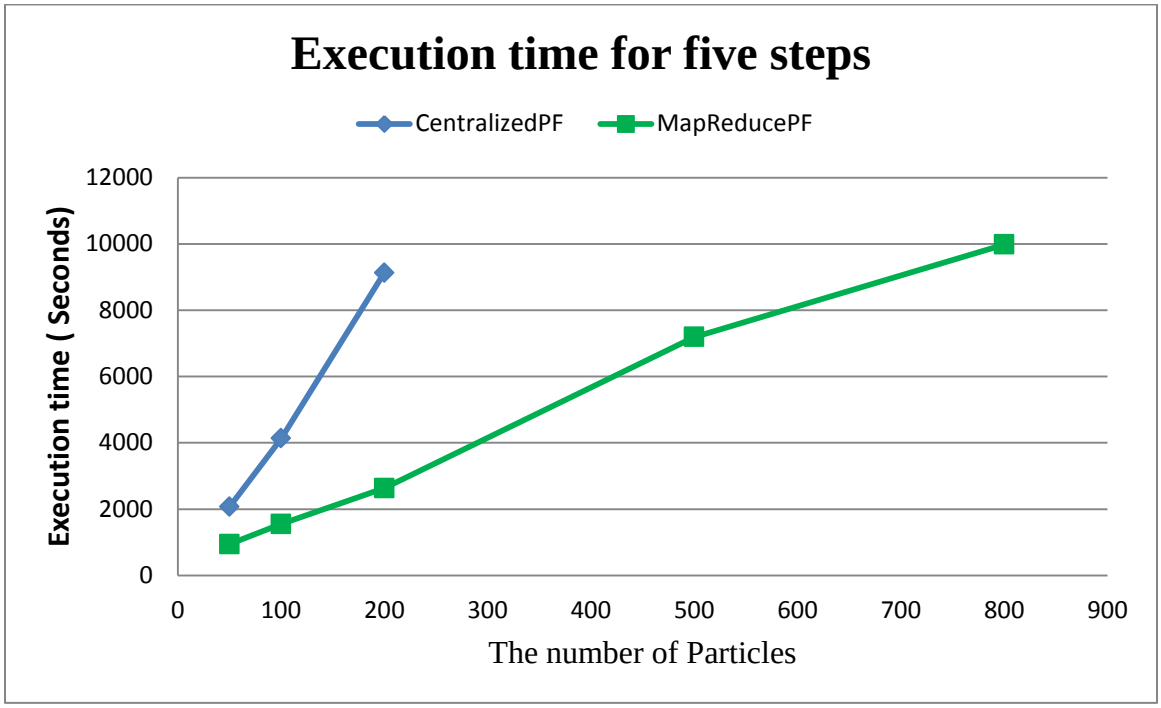


Figure 6.4 Execution time for five steps

The experiment results showed the MapReducePF significantly increases the performance for data assimilation using large number particles. Although in our current experiment we used up to 200 particles, we expect the performance will be further improved for larger number of particles. This work builds a foundation where future work can be carried out. Future work includes develop new ways that utilize the MapReduce programming model for further improving the

data assimilation performance, and to build a framework for parallel particle filtering based on MapReduce for general applications other than the wildfire application considered.

7 CONCLUSION AND DISCUSSIONS

7.1 Conclusions

In this work, we developed distributed PFs for larger-scale spatial temporal systems in order to improve the performance of data assimilation. We reviewed several distributed particle filtering algorithms that have already developed in literature, and discussed the merits and demerits of these algorithms based on the different steps of the PF algorithm. Although these algorithms have already attained a good performance, they mainly differ in how the resampling is carried out, and less research has been conducted to investigate how to route particles among PUs after resampling. Efficient particle routing is highly critical for reducing the communication costs in distributed PFs, due to the following reasons: 1) Particle routing is necessary because the numbers of particles on different PUs are unbalanced after resampling. 2) As the number of PUs increases, the communication overhead rises. The unbalanced particles on PUs are caused by the fact that particles have different importance weights. 3) The size of each particle is very large due to the high dimensional state it represents in high dimensional spatial temporal simulations. Therefore, we developed two efficient particles routing policies named *minimal transfer particle routing policy* and the *maximal balance particle routing policy*, and showed their impacts on distributed PFs with centralized resampling. We evaluated the proposed methods based on data assimilation of a large-scale wildfire spread simulation. Experimental results showed that the minimal transfer particle routing policy is the best choice for centralized resampling, since it can achieve the same data assimilation results with the lowest number of state transfers as compared to the random routing policy and the maximal balance routing policy.

