Data Assimilation
in Discrete Event Simulations

Xu Xie

Data Assimilation in Discrete Event Simulations

Xu Xie
Delft University of Technology

Data Assimilation in Discrete Event Simulations

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
dinsdag 27 februari 2018 om 15:00 uur
door

Xu XIE
Master of Engineering in Control Science and Engineering
National University of Defense Technology, China
geboren te Hanzhong, Shaanxi, China

This dissertation has been approved by the promotor:
Prof. dr. ir. A. Verbraeck

Composition of the doctoral committee:

Rector Magnificus	Chairman
Prof. dr. ir. A. Verbraeck	Delft University of Technology, promotor
Independent members:	
Prof. dr. R.R. Negenborn	Delft University of Technology
Prof. dr. B.A. Van de Walle	Delft University of Technology
Prof. dr. ir. J.H. van Schuppen	Delft University of Technology
Prof. dr. H.L.M. Vangheluwe	University of Antwerp, Belgium
Dr. X. Hu	Georgia State University, United States
Other members:	
Prof. dr. ir. J.W.C. van Lint	Delft University of Technology

This research was funded by the China Scholarship Council (CSC) under Grant 201306110027.

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems. SIKS Dissertation Series No. 2018-09.

Cover design by Stevan Stojic `www.24-design.com`
Published and distributed by: Xu Xie
E-mail: x.xie@hotmail.com

ISBN 978-94-6186-893-0

To my family

Acknowledgment

The four-year studying experience in TU Delft is precious and unforgettable for me. I feel so lucky to meet so many nice people during this period, who have helped me in different ways. It is my great pleasure to take this opportunity to express my sincere gratitude for their time, help, support and friendship. If I forget to mention someone, please accept my apologies beforehand. My mind is forgetful but my heart is not.

First of all, I would like to express my deepest thanks to my promotor, Prof. Alexander Verbraeck. Alexander, thank you for offering me the opportunity to conduct this research under your supervision. Being an extremely busy professor, you still tried your best to guarantee that as my promotor, you provided me sufficient supervision on research in general, and as my daily supervisor, you timely gave constructive feedbacks on my every proposals, patiently answered every tiny and sometimes stupid questions, and quickly rescued me from any technical difficulties. Without your guidance, this dissertation would not have been possible. I hope you can work a bit less and have a bit more rest in the future. I would like to thank Prof. Kedi Huang, my advisor in China, for bringing me to the field of modeling and simulation, and for supporting me to pursue my PhD degree abroad. I would like to thank Prof. Hans van Lint for his huge contribution to the urban traffic case and also to my dissertation. Hans, I learned a lot from you through the joint work, and this precious experience means a lot to me, since it makes me more enthusiastic on research and also gives me more confidence on doing good research. I also thank you for accepting to be in my doctoral committee and for your good comments on my dissertation. I would like to thank Prof. Rudy Negenborn, Prof. Bartel Van de Walle, Prof. Jan H. van Schuppen, Prof. Hans Vangheluwe and Dr. Xiaolin Hu for being part of my doctoral committee and for their insightful and constructive comments that have greatly improved the quality of this dissertation. Special thanks to Dr. Feng Gu for many useful discussions and good suggestions on my research.

I would like to thank all my colleagues and friends in both the current section and the previous section for making the work environment lively and enjoyable. Special thanks to Yilin for many nice discussions, suggestions and encouragement both scientifically and personally. Thanks to Mingxin for his kind help since we knew each other, and especially for introducing me to Delft. Thanks to Shalini! I really enjoyed the interesting chats, delicious dinners and tasty snacks with you. Thanks to Yan for being such a wonderful roommate, and for many talks on various interesting topics. Thanks to Shalini and Yan again for accepting to be my paranymphs and for many other preparation work for my defense. Thanks to Lotte! It was a lot of fun to cycle along the river to Rotterdam together in summer. Thanks to Marlies, Monique, Diones and Everdine for their secretarial support and cheerfulness.

I would like to thank my friends in the Netherlands for making my life more enjoyable. Thanks to Laobing and Xingxing for their company. We spent almost every holiday together and had so many nice dinners and talks, and I really enjoyed a lot. Thanks to Changjie for being such a good friend with whom I can talk everything. I will miss you! Thanks to Xu, Shanshan, Yu, and Shilong for their friendship. I will always miss the time we spent together. Thanks to Binod, I really enjoyed the breaks, discussions and dinners with you and your family! Thanks to all the members of the ChInE peer group! It was very nice to have so many interesting discussions with all of you. Thanks to all my fellow PhD students and friends in and outside the faculty, and I wish you all success!

A special thanks to the China Scholarship Council (CSC) for providing me the financial support for this PhD study.

I would like to thank my parents-in-law. You are so nice and kind, and I especially thank you for educating such a wonderful wife for me. I am very grateful to my parents and sister for their unconditional love and support.

Finally, my biggest hug and kiss to my wife Zhangling. I really admire your independence, your optimistic attitude towards difficulties, and your kindness and tolerance to people. You are always a model for me. I wish you happy forever and success in your study. Our hearts are always together!

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, the motivation for this research is introduced by presenting the research gap. To fill the research gap, the research objective and corresponding research questions are proposed. Subsequently, the research philosophy and research instruments are selected based on the characteristics of this research. Finally, the organization of this thesis is presented.

1.1 Research motivation

Modeling & Simulation are a method of choice for studying and predicting dynamic behavior of complex systems. In the application of simulation, the real system is first abstracted by a conceptual model, which is then translated into a (computer-executable) simulation model (Banks, 1998). The simulation model needs to be verified (whether the computer implementation of the conceptual model is correct) and validated (whether the conceptual model can replace the real system for the purposes of experimentation) iteratively until the simulation model can represent the real system accurately, in the sense that the discrepancy between the simulation output and the relevant system measurements is within pre-specified acceptance criteria (Banks, 1998; Sargent, 2011). During this iterative model construction process, data is used *off-line*, e.g., for model calibration (adjusting model parameters) (Kesting and Treiber, 2008; Ciuffo et al., 2012), or for automated model generation (Huang, 2013). Once the simulation model is verified and validated, we can experiment with the simulation model to predict the behavior of the real system, while the data itself is not used in the simulation process. However, models inevitably contain errors, which arise from many sources in the modeling process, such as inadequate sampling of the real system when constructing the behavior database for the source system (Zeigler et al., 2000), or conceptual abstraction in the modeling process (Lahoz et al., 2010). Due to these inevitable errors, even elaborate complex models of systems cannot model the reality perfectly, and consequently, results produced by these imperfect simulation models will diverge from or fail to predict the real behavior of those systems (Darema, 2004, 2005).

With the advancement of measurement infrastructures, such as sensors, data storage technologies, and remote data access, the availability of data, whether real-time on-line or archival, has greatly increased (Darema, 2004, 2005). This allows for a new paradigm – *dynamic data driven simulations*, in which the simulation is continuously influenced by fresh data sampled from the real system (Hu, 2011). Figure 1.1 shows a general dynamic data driven simulation, which consists of 1) a simulation model, describing the dynamic behavior of the real system; 2) a data acquisition component, which essentially consists of sensors that collect data from the real system; and 3) a data assimilation component, which carries out state estimations based on information from both measurements and the simulation. The dynamic data driven simulation integrates computational (i.e. behavior predicted by the simulation model) and measurement (i.e. real-time data from the real system collected by sensors) aspects of a system. This can lead to more accurate simulation results (i.e. the estimated model state is closer to the real system state) than using a single source of information from either the simulation model or the measurements. Integrating data from the real system also helps the simulation to prune unrealistic states, since actual system data naturally contains correlation information which is easily lost in the modeling process (e.g., by falsely assuming that state variables are independently distributed). Such an integration is achieved by an analysis technique, *data assimilation*, which incorporates measured observations into a dynamical system model in order to produce a time sequence of estimated system states (Bouttier and Courtier, 1999; Nichols, 2003). By assimilating actual data, the simulation can dynamically update its current state to be closer to the real system state, which facilitates real-time applications of simulation models, such as real-time control and analysis, real-time decision making, and understanding the current state of the real system. Besides, if the model state is extended to include model parameters, on-line model parameter calibration can be achieved together with the state estimation (Bai et al., 2011). With more accurate model state and model parameters adjusted by assimilating real-time data, we can experiment (off-line) on the simulation model with the adjusted state and parameters, which will lead to more accurate results for follow-on simulations. In reverse, the information from data assimilation can also be fed back to the data acquisition component to guide the measurement process, for example, to optimize the sensor deployment (Heaney et al., 2007; Xue and Hu, 2012; Kouichi et al., 2016).

The data driven idea shown in Figure 1.1 has been applied in many (continuous systems) applications, such as weather forecasting (Huang et al., 2009), chemical data assimilation (Constantinescu et al., 2007), and ocean data assimilation (Carton and Giese, 2008). In these applications, the system state is a (high dimensional) vector consisting of continuous variables (taking values from $\mathbb{R}$), and its evolution is commonly modeled as a discrete time state space model. However, very little data assimilation research has been found for discrete event simulations, in which the state is defined as a combination of continuous/discrete variables, and its updates are triggered by events (happening irregularly on a continuous time axis). The discrete event approach is a natural modeling method for a big proportion of systems in reality, such as manufacturing processes, and urban traffic systems (modeling signal control logic, vehicle arrivals, etc.). With the application of new sensor technologies and communication solutions (e.g., smart sensors, Internet of Things (Atzori et al., 2010)), collecting data in these discrete event systems has become a simple

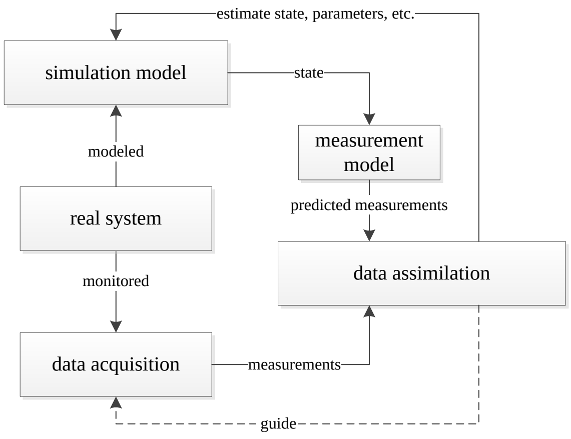


Figure 1.1: A general dynamic data driven simulation

exercise (Lee et al., 2013), and the data availability has greatly increased as well, such as data from machines and processes (Lee et al., 2013), or high-resolution event data in traffic systems (Liu et al., 2009; Wu and Liu, 2014).

The increased data availability for discrete event systems but the lack of related data assimilation research thus motivates this work on data assimilation in discrete event simulations. The outcome of this research will make incorporating real-time data into discrete event simulations possible, which will lead to more accurate state estimation in these systems for better analysis and control. Examples could be estimating job arrivals for optimizing the production line in manufacturing processes, estimating queues behind traffic lights on urban arterials for better traffic management (Marinică et al., 2013), etc.

1.2 Research objectives and questions

To fill the research gap, we conduct this research which aims to *develop a data assimilation framework for discrete event simulations*. To achieve this goal, the following two research questions (RQ) are addressed:

- *RQ* 1*: What existing or adapted data assimilation technique is suitable for discrete event simulations in order to obtain accurate data assimilation results*?

  In literature, many data assimilation techniques exist (they will be reviewed in chapter 2). Can any of them (probably with certain adaptations) be applied in discrete event simulations in order to obtain accurate data assimilation results? If not, why not? If yes, what are the challenges to apply these methods in discrete event simulations, and based on the (adapted) methods, how to design a systematic approach (i.e., a data assimilation framework) which can address these challenges? The accuracy of the data assimilation results is evaluated in terms of the error between the estimated value and the ground-truth value of the variable of interest.

- *RQ* 2*: How do the parameters of key components in the data assimilation framework affect the data assimilation results?*

  At least three key components are involved in a data assimilation framework, i.e. the (erroneous) model, the (noisy) data, and the data assimilation technique. The parameters of these components are those that describe their key features. For example, in the urban traffic case which is studied in chapter 4, noisy vehicle passage data containing miss-counts and over-counts is assimilated into a microscopic traffic simulation model (focusing on car-following behavior (Treiber and Kesting, 2013)) using particle filters (Arulampalam et al., 2002; Djurić et al., 2003; van Leeuwen, 2009). The parameters are the detection accuracy (quantifying miss-counts), the occurrence rate of over-counts, which characterize the noisy data, the difference between car-following models (i.e., the car-following model generating the ground-truth data and the car-following models used in the case study) which characterizes the model errors, the number of particles which characterizes the data assimilation technique, etc. This research question aims to analyze the sensitivity of the data assimilation results to the parameters of these components in the data assimilation framework.

1.3 Research philosophy and instruments

A research philosophy refers to the set of beliefs concerning the nature of the reality being investigated (Bryman, 2015). It is fundamental since it determines the selection of proper approaches and instruments in the research process (Saunders et al., 2009). In literature, positivism and interpretivism are the two main (and often seen as opposing) research philosophies (Weber, 2004). Positivism assumes that all knowledge about reality is objectively given and the observer is capable of studying it without influencing it, while interpretivism believes that all knowledge about reality is constructed and depends on human perception and experience. This research aims to develop a data assimilation framework for discrete event simulations, therefore *positivism* is the main philosophical view to follow during the research process.

Based on the positivist philosophical view, we choose a *deductive* research approach (Kothari, 2004), in which the general data assimilation technique is adapted to and applied in (specific) discrete event simulations. In this research, we use computer simulations to generate data to test and validate the proposed data assimilation framework, therefore our research is highly *quantitative*.

Given the chosen research philosophy and research approach, the research instruments are employed accordingly. First, a systematic knowledge on modeling and simulation (with a focus on discrete event simulations), and data assimilation needs to be obtained through *literature review*. Based on this knowledge, the research gap can be clarified, and adaptations to the chosen existing data assimilation techniques can be made to fill this gap by proposing a data assimilation framework for discrete event simulations. Second, *case studies* are used to provide concrete cases on which *controlled experiments* are conducted to test and validate the proposed data assimilation framework. Finally, when implementing the data assimilation framework, *design science* and *software engineering techniques* are adopted.

1.4 Organization of the thesis

This thesis consists of six chapters. Since the detailed description of these chapters requires concepts that will be introduced in chapter 2, we only briefly depict the organization of the thesis (see Figure 1.2). After we elaborate on the main concepts and clarify the research gap in chapter 2, a more detailed organization of this thesis will be presented. Chapters 3 and 4 present a data assimilation framework for discrete event simulation, which answers research question 1. Chapter 5 conducts an extensive sensitivity analysis to explore how the data assimilation results are affected by the parameters of the key components in the proposed data assimilation framework, which answers research question 2. Finally, the thesis is concluded in chapter 6, in which the research questions and corresponding answers are summarized, and future research directions are suggested.

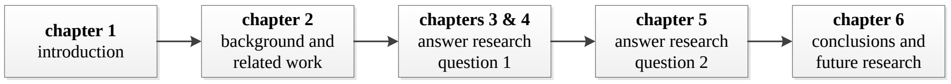

chapter 1
introduction
chapter 2
background and related work
chapters 3 & 4
answer research question 1
chapter 5
answer research question 2
chapter 6
conclusions and future research

Figure 1.2: Brief organization of the thesis

CHAPTER 2

Background and related work

In chapter 1, the research motivation, research objective, and research questions were presented. In this chapter, basic background knowledge, such as modeling and simulation, and data assimilation techniques, are introduced. Based on the characteristics of discrete event simulation and data assimilation techniques, the potentially applicable data assimilation technique for discrete event simulations is chosen, and corresponding challenges are analyzed. Finally, a detailed organization of this thesis is presented.

2.1 Modeling and simulation

As stated by Shannon (1975), modeling and simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system. This process thus involves several basic entities: a real system, its model, the execution of the model, and purposes & conditions of conducting experiments on the model. These entities and their relationships are defined by the modeling and simulation framework. Understanding these concepts will help everyone involved in a simulation modeling project—analysts, programmers, managers, and users—to better carry out their tasks and communicate with each other. Based on this framework, the basic issues and problems encountered in performing M&S activities can be better understood and coherent solutions can be developed (Zeigler et al., 2000).

2.1.1 Framework for modeling and simulation

The modeling and simulation (M&S) framework defines *entities* and their *relationships* that are central to the M&S enterprise (Zeigler et al., 2000). As illustrated in Figure 2.1,

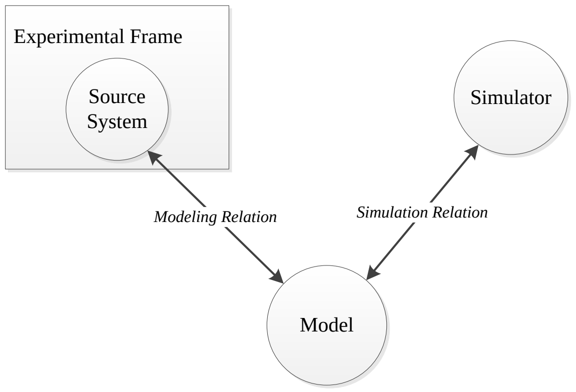


Figure 2.1: Basic entities in modeling and simulation and their relationships

the basic entities of the M&S framework are (Zeigler et al., 2000):

- *source system*, which is the real or virtual environment that we are interested in for modeling. The source system is viewed as a *source of observable data*, in the form of time-indexed trajectories of variables.
- *experimental frame*, which specifies the conditions under which the system is observed or experimented with.
- *model*, which is a set of *instructions*, *rules*, *equations*, or *constraints* for generating I/O behavior.
- *simulator*, which is a computation system capable of executing a model to generate its behavior.

The entities – *system*, *experimental frame*, *model*, *simulator* – become truly significant only when properly related to each other (Zeigler et al., 2000). The two most fundamental are the *modeling* and *simulation* relations. The modeling relation is concerned with how well model generated behavior agrees with the observed system behavior, while the simulation relation is concerned with ensuring that the simulator carries out the model instructions correctly.