In the distributed resampling schema (more specifically, the distributed RNA), communications are constrained between neighboring PUs. This local communication schema

supports a large degree of parallelism due to elimination of the centralized resampling step. However, it also results in slow propagation of high weight particles, and thus reduces the convergence rate of the particles. To address this issue, we develop a hybrid particle routing approach that uses both local and global particle routing in distributed resampling with non-proportional allocation (RNA). In this approach, we mainly use local routing to ensure scalability and low communication costs, and occasionally invoke global routing to support faster propagation of "good" particles. We showed how the minimal transfer particle routing policy and the maximal balance routing policy can be used in the global routing step, and their impacts on the performance and accuracy of particle filtering. We also evaluated and compared the different particle routing methods based on the application of data assimilation for large-scale wildfire spread simulations.

For the hybrid approach of particle routing in distributed resampling with RNA, maximal balance particle routing policy is preferred in the global routing step because it can attain the best data assimilation results with a slightly higher number of state transfers compared to the minimal transfer routing policy. Moreover, our work used cloud MapReduce and Hadoop to provide another solution to improve the performance of data assimilation for larger-scale spatial temporal systems based on PFs. Our work built the foundation algorithm by using MapReduce and Hadoop to improve the performance of data assimilation for larger-scale spatial temporal systems. The experiment results showed that the MapReducePF and Hadoop can significantly increase performance for data assimilation by using 200 particles, 500 particles and 800 particles.

7.2 Discussions and future work

We developed two efficient particle routing policies in particle routing according to the PF algorithm, and showed their impacts on distributed PFs for larger-scale spatial temporal

systems. We implemented both the minimal transfer particle routing policy and maximal balance particle routing policy in an intuitive manner without formally proving that the algorithms will always guarantee the best results. A formal analysis of these algorithms wills an imminent task in our future work. The experimental results shown in this work are based on a specific application of data assimilation of wildfire spread simulations. These results provide a guideline for choosing different particle routing policies for other applications. In general, the performances of different particle routing policies are dependent on the distribution of particles' weights among PUs. If all PUs has a balanced distribution of particles' weights, the different policies will not lead to results that are much different, since there is little need to transfer particles between PUs. On the other hand, if all the high weight particles are concentrated on a single PU, the different policies will not lead to very different results either, since they all result in transferring particles from the dominant PU to others. Systematically and formally analyzing in what conditions the different routing policies perform the best is another task that we plan to carry out in future research.

Moreover, the creation of a cloud MapReduce and Hadoop builds a foundation where future investigation can be carried out. Future tasks include developing new ways to utilize the MapReduce programming model in order to further improve data assimilation performance, and building a framework for parallel particle filtering based on MapReduce for general applications other than the wildfire application considered.

REFERENCES

1. *Grossi, P. 2007. The 2007 U.S. wildfire season lessons from southern California. Last accessed in Aug. 2009 fromhttp://www.rms.com/Publications/2007_US_Wildfire_Season.pdf.*
2. *Source(s): Insurance Institute for Business and Home Safety (IBHS). Date: 2nd, May, 2012.*
3. *http://www.bloomberg.com/news/2014-07-03/climate-driven-wildfires-consume-forest-service-budget.html.*
4. *Jun Bi, Can Chang and Yang Fan. Particle Filter for Estimating Freeway Traffic State in Beijing. Mathematical Problems in Engineering, Volume 2013 (2013), Article ID 382042.*
5. *http://en.wikipedia.org/wiki/Traffic_congestion.*
6. *Dongling Ma ; Ning Ding ; Jingwei Wang ; Jian Cui, Research on flood submergence analysis system based on ArcEngine component library. Agro-Geoinformatics (Agro-Geoinformatics), 2012 First International Conference on Digital Object Identifier: 10.1109/Agro-Geoinformatics.2012.6311730.*
7. *FINNEY, M.A. 1998. FARSITE: fire area simulator–model development and evaluation. United States Department of Agriculture Forest Service Rocky Mountain Research Station Research Paper, RMRS-RP-4 Revised March 1998, revised February 2004.*
8. *ANDREWS, P.L., BEVINS, C.D., AND SELI, R.C. 2005. BehavePlus fire modeling system, version 3.0: User's Guide Gen. Tech. Rep. RMRS-GTR-106WWW Revised. Ogden, UT: Department of Agriculture, Forest Service, Rocky Mountain Research Station, 132.*
9. *X. Hu, Y. Sun, L. Ntaimo, DEVS-FIRE: Design and Application of Formal Discrete Event Wildfire Spread and Suppression Models, SIMULATION, Vol. 88, No. 3, pp. 259-279, 2012.*

10. *MORAIS, M. 2001. Comparing spatially explicit models of fire spread through Chaparral fuels: A new model based upon the Rothermel fire spread equation. MA Thesis, University of California, Santa Barbara.*
11. *Furtlehner, C.; Yufei Han; Lasgouttes, J.-M. ; Martin, V.; Marchal, F. Spatial and temporal analysis of traffic states on large scale networks. ; Moutarde, F.Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference.*
12. *H. S. Wheater, R. E. Chandler, C. J. Onof, V. S. Isham, E. Bellone, C. Yang, D. Lekkas, G. Lourmas, M.-L. Segond. Spatial-temporal rainfall modelling for flood risk estimation. Stochastic Environmental Research and Risk Assessment, December 2005, Volume 19, Issue 6, pp 403-416.*
13. *BOUTTIER, F. AND COURTIER, P. 1999. Data assimilation concepts and methods. Training course notes of ECMWF.*
14. *DALEY, R. 1991. Atmospheric data analysis. Cambridge University Press.*
15. *KALNAY, E. 2003. Atmospheric Modeling, Data Assimilation and Predictability. Cambridge University Press.*
16. *Doucet, A., N. D. Freitas, N. Gordon (eds.).2001.Sequential Monte Carlo methods in practice. New York: Springe-Verlag.*
17. *Jonathan Briggs, Particle Filters for High Dimensional Spatial Systems, Masters Theses, The University of Auckland, 2011.*
18. *Lyudmila Mihaylova, Donka Angelova, Anna Zvikhachevskaya. Sequential Monte Carlo Methods for Localization in Wireless Networks.*
19. *Frank Dellaert, A Sample of Monte Carlo Methods in Robotics and Vision, Georgia Institute of Technology.*

20. *Sotá k, M., Sopata, M., & Kmec. Navigation Systems Using Monte Carlo Method, F. 6th International ESA Conference on Guidance, Navigation and Control Systems, held 17-20 October 2005 in Loutraki, Greece.*
21. *Zigang Yang; Xiaodong Wang, Joint mobility tracking and hard handoff in cellular networks via sequential Monte Carlo filtering, INFOCOM 2002. Twenty-First Annual Joint Conferences of the IEEE Computer and Communications Societies.*
22. *Fan Lin-bo; Kang Li; Wu Ying-cheng; Zhao Ming. Study of Multi-target Tracking and Data Association Based on Sequential Monte Carlo Algorithm. Future BioMedical Information Engineering, 2008. FBIE '08.*
23. *Van Leeuwen, P.J. Particle filtering in geophysical systems. Monthly Weather Review, 137:4089-4114, 2009M. J. Coates, "Distributed particle filtering for sensor networks," in Proc. of Int. Symp. Information Processing in Sensor Networks (IPSN), Berkeley, CA, April 2004.*
24. *C. Snyder, T. Bengtsson, P. Bickel, J. Anderson. Obstacles to High-Dimensional Particle Filtering. American Meteorological Society. May 2008.*
25. *Muhammad Shakir Hussain, Real-Coded Genetic Algorithm Particle Filters for High-Dimensional State Spaces, dissertation, University College London.*
26. *Nakano, S., G. Ueno, and T. Higuchi, 2007: Merging particle filter for sequential data assimilation, Nonlin. Processes Geophys., 14, 395-408.*
27. *M. J. Coates, "Distributed particle filtering for sensor networks," in Proc. of Int. Symp. Information Processing in Sensor Networks (IPSN), Berkeley, CA, April 2004.*