A system can be formally defined (i.e., modeled) as a 7-tuple (Ören and Zeigler, 2012):

$$S =< T, X, \Omega, Q, \delta, Y, \lambda > \tag{2.1}$$

where

- $T \subset \mathbb{R}^+_{0,\infty}$ is the time base, where $\mathbb{R}^+_{0,\infty}$ is the positive reals with 0 and ∞;
- X is the input set;
- $\Omega = \{\omega : T \to X\}$ is the input segment set or function;
- Q is the state set;

- $\delta : \Omega \times Q \to Q$ is the transition function;
- Y is the output set;
- $\lambda : Q \to Y$ is the output function.

The *time base* can be *continuous*, where time evolves continuously (time is represented by a real number), or *discrete*, where time evolves by advancing in discrete portions (time is represented by an integer number). Similarly, *values of the state variables* can also be *continuous*, where the variables take values from a continuous set represented as a real number, or *discrete*, where the variables are discrete and can be represented as a finite set of integer numbers. According to how the state variables and time are represented, a system can be categorized into four classes (Wainer, 2009), as shown in Figure 2.2:

- *continuous variable dynamic systems*: both the state variables and time are continuous, as shown in Figure 2.2(a).
- *discrete time dynamic systems*: the state variables are continuous, but time is discrete, as shown in Figure 2.2(b).
- *discrete event dynamic systems*: the state variables are discrete, but time is continuous, as shown in Figure 2.2(c).
- *discrete dynamic systems*: both the state variables and time are discrete, as shown in Figure 2.2(d). These kind of systems are a specialization of discrete event dynamic systems in which events occur at a limited set of times.

According to the same classification criteria, three basic system specification formalisms exist that are suitable to model a particular system: *Differential Equation System Specification (DESS)*, *Discrete Time System Specification (DTSS)*, and *Discrete Event System Specification (DEVS)*. These three basic formalisms are introduced briefly in section 2.1.2. Since this thesis focuses on data assimilation in discrete event simulations, we use two separate sections, sections 2.1.3 and 2.1.4, to introduce discrete event simulations in detail.

2.1.2 Basic modeling formalisms

Modeling is not done by writing a dynamic system structure itself, but indirectly, by using a *system specification formalism* (Zeigler et al., 2000). A system specification formalism is a shorthand means of specifying a system, which implicitly sets constraints on the elements of the dynamic system (Zeigler et al., 2000). A system specification formalism usually allows for the description of system behavior at two levels, the *basic* level and the *coupled* level. Basic system specification formalisms allow for a local description of the dynamic behavior of the system, while coupled system specifications emphasize how to create networks built from components (Zeigler et al., 2000). There are three basic system specification formalisms:

- *Discrete Time System Specification (DTSS).* This formalism represents systems over a discrete time base. It assumes a stepwise mode of execution (Zeigler et al., 2000).

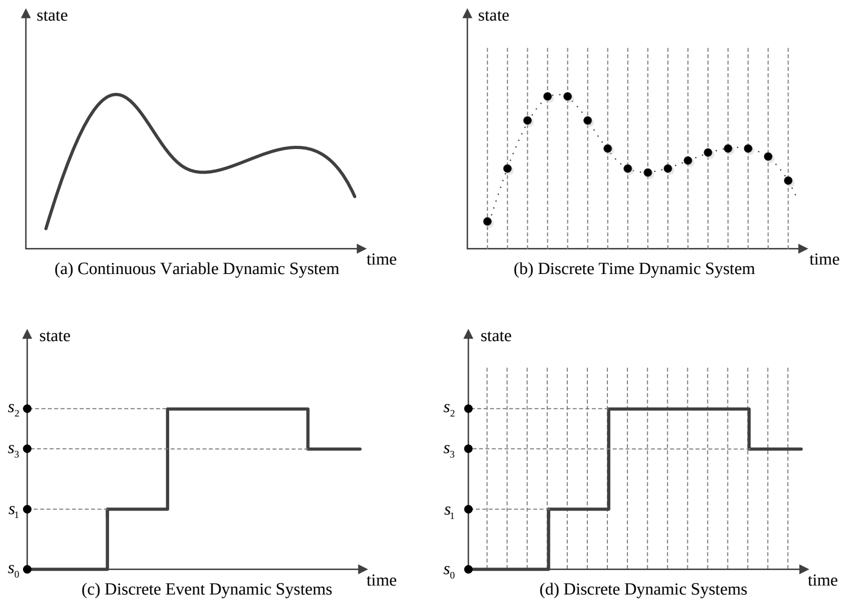


Figure 2.2: System classification according to the representation of time base/state variables

At a particular time instant the model is in a particular state and it defines how this state changes. If the state at time k is $q(k)$, and the input at time k is $x(k)$, the state at time $k+1$ is $q(k+1) = \delta(q(k), x(k))$, where δ is the state transition function. *Difference equations* are an example of DTSS.

- *Differential Equation System Specification (DESS).* This formalism represents systems with continuous state over a continuous time base. It does not specify a next state directly through a state transition function, but specifies the rate of change of the state variables $q_i, i = 1, \ldots, n$ through a derivative function f. This means that at any particular instant, given a state q and an input value x, we know the rate of change of the state, i.e., $\frac{dq_i}{dt} = f(q_1(t), q_2(t), \ldots, q_n(t), x(t)), i = 1, \ldots, n$, and can thus compute the state at any instant in the future using integration methods. *Differential equations* are an example of DESS.

- *Discrete Event System Specification (DEVS).* This formalism represents systems as piecewise constant state trajectories over a continuous time base. The state trajectories are produced by state transition functions δ_{int} and δ_{ext} that are activated by internal or external events. This formalism is presented in detail in section 2.1.4.

2.1.3 Discrete event simulations and world views

Discrete event systems are usually man-made dynamic systems, for example, production or assembly lines, computer/communication networks, or traffic systems. These systems are not easily described by (partial) differential equations or difference equations, instead they are modeled and simulated by the discrete event approach (Ho, 1989). This approach abstracts the physical time and the state of the physical system as a continuous simulation time and a collection of state variables, respectively. A point on this continuous time axis at which at least one state variable changes is called *instant* (Nance, 1981). State changes are only captured at discrete, but possibly random, instants (Schriber et al., 2012), where such a change in state occurring at an instant is called an *event* (Nance, 1981). Since the discrete event approach jumps from one event to the next, omitting the behavior in between, it can be very efficient.

A simulation modeling *world view* precisely defines the dynamic state transitions that occur over time (Pegden, 2010). There are three main types of world views employed in discrete event simulations – *event scheduling*, *activity scanning*, and *process interaction* (Zeigler et al., 2000), which provides different types of *locality* (Overstreet and Nance, 2004) that is defined by Weinberg as "that property when all relevant parts of a program are found in the same place" (Weinberg, 1971). The event scheduling world view provides locality of *time*; the focus of the model is on events (Fujimoto, 2000), and each event processing procedure describes actions that should occur in one instant (Overstreet and Nance, 2004). The activity scanning world view, which provides locality of *state*, describes the actions of objects comprising the model that should occur due to the model assuming a particular state (i.e., due to a particular condition becoming true) (Nance, 1981; Overstreet and Nance, 2004). The process interaction world view provides locality of *object*, in which each process encapsulates the behavioral description of a particular model object (Overstreet and Nance, 2004), and each process advances in a somewhat autonomous fashion through simulation time and interacts with other processes by competing for shared resources (Fujimoto, 2000).

To develop discrete event simulations in general purpose programming languages, the event scheduling strategy is favored by many because it is very flexible (Pegden, 2010), relatively simple to implement (Seck and Verbraeck, 2009), and can be used to efficiently model a wide range of complex systems (Pegden, 2010). However, simulations based on the event scheduling world view are difficult to understand, debug, and maintain, because the behavioral description for a single model object is scattered across the entire program in the different event procedures (Fujimoto, 2000). An object-oriented implementation of the event scheduling world view tackles these problems. As shown in Figure 2.3, object orientation views physical systems as collections of objects that interact in some fashion (Joines and Roberts, 1999), but the interaction is achieved not by a direct method invocation, but by scheduling them via constructing a simulation event (Jacobs et al., 2002). The object-oriented implementation of the event scheduling world view not only makes running a simulation model very simple by only executing ordered simulation events, but also greatly improves the modularity and maintainability of the simulation program.

2.1.4 Discrete Event System Specification (DEVS)

There are a variety of discrete event modeling formalisms, such as the Max-Plus Algebra (De Schutter and Van den Boom, 2008), Petri Nets (DiCesare et al., 1994), and Discrete Event System Specification (DEVS) (Zeigler et al., 2000). Among these formalisms, DEVS is the most widely used formalism in the simulation community, and it is also the common denominator of other discrete event modeling formalisms (Vangheluwe, 2000),

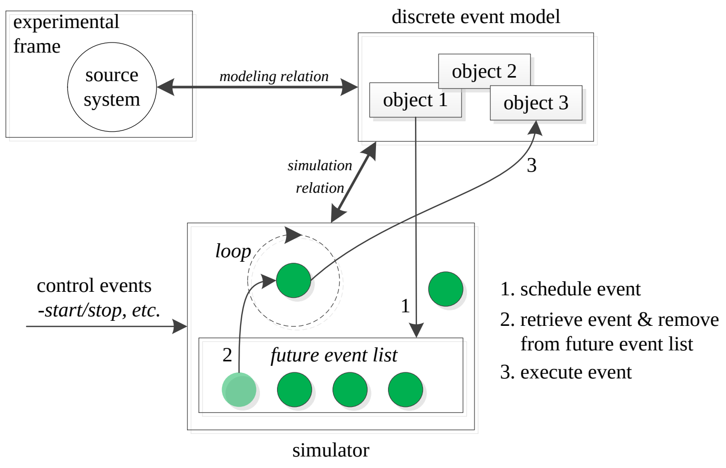


Figure 2.3: The object-oriented implementation of the event scheduling world view (Jacobs et al., 2002)

which means that models in other discrete event modeling formalisms can be mapped to DEVS models. Given its importance, we introduce the DEVS formalism in more detail in this section.

Discrete Event System Specification (DEVS) is a formalism that was developed by Zeigler as a general methodology for describing discrete event systems (Zeigler et al., 2000). The formalism is based on general system theory, and it allows us to represent all the systems whose input/output behavior can be described by sequences of events. DEVS allows for the description of system behavior at two levels: the *atomic* level and the *coupled* level. An atomic DEVS model describes the autonomous behavior of a discrete event system as a sequence of deterministic transitions between sequential states over time, as well as how it reacts to external input (events) and how it generates output (events). Formally, an atomic DEVS model M is defined by the following structure:

$$M =< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$$

where

- X and Y are the sets of input and output events
- S is a set of sequential states
- $\delta_{int} : S \rightarrow S$ is the internal state transition function
- $\delta_{ext} : Q \times X \rightarrow S$ is the external state transition function, where

  $Q = \{(s, e)|s \in S, 0 \leq e \leq ta(s)\}$ is the total state set

  e is the time elapsed since the last transition
- $\lambda : S \rightarrow Y$ is the output function

- $ta : S \rightarrow R^+_{0,\infty}$ is the time advance function, where $\mathbb{R}^+_{0,\infty}$ is the positive reals with 0 and ∞

Functions ta, δ_{int}, λ define the autonomous behavior of an atomic DEVS model when no external events occur (Zeigler et al., 2000). At any time, the system is in some state, $s \in S$. Each possible state s has an associated time advance calculated by the time advance function $ta(s)$, which indicates that the system will stay in state s for $ta(s)$ units of time, if no external event occurs. When the calculated time advance expires, i.e., when the elapsed time $e = ta(s)$, the system outputs the value, $\lambda(s)$, and changes to a new state $\delta_{int}(s)$.

If an external event $x \in X$ occurs before this time advance, i.e., when the system is in total state (s, e) with $e \leq ta(s)$, the system changes to a new state $\delta_{ext}(s, e, x)$ (Zeigler et al., 2000). Thus, the internal transition function dictates the system's new state when no events occur since the last transition; while the external transition function dictates the system's new state when an external event occurs – this state is determined by the input, x, the current state, s, and how long the system has been in this state, e. In both cases, the system is then in some new state s' with some new time advance, $ta(s')$, and the same story continues. Figure 2.4 shows how a discrete event model evolves over time.

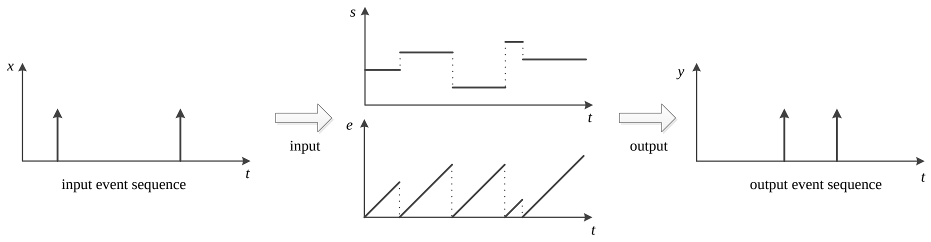


Figure 2.4: DEVS trajectories

Atomic models can be coupled to form a lager model. A coupled DEVS model N is defined by the structure:

$$N =< X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, Select >$$

where

- X and Y are the sets of input and output events of the coupled model
- D is a set of component names, and for each $i \in D$, M_i is an atomic DEVS model defined as
$$M_i =< X_i, Y_i, S_i, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i >, \forall i \in D$$
- for each $i \in D \cup \{N\}$, I_i is the set of components which are influenced by component i, and $I_i \subseteq D \cup \{N\}, i \notin I_i$
- for each $j \in I_i$, $Z_{i,j}$ is the output-to-input translation function, where

$$Z_{i,j} : \begin{cases} X \rightarrow X_j & \text{if } i = N \text{ and } j \in D \\ Y_i \rightarrow Y & \text{if } i \in D \text{ and } j = N \\ Y_i \rightarrow X_j & \text{if } i \in D \text{ and } j \in D \end{cases}$$

- $Select : 2^D \rightarrow D$ is a tie-breaking function with $Select(E) \in E$ to arbitrate the occurrence of simultaneous events. In other words, when multiple (atomic) models have to change their state at the same time, $Select$ determines the order in which the (atomic) models are allowed to update their state one by one.

DEVS models are closed under coupling, i.e., the coupling of DEVS models defines an equivalent atomic DEVS model (Vangheluwe, 2001).

2.2 Data assimilation techniques

The aim of data assimilation is to incorporate measured (noisy) observations into a dynamical system model in order to produce accurate estimates of all the current (and future) state variables of the system (Nichols, 2003). Therefore, data assimilation relies on the following elements to work (Ide et al., 1997; Bouttier and Courtier, 1999):

- *system model*, describing the evolution of the state over time, which is usually defined in a discrete time state space form:

$$s_k = f_k(s_{k-1}) + \nu_{k-1}, k = 1, 2, \ldots, \tag{2.2}$$

  in which f_k is a (possibly nonlinear) function of the state vector s_{k-1}, and ν_{k-1} represents a system noise process.

- *measurement model*, relating noisy observations to the state, which is also defined as a discrete time equation:

$$m_k = g_k(s_k) + \varepsilon_k, k = 1, 2, \ldots, \tag{2.3}$$

  in which g_k is a (possibly nonlinear) function that maps the state to the measurements, and ε_k represents a measurement noise process.

- *data assimilation techniques*, that carry out state estimation based on information from both the model and the measurements, and in the process address measurement and modeling errors.

There are two distinct classes of data assimilation techniques. One is the class of *variational techniques*, and the other is the class of *sequential methods*, which are introduced in section 2.2.1 and section 2.2.2, respectively. Particle filtering belongs to the category of sequential data assimilation techniques, but due to its importance, we introduce it in section 2.2.3 separately.

2.2.1 Variational techniques

3-Dimensional Variational Analysis (3D-VAR) (Wu et al., 2002) and 4-Dimensional Variational Analysis (4D-VAR) (Lorenc and Rawlins, 2005) are two typical variational techniques in use. 3D-VAR minimizes the cost function J shown in equation (2.4) which measures the misfit between s_k and the background state s_k^b (commonly derived from a short-range forecast), and also between s_k and the observation m_k. In equation 2.4, $\mathbf{Q}$ and

$\mathbf{R}$ are the covariance matrix of the system noise and the measurement noise, respectively. The minimization of J is done with respect to s_k, and the resultant s_k is the estimated system state (also called *analysis*).

$$J(s_k) = \frac{1}{2}(s_k - s_k^b)^T \mathbf{Q}^{-1}(s_k - s_k^b) + \frac{1}{2}(m_k - g_k(s_k))^T \mathbf{R}^{-1}(m_k - g_k(s_k)) \quad (2.4)$$

In 3D-VAR, all observations in the time-window (i.e. measurement interval) are treated as if they occurred at the same time (Lorenc and Rawlins, 2005). This introduces some error because real systems are evolving over time. 4D-VAR addresses this problem by introducing the time dimension into assimilation (Lahoz and Schneider, 2014). In 4D-VAR, the observation operators are generalized to include a forecast model that will allow a comparison between the model states and the observations distributed over the time window (Bouttier and Courtier, 1999).