28. *Y. Sheng, X. Hu, and P. Ramanathan, "Distributed particle filter with GMM approximation for multiple targets localization and tracking in wireless sensor networks," in Proc. of the 4th Int. Symposium on Information Processing in Sensor Networks, Apr. 2005.*
29. *L. Zuo, K. Mehrotra, P. Varshney, and C. Mohan, "Bandwidth-efficient target tracking in distributed sensor networks using particle filters," in Proc. of 14th European Signal Processing Conference EURASIP2006, Florence, Italy, Sept. 2006.*
30. *A. S. Bashi, V. P. Jilkov, X. R. Li, and H. Chen, "Distributed implementations of particle filters," in Proc. 2003 International Conf. Information Fusion, Cairns, Australia, July 2003, pp. 1164–1171.*
31. *J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: Proceedings of Operating Systems Design and Implementation (OSDI), San Francisco, CA, 2004, pp. 137–150.*
32. *J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Communications of the ACM 51 (2008) 107–113.*
33. *M. Boli ć, Architectures for Efficient Implementation of Particle Filters, Ph.D. thesis, State University of New York at Stony Brook, 2004.*
34. *http://en.wikipedia.org/wiki/Data_assimilation.*
35. *Xiang-Yu Huang and Henrik Vedel. AN INTRODUCTION TO DATA ASSIMILATION. Danish Meteorological Institute, Lyngbyvej 100, DK-2100 K øbenhavn Ø, Denmark.*
36. *Rui Li ; Cunjun Li ; Feng Liu ; Xiaodong Yang ; Jihua Wang. Methods and algorithms of data assimilation and its application in agriculture. World Automation Congress (WAC), 2010.*

37. *G Evensen, "Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast statistics", Geophysics. Res. 99(C5):10143–10162.*
38. *B.Ristic, S.Aruampalam, N.Gordon, Beyond the Kalman Filter [M], Artech House,Boston/London,2004.*
39. *A. Smith, A. Doucet, N.D. Freitas, N. Gordon, Sequential Monte Carlo methods in practice [M], Springer,New York, 2005.*
40. *Q.Y.Duan, S.Sorooshian, V.K.Gupta, "Effective and efficient global optimization for conceptual rainfall-runoff models ". Water Resource Research. 1992. 28(4):1015–1031.*
41. *Q.Y. Duan, V.K. Gupta, Sorooshian S, "Shuffled complex evolution approach for effective and efficient global minimization", J. Optim. Theor. Appl, 1993.76(3):501–521.*
42. *L.Ingber ,"Very fast simulated re-annealing", Math. Comput. Model, 1989, 12(8): 967–973.*
43. *X. Li, T. Koike, Pathmathevan M., "A very fast simulated re-annealing (VFSA) approach for land data assimilation". Computer and Geoscience, 2004, 30(3):239–248.*
44. *R. Storn, K. Price. "Differential Evolution – a simple and efficient heuristic for global optimization over continuous space". J. Global Optim. 1997, 11:341–359.*
45. *D.E.Goldberg, Genetic algorithms in search, optimization and machine learning [M], Addison Wesley, Reading, MA, 1989.*
46. *Bouttier, F., and Courtier, P. 1999. Data assimilation concepts and methods. Training course notes of ECMWF.*
47. *F. Gu, Dissertations: Dynamic Data Driven Application System for Wildfire Spread Simulation, Dept. Computer Science, Georgia State University, December 2010.*

48. *Zou X, Vandenberghe F, Wang B, Gorbunov ME, Kuo Y-H, Sokolovskiy S, Chang JC, Sela JG, Anthes RA. 1999. A ray-tracing operator and its adjoint for the use of GPS/MET refraction angle measurements. Journal of Geophysical Research. 104(D18):22301-22318.*

49. *Kurihara, Y., M. A. Bender, R. E. Tuleya, and R. J. Ross, 1990: Prediction experiments of Hurricane Gloria (1985) using a multiply nested movable mesh model. Mon. Wea. Rev., 118, 2185–2198.*

50. *Xulin Ma; Xiaolei Zou ; Gang Li. Diagnosis of surface data assimilation with GRAPES 3D-VAR, Advanced Computational Intelligence (ICACI), 2012 IEEE Fifth International Conference on. Digital Object Identifier.*