2.2.2 Sequential methods

In this thesis, we are interested in sequential data assimilation techniques, which assimilate data sequentially over time and whose main objective is to correct the estimated state at every time an observation becomes available (Arulampalam et al., 2002; Pelc, 2013). The sequential data assimilation techniques consist of two steps:

- *prediction step*
$$s_k^f = f_k(s_{k-1}^a) \quad (2.5)$$
- *update step*
$$s_k^a = s_k^f + \mathbf{K}_k(m_k - g_k(s_k^f)) \quad (2.6)$$

where superscript $(\cdot)^f$ and superscript $(\cdot)^a$ mean forecast (a priori estimate) and analysis (a posteriori estimate), respectively. $\mathbf{K}_k$ is a gain matrix which is chosen to minimize the *analysis error covariance matrix* $\mathbf{P}_k^a = \overline{(s_k^t - s_k^a)(s_k^t - s_k^a)^T}$, where superscript $(\cdot)^t$ represents the true value (i.e. ground truth). If f_k and g_k are linear, and ν_{k-1} and ε_k are stationary zero-mean white noise (Arulampalam et al., 2002), the *forecast error covariance matrix* $\mathbf{P}_k^f = \overline{(s_k^t - s_k^f)(s_k^t - s_k^f)^T}$ and the analysis error covariance matrix $\mathbf{P}_k^a$ can be accurately calculated. This yields the *optimal* filter, also known as the Kalman filter (KF). However, in many situations of interest, the linear assumption does not hold, which poses great difficulties on the computation of error covariance matrices $\mathbf{P}_k^f$ and $\mathbf{P}_k^a$, therefore many approximation methods are proposed to tackle these difficulties. The extended Kalman filter (EKF) linearizes the nonlinear system model locally around s_k^f to ease the computation of $\mathbf{P}_k^f$ and $\mathbf{P}_k^a$ (Gillijns et al., 2006). Although EKF is effective in many practical cases, the method fails to account for the fully nonlinear dynamics in propagating the error covariance, which, in turn, fails to represent the error probability density (Gillijns et al., 2006). Another approach that tackles nonlinearity very well is the ensemble Kalman filter (EnKF) (Evensen, 2003). In EnKF, the error covariance matrices are approximated by using an ensemble of model states. Theoretically, full error statistics can be exactly represented by an infinite ensemble of model states (Evensen, 2003). EnKF does not involve an approximation of the nonlinearity of f_k and g_k. The computational burden of evaluating the Jacobians is hence absent (Gillijns et al., 2006).

Notice that these aforementioned sequential data assimilation techniques require the errors to be Gaussian. Another powerful sequential data assimilation technique, particle filters, does not impose any restrictions on the model and the data, and due to its importance, we explain its principle in section 2.2.3 separately.

2.2.3 Particle filters

Given the system model and the measurement model described in equation 2.2 and equation 2.3, the objective is to estimate the conditional distribution of all states up to time k given all available measurements, i.e., $p(s_{0:k}|m_{1:k})$, where $s_{0:k} = \{s_i, i = 0, 1, \dots, k\}, m_{1:k} = \{m_i, i = 1, 2, \dots, k\}$. Since the analytic solution of $p(s_{0:k}|m_{1:k})$ is usually intractable, we use a sufficient large set of Monte Carlo samples (particles) from $p(s_{0:k}|m_{1:k})$ with their associated weights to approximate this posterior distribution. When the number of samples becomes very large, this Monte Carlo characterization becomes an equivalent representation of the usual functional description of the posterior distribution. That is the basic idea behind the particle filters (Arulampalam et al., 2002; Djurić et al., 2003; van Leeuwen, 2009). Based on the samples, estimates can be obtained by standard Monte Carlo integration techniques.

Let $\chi_k = \{s^i_{0:k}, w^i_k\}_{i=1}^{N_p}$ represent a random measure that characterizes the posterior distribution $p(s_{0:k}|m_{1:k})$, where $\{s^i_{0:k}\}_{i=1}^{N_p}$ is a set of support points, and $\{w^i_k\}_{i=1}^{N_p}$ is the set of their weights, then $p(s_{0:k}|m_{1:k})$ can be approximated as

$$p(s_{0:k}|m_{1:k}) \approx \sum_{i=1}^{N_p} w^i_k \delta(s_{0:k} - s^i_{0:k}) \tag{2.7}$$

where $\delta(\cdot)$ is the Dirac delta function. The weights are computed using the principle of importance sampling, i.e., if the samples $s^i_{0:k}$ are drawn from an importance density $q(s_{0:k}|m_{1:k})$, then the weights in equation 2.7 are defined by

$$w^i_k \propto \frac{p(s^i_{0:k}|m_{1:k})}{q(s^i_{0:k}|m_{1:k})}$$

Based on Bayes' theorem, $p(s_{0:k}|m_{1:k})$ can be expressed as

$$p(s_{0:k}|m_{1:k}) = \frac{p(s_{0:k})p(m_{1:k}|s_{0:k})}{p(m_{1:k})}$$

Similarly we have $p(s_{0:k-1}|m_{1:k-1}) = \frac{p(s_{0:k-1})p(m_{1:k-1}|s_{0:k-1})}{p(m_{1:k-1})}$. Therefore we can obtain a sequential update equation as

$$\begin{aligned} p(s_{0:k}|m_{1:k}) &= \frac{p(m_k|s_k)p(s_k|s_{k-1})p(s_{0:k-1}|m_{1:k-1})}{p(m_k|m_{1:k-1})} \\ &\propto p(m_k|s_k)p(s_k|s_{k-1})p(s_{0:k-1}|m_{1:k-1}) \end{aligned} \tag{2.8}$$

If the importance density is chosen to factorize such that

$$q(s_{0:k}|m_{1:k}) = q(s_k|s_{0:k-1}, m_{1:k})q(s_{0:k-1}|m_{1:k-1}),$$

then the random measure $\chi_{k-1} = \{s^i_{0:k-1}, w^i_{k-1}\}_{i=1}^{N_p}$ can be updated in a sequential manner when a new measurement m_k is available:

- samples $s^i_{0:k} \sim q(s_{0:k}|m_{1:k})$ are obtained by augmenting samples from the previous time step $s^i_{0:k-1} \sim q(s_{0:k-1}|m_{1:k-1})$ with the new state $s^i_k \sim q(s_k|s^i_{0:k-1}, m_{1:k})$;
- weights are updated by

$$\begin{aligned} w^i_k \propto \frac{p(s^i_{0:k}|m_{1:k})}{q(s^i_{0:k}|m_{1:k})} &= \frac{p(m_k|s^i_k)p(s^i_k|s^i_{k-1})p(s^i_{0:k-1}|m_{1:k-1})}{q(s^i_k|s^i_{0:k-1}, m_{1:k})q(s^i_{0:k-1}|m_{1:k-1})} \\ &= \frac{p(m_k|s^i_k)p(s^i_k|s^i_{k-1})}{q(s^i_k|s^i_{0:k-1}, m_{1:k})} w^i_{k-1} \end{aligned} \tag{2.9}$$

If we assume that $q(s_k|s_{0:k-1}, m_{1:k}) = q(s_k|s_{k-1}, m_k)$, i.e., the importance density is only dependent on s_{k-1} and m_k, then we have

$$w^i_k \propto \frac{p(m_k|s^i_k)p(s^i_k|s^i_{k-1})}{q(s^i_k|s^i_{k-1}, m_k)} w^i_{k-1} \tag{2.10}$$

In practice, the system transition density is often chosen as the importance density, i.e., $q(s_k|s_{k-1}, m_k) = p(s_k|s_{k-1})$. Then equation 2.10 is simplified to

$$w^i_k \propto p(m_k|s^i_k) w^i_{k-1} \tag{2.11}$$

A major problem of particle filters is that the discrete random measure degenerates quickly (Arulampalam et al., 2002; Djurić et al., 2003; van Leeuwen, 2009). In other words, most particles except for a few are assigned negligible weights. The solution is to resample the particles after they are updated. Different resampling algorithms and methods exist to determine when resampling is necessary (Arulampalam et al., 2002; Djurić et al., 2003; van Leeuwen, 2009; Douc et al., 2005). A simple and often adopted resampling method is to replicate particles in proportion to their weights. It has been shown that a sufficiently large number of particles are able to converge to the true posterior distribution even in nonlinear, non-Gaussian dynamic systems (Arulampalam et al., 2002; Djurić et al., 2003; van Leeuwen, 2009).

2.3 Data assimilation in discrete event simulations

2.3.1 Characteristics of discrete event simulations

As introduced in section 2.1.3 and section 2.1.4, the key characteristics of discrete event simulations can be summarized as follows:

- *The model state is defined as a collection of atomic model states, each of which is represented by a combination of continuous and discrete variables.* Take the case study in the gold mine system (see chapter 3) as an example. The position of the elevator is a continuous state variable; the number of trucks that are waiting for loading is a discrete state variable, and the status of the miner, i.e. busy or idle, is also a discrete state variable.

- *The behavior of discrete event simulations is highly nonlinear, non-Gaussian.* In a discrete event simulation, the state evolution is usually based on rules, which define what the next state will be when the time advance expires, how to react when external events occur, etc. These functions are highly nonlinear step functions, because state changes in a discrete event simulation happen instantaneously at the event. The Gaussian error assumption is easily violated, since both state variables and measurements can be non-numerical.
- *The state updates in a discrete event simulation happen locally and asynchronously within each atomic model component; for each atomic model component, its state is updated at time instants lying irregularly on a continuous time axis, and the duration between two consecutive state updates is usually not fixed.* The state trajectory in a discrete event simulation is thus piecewise constant as shown (in blue) in Figure 2.5, which only captures changes of interest in the real state evolution.
- *Many discrete event systems are open systems.* In these systems (e.g., urban traffic systems), entities can flow in and flow out through system boundaries, which leads to a dynamic number of components in the system.

2.3.2 Data assimilation technique for nonlinear, non-Gaussian applications

The main sequential data assimilation techniques introduced in section 2.2 are compared in Table 2.1. Though EKF and EnKF can deal with nonlinear models, they still rely on Gaussian error assumptions. Since particle filters approximate a probability density function by a set of particles and their associated importance weights, they put little or no assumption on the properties of the system model, which thus are *in principle* applicable to nonlinear and/or non-Gaussian applications (Bai et al., 2016).

Since discrete event simulations are highly nonlinear, non-Gaussian models, particle filters are *in principle* applicable in discrete event simulations. However, applying particle filtering in discrete event simulations still encounters several theoretical and practical problems which will be explained in detail in section 2.3.4.

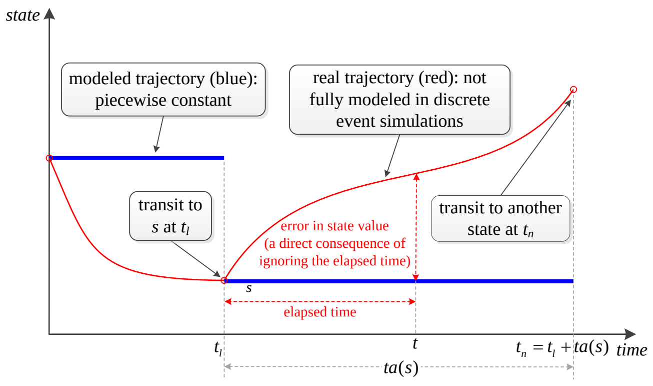
Table 2.1: Comparison of main sequential data assimilation techniques (Yuan, 2013)

	Kalman filter (KF)	extended Kalman filter (EKF)	ensemble Kalman filter (EnKF)	particle filter (PF)
system model	Gaussian errors linear	Gaussian errors continuously differentiable	Gaussian errors	no restrictions
measurement model	Gaussian errors linear	Gaussian errors continuously differentiable	Gaussian errors	no restrictions

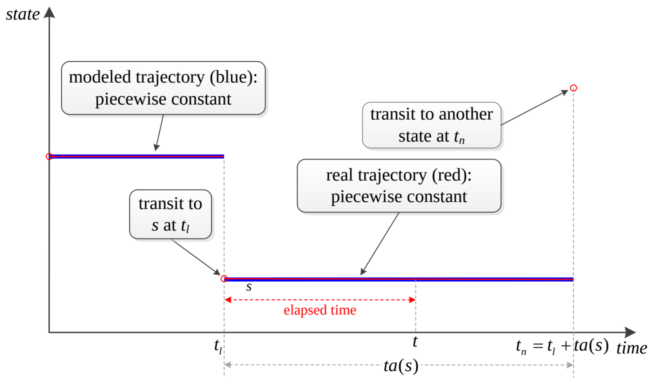
2.3.3 Comments on data assimilation in DEVS-FIRE

The Systems Integrated Modeling and Simulation (SIMS) Lab[1] in Georgia State University has done extensive work on data assimilation in wildfire spread simulations (Gu and

[1]More information can be found at `https://grid.cs.gsu.edu/~cscxlh/`.


state
modeled trajectory (blue): piecewise constant
real trajectory (red): not fully modeled in discrete event simulations
transit to s at t_l
error in state value (a direct consequence of ignoring the elapsed time)
transit to another state at t_n
s
elapsed time
t_l
t
$t_n = t_l + ta(s)$
time
$ta(s)$

(a) Continuous state


state
modeled trajectory (blue): piecewise constant
transit to another state at t_n
real trajectory (red): piecewise constant
transit to s at t_l
s
elapsed time
t_l
t
$t_n = t_l + ta(s)$
time
$ta(s)$

(b) Discrete state

Figure 2.5: The modeled state trajectory and the real state trajectory in discrete event simulations ($ta(s)$ is the time advance of state s; state updates captured in the discrete event simulation (red circles) can be different from those in the real state trajectory; since we focus on the effect of ignoring the elapsed time when retrieving the model state, we do not show the difference in states explicitly in the figure)

Hu, 2008; Gu et al., 2009; Gu, 2010; Hu, 2011; Xue et al., 2012). In their work, the simulation model for wildfire spread is a cellular automaton based discrete event simulation model called DEVS-FIRE (Ntaimo et al., 2008; Hu et al., 2012); the measurements are temperature values from sensors deployed in the fire field; particle filters are employed to assimilate these measurements into the DEVS-FIRE model to estimate the spread of the fire front.

Since the measurement in the wildfire application is the temperature at a time instant, and it is only related to the system state (fire front) at the same time, their system model can be formalized as a discrete time state space model which only focuses on the state evolution at time instants when measurements are available, and the detailed evolution in between (not of interest in their application) is done with the DEVS-FIRE model. However, when retrieving the system state at the time instant when a measurement is available, the retrieved state is only a combination of sequential states of all atomic components (i.e. cells), which do not reflect any elapsed time information. As a result, errors exist as explained in Figure 2.5.

2.3.4 Challenges of applying particle filtering in discrete event simulations

In most applications of particle filtering, the system model is usually formulated as a standard discrete time state space model, and the system state is updated at the same pace with the measurement process, as assumed in equation 2.2 and equation 2.3. However, in discrete event simulations, state updates happen *locally* and *asynchronously* within each (atomic) model component, and the system state takes a new value when one of its (atomic) components has a state update. Consequently the time between two consecutive (system) state updates is usually not fixed, i.e., the system state process is asynchronous with the measurement process, which usually feeds data at fixed times (Bouttier and Courtier, 1999).

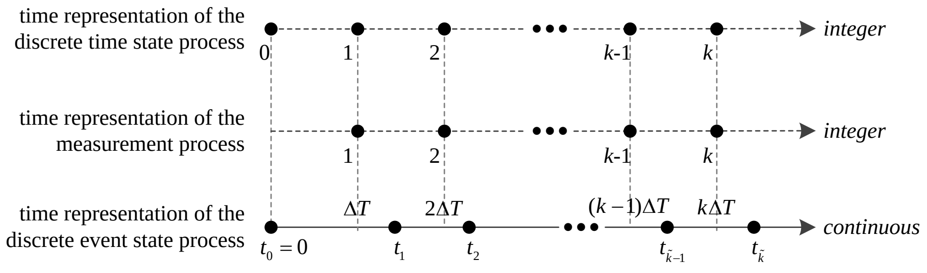

time representation of the discrete time state process
0
1
2
k-1
k
integer
time representation of the measurement process
1
2
k-1
k
integer
time representation of the discrete event state process
ΔT
2ΔT
(k − 1)ΔT
kΔT
continuous
$t_0 = 0$
t_1
t_2
$t_{\tilde{k}-1}$
$t_{\tilde{k}}$

Figure 2.6: Time representation in the discrete time state process, the measurement process, and the discrete event state process (in the state processes, each black dot indicates a state update, while in the measurement process, each black dot represents an arrival of a measurement)

Figure 2.6 illustrates how the time is represented in the discrete time state process, the measurement process, and the discrete event state process. In a discrete time (state or measurement) process, the time is represented as a nonnegative integer k (see section 2.2), while in a discrete event state process, the time is represented as a nonnegative real number. In a discrete event state process, since the number of state updates during any time interval is a bounded number (Zeigler et al., 2000), the time when the state updates happen can be indexed by an integer $\tilde{k}$, i.e., $t_{\tilde{k}}$ indicates the time instant when the model makes the $\tilde{k}$-th state update.

The mismatch between the (discrete time) measurement process and the discrete event state process results in two problems that hinder the application of particle filtering in discrete event simulations.

- *The state retrieved from a discrete event simulation model is a combination of sequential states of atomic components that were updated at past time instants, with* *which inaccurate estimation results will be obtained.* If we directly retrieve the state of an atomic model, the retrieved state is just a sequential state that does not contain the elapsed time, as shown in Figure 2.5. In particle filter based data assimilation, the likelihood of each particle is evaluated largely based on $p(m_k|s_k)$ (see equation 2.9, 2.10 and 2.11), i.e., evaluating the state at time k based on the measurement at the same time. Ignoring the elapsed time, $p(m_k|s_k)$ will be computed wrongly as $p(m_k|s)$ where s is a state updated at the last state transition (see Figure 2.5), and consequently the particle will be assigned a wrong weight, therefore inaccurate estimation results will be obtained. This effect is evident for continuous states (see Figure 2.5(a)); for discrete states, in order to compute $p(m_k|s_k)$, one probably needs the elapsed time to define a proper measurement model that relates the discrete state to the measurement. However, ignoring the elapsed time will also make this definition and computation inaccurate.

- *The dimension of the state trajectory during a fixed time interval is a random variable, which makes the standard sequential importance sampling algorithm inapplicable.* Since the time between two consecutive state updates (at both the coupled level and the atomic level) in discrete event simulations is usually a random variable, the number of state points in a state trajectory during a fixed time interval is also random.

For discrete event simulations of open systems (e.g., urban traffic systems), where entities can flow in and flow out through the system boundaries, the variable dimension problem arises in a different way. Entities flowing in and flowing out lead to a dynamic (random) number of components in the system, and as a result, the system state has a variable dimension (and thus the state trajectory has a variable dimension). Other practical problems that mainly relate to data issues, such as non-numerical data, e.g., event sequences, also make particle filtering in discrete event simulations highly problematic.

Given the existing research gap, a particle filter based data assimilation framework for discrete event simulations (of both closed and open systems) needs to be developed, in which the aforementioned challenges have to be dealt with. Other issues which closely relate to the proposed framework, such as sensitivity analysis with respect to parameters of the particle filtering method (e.g., the number of particles), and implementation for discrete event simulation languages, need to be investigated as well.