51. *Qifeng Lu ; Xuebao Wu ; Peng Zhang ; Songyan Gu ; Chaohua Dong ; Jiandong Gong ; Xueshun Shen ; Chenli Qi ; Gang Ma; Assimilating FY-3A VASS data into Chinese 3Dvar assimilation system (Grapes 3Dvar). Geoscience and Remote Sensing Symposium, 2009 IEEE International, IGARSS 2009.*

52. *BEI, N., DE FOY, B., LEI, W., ZAVALA, M., AND MOLINA, L.T. 2008. Using 3DVAR data assimilation system to improve ozone simulations in the Mexico City Basin. Atmos. Chem. Phys. 8, 7353-7366.*

53. *Eric Be ̂langer, Alain Vincent, Data assimilation (4D-VAR) to forecast flood in shallow-waters with sediment erosion. Journal of Hydrology 300 (2005) 114–125.*

54. *Jim Kao, Dawn Flicker, Rudy Henninger , Sarah Frey, Michael Ghil, Kayo Ide, Data assimilation with an extended Kalman filter for impact-produced shock-wave dynamics, Journal of Computational Physics 196 (2004) 705–723.*

55. *P. L. Houtekamer AND Herschel L. Mitchell, A Sequential Ensemble Kalman Filter for Atmospheric Data Assimilation MONTHLY WEATHER REVIEW JANUARY 2001.*

56. *Rolf H. Reichle, Dennis B. Mclaughlin, and Dara Entekhabi, R e i c h l e e t a l. Hydrologic Data Assimilation with the Ensemble Kalman Filter, JANUARY 2002.*
57. *Haidong Xue ; Xiaolin Hu ,An effective proposal distribution for sequential Monte Carlo methods-based wildfire data assimilation, Simulation Conference (WSC), 2013 Winter.*
58. *BRADLEY, J.M. 2007. Particle filter based mosaicking for forest fire tracking. Master thesis, Brigham Young University.*
59. *Flood forecasting and uncertainty assessment with sequential data assimilation using a distributed hydrologic model, Seong Jin Noh ; Tachikawa, Y. ; Kyoungjun Kim ; Shiiba, M. ; Yeonsu Kim Information Fusion (FUSION), 2013 16th International Conference.*
60. *M. Wang, X. Hu, Data Assimilation in Agent Based Simulation of Smart Environment, Proc. 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS), 2013.*
61. *S. Rai, X. Hu, Behavior Pattern Detection for Data Assimilation in Agent-Based Simulation of Smart Environments, Proc. 2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-13), 2013.*
62. *Arnaud Doucet, Nando de Freitas and Neil Gordon.An Introduction to Sequential Monte Carlo Methods.*
63. *N. J. Gordon. Beyond the Kalman Filter:Particle filters for tracking applications. Tracking and Sensor Fusion Group Intelligence, Surveillance and Reconnaissance Division Defence Science and Technology Organisation.*
64. *Hong Sangjin, Petar M. Djurić, Bolić Miodrag. Resampling Algorithms for Particle Filters: A Computational Complexity Perspective. EURASIP Journal on Advances in Signal Processing 01/2004;*

65. *D. B. Rubin, J. M. Bernardo and M. H. De Groot and D. V. Lindley and A. F. M. Smith, "Bayesian Statistics 3," Oxford: University Press, pp 395-402, 1988.*
66. *A. Kong and J.S Liu and W.H. Wong, "Sequential Imputations and Bayesian Missing Data Problems," Journal of American Statistical Association, Vol. 89, no. 425, pp. 278- 288, 1994.*
67. *J. S. Liu and R. Chen, "Blind deconvolution via sequential imputations," Journal of the American Statistical Association, vol. 90, pp. 567- 576, 1995.*
68. *J. S. Liu, R. Chen, and T. Logvinenko, "A Theoretical Framework for Sequential Importance Sampling and Resampling," in Sequential Monte Carlo Methods in Practice, A. Doucet, N. de Freitas, and N. Gordon, Eds., New York: Springer Verlag, 2001.*
69. *B. Ristic, S. Arulampalam, N. Gordon, Beyond the Kalman Filter: Particle Filters for Tracking Applications, Artech House, 2004.*
70. *Fan Lin-bo; Kang Li; Wu Ying-cheng; Zhao Ming. Study of Multi-target Tracking and Data Association Based on Sequential Monte Carlo Algorithm. Future BioMedical Information Engineering, 2008. FBIE '08.*
71. *Lyudmila Mihaylova, Donka Angelova, Anna Zvikhachevskaya. Sequential Monte Carlo Methods for Localization in Wireless Networks,*
72. *Aline Baggio and Koen Langendoen. Monte-Carlo Localization for Mobile Wireless Sensor Networks. Ad Hoc Networks. Volume 6, Issue 5, July 2008, Pages 718–733.*
73. *Azzabou, N., Paragios, N., and Guichard, F. Application of particle filtering to image enhancement. CERTIS 05-18, 2005.*

74. *Zhang, J., Chen, R., Tang, C., and Liang, J. Origin of scaling behavior of protein packing density: A sequential Monte Carlo study of compact long chain polymers. Journal of Chemical Physics, 118(13): 5102-610.*

75. *http://www.dddas.org/*

76. *Frederica Darema, Fellow, IEEE. Grid Computing and Beyond: The Context of Dynamic Data Driven Applications Systems. Invited paper.*

77. *Darema, F. 2000. Dynamic data driven application systems (Symbiotic measurement & simulation systems): A new paradigm for application simulations and a new paradigm for measurement systems. NSF sponsored workshop.*

78. *Farhat, C., Michopoulos, J., Chang, F.K., Guibas, L.J., and Lew, A.J. 2006. Towards a dynamic data driven system for structural and material health monitoring. International Conference on Computational Science (3) 2006:456-464.*

79. *Fujimoto, R.M., Guensler, R., Hunter, M., Kim, H.K., Lee, J., Leonard, J., Palekar, M., Schwan, K., and Seshasayee, B. 2006. Dynamic data driven application simulation of surface transportation systems. International Conference on Computational Science (3) 2006:425-432.*

80. *Liqian Peng Doug Lipinski Kamran Mohseni, Dynamic Data Driven Application System for Plume Estimation Using UAVs, J Intell Robot Syst (2014) 74:421–436.*

81. *Allen, G. 2007. Building a dynamic data driven application system for hurricane forecasting. International Conference on Computational Science (1) 2007:1034-1041.*

82. *Oden, J.T., Diller, K.R., Bajaj, C.L., Browne, J.C., Hazle, J., Babuska, I., Bass, J., Demkowicz, L.F., Feng, Y., Fuentes, D., Prudhomme, S., Rylander, M.N., Stafford, R.J., and*

Zhang. Y. 2006. Development of a computational paradigm for laser treatment of cancer. International Conference on Computational Science (3) 2006:530-537.

83. *X. Yan, F. Gu, X. Hu, S. Guo, A Dynamic Data Driven Application System for Wildfire Spread Simulation, Proc. 2009 Winter Simulation Conference (WSC09), 2009.*
84. *Particle Filters for High Dimensional Spatial Systems, Jonathan Briggs, Masters Theses, The University of Auckland, 2011*
85. *Bolic, Miodrag ; Djuric, P.M. ; Sangjin Hong. Resampling algorithms and architectures for distributed particle filters, Signal Processing, IEEE Transactions on Volume: 53, Issue: 7, 2005, Page(s): 2442 – 2450.*
86. *Bolic, M.; Djuric, P.M.; Sangjin Hong. New resampling algorithms for particle filters. Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference.*
87. *Joaqu´ın M´ıguez. Analysis of Parallelizable Resampling Algorithms for Particle Filtering. Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*
88. *M. Boli´c, P. M. Djuri´c, and S. Hong, "Resampling Algorithms for Particle Filters: A Computational Complexity Perspective," submitted to the EURASIP Journal of Applied Signal Processing, 2003.*
89. *J. L. Dekeyser, C. Fonlupt, and P. Marquet, "Analysis of synchronous dynamic load balancing algorithms," Advances in Parallel Computing, vol 11, pp. 455-462, 1995.*
90. *Bashi, A. S., et al Distributed implementations of particle filters Proceedings of the Sixth International Conference on Information Fusion, vol. 2, Cairns, Australia, July 2003,1164—1171.*