2.4 Outlook of subsequent chapters

In this chapter, background knowledge for the research, such as modeling and simulation (with a focus on discrete event simulations), and data assimilation techniques, were introduced. Since discrete event simulations are highly nonlinear, non-Gaussian systems, particle filters are in principle applicable to discrete event simulations. However, applying particle filtering in discrete event simulations still encounters several theoretical and practical problems, that thus asks for a particle filter based data assimilation framework for discrete event simulations (of both closed and open systems), in which these theoretical and practical problems can be addressed.

As explained in section 2.3.4, the key challenges of applying particle filtering in discrete event simulations are twofold. On the one hand, we need to obtain correct state values when measurements are available; one the other hand, we need to come up with a way to update the random measure that approximates the posterior distribution of the state trajectory with a variable dimension, since the standard sequential importance sampling algorithm cannot be applied due to the variable dimension problem. To deal with these challenges, we present a particle filter based data assimilation framework in chapters 3 and 4, in which we address the state retrieval problem and the variable dimension problem. Specifically, in chapter 3, we solve these problems for discrete event simulations of closed systems (i.e., system entities do not change over time), while in chapter 4, we tackle these problems for discrete event simulations of open systems. We employ case studies to demonstrate the working of the proposed data assimilation framework. In chapter 3, noisy data (event sequences, entity positions) is assimilated into a gold mine simulation (of a closed goldmine system) in order to estimate the truck arrival times at the bottom of the vertical shaft, while in chapter 4, noisy data (vehicle passages, traffic signal timings, travel time observations) is assimilated into a microscopic traffic simulation (of an open urban traffic system) to reconstruct vehicle trajectories on signalized urban arterials. These cases also provide concrete data for the sensitivity analysis in chapter 5, in which we analyze in detail the characteristics of the proposed data assimilation framework and its sensitivity to a number of parameters in the framework. Finally in chapter 6 we conclude this thesis by summarizing the research findings and suggesting future research directions. To conduct controlled experiments in case studies, an implementation of the proposed data assimilation framework is needed. Therefore, in appendix A, we provide a reference implementation based on DSOL (**D**istributed **S**imulation **O**bject **L**ibrary) (Jacobs et al., 2002; Jacobs, 2005) which is the simulation environment chosen for this research.

Based on the explanation above, a detailed organization of this thesis is depicted in Figure 2.7.

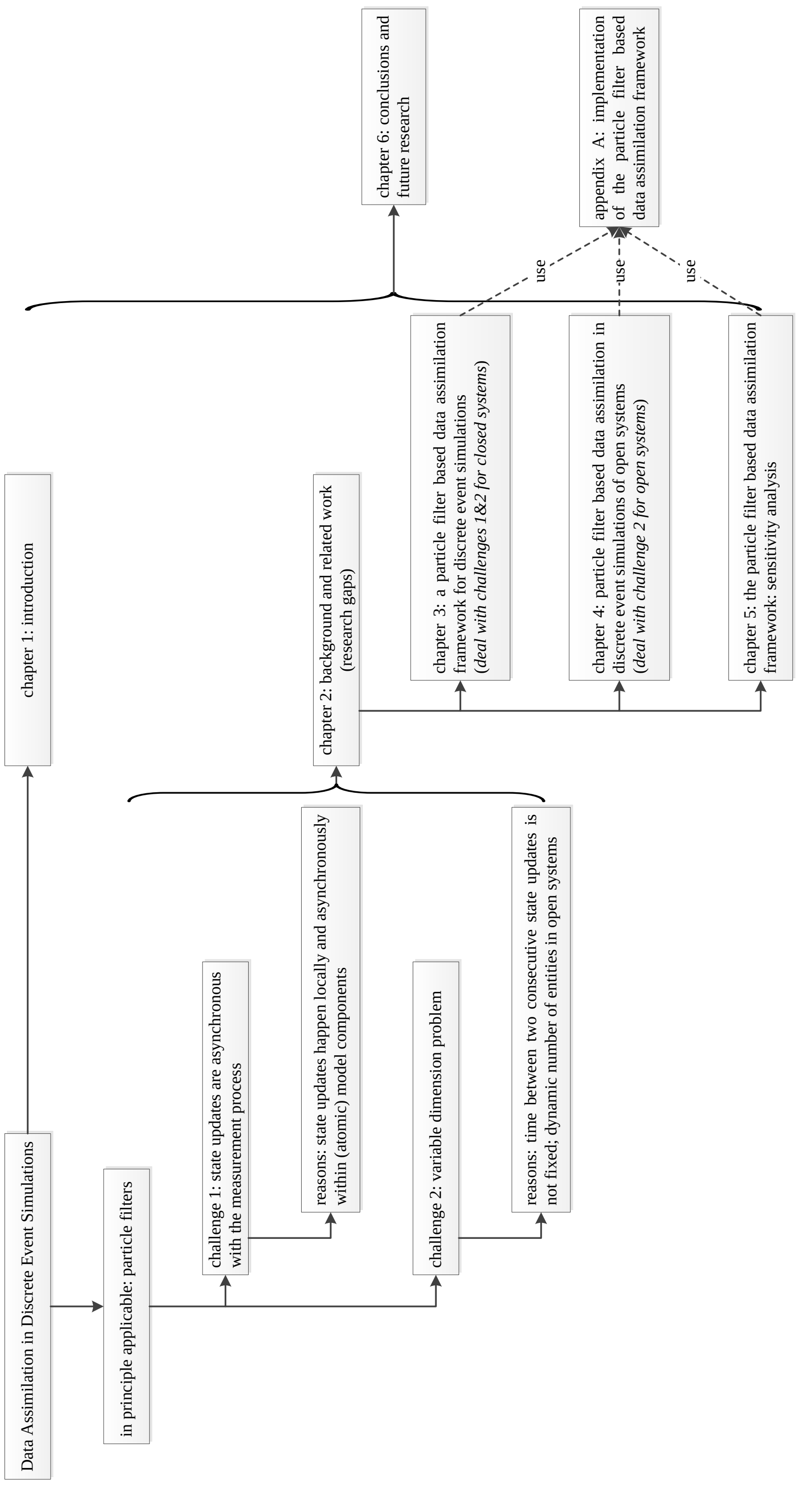

Data Assimilation in Discrete Event Simulations
in principle applicable: particle filters
challenge 1: state updates are asynchronous with the measurement process
reasons: state updates happen locally and asynchronously within (atomic) model components
challenge 2: variable dimension problem
reasons: time between two consecutive state updates is not fixed; dynamic number of entities in open systems
chapter 1: introduction
chapter 2: background and related work (research gaps)
chapter 3: a particle filter based data assimilation framework for discrete event simulations (deal with challenges 1&2 for closed systems)
chapter 4: particle filter based data assimilation in discrete event simulations of open systems (deal with challenge 2 for open systems)
chapter 5: the particle filter based data assimilation framework: sensitivity analysis
chapter 6: conclusions and future research
appendix A: implementation of the particle filter based data assimilation framework
use
use
use

Figure 2.7: Detailed organization of the thesis

CHAPTER 3

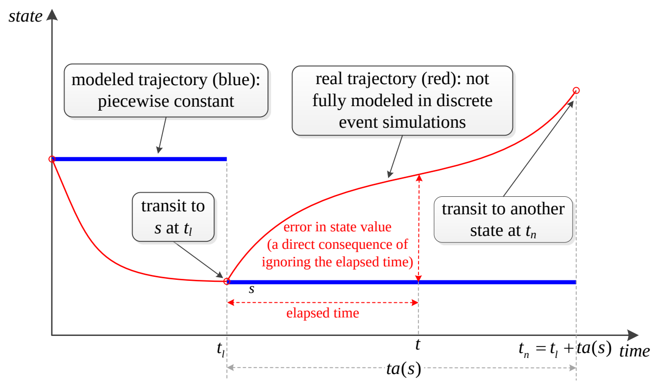
A particle filter based data assimilation framework for discrete event simulations[1]

Since discrete event simulations are typical nonlinear, non-Gaussian systems, particle filtering is adopted to carry out state estimation by fusing noisy data into discrete event simulations. However, applying particle filtering in discrete event simulations still encounters several theoretical and practical problems as explained in chapter 2. In this chapter, we propose a particle filter based data assimilation framework for discrete event simulations, in which the aforementioned problems in chapter 2 are addressed. In this chapter, we focus on discrete event simulations of closed systems, in which the system components do not change over time. In the rest of this chapter, unless specifically stated, the discrete event simulations are the simulations of closed systems.

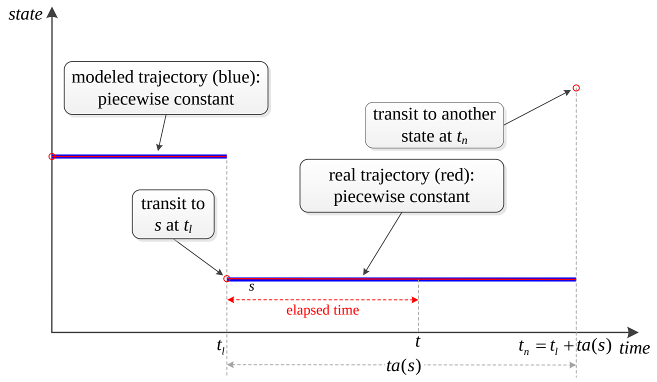
3.1 Revisiting the challenges of applying particle filtering in discrete event simulations

In applications of particle filtering in standard discrete time state space models, the system state is updated at the same pace with the measurement process (Wang and Hu, 2015; Wu et al., 2015). However, in discrete event simulations, state updates happen *locally* and *asynchronously* within each (atomic) model component, and the system state takes a new

[1]This chapter is a revised version of a paper submitted to *Simulation: Transactions of the Society for Modeling and Simulation International*: Xie, X., Verbraeck, A.: *A particle filter based data assimilation framework for discrete event simulations*.


state
modeled trajectory (blue): piecewise constant
real trajectory (red): not fully modeled in discrete event simulations
transit to s at t_l
transit to another state at t_n
error in state value (a direct consequence of ignoring the elapsed time)
s
elapsed time
t_l
t
$t_n = t_l + ta(s)$
time
$ta(s)$

(a) Continuous state


state
modeled trajectory (blue): piecewise constant
transit to another state at t_n
real trajectory (red): piecewise constant
transit to s at t_l
s
elapsed time
t_l
t
$t_n = t_l + ta(s)$
time
$ta(s)$

(b) Discrete state

Figure 3.1: The modeled state trajectory and the real state trajectory in discrete event simulations ($ta(s)$ is the time advance of state s; state updates captured in the discrete event simulation (red circles) can be different from those in the real state trajectory; since we focus on the effect of ignoring the elapsed time when retrieving the model state, we do not show the difference in states explicitly in the figure)

value when one of its components has a state update. Consequently the time between two consecutive state updates (at both the coupled level and the atomic level) is usually not fixed, i.e., the system state process is asynchronous with the measurement process, which usually feeds data at fixed times (Bouttier and Courtier, 1999). The mismatch between the (discrete time) measurement process and the discrete event state process incurs two problems which hinder the application of particle filtering in discrete event simulations. The first problem is that the state retrieved from a discrete event simulation model is a combination of sequential states of atomic components that were updated at past time instants, with which inaccurate estimation results will be obtained. Retrieving the state of an atomic model directly will only get a sequential state that does not contain the elapsed time, as shown in Figure 3.1. In particle filter based data assimilation, ignoring the elapsed time will make particles being wrongly evaluated, since $p(m_k|s_k)$ (see the weight computation equations 2.9, 2.10 and 2.11) will be computed wrongly as $p(m_k|s)$ where s is a state updated at the last state transition (see Figure 3.1). As a result, inaccurate estimation results will be obtained. This effect is evident for continuous states (see Figure 3.1(a)); for discrete states, in order to compute $p(m_k|s_k)$, one probably needs the elapsed time to define a proper measurement model that relates the discrete state to the measurement. However, ignoring the elapsed time will also make this definition and computation inaccurate. The second problem is the variable dimension problem, i.e., the dimension of the state trajectory during a fixed time interval is a random variable, which makes the standard sequential importance sampling algorithm inapplicable. Other practical problems which mainly relate to data issues, such as non-numerical data, e.g., event sequences, also make particle filtering in discrete event simulations highly problematic.

In this chapter, we propose a particle filter based data assimilation framework for discrete event simulations, in which the aforementioned problems will be addressed. Section 3.2 presents such a framework, which includes the system model (integer indexed state process), measurement model (based on the interpolated states), and the particle filtering algorithm for discrete event simulations. A case in a gold mine system is studied in section 3.3 (tailor the generic data assimilation framework to the specific estimation problem), section 3.4 (qualitative analysis), and section 3.5 (quantitative analysis) to demonstrate the working of the proposed data assimilation framework.

3.2 The particle filter based data assimilation framework for discrete event simulations

In this section, the proposed data assimilation framework for discrete event simulations is presented. In order to formalize the data assimilation problem, we need to formalize the state transitions in a discrete event model as an integer indexed state process (i.e., in the same form with a discrete time model), therefore, in section 3.2.1, we show how to achieve such formalization. In section 3.2.2, the interpolation operation is introduced in order to obtain updated state values, and the measurement model is formalized accordingly. On the basis of the integer indexed state process and the measurement model, the particle filtering algorithm is formalized in section 3.2.3, in which the variable dimension problem is solved. Finally some practical remarks which can help simplify the application of the data assimilation framework are given in section 3.2.4.

3.2.1 System model

In order to describe the discrete event models formally, we use the DEVS formalism (see section 2.1.4). First we introduce how the state is evolved in a DEVS model. Subsequently, we show how to formalize the state transitions in a DEVS model as an integer indexed state process. Though the integer indexed state process has the same form with a discrete time model, the duration between two consecutive state updates is not fixed.

3.2.1.1 State evolution in a coupled DEVS model

Consider a coupled DEVS model $N =< X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, Select >$, where $\forall i \in D$, M_i is an atomic component defined as $M_i =< X_i, Y_i, S_i, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i >$ (please refer to section 2.1.4 for more details). For any atomic component M_i, its state evolution is achieved by executing (triggered by the simulator) internal state transition $\delta_{int,i}(s_i)$ and external state transition $\delta_{ext,i}(s_i, e_i, x_i)$. In this section, we clarify how state evolution of the coupled DEVS model N is driven by state evolutions of its components.

Since DEVS models are closed under coupling (Vangheluwe, 2001), the coupled DEVS model N is equivalent to an atomic DEVS model $M =< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$ (the construction of M can be found in Vangheluwe (2001)). The sequential state of M (equivalent to the coupled DEVS model N) can be represented as

$$s = (\dots, (s_i, e_i), \dots) \in S = \times_{i \in D} Q_i \tag{3.1}$$

where $Q_i = \{(s_i, e_i) | s_i \in S_i, 0 \le e_i \le ta_i(s_i)\}$. The state evolution of the coupled DEVS model is triggered by either an internal state transition of the selected imminent component i^* (Vangheluwe, 2001) which transforms the different parts of the total state as follows:

$$\begin{aligned}
&\delta_{int}(s) = (\dots, (s'_i, e'_i), \dots) \\
&\text{where } (s'_i, e'_i) = \begin{cases} (\delta_{int,i}(s_i), 0) & \text{if } i = i^* \\ (\delta_{ext,i}(s_i, e_i + ta(s), Z_{i^*,i}(\lambda_{i^*}(s_{i^*}))), 0) & \text{if } i \in I_{i^*} \\ (s_i, e_i + ta(s)) & \text{otherwise} \end{cases} \\
&\text{where } ta(s) = min\{\sigma_i = ta_i(s_i) - e_i | i \in D\}
\end{aligned}$$

or an external state transition which transforms the different parts of the total state as follows:

$$\begin{aligned}
&\delta_{ext}(s, e, x) = (\dots, (s'_i, e'_i), \dots) \\
&\text{where } (s'_i, e'_i) = \begin{cases} (\delta_{ext,i}(s_i, e_i + e, Z_{N,i}(x)), 0) & \text{if } i \in I_N \\ (s_i, e_i + e) & \text{otherwise} \end{cases}
\end{aligned}$$

3.2.1.2 Formalize discrete event state evolution as integer indexed state process

In order to formalize the data assimilation problem, we need to formalize the state transitions in a DEVS model as an integer indexed state process:

$$\begin{aligned}
x_{\tilde{k}} &= (s_{\tilde{k}}, t_{\tilde{k}}) \\
&= ((\dots, (s_{i,\tilde{k}_i}, e_{i,\tilde{k}_i}), \dots), t_{\tilde{k}}), i \in D; \tilde{k} = 0, 1, 2, \dots; \tilde{k}_i = 0, 1, 2, \dots
\end{aligned} \tag{3.2}$$

where $s_{\tilde{k}} \in S$ is a sequential state of a coupled DEVS model as defined in equation 3.1, and $t_{\tilde{k}} \in \mathbb{R}^+_{0,\infty}$ is the time instant when the model transfers to state $s_{\tilde{k}}$, and we assign $t_0 = 0$. $s_{i,\tilde{k}_i} \in S_i$ is the sequential state of component $i \in D$; $e_{i,\tilde{k}_i} = t_{\tilde{k}} - t_{i,\tilde{k}_i}$ is the time elapsed since component i made a state transition to state $s_{i,\tilde{k}_i} \in S_i$ at time $t_{i,\tilde{k}_i} \in \mathbb{R}^+_{0,\infty}$. Essentially $x_{i,\tilde{k}_i} = (s_{i,\tilde{k}_i}, t_{i,\tilde{k}_i})$ also defines an integer indexed state process for atomic DEVS component $i \in D$. Since state evolutions of different components are again asynchronous with each other, the state index is different from component to component at the same time, therefore, the state index $\tilde{k}$ is associated with the component index i, i.e., $\tilde{k}_i$. Obviously, $\forall t_{\tilde{k}}, \exists i \in D, s.t.\ t_{\tilde{k}} = t_{i,\tilde{k}_i}$, which means that a coupled model takes a new state value when one of its atomic components has a state update. The integer indexed state process is illustrated in Figure 3.2.