91. *M. Boli´c, P. M. Djuri´c, and S. Hong,"Resampling Algorithms for Particle Filters: A Computational Complexity Perspective," submitted to the EURASIP Journal of Applied Signal Processing, 2003.*
92. *B.P. Zeigler, H. Praehofer, and T.G. Kim, Theory of Modeling and Simulation (2nd edition). Academic Press, UT: Salt Lake City, 2000.*
93. *Rothermel, Richard C. 1972. A mathematical model for predicting fire spread in wildland fuels. Research Paper INT-115. Ogden, UT: U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station. 40 p.*
94. *X. Hu, and L. Ntaimo, Integrated Simulation and Optimization for Wildfire Containment, The ACM Transactions on Modeling and Computer Simulation (TOMACS), Vol. 19, No. 4, 2009*
95. *Finney, M. A. 1998. "FARSITE, Fire Area Simulator--Model Development and Evaluation." In Research paper RMRS, RP-4. Ogden, UT (324 25th St., Ogden 84401): U.S. Dept. of Agriculture, Forest Service, Rocky Mountain Research Station.*
96. *H. Xue, F. Gu, X. Hu, Data Assimilation Using Sequential Monte Carlo Methods in Wildfire Spread Simulation, The ACM Transactions on Modeling and Computer Simulation (TOMACS), Vol. 22, No. 4, Article No. 23, 2012.*
97. *CRISAN, D. 2001. Particle filters—A theoretical perspective. Sequential Monte Carlo Methods in Practice (eds A. Doucet, J. F. G. de Freitas and N. J. Gordon). New York: Springer-Verlag.*
98. *GORDON, N.J., SALMOND, D.J., AND SMITH, A.F.M. 1993. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In IEE Proceedings on Radar and Signal Processing 140, 107-113.*

99. *W. Wagner, A. Ulrich, T. Melzer, C. Briese, and K. Kraus, "From Single-Pulese to Full-Waveform Airborne Laser Scanners: Potential and Practical Challenges," International Archivies of the Photogrammetry, Remoe Sensing, and Geoinformation Sciences, pp. 414-419, 2004.*

100. *M. Mutlu, S.C. Popescu, C. Stripling, and T. Spencer, "Assessing Surface Fuel Models Using LiDAR and Multispectral Data Fusion," Remote Sensing of Enviroment, vol. 112, no. 1, pp. 274-285, 2008.*

101. *S. Derin Babacan. Parameter Estimation in TV Image Restoration Using Variational Distribution Approximation. IEEE VOL. 17, NO. 3 MARCH 2008.*

102. *V. Dua. A Decomposition Approach for Parameter Estimation of System of Ordinary Differential Equations. – ESCAPE20.*

103. *Y.Lin and M.A.Stadtherr. Deterministic Global Optimization for Parameter Estimation of Dynamic Systems.*

104. *G.Heinrich, Parameter estimation for text analysis.*

105. *A.John (1997). "R. A. Fisher and the making of maximum likelihood 1912–1922". Statistical Science 12 (3): 162–176.*

106. *Natimo, L., X. Hu, and Y. Sun. 2008. "DEVS-FIRE:Towards an Integrated Simulation Environment for Surface Wildfire Spread and Containment," SIMULATION:, Vol. 84, Issue 4,April 2008, pp 137-155.*

107. *Byram, G. M. 1959. Combustion of forest fuels. In: Davis, K. P., ed. Forest Fire: Control and Use. New York: McGraw Hill.*

108. *Pyne, S.J.; Andrews, P.L.; Laven, R.D. 1996. Introduction to wildland fire. 2nd ed. New York, NY: John Wiley & Sons. 808*

109. Miller, M. 1994. Fuels. Fire Effects Guide. National Wildfire Coordinating Group, NFES.

110. http://www.forestencyclopedia.net/p/p509

111. http://www.fire.org/downloads/farsite/WebHelp/technicalreferences/tech_dead_fuel_moisture.htm

112. Ntaimo, X. Hu, and Y. Sun. "DEVS-FIRE: Towards an Integrated Simulation Environment for Surface Wildfire Spread and Containment," Simulation, vol. 84, no. 4, pp. 137-155, 2008.

113. F. Bai, F. Gu, X. Hu, Particle Routing in Distributed Particle Filters for Large-Scale Spatial Temporal Systems, IEEE Transactions on Parallel and Distributed Systems (TPDS), under review, 2014.