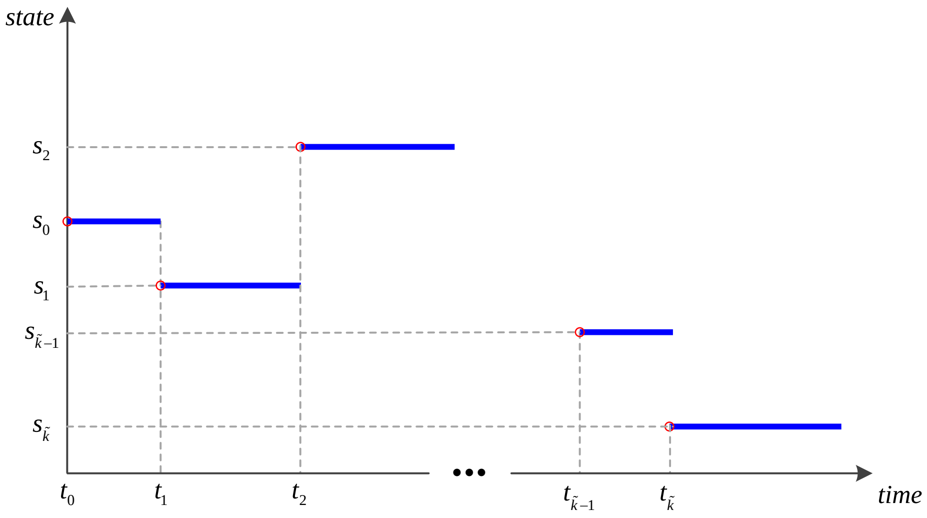


Figure 3.2: The integer indexed state process (each red circle represents a state point $x_{\tilde{k}} = (s_{\tilde{k}}, t_{\tilde{k}})$)

We denote the input event segment for the coupled DEVS model as $w : (t_{\tilde{k}}, t_{\tilde{k}} + ta(s_{\tilde{k}})] \rightarrow X^{\emptyset} = X \cup \{\emptyset\}$, where $ta(s_{\tilde{k}}) = ta_{i^*}(s_{i^*,\tilde{k}_{i^*}}) - e_{i^*,\tilde{k}_{i^*}} = min\{\sigma_{i,\tilde{k}_i} = ta_i(s_{i,\tilde{k}_i}) - e_{i,\tilde{k}_i} | i \in D\}$, i.e., i^* is the selected imminent component. Based on $x_{\tilde{k}} = (s_{\tilde{k}}, t_{\tilde{k}})$ and the input segment w, the next state $x_{\tilde{k}+1} = (s_{\tilde{k}+1}, t_{\tilde{k}+1})$ is defined as follows:

- if there is no external event during $(t_{\tilde{k}}, t_{\tilde{k}}+ta(s_{\tilde{k}})]$, i.e., $\nexists t \in (t_{\tilde{k}}, t_{\tilde{k}}+ta(s_{\tilde{k}})],\ s.t.\ w(t) \neq \emptyset$, $x_{\tilde{k}+1} = (s_{\tilde{k}+1}, t_{\tilde{k}+1})$ is determined as

$$\begin{aligned} s_{\tilde{k}+1} &= \delta_{int}(s_{\tilde{k}}) = (\ldots, (s_{i,\tilde{k}'_i}, e_{i,\tilde{k}'_i}), \ldots) \\ t_{\tilde{k}+1} &= t_{\tilde{k}} + ta(s_{\tilde{k}}) \end{aligned} \tag{3.3}$$

  where $(s_{i,\tilde{k}'_i}, e_{i,\tilde{k}'_i})$ is defined as

$$(s_{i,\tilde{k}'_i}, e_{i,\tilde{k}'_i}) = \begin{cases} (\delta_{int,i}(s_{i,\tilde{k}_i}), 0) = (s_{i,\tilde{k}_i+1}, 0) & \text{if } i = i^* \\ (\delta_{ext,i}(s_{i,\tilde{k}_i}, e_{i,\tilde{k}_i} + ta(s_{\tilde{k}}), Z_{i^*,i}(\lambda_{i^*}(s_{i^*,\tilde{k}_{i^*}}))), 0) = (s_{i,\tilde{k}_i+1}, 0) & \text{if } i \in I_{i^*} \\ (s_{i,\tilde{k}_i}, e_{i,\tilde{k}_i} + ta(s_{\tilde{k}})) & \text{otherwise} \end{cases}$$

- if there exists external events during $(t_{\tilde{k}}, t_{\tilde{k}}+ta(s_{\tilde{k}})]$, i.e., $\exists t \in (t_{\tilde{k}}, t_{\tilde{k}}+ta(s_{\tilde{k}})],\ s.t.\ w(t) \neq$

$\emptyset \cap \nexists t' \in (t_{\tilde{k}}, t),\ s.t.\ w(t') \neq \emptyset$, $x_{\tilde{k}+1} = (s_{\tilde{k}+1}, t_{\tilde{k}+1})$ is determined as

$$\begin{aligned} x_{\tilde{k}+1} &= \delta_{ext}(s_{\tilde{k}}, t - t_{\tilde{k}}, w(t)) = (\dots, (s_{i,\tilde{k}'_i}, e_{i,\tilde{k}'_i}), \dots) \\ t_{\tilde{k}+1} &= t \end{aligned} \tag{3.4}$$

where $(s_{i,\tilde{k}'_i}, e_{i,\tilde{k}'_i})$ is defined as

$$(s_{i,\tilde{k}'_i}, e_{i,\tilde{k}'_i}) = \begin{cases} (\delta_{ext,i}(s_{i,\tilde{k}_i}, e_{i,\tilde{k}_i} + t - t_{\tilde{k}}, Z_{N,i}(w(t))), 0) = (s_{i,\tilde{k}_i+1}, 0) & \text{if } i \in I_N \\ (s_{i,\tilde{k}_i}, e_{i,\tilde{k}_i} + t - t_{\tilde{k}}) & \text{otherwise} \end{cases}$$

Finally, we can formalize the state evolution of a coupled DEVS model as an integer indexed state process

$$x_{\tilde{k}+1} = SIM(x_{\tilde{k}}, w) + \nu_{\tilde{k}}, \tilde{k} = 0, 1, 2, \dots \tag{3.5}$$

where w is the input event segment, and SIM is a discrete event simulation model which transfers state $x_{\tilde{k}}$ to $x_{\tilde{k}+1}$ based on equation 3.3 and equation 3.4; $\nu_{\tilde{k}}$ is the process noise. Notice that the time duration between two consecutive state points, i.e., $t_{\tilde{k}+1} - t_{\tilde{k}}$, is not a constant, but a random variable. In this chapter, we focus on closed systems, therefore, $w = \emptyset$.

3.2.2 Measurement model

The (discrete time) measurement model relates noisy observations to the system state:

$$m_k = g_k(s_k) + \varepsilon_k, k = 1, 2, \dots \tag{3.6}$$

where ε_k is the measurement noise. Notice that the measurement process is assumed to feed data at fixed times, i.e. every ΔT time units, therefore the time of the measurement process can be represented as an integer k (the corresponding simulation time is $t = k\Delta T$; see Figure 3.3). The state points in the discrete event state process can also be indexed by an integer $\tilde{k}$ (see section 3.2.1.2), but since these state points lie irregularly on the continuous axis, we need to explicitly represent the time instants (i.e. $t_{\tilde{k}}$ which is a continuous variable) when the system transfers to these states.

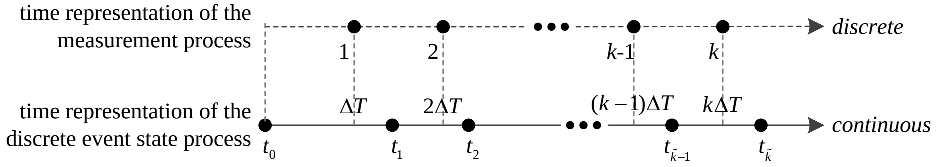

time representation of the measurement process
1
2
k-1
k
discrete
time representation of the discrete event state process
ΔT
2ΔT
(k − 1)ΔT
kΔT
continuous
t_0
t_1
t_2
$t_{\tilde{k}-1}$
$t_{\tilde{k}}$

Figure 3.3: Time representation of the discrete event state process (each black dot indicates a state update) and the (discrete time) measurement process (each black dot represents an arrival of a measurement)

In a discrete event simulation, the state values are only updated when events happen. As shown in Figure 3.1, if we directly retrieve the model state at a time instant t, the retrieved value will be a combination of sequential states of all atomic components, which were updated at past time instants. To get updated (thus more accurate) state value at a time instant t, we need to consider the time elapsed since the model transfers to the current (sequential) state as well. Therefore we introduce an interpolation operation to obtain updated state value, which infers the state value at a time instant t based on the states lying around that time (i.e., neighborhood of t). How many states are involved in the interpolation is determined by the interpolation method we use. In the measurement model, the time is represented by an integer k, therefore, we define how to obtain the state value at time k (i.e. $k\Delta T$) given the integer indexed state process $x_{\tilde{k}}$. To this end, we first define a neighborhood of states around time k:

$$x_{\mathcal{N}_k} = \{x_{\tilde{k}}, \tilde{k} \in \mathcal{N}_k(x_{0:\infty})\}$$

where $x_{0:\infty} = \{x_i, i = 0, 1, 2, \dots\}$ is a sequence of state points defined in equation 3.2; $\mathcal{N}_k(x_{0:\infty})$ defines a set of indexes of states that are required for the interpolation operation in order to compute the state at time k. For example, in Figure 3.3, if we use the linear interpolation, $\mathcal{N}_k(x_{0:\infty}) = \{\tilde{k} - 1, \tilde{k}\}$. Then we can compute an updated state by interpolation: $\hat{s}_k = interpolate(x_{\mathcal{N}_k})$. Based on $\hat{s}_k$, we can now formalize the measurement model between $\hat{s}_k$ and m_k:

$$m_k \sim p(m_k|\hat{s}_k) = p(m_k|x_{\mathcal{N}_k}) \tag{3.7}$$

which is just a reformulation of equation 3.6.

In this research, we want to generalize the measurement model to include situations where measurements are dependent on the state trajectory, i.e., the history of state transitions, which means that m_k will contain observations that are distributed over the last measurement interval $[(k-1)\Delta T, k\Delta T]$. This assumption holds in many applications, such as vehicle passages (event data) collected at a loop detector during one minute (Wu and Liu, 2014). In this case, the measurement m_k is not only related to a specific state at a time instant, but also related to a sequence of states over a period of time. Therefore, we define a generalized form of measurement model

$$m_k \sim p(m_k|x_{\mathcal{N}^+_{k-1}+1:\mathcal{N}^+_k}) \tag{3.8}$$

where $\mathcal{N}^+_k = max\{i \in \mathcal{N}_k\}$, and $x_{\mathcal{N}^+_{k-1}+1:\mathcal{N}^+_k}$ represents a sequence of states which are indexed from $\mathcal{N}^+_{k-1} + 1$ to $\mathcal{N}^+_k$.

3.2.3 State estimation using particle filters

Consider a discrete event system with sensors deployed to monitor its operation. The measurement fed at time k, i.e., m_k, contains the partial observations of the system collected during the last measurement interval $[(k-1)\Delta T, k\Delta T]$. We are interested in the conditional distribution of the state trajectory $x_{0:\mathcal{N}^+_k}$ given all measurements, i.e., $p(x_{0:\mathcal{N}^+_k}|m_{1:k})$. Based on Bayes' theorem, $p(x_{0:\mathcal{N}^+_k}|m_{1:k})$ can be expressed as

$$p(x_{0:\mathcal{N}^+_k}|m_{1:k}) = \frac{p(x_{0:\mathcal{N}^+_k})p(m_{1:k}|x_{0:\mathcal{N}^+_k})}{p(m_{1:k})} \tag{3.9}$$

Similarly, we have $p(x_{0:\mathcal{N}_{k-1}^+}|m_{1:k-1}) = \dfrac{p(x_{0:\mathcal{N}_{k-1}^+})p(m_{1:k-1}|x_{0:\mathcal{N}_{k-1}^+})}{p(m_{1:k-1})}$. Then we have $\dfrac{p(x_{0:\mathcal{N}_k^+}|m_{1:k})}{p(x_{0:\mathcal{N}_{k-1}^+}|m_{1:k-1})} = \dfrac{p(m_k|x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+})p(x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+}|x_{\mathcal{N}_{k-1}^+})}{p(m_k|m_{1:k-1})}$. Consequently, we can obtain a sequential update equation

$$\begin{aligned} p(x_{0:\mathcal{N}_k^+}|m_{1:k}) &= \frac{p(m_k|x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+})p(x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+}|x_{\mathcal{N}_{k-1}^+})}{p(m_k|m_{1:k-1})} \times p(x_{0:\mathcal{N}_{k-1}^+}|m_{1:k-1}) \\ &\propto p(m_k|x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+})p(x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+}|x_{\mathcal{N}_{k-1}^+}) \times p(x_{0:\mathcal{N}_{k-1}^+}|m_{1:k-1}). \end{aligned} \tag{3.10}$$

This sequential update equation is similar in form to that in equation 2.8, but an important difference here is that $\mathcal{N}_k^+$ is a random variable, which means that the dimension of $x_{0:\mathcal{N}_k^+}$ is also random. The variable dimension problem will lead to inapplicability of the standard sequential importance sampling algorithm (see section 2.2.3) (Godsill and Vermaak, 2005; Godsill et al., 2007).

In Godsill et al. (2007), the authors proposed a solution to solve the variable dimension problem. Instead of estimating $p(x_{0:\mathcal{N}_k^+}|m_{1:k})$ directly, they estimate $p(x_{0:K}|m_{1:k})$ where $x_{0:K}$ consists of two segments: $x_{0:\mathcal{N}_k^+}$ (our interest) and $x_{\mathcal{N}_k^+ +1:K}$ (extension). K is a sufficient large constant integer such that for every k, the neighborhood $x_{\mathcal{N}_k}$ is complete. If $x_{\mathcal{N}_k}$ contains all state points that are required for interpolation at time k, we say that $x_{\mathcal{N}_k}$ *is complete*. Since $x_{0:K}$ has fixed dimension, the standard sequential importance sampling algorithm can be applied. Once samples from joint distribution $p(x_{0:K}|m_{1:k})$ are available, samples from its marginal $p(x_{0:\mathcal{N}_k^+}|m_{1:k})$ can be obtained from the original joint samples by simply discarding the components (i.e., $x_{\mathcal{N}_k^+ +1:K}$) that are not of interest, and retaining the original weights. Finally, the weight is updated by

$$\begin{aligned} w_k &= \frac{p(x_{0:K}|m_{1:k})}{q(x_{0:K}|m_{1:k})} \\ &\propto \frac{p(m_k|x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+})p(x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+}|x_{\mathcal{N}_{k-1}^+})}{q(x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+}|x_{0:\mathcal{N}_{k-1}^+}, m_{1:k})} \times w_{k-1} \end{aligned} \tag{3.11}$$

where $q(\cdot)$ is the importance density. The weight update is independent of states $x_{\mathcal{N}_k^+ +1:K}$, and as a result, the extension $x_{\mathcal{N}_k^+ +1:K}$ is never generated in practice. More detailed proof can be found in Godsill and Vermaak (2005); Godsill et al. (2007).

Suppose we have a large number N_p of weighted samples $\chi_{k-1} = \{x^i_{0:\mathcal{N}_{k-1}^+}, w^i_{k-1}\}_{i=1}^{N_p}$, which approximate the posterior distribution $p(x_{0:\mathcal{N}_{k-1}^+}|m_{1:k-1})$ at previous time step; when new measurement m_k is available, samples $\chi_k = \{x^i_{0:\mathcal{N}_k^+}, w^i_k\}_{i=1}^{N_p}$ which approximate the posterior distribution $p(x_{0:\mathcal{N}_k^+}|m_{1:k})$ at time k can be obtained by Algorithm 1.

Algorithm 1: A generic particle filter for discrete event simulations

```
1  % initialization of particles at $k = 0$
2  for $i = 1 : N_p$ do
3      generate the $i$-th sample $x_0^i = (s_0^i, t_0^i)$, where $s_0^i \sim p(s_0)$ ($p(s_0)$ is the
         probability distribution of the initial state), and $t_0^i = 0$
4      set weight $w_0^i = 1/N_p$
5  end
6  % the sampling step for any time $k \geq 1$
7  for $i = 1 : N_p$ do
8      sample particles according to the importance density $q(\cdot)$:
           • set $j = {\mathcal{N}_{k-1}^{+}}^{i}$
           • while ${\mathcal{N}_k}^i$ is incomplete:
               – set $j = j + 1$
               – sample $x_j^i \sim q(x_j | x_{0:j-1}^i, m_{1:k})$
       set ${\mathcal{N}_k^{+}}^{i} = j$, and append the newly generated states to particle:
         $x_{0:\mathcal{N}_k^+}^i = (x_{0:\mathcal{N}_{k-1}^+}^i, x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+}^i)$, where $\mathcal{N}_0^+ \equiv 0$
       update weight:
           $$w_k^i \propto \frac{p(m_k | x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+}^i) p(x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+}^i | x_{\mathcal{N}_{k-1}^+}^i)}{q(x_{\mathcal{N}_{k-1}^+ +1:\mathcal{N}_k^+}^i | x_{0:\mathcal{N}_{k-1}^+}^i, m_{1:k})} \times w_{k-1}^i$$
9  end
10 normalize the weights, such that $\sum_{i=1}^{N_p} w_k^i = 1$
11 % the resampling step
12 resample particles $\{x_{0:\mathcal{N}_k^+}^i, w_k^i\}_{i=1}^{N_p}$ based on the chosen resampling method, which
     can be found in Douc et al. (2005).
```

3.2.4 Practical remarks

3.2.4.1 The sampling procedure

As shown in Algorithm 1, once $\mathcal{N}_k^+$ is complete, one can stop generating new state points. This stopping condition is quite straightforward to check in simple models, e.g., equation based model. However, in discrete event simulations which involve large number of interacting components, this stopping condition is not easy to capture since the model is separated from its simulator. One possible solution is to put a little more effort on modeling by adding certain attributes which can make the interpolation operation conducted at a time instant independent of the states beyond that time. This solution is reasonable since the causal relationship should be obeyed in the modeling process, which means that the current state should not be influenced by events which will happen in the future. For example in the gold mine case which will be studied in subsequent sections, we have a speed attribute for moving entities; as a consequence when we need to get entity position at a time instant, we only need the last updated state (contains speed and location) and the elapsed time to fulfill linear interpolation in order to get updated entity positions.

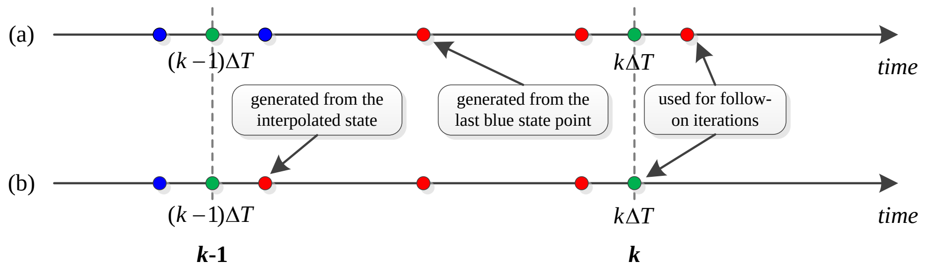

(a)
$(k-1)\Delta T$
$k\Delta T$
time
generated from the interpolated state
generated from the last blue state point
used for follow-on iterations
(b)
$(k-1)\Delta T$
$k\Delta T$
time
k-1
k

Figure 3.4: The state points generation process

The two state generation processes are compared in Figure 3.4, where Figure 3.4(a) shows the state generation process for Algorithm 1, and Figure 3.4(b) illustrates the state generation process when the interpolation operation at time k is independent of state points beyond that time. In Figure 3.4, the state points generated from previous iteration (i.e., $k-1$) are represented in blue, the newly generated state points in current iteration (i.e., k) are depicted in red, and the interpolated states are shown in green. In Figure 3.4(a), the size of $\mathcal{N}_k$ is 2, i.e., we need the two state points lying in the left and right side of k to interpolate; while in Figure 3.4(b), the size of $\mathcal{N}_k$ is 1, since we only need one state point which lies in the left side of k and the elapsed time to fulfill interpolation. The benefit of the state generation process in Figure 3.4(b) is that we do not need to check the stopping condition any more, and we can simply stop state generation (e.g., the simulation execution) at time k and all information is already sufficient for interpolation. In follow-on iterations, new states will then be generated from the interpolated state. In such a case, the sequential update rule in equation 3.10 will be simplified to

$$\begin{aligned} p(x_{0:\mathcal{N}_k^+}|m_{1:k}) &= p(s_{0:k}|m_{1:k}) \\ &\propto p(m_k|s_{k-1:k})p(s_{k-1:k}|\hat{s}_{k-1})p(s_{0:k-1}|m_{1:k-1}) \end{aligned} \tag{3.12}$$

where the partial state trajectory $s_{k-1:k}$ and the full state trajectory $s_{0:k}$ are defined as follows:

$$\begin{aligned} s_{k-1:k} &= \{s_{\tilde{k}}|x_{\tilde{k}} = (s_{\tilde{k}}, t_{\tilde{k}}) \cap (k-1)\Delta T \leq t_{\tilde{k}} \leq k\Delta T\} \cup \{\hat{s}_k\} \\ s_{0:k} &= s_{0:k-1} \cup s_{k-1:k} \end{aligned} \tag{3.13}$$

The weight update in equation 3.11 will thus be modified to

$$w_k = \frac{p(x_{0:K}|m_{1:k})}{q(x_{0:K}|m_{1:k})} \propto \frac{p(m_k|s_{k-1:k})p(s_{k-1:k}|\hat{s}_{k-1})}{q(s_{k-1:k}|s_{0:k-1}, m_{1:k})} \times w_{k-1} \tag{3.14}$$

3.2.4.2 Generating initial particles

Generating initial particles boils down to generating initial model states. For a discrete event simulation model, we cannot generate its initial state arbitrarily (i.e. cannot generate the initial state of each atomic model independently), since an arbitrary combination (of atomic model states) might be infeasible in reality. For example, in the gold mine case which will be studied in subsequent sections, if we generate initial states arbitrarily, we might generate a system state which indicates that the miner is drilling while no trucks are present. Therefore, initial states should be generated from a set of feasible combinations of atomic model states. We denote this set as $FS \subseteq \times_{i \in D} S_i$, where D is the set of names of components of the discrete event model (i.e. a coupled DEVS model), and S_i is the set of sequential states of component i. We denote the combination of initial states of all atomic components as a random variable S_0, and it should take value from FS. Since S_0 is a discrete random variable, we formalize its probability distribution as

$$P(S_0 = s_0^j) = p_j, s_0^j \in FS, p_j \in (0,1) \tag{3.15}$$

and $\sum_{j=1}^{|FS|} p_j = 1$. Notice that $s_0^j = (\dots, s_{0,i}^j, \dots), i \in D, s_{0,i}^j \in S_i$.

Based on the discrete probability distribution in equation 3.15, generating an initial model state is done as follows:

- Generate a feasible combination of initial states of all atomic components: $s_0^j = (\dots, s_{0,i}^j, \dots) \in FS, i \in D, s_{0,i}^j \in S_i$, by sampling the discrete probability distribution $P(S_0)$.
- Compute time advance for each sequential state $s_{0,i}^j$, and the computed value is denoted as $ta(s_{0,i}^j)$.
- If $ta(s_{0,i}^j) < +\infty$, generate an elapsed time e_i for each sequential state $s_{0,i}^j$. Since $0 \leq e_i \leq ta(s_{0,i}^j)$, we can generate the elapsed time by sampling from certain continuous probability distribution (lying within $[0, ta(s_{0,i}^j)]$) which is deemed appropriate. For example, we can sample from a Normal distribution $\mathcal{N}(0,1)$ which lies within $[0, ta(s_{0,i}^j)]$, i.e. $e_i \sim \mathcal{N}(0,1) \bigcap e_i \in [0, ta(s_{0,i}^j)]$. In the initialization, the next state update (i.e. transfer from $s_{0,i}^j$ to another sequential state) is scheduled at $t_{now} + ta(s_{0,i}^j) - e_i = ta(s_{0,i}^j) - e_i$ (in the initialization, $t_{now} = 0$). If $ta(s_{0,i}^j) = +\infty$, i.e. atomic component i is passive, we set $e_i = 0$.
- Finally, combining the generated sequential state $s_{0,i}^j$, the time advance $ta(s_{0,i}^j)$, and the elapsed time e_i for each atomic component $i \in D$, we can initialize the discrete event simulation model.

3.3 Case study – estimating truck arrivals in a gold mine system

In this section and subsequent sections, we study a case in a gold mine system, to illustrate the working of the particle filter based data assimilation framework introduced in section 3.2. In this section, we focus on how to tailor the generic data assimilation framework to the specific estimation problem in the gold mine system.

3.3.1 Scenario description

A gold mine system is shown in Figure 3.5, and its operation is based on the coordination among miners, two trucks, and an elevator:

- miners drill at the mine shaft end, and they can only drill when an empty truck is present. Loading a truck varies very much. Creating a full truckload takes minimally 15 minutes, maximally 30 minutes.
- two trucks are available to transport ore; each truck travels 5 km/h when full through the mine shaft, and 10 km/h when empty. The current mine shaft is 400 meters long.
- an elevator can take a batch of gold ore up. The depth of the elevator shaft is 100 meters; it takes the elevator 8 minutes to go up with ore, and 3 minutes to go down empty.

When a truck is full, the miners ask the elevator to come down, so it will be at the bottom of the vertical shaft when the full truck arrives. When a truck of ore arrives at the bottom of the vertical shaft, it needs to be unloaded from the truck before the elevator can go up. Unloading takes between 5 and 10 minutes. After that, the elevator can go up, and the truck can go back. Unloading at the top of the vertical shaft takes between 2 and 4 minutes before the load can be put on a 100-meter long conveyor belt that transports the gold ore to a processing plant. The conveyor belt has a speed of 0.6 km/h.

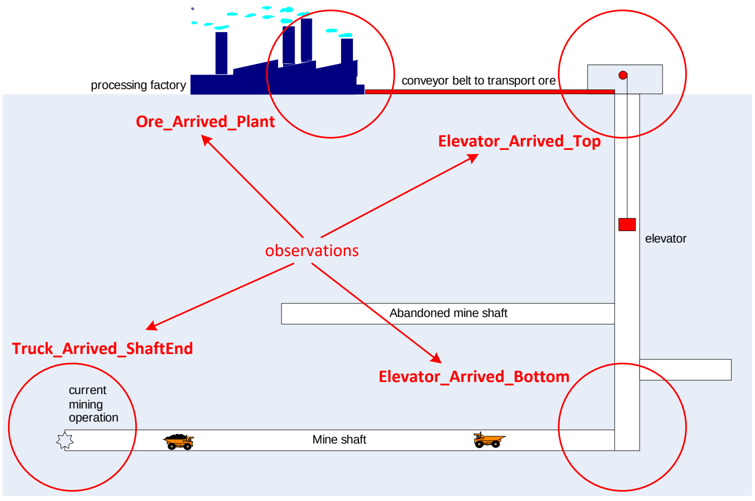


Figure 3.5: The gold mine system

The gold mine is monitored by multiple sensors, which can provide partial observations of the gold mine system (the detailed available data will be explained in section 3.3.4). The problem is that: given these partial observations, can we estimate when the trucks arrive at the bottom of the vertical shaft? The arrival information is important for efficient operation of the elevator, which may improve the overall performance of the gold mine system.

3.3.2 Modeling the gold mine system in the DEVS formalism

The scenario described in section 3.3.1 is a typical discrete event system, therefore we model it using the DEVS formalism (Zeigler et al., 2000), as shown in Figure 3.6. Notice that the gold mine simulation model has no external inputs. We model each component into different phases (Honig and Seck, 2012), and each phase has a name and a life time, where the name indicates the activity that the component is undergoing, and the life time tells how long the entity will stay in that phase. The phases and associated parameters (i.e. state variables) of several key components (i.e. Miner, Truck, and Elevator) are listed in Table 3.1, while other components (such as Queue, Conveyor, Observer) are quite simple, therefore we do not describe them in detail due to space limitations.

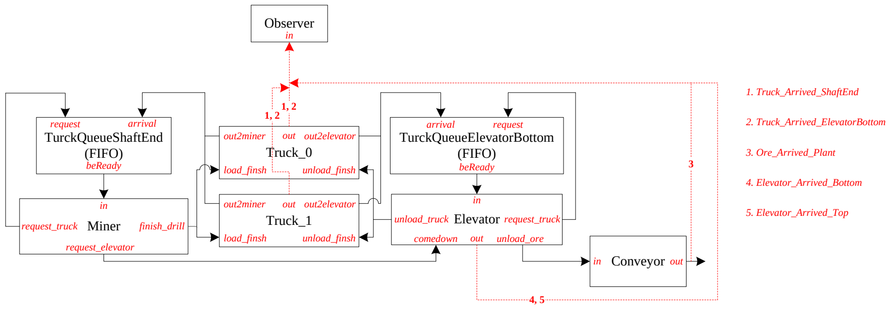


Figure 3.6: The DEVS model of the gold mine system

Table 3.1: State variables of key components in the DEVS gold mine model

component type	phases	parameters	description
Miner	TRANSIENT_PHASE HAVE_REQUEST DRILLING	serving_truck	the truck that is being loaded
Truck	TRAVEL_TO_MINER TRAVEL_TO_ELEVATOR TRANSIENT_PHASE WAITING	pos v	the position of the truck the velocity of the truck
Elevator	IDLE_AT_TOP GO_DOWN_EMPTY TRANSIENT_PHASE HAVE_REQUEST UNLOAD_TRUCK_AT_BOTTOM GO_UP_WITH_ORE UNLOAD_ORE_AT_TOP	pos v serving_truck hasUnprocessedRequest	the position of the elevator the velocity of the elevator the truck that is being unloaded if there is any unprocessed request from miner

As shown in Table 3.1, each component has a transient phase, i.e. TRANSIENT_PHASE, which has zero length of life time and is used to request resources or jobs. For example, when Miner finishes drilling and loading, it will first make a transition from DRILLING to TRANSIENT_PHASE; since TRANSIENT_PHASE has zero length of life time, a message is immediately sent to TruckQueueShaftEnd to tell that Miner is idle and can drill & load other trucks if there is any; then Miner transfers to HAVE_REQUEST (i.e, idle) to wait for new trucks. Truck and Elevator work in a similar way. The movement of the elevator and the trucks is assumed with constant speed (though not realistic).

The unloading times at the bottom and the top of the vertical shaft are modeled as Uniform distribution $U(5.0, 10.0)$ and Uniform distribution $U(2.0, 4.0)$, respectively. The drilling time of the Miner is modeled as a Triangular distribution with varying mode (shown in Figure 3.7). The purpose of varying mode is to simulate miners' tiredness, which means that miners can become tired, i.e., the longer time they has been working, the longer time they spend to load a truck. In the beginning ($t = t_s$), the mode $c = c_{t_s}$; while in the end ($t = t_e$), the mode will increase to $c = c_{t_e}$; at any time instants $t_1, t_2 \in (t_s, t_e)$, if $t_1 < t_2$, we have $c_{t_1} < c_{t_2}$. In our simulation, the run length is 480, therefore, we set $a = 15, b = 30, t_s = 0, t_e = 480, c_{t_s} = a + \frac{1}{4}(b-a), c_{t_e} = a + \frac{3}{4}(b-a)$; for any $t \in (t_s, t_e)$, we have $c_t = a + (\frac{1}{4} + \frac{1}{2} \times \frac{t-t_s}{t_e-t_s}) \times (b-a)$. The unit of time is minute.

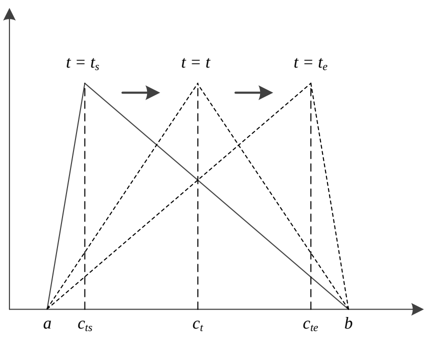


Figure 3.7: Triangular distribution with varying mode

We denote the set of component names as *D* = {*TruckQueueShaftEnd*, *TruckQueueElevatorBottom*, *Miner*, *Truck*_0, *Truck*_1, *Elevator*, *Conveyor*, *Observer*}. For any component $i \in D$, the (sequential) state of component i can be represented as $s_i = \{p_i, \theta_i\}$, where p_i is the phase (name), and θ_i is the corresponding state parameters (variables). Consequently, the sequential state of the gold mine model can be represented as

$$s = (\dots, (s_i, e_i), \dots) \in S = \times_{i \in D} Q_i \tag{3.16}$$

where $Q_i = \{(s_i, e_i) | s_i \in S_i, 0 \leq e_i \leq ta_i(s_i)\}$. Based on the derivation shown in section 3.2.1.2, we can easily formalize the state evolution of the gold mine model as an integer indexed state process (i.e., the system model of the gold mine system):

$$\begin{aligned} x_{\tilde{k}} &= ((\dots, (s_{i,\tilde{k}_i}, e_{i,\tilde{k}_i}), \dots), t_{\tilde{k}}), i \in D \\ x_{\tilde{k}+1} &= GoldMineSim(x_{\tilde{k}}) + \nu_{\tilde{k}}, \tilde{k} = 0, 1, 2, \dots \end{aligned} \tag{3.17}$$

where $GoldMineSim$ is the (discrete event) gold mine simulation model, $\nu_{\tilde{k}}$ is the system noise, such as position uncertainty incurred by small deviations on speed.

3.3.3 Interpolation operation

In this section, we introduce the interpolation method used in our gold mine case, and show the difference between the simulated state trajectory and the interpolated state trajectory.

Considering that discrete state variables cannot be interpolated, we distinguish continuous states from discrete states as shown in Figure 3.1. The implementation of the interpolation operation can be found in section A.2.5.1.

3.3.3.1 Continuous state

Continuous states can be interpolated. We take the elevator as an example, whose (sequential) state is represented as $s = (phase, pos, v)$ (component index is omitted here), where $phase$ is the phase name, pos and v are its position and velocity, respectively. Though the state contains a string-type variable (phase name), we still consider it as a continuous state since our focus is the elevator's movement.

As introduced in section 3.3.2, the elevator moves with constant speed. Therefore, we use linear interpolation to update the elevator's state. Suppose that the last state update was at time t_l due to the occurrence of an internal or external event, and the state was updated to $s(t_l) = (phase_l, pos_l, v_l)$; in that event handler, the next state update was scheduled at time t_n, i.e. $ta(s_l) = t_n - t_l$. Since we have velocity in the state definition, we can obtain the updated state at time $t \in (t_l, t_n)$ based on the state at t_l and the elapsed time e:

$$\begin{aligned} &\hat{s}(t) = interpolate(s(t_l), e) \\ &\text{where} \begin{cases} phase_t & = phase_l \\ pos_t & = pos_l + v_l \times e = pos_l + v_l \times (t - t_l) \\ v_t & = v_l \end{cases} \end{aligned} \tag{3.18}$$

which is independent of the states beyond time t.

3.3.3.2 Discrete state

Discrete states cannot be interpolated. For example, the (sequential) state of the miner is $s = (phase, \text{serving\_truck})$, where $phase$ is the phase name, and serving_truck is the name of the truck which is being loaded. Suppose that the last state update was at time t_l, and the state was updated to $s(t_l)$; in that event handler, the next state update was scheduled at time t_n. Since the discrete state cannot be interpolated, the interpolation operation gives

$$\hat{s}(t) = interpolate(s(t_l), e) = (s(t_l), e) \tag{3.19}$$

where the elapsed time $e = t - t_l$. We still denote $(s(t_l), e)$ as $\hat{s}(t)$, i.e. $(s(t_l), e)$ is equivalent to those continuous states that can be interpolated (e.g., equation 3.18). Since $s(t_l)$ cannot be interpolated, we need an elapsed time e to reflect the state evolution. If the measurement is related to the discrete state, one probably needs the elapsed time to define a measurement model that relates the discrete state to the measurement.