114. P.J. van Leeuwen, "A Variance-minimizing Filter for Large-Scale Applications," Monthly Weather Review, vol. 131, pp. 2071-2084, 2003.

115. Y. Zhou, D. McLaughlin, and D. Entekhabi, "Accessing the Performance of the Ensemble Kalman Filter for Land Surface Data Assimilation," Monthly Weather Review, vol. 134, 2128-2142, 2006.

116. L. Mihaylova and A. Carmi, "Particle Algorithm for Filtering in High Dimensional State Spaces: A Case Study Example in Group Object Tracking," 2011 IEEE International Conference on Acoustic, Speech and Signal Processing, pp. 5932-5935, 2011.

117. N. Lingala, N.S. Namachchivaya, N. Perkowski, and H.C. Yeong, "Particle Filtering in High-diemensional Chaotic Sys-tems," Chaos: An interdisciplinary Journal of Nonlinear Science, vol. 22, no. 4, pp. 047509-1-047509-18, 2012.

118. C. Snyder, T. Bengtsson, P. Bickel, and J. Anderson, "Obstacles to High-Dimensional Particle Filtering,", Monthly Weather Review, vol. 136, pp. 4629-4640, 2008.

119. B.W. Silverman, Density Estimation for Statistics and Data Analysis. London: Chapman and Hall, pp. 175, 1986.

120. A. Hegiy, L. Mihaylova, R. Boel, and L. Lendek, "Parallellized Particle Filtering for Freeway Traffic State Tracking," 9th European Control Conference, 2007.

121. M. Rosencrantz, G. Gordon, and S. Thrun, "Decentralized Sensor Fusion with Distributed Particle Filters," Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence, pp. 493-500, 2002.

122. L.-L. Ong, T. Bailey, H. Durrant-Whyte, and B. Upcroft, "Decentralised Particle Filtering for Multiple Target Tracking in Wireless Sensor Networks," 2008 11th International Conference on Information Fusion, pp. 1-8, 2008.

123. M. Coates, "Distributed Particle Filters for Sensor Networks," Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks, pp. 99-107, 2004.

124. Junbo Zhang, Tianrui Li, Yi Pan, Parallel rough set based knowledge acquisition using MapReduce from big data. August 2012. Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications ACM.

125. Hadoop: Open source implementation of MapReduce, <http://hadoop.apache.org/mapreduce/ >.

126. C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In Proceedings of the 2007 IEEE 13th

International Symposium on High Performance Computer Architecture, HPCA'07, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society.

127. B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang. Mars: a mapreduce framework on graphics processors. In Proceedings of the 17th international conference on Parallel architectures and compilation techniques, PACT'08, pages 260–269, New York, NY, USA, 2008. ACM.

128. J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative mapreduce. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC'10, pages 810–818, New York, NY, USA, 2010. ACM.

129. Mahout: Scalable machine learning and data mining, < http : //mahout.apache.org/>

130. R. K. Menon, G. P. Bhat, and M. C. Schatz. Rapid parallel genome indexing with mapreduce. In Proceedings of the second international workshop on MapReduce and its applications, MapReduce'11, pages 51–58, New York, NY, USA, 2011. ACM.

131. S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In Proceedings of the 2010 international conference on Management of data, SIGMOD'10, pages 975–986, New York, NY, USA, 2010. ACM.

132. A. Ene, S. Im, and B. Moseley. Fast clustering using mapreduce. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD'11, pages 681–689, New York, NY, USA, 2011. ACM.

133. C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In Proceedings of the 2007 IEEE 13th

International Symposium on High Performance Computer Architecture, HPCA'07, pages 13–24, Washington, DC, USA, 2007. IEEE

134. Cloud MapReduce for Monte Carlo bootstrap applied to Metabolic Flux Analysis Tolga Dalman, Tim Dörnemann, Ernst Juhnke, Michael Weitzel , Wolfgang Wiechert , Katharina Nöha, Bernd Freisleben. Future Generation Computer Systems. Volume 29, Issue 2, February 2013, Pages 582–590

135. Monte Carlo simulation of photon migration in a cloud computing environment with MapReduce. Guillem Pratx and Lei Xing, Journal of Biomedical Optics 16(12), 125003 (December 2011).

136. http://cs.gsu.edu/?q=cheetah