3.3.3.3 Interpolated state

Suppose that the (sequential) state of the coupled model at time instant t_l is $s(t_l) = (\dots, (s_i, e_i), \dots), i \in D$, and $ta(s(t_l)) = min\{\sigma_i = ta_i(s_i) - e_i, i \in D\}$. At any time

$t \in (t_l, t_l + ta(s(t_l)))$, the interpolated state can be represented as

$$\hat{s}(t) = interpolate(s(t_l), e) = (\dots, (s'_i, e'_i), \dots), \text{ where}$$
$$(s'_i, e'_i) = \begin{cases} (interpolate(s_i, e_i), 0) & \text{if } s_i \text{ can be interpolated (see equation 3.18)} \\ (s_i, e_i + t - t_l) & \text{if } s_i \text{ cannot be interpolated (see equation 3.19)} \end{cases} \tag{3.20}$$

Notice that the time advance of state $interpolate(s_i, e_i)$ will be $ta(s_i) - e_i$. In section 3.2.2, when computing $\hat{s}_k, k = 1, 2, \dots,$ we essentially compute $\hat{s}_k = \hat{s}(k\Delta T)$ based on equation 3.20.

3.3.3.4 Simulated state trajectory versus interpolated state trajectory

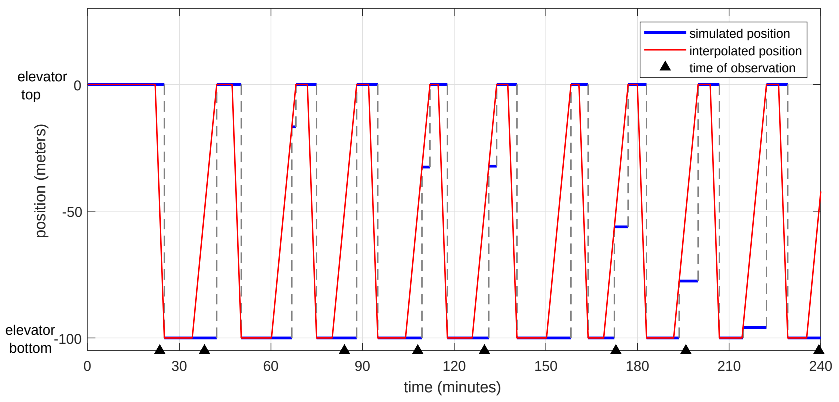
In this section, we show the difference between the simulated state trajectory and the interpolated state trajectory. We take the state of the elevator in terms of position as an example. As shown in Figure 3.8, the positions of the elevator in the discrete event simulation is captured in blue, while the interpolated state trajectory is depicted in red. Since states only change when events occur, the simulated state trajectory of the elevator in terms of position is a piecewise constant curve; while the interpolated state trajectory is a piecewise linear curve since the velocity is constant and we adopt the liner interpolation method.

As explained in the previous section, the elevator moves with constant speed. Therefore, the true state trajectory of the elevator in terms of position is also a piecewise linear curve, which overlaps the interpolated state trajectory. Notice that if the elevator has a different speed profile, for example, accelerate – constant speed – decelerate, the true state trajectory in terms of position and the interpolated state trajectory will not overlap any more. From Figure 3.8, we can clearly see that if we retrieve the state of a discrete event simulation model without interpolation, the retrieved state is only a past state that was updated at a past time instant, which cannot reflect real-time evolutions of the state, therefore, errors would be incurred if the outdated states are used for estimation. This will be proven in section 3.5.

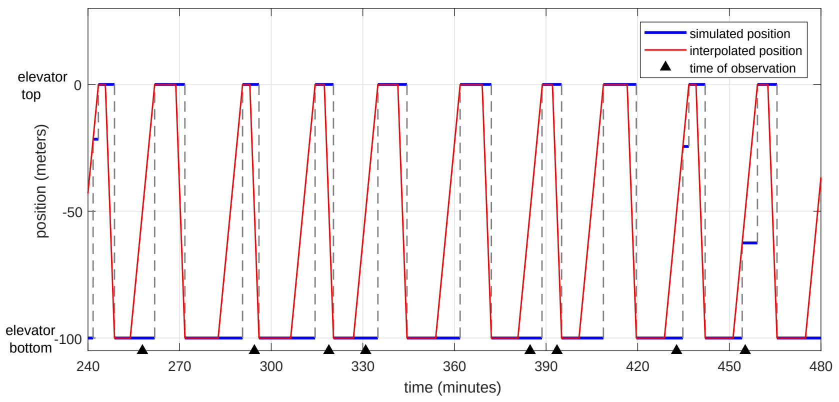
3.3.4 Available data and measurement model

The simulated data is generated by running the gold mine simulation (section 3.3.2) for 480 minutes. During the run, all events are recorded; the states of the elevator and the trucks are sampled (using interpolation) and recorded very densely (every 0.01 minute) in order to obtain their detailed evolutions; the data recorded for the elevator and the trucks includes phase names and their real-time positions. This ground truth data is then processed as follows:

- we extract the event sequence that only contains following types of events (as shown in Figure 3.5): trucks arriving at the shaft end (*Truck_Arrived_ShaftEnd*), the elevator arriving at the top or the bottom of the vertical shaft (*Elevator_Arrived_Top*, *Elevator_Arrived_Bottom*), and a batch of ore arriving at the plant (*Ore_Arrived_Plant*). This event sequence is partial, but accurate (i.e., no missed events, and occurrence times are accurate).



(a) positions between $[0\ min, 240\ min]$



(b) positions between $[240\ min, 480\ min]$

Figure 3.8: The state trajectory of the elevator in terms of position

- we add Gaussian noise to the positions of the elevator and the trucks, respectively; specifically, we add noise drawn from $\mathcal{N}(0, \sigma_e^2)$ for the elevator, and add noise drawn from $\mathcal{N}(0, \sigma_t^2)$ for the trucks.

The noisy dataset is used for data assimilation, and we set the measurement interval to $\Delta T = 30$ minutes. The measurement at time k is denoted as m_k^o, which contains following noisy data collected during $[(k-1)\Delta T, k\Delta T]$:

- event sequence $E_k = \{(t_1, e_1), (t_2, e_2), \ldots, (t_n, e_n)\}, (k-1)\Delta T \leq t_1 \leq t_2 \leq \cdots \leq t_n \leq k\Delta T; e_i \in \{$*Truck_Arrived_ShaftEnd*, *Elevator_Arrived_Top*, *Elevator_Arrived_Bottom*, *Ore_Arrived_Plant*$\}$.

- $PX_k = \{(phase^j(t_j), pos^j(t_j)) | j \in \{Elevator, Truck\_0, Truck\_1\}, t_j \in [(k-1)\Delta T, k\Delta T]\}$, which represents the phase and position of the elevator and the trucks, where $phase^j(t_j)$ indicates the name of the phase of component j at time

t_j, while $pos^j(t_j)$ is the noisy position of component j at time t_j. Notice that during $[(k-1)\Delta T, k\Delta T]$, there is only one observation for each component in $\{Elevator, Truck\_0, Truck\_1\}$; the times of observation for different components are not necessarily the same. As shown in Figure 3.8, the black triangles represent the time instants when noisy observations from the elevator are available. These observation times are randomly chosen, but in order to illustrate the effect of interpolation, we choose time instants when the component (either the elevator or the trucks) is moving, since when components are still, their position does not change, whether interpolate or not has no difference.

To summarize, the measurement available at time k can be represented as

$$m_k^o = \{E_k, PX_k\}, \tag{3.21}$$

and the measurement model can be formalized as

$$m_k^o \sim p(m_k^o | x_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+})$$

where $x_{\tilde{k}} = (s_{\tilde{k}}, t_{\tilde{k}}), \tilde{k} = 0, 1, 2, \dots$ is defined in equation 3.17. As introduced in section 3.3.3, the interpolation operation is independent of states beyond the time instant when the operation is invoked, therefore, the measurement model can be modified to

$$m_k^o \sim p(m_k^o | s_{k-1:k}) \tag{3.22}$$

where $s_{k-1:k} = \{s_{\tilde{k}} | x_{\tilde{k}} = (s_{\tilde{k}}, t_{\tilde{k}}) \cap (k-1)\Delta T \leq t_{\tilde{k}} \leq k\Delta T\} \cup \{\hat{s}_k\}$, and $\hat{s}_k$ is computed based on equation 3.20 ($\hat{s}_k = \hat{s}(k\Delta T)$).

3.3.5 Estimating truck arrivals using particle filters

Having formalized the system model (section 3.3.2) and the measurement model (section 3.3.4), in this section, we implement (on the algorithmic level) the particle filtering framework (section 3.2) in the (discrete event) gold mine simulation to illustrate the working of the framework by estimating the truck arrivals at the bottom of the vertical shaft.

3.3.5.1 Particle filtering for truck arrivals estimation

Algorithm 2 describes in detail how the generic particle filter shown in Algorithm 1 are applied in the specific gold mine case to fulfill the truck arrival estimation task. Since the interpolation operation at any time instant t is independent of states beyond that time, the formalization of Algorithm 2 is focused on system states $s_{\tilde{k}}$, where $x_{\tilde{k}} = (s_{\tilde{k}}, t_{\tilde{k}})$. The main steps of the proposed algorithm are summarized as below.

- **Initialization**. In the initialization step (line 2 $\sim$ 5 in Algorithm 2), the i-th sample x_0^i is actually a guess of possible initial states (i.e., s_0^i) of the gold mine model. The process of generating initial particles is detailed in section 3.3.5.2.

- **Sampling**. In this case, we adopt the system transition density (a reformulation of $GoldMineSim(\cdot)$ in equation 3.17) as the importance density. Therefore, generating state points is done by running the gold mine simulation (line 8 in Algorithm 2). Since the interpolation operation at a time instant t is independent of state points beyond that time (see the explanation in section 3.3.3), we just stop the simulation at time $t = k\Delta T$, and then update its weight based on newly available data m_k^o (line 9 in Algorithm 2); detailed computation of the weight is presented in section 3.3.5.3.

- **Resampling**. To solve the degeneracy problem, we resample the particles using the standard resampling scheme which samples particles in proportion to their weights.

- **Estimation**. We scan the state trajectory $s_{k-1:k}^i$, and record the time instants when event *Truck_Arrived_ElevatorBottom* occurs. Each particle gives an estimation of the truck arrival, and estimations from all particles will form a distribution of truck arrival. These (raw) estimations will be processed to give more informative results in section 3.5.

3.3.5.2 Generating initial particles

In this case study, initial particles are generated based on the procedure introduced in section 3.2.4.2. For illustration purpose, we only enumerate two feasible combinations which are listed in Table 3.2, although there are many more feasible choices. We assume $P(s_0^1) = P(s_0^2) = 0.5$, and generate elapsed time using $\mathcal{N}(0, 1)$ lying within $[0, ta(s_{0,i}^j)], j \in \{1, 2\}, i \in \{$*Miner*, *Truck_0*, *Truck_1*, *Elevator*$\}$. For other atomic components in the gold mine model, i.e. *TruckQueueShaftEnd*, *TruckQueueElevatorBottom*, *Conveyor*, *Observer*, we initialize them as passive (i.e. time advance is $+\infty$).

3.3.5.3 Weight computation

In this section, we detail how the weight is computed, i.e., utilize $w_k^i = p(m_k^o|s_{k-1:k}^i) \times w_{k-1}^i$. The measurement at time k is $m_k^o = \{E_k, PX_k\}$, where E_k is the observed event sequence during time interval $[(k-1)\Delta T, k\Delta T]$, $PX_k = \{(phase^j(t_j), pos^j(t_j))|j \in \{Elevator, Truck\_0, Truck\_1\}, t_j \in [(k-1)\Delta T, k\Delta T]\}$ represents phase and position observations from the elevator and the trucks. Since the two types of observations are conditionally independent given $s_{k-1:k}^i$, we have $p(m_k^o|s_{k-1:k}^i) = p(E_k|s_{k-1:k}^i)p(PX_k|s_{k-1:k}^i)$.

Event sequences

Given state points $s_{k-1:k}^i$, it is very easy to retrieve an event sequence which only contains the four types of events shown in Figure 3.5 (i.e., types of observed events). We denote such event sequence retrieved from the i-th particle as E_k^i, then $p(E_k|s_{k-1:k}^i) = p(E_k|E_k^i)$. Subsequently, we first define a distance measure between two event sequences, and based on the distance measure, we then define $p(E_k|E_k^i)$.

An *event* can be modeled as a two-tuple (t, e), where e is the event type, and t is the occurrence time. An *event sequence* S is an ordered sequence of events:

$$S = \{(t_1, e_1), (t_2, e_2), \ldots, (t_n, e_n)\}, t_1 \leq t_2 \leq \cdots \leq t_n$$

Algorithm 2: The particle filter for truck arrival estimation

```
1  % initialization of particles at $k = 0$
2  for $i = 1 : N_p$ do
3      generate the $i$-th sample $x_0^i = (s_0^i, t_0^i)$ where $t_0^i = 0$
4      set weight $w_0^i = 1/N_p$
5  end
6  % the sampling step for any time $k \geq 1$
7  for $i = 1 : N_p$ do
8      run the gold mine simulation to time $t = k\Delta T$ with initial state $\hat{s}_{k-1}^i$, where
         $\hat{s}_{k-1}^i$ is obtained based on equation 3.20 ($t = (k-1)\Delta T$); the newly
         generated partial state trajectory is
         $s_{k-1:k}^i = \{s_{\tilde{k}}^i | x_{\tilde{k}}^i = (s_{\tilde{k}}^i, t_{\tilde{k}}^i), (k-1)\Delta T \leq t_{\tilde{k}}^i \leq k\Delta T\} \cup \{\hat{s}_k^i\}$; the full state
         trajectory is thus updated to $s_{0:k}^i = (s_{0:k-1}^i, s_{k-1:k}^i)$
9      compute weight: $w_k^i = p(m_k^o | s_{k-1:k}^i) \times w_{k-1}^i$
10 end
11 normalize the weights, denote them as $\{s_{0:k}^i, w_k^i\}_{i=1}^{N_p}$
12 % the resampling step
13 resample $\{s_{0:k}^i, w_k^i\}_{i=1}^{N_p}$ using the standard resampling method which samples
     particles in proportion to their weights; the resampled results are again denoted as
     $\{s_{0:k}^i, w_k^i\}_{i=1}^{N_p}$
14 for $i = 1 : N_p$ do
15     $w_k^i = 1/N_p$
16 end
17 % record data for estimation
18 for $i = 1 : N_p$ do
19     scan $s_{k-1:k}^i$, and record the time instants when event
         Truck_Arrived_ElevatorBottom occurs
20 end
```

We adopt the *edit distance* (Mannila and Ronkainen, 1997) to define the 'distance' between two event sequences. The edit distance satisfies following conditions:

$$\begin{aligned} &d(S,T) \geq 0 \\ &d(S,T) = 0 \text{ if and only if } S = T \\ &d(S,T) = d(T,S) \\ &d(S,T) + d(T,U) \geq d(S,U) \end{aligned}$$

The edit distance defines 'the amount of work that has to be done to convert one sequence to another'. To compute the edit distance, a set of transformation operations and their associated costs are defined:

- $insert(t, e)$ that inserts an event of type e at time t, i.e., insert a two-tuple (t, e) to

Table 3.2: The set (denoted as FS) of feasible combinations of initial states of atomic components

elements in FS	**component name**	**phase**	**parameters**	**value**	**time advance (min)**
s_0^1	Miner	TRANSIENT_PHASE	serving_truck	NULL	0
	Truck_0	TRANSIENT_PHASE	pos v	0 m 0 km/h	0
	Truck_1	TRANSIENT_PHASE	pos v	0 m 0 km/h	0
	Elevator	IDLE_AT_TOP	pos v serving_truck hasUnprocessedRequest	0 m 0 km/h NULL NULL	$+\infty$
s_0^2	Miner	TRANSIENT_PHASE	serving_truck	NULL	0
	Truck_0	TRANSIENT_PHASE	pos v	0 m 0 km/h	0
	Truck_1	TRANSIENT_PHASE	pos v	0 m 0 km/h	0
	Elevator	GO_DOWN_EMPTY	pos v serving_truck hasUnprocessedRequest	-20 m 2 km/h NULL NULL	2.4

the sequence. The cost of $insert(t, e)$ is defined as

$$c(insert(t, e)) = w(e) \propto \frac{1}{occ(e)}$$

where $occ(e)$ is the occurrence rate of a e-type event in a long reference sequence.

- $delete(t, e)$ that deletes a two-tuple (t, e) from the sequence. The cost of $delete(t, e)$ is defined to be the same as that of an $insert$-operation, i.e., $c(delete(t, e)) = w(e)$.
- $move(t, e, t')$ that moves an existing event (t, e) from time t to time t'. The cost of a $move$-operation is defined as

$$c(move(t, e, t')) = v \cdot |t - t'|$$

where v is a constant that should satisfy $v|t - t'| < 2 \cdot w(e), \forall t, t', e$. Otherwise, it is never useful to move an event e, one can always delete and insert an event instead.

Suppose $O = \{o_1, o_2, \ldots, o_n\}$ is an operation sequence that transforms S to T, and the cost of O is defined as

$$c(O) = \sum_{i=1}^{n} c(o_i),$$

then the edit distance between event sequence S and event sequence T is defined as the minimum cost that is needed to transform S to T, i.e.,

$$d(S, T) = min\{c(O_j)|O_j\}$$

where O_j is an arbitrary operation sequence that transforms S to T. Given two event sequences $S = \{(t_1, e_1), (t_2, e_2), \ldots, (t_n, e_n)\}$, and $T = \{(u_1, f_1), (u_2, f_2), \ldots, (u_m, f_m)\}$, dynamic programming is employed to compute $d(S, T)$. Suppose $r(i, j)$ is the minimum cost of operations needed to transform the first i events of S to the first j events of T, then $d(S,T) = r(n,m)$. The base conditions and the recurrence relation for $r(i,j)$ are

$$\begin{aligned} r(0,0) &= 0 \\ r(i,0) &= r(i-1,0) + w(e_i) \\ r(0,j) &= r(0,j-1) + w(f_i) \\ r(i,j) &= min\{r(i-1,j) + w(e_i), r(i,j-1) + w(f_j), r(i-1,j-1) + k(i,j)\} \end{aligned}$$

where

$$k(i,j) = \begin{cases} v \cdot |t_i - u_j|, & \text{if } e_i = f_j \\ w(e_i) + w(f_j), & \text{if } e_i \neq f_j. \end{cases}$$

Once the distance between two event sequences can be computed, we can now define $p(E_k|E_k^i)$ as follows:

$$p(E_k|s_{k-1:k}^i) = p(E_k|E_k^i) = \exp(-\frac{d(E_k, E_k^i)}{d_m}) \tag{3.23}$$

where $d_m = d(E_k, \emptyset)$.

Phases and positions

Given state points $s_{k-1:k}^i$, we can straightforwardly obtain the phase (name) and position of any component at any time based on interpolation explained in section 3.3.3. We denote such phase and position pairs for entities in D_c = {*Elevator*, *Truck*_0, *Truck*_1} as PX_k^i, then $p(PX_k|s_{k-1:k}^i) = p(PX_k|PX_k^i)$.

For phase and position data, we need to consider them as a whole. For example, we assume that the observation from the elevator is $\{$GO_DOWN_EMPTY, $-10\}$; in the first particle, we have $\{$GO_DOWN_EMPTY, $-10\}$, while in the second particle, we have $\{$GO_UP_WITH_ORE, $-10.0\}$. Obviously, the first particle should be assigned a larger weight than the second one given the observation. However, if we do not consider the phase difference, we cannot differentiate the two particles. Therefore we propose a phase match method to define a distance measure for phases.

The phase match method works as follows. Suppose the phase is represented as $\{p_i, \theta_i\}$, where p_i is the name of the phase, and θ_i is the corresponding parameters. The distance between phases is defined based on the phase transition graph shown in Figure 3.9. The phase transition graph is actually a simplified version of the model of the corresponding component. For convenience, we assume that the index of one phase in the two phases that we want to compare is 0, while the index of the other is n, their distance is defined as

$$d(0,n) = min\{\sum_{i=0}^{n-1} d(i,i+1), \sum_{i=n}^{N-1} d(i,i+1) + d(N,0)\}$$

where $d(i,j)$ is the distance between phase i and phase j. The distance function can be defined in many ways, for example, we can define $d(i,i+1)$ as the time that the system stays in phase i before it makes a transition to phase $i+1$. In our case, we choose a simple distance function as $d(i,i+1) = 1$.

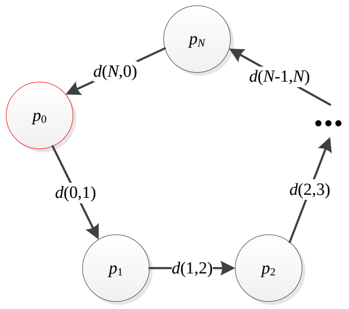


Figure 3.9: The phase transition graph

In our case, the parameter is the position with Gaussian noise, therefore we define $p(PX_k|PX_k^i)$ as follows:

$$p(PX_k|s_{k-1:k}^i) = p(PX_k|PX_k^i) = \prod_{j \in D_c} p(\{phase^j, pos^j\}|\{phase^{i,j}, pos^{i,j}\}) \quad (3.24)$$

where D_c = {*Elevator*, *Truck*_0, *Truck*_1}; $p(\{phase^j, pos^j\}|\{phase^{i,j}, pos^{i,j}\})$ is defined as

$$p(\{phase^j, pos^j\}|\{phase^{i,j}, pos^{i,j}\}) = \begin{cases} max\{p_{min}, \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(pos^{i,j} - pos^j)^2}{2\sigma_j^2}}\} & \text{if } phase^{i,j} = phase^j \\ \frac{p_{min}}{d(phase^{i,j}, phase^j)+1} & \text{if } phase^{i,j} \neq phase^j \end{cases}$$

We argue that the weight of a particle in which the phase is the same with the observed phase (i.e., $phase^{i,j} = phase^j$) should be absolutely larger than that of a particle which has different phase with the observed phase (i.e., $phase^{i,j} \neq phase^j$). Therefore, we define a threshold value p_{min} to guarantee this.

3.4 Case study in the gold mine system – qualitative analysis

In this section, a qualitative analysis is conducted to compare the estimation results without and with assimilating noisy observations; the objective of this comparison is to prove the necessity to assimilate observations into discrete event simulations in order to get better estimation results.

If we do not assimilate noisy observations, we can run the simulation multiple times with different random seeds to generate data for estimation. Therefore, we run the gold mine simulation 2000 times with different random seeds and record the time instants when trucks arrive at the bottom of the vertical shaft. The estimation results are shown in Figure 3.10a. The results show that if there is no real-time data from the real system assimilated, the discrepancy between the simulation and the real system will become larger and larger as time advances. Consequently, the simulation without data assimilation will gradually lose its prediction ability. Based on our example, from $t = 150$ minutes onwards, the gold mine simulation can no longer provide any useful information for truck arrivals at the bottom of the vertical shaft.

In contrast, we use the same simulation model to assimilate the noisy dataset ($\sigma_e = 3.0, \sigma_t = 3.0$) every $\Delta T = 30$ minutes with 2000 particles to estimate the truck arrival times. The estimation results are depicted in Figure 3.10b. The results show that if we assimilate noisy observations into the same simulation model using similar effort (i.e., 2000 particles versus 2000 runs), the simulation can provide reasonable estimations for truck arrivals during the whole simulation period (480 minutes). Therefore it is necessary to assimilate data if there are any into the discrete event simulation in order to obtain better estimation results of the variable of interest.

We present the estimation results of the truck arrival times at the bottom of the vertical shaft in one time step (i.e., $[(k-1)\Delta T, k\Delta T]$) in Figure 3.11. Since the minimal drilling time is 15 minutes, there are at most two arrivals during one time step of duration $\Delta T = 30$ minutes. Notice that the estimation results actually gives a distribution of the truck arrival times. In order to know how accurate the estimation results are and also explore the influences of factors, such as data errors, model errors, and the number of particles employed, in section 3.5.2, we define a set of performance indicators, and conduct quantitative analysis accordingly.

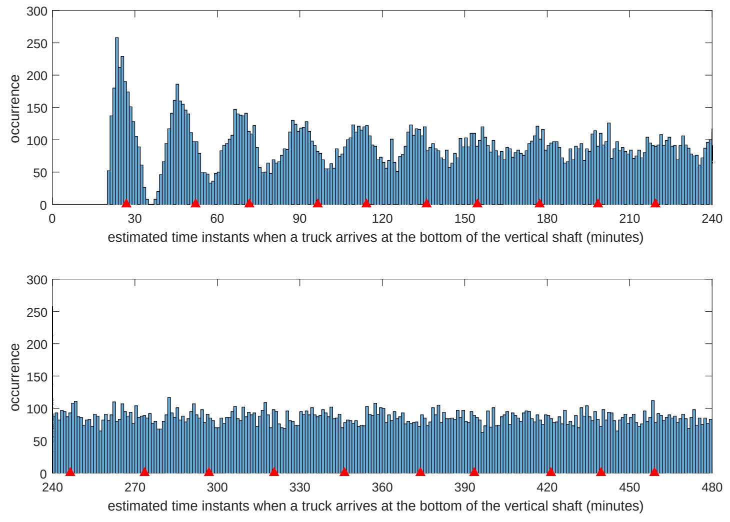
3.5 Case study in the gold mine system – quantitative analysis

The particle filtering method shown in Algorithm 2 gives us raw estimation results of truck arrivals, which are depicted in Figure 3.10b. In this section, we show how these raw data is processed in order to conduct more informative analysis; based on the processed data, a set of performance indicators are proposed to quantify how accurate the estimation results are; finally, results computed based on these performance indicators are presented and analyzed.

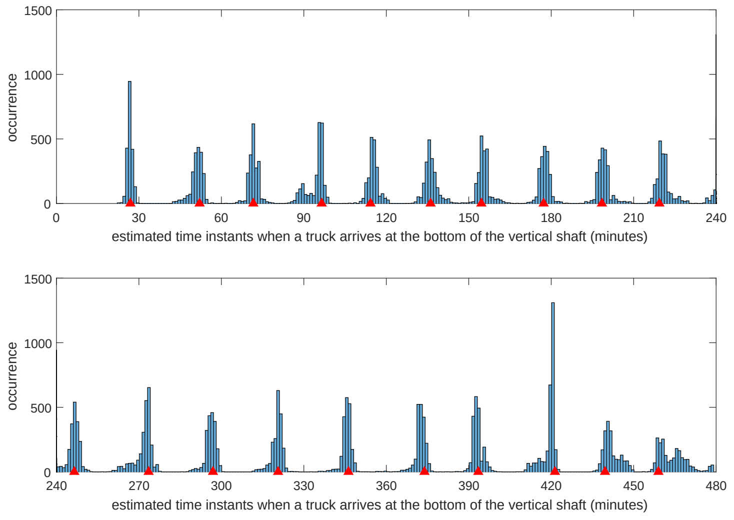
3.5.1 Data processing

3.5.1.1 Data preparation for estimating the dimension of the state trajectory

As explained in previous sections, a big difference of discrete event simulations from discrete time state space models is that in discrete event simulations the dimension of the state trajectory $x_{0:\mathcal{N}_k^+}$, i.e., $\mathcal{N}_k^+$, is a random variable. Since the gold mine simulation model has no external inputs, we can obtain $\mathcal{N}_k^+$ by summing in each atomic DEVS component the number of internal state transitions that happen before time $t = k\Delta T$. This can be done by simply counting the number of corresponding output events ($\lambda(s_i), i \in D$), since output is only possible just before internal state transitions (Zeigler et al., 2000). In Figure 3.12, we show the histogram of $\mathcal{N}_{16}^+$ (i.e. $t = 480$ minutes) estimated by particle filtering (the corresponding ground truth value is 157). The results confirm that in different particles, the number of state points in $x_{0:\mathcal{N}_k^+}$ differs from particle to particle,

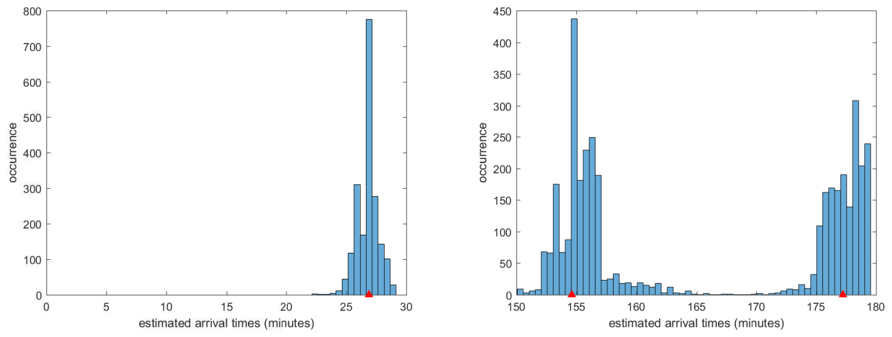


(a) The truck arrivals estimated from 2000 independent runs



(b) The truck arrivals estimated by assimilating the noisy dataset ($\sigma_e = 3.0, \sigma_t = 3.0$) every $\Delta T = 30$ minutes with 2000 particles

Figure 3.10: A general view of the estimation results of truck arrivals at the bottom of the vertical shaft with and without assimilating noisy data (each red triangle represents a truck arrival in ground truth)



(a) Single arrive during $[0\ min, 30\ min]$ (b) Two arrives during $[150\ min, 180\ min]$

Figure 3.11: Histogram of estimated truck arrival times at the bottom of the vertical shaft during one step $[(k-1)\Delta T, k\Delta T]$, where $\Delta T = 30\ min$ (each red triangle represents a truck arrival in ground truth)

and the particle filtering can estimate the distribution (approximated by a histogram) of the dimension.

3.5.1.2 Data preparation for estimating the truck arrivals

As shown in Figure 3.11b, the estimated truck arrival times obviously belong to two groups, each of which approximates the distribution of a truck arrival. Therefore, we cluster the estimated arrival times into groups (for example, using k-means clustering algorithm (Kanungo et al., 2002)), and each group estimates one truck arrival. Suppose that there are m such clusters: $\{C_c | C_c = \{t_1^c, t_2^c, \ldots, t_{n_c}^c\}\}_{c=1}^{m}$; based on the data in each cluster, we can fit a probability distribution of truck arrival times by whatever means. In our case, we fit a kernel distribution using the Normal kernel to the data in each cluster, for example, in Figure 3.13, we show the obtained kernel distribution fitted to the data belonging to the cluster at the right side in Figure 3.11b.

If we denote the fitted probability distribution from data in cluster C_c as $f_c(t)$, and the cumulative distribution function as $F_c(t)$, the probability that a truck arriving at the bottom of the vertical shaft during a very small interval $[t-\varepsilon, t+\varepsilon]$ can be computed as

$$Prob(\text{arriving during } [t-\varepsilon, t+\varepsilon]) = F_c(t+\varepsilon) - F_c(t-\varepsilon),$$

and for convenience, we denote this probability as $P_c(t, \varepsilon)$. $P_c(t, \varepsilon)$ thus represents the probability that a truck arriving at the bottom of the vertical shaft during $[t-\varepsilon, t+\varepsilon]$; the subscript c indicates that the probability is computed from the probability distribution fitted to the data in cluster C_c.

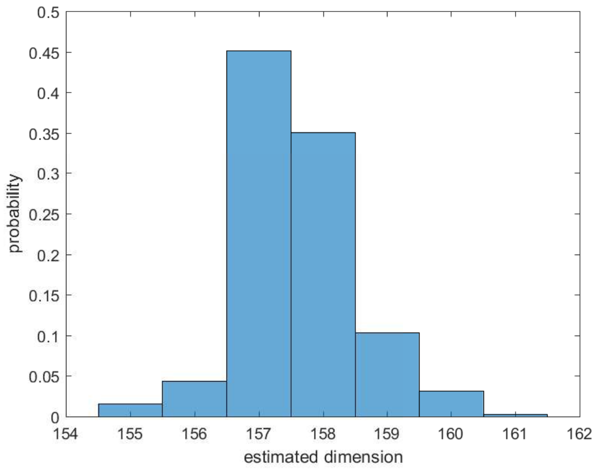


Figure 3.12: The estimated dimension of the state trajectory $x_{0:\mathcal{N}_k^+}$ at time step $k = 16$ (the corresponding ground truth value is 157)

3.5.2 Evaluation criteria

3.5.2.1 Dimension of the state trajectory

Given the distribution (histogram) of $\mathcal{N}_k^+$, we can estimate $\mathcal{N}_k^+$ as follows:

$$\hat{\mathcal{N}}_k^+ = \frac{1}{N_p} \sum_{i=1}^{N_p} \mathcal{N}_k^{+i} \tag{3.25}$$

where N_p is the number of particles, and $\mathcal{N}_k^{+i}$ is the dimension estimated from the i-th particle. The estimation error of the state trajectory dimension can be defined as

$$Err_k = \mathcal{N}_k^+ - \hat{\mathcal{N}}_k^+ \tag{3.26}$$

where $\mathcal{N}_k^+$ is the ground truth value. The *average dimension error* can be defined as

$$\bar{E} = \frac{\Delta T}{span} \sum_{k=1}^{span/\Delta T} |Err_k| \tag{3.27}$$

where $span$ is the simulation run length, and ΔT is the measurement interval.

3.5.2.2 Truck arrivals

Assume that the ground truth value of truck arrivals is $A = \{t_1, t_2, \dots, t_n\}$. After data processing, we obtain m clusters $\{C_c | C_c = \{t_1^c, t_2^c, \dots, t_{n_c}^c\}\}_{c=1}^m$; from each cluster, we have a fitted probability density function. The format of the ground truth data and the

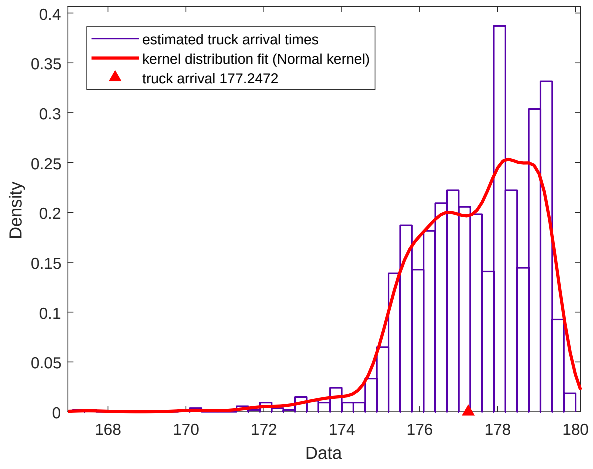


Figure 3.13: Fitting a kernel probability distribution using the Normal kernel to the truck arrival times in one cluster (this group of data belongs to the cluster at the right side in Figure 3.11b; the red triangle represents a truck arrival in ground truth)

estimated data can thus be shown in Figure 3.14. The performance indicators are defined as follows.

For each arrival $t_i \in A$, if there exists a cluster C_{c_i} such that

$$\frac{P_{c_i}(t_i, \varepsilon)}{max\{P_{c_i}(t, \varepsilon)\}} > \delta, \tag{3.28}$$

we regard that the arrival t_i is successfully estimated by C_{c_i}. $P_{c_i}(t, \varepsilon)$ should get its maximum value (i.e., $max\{P_{c_i}(t, \varepsilon)\}$) around the time instant when the probability density function $f_{c_i}(t)$ reaches its peak; $\delta \in [0, 1)$ is a threshold value we can arbitrarily set, i.e., if the probability $P_{c_i}(t_i, \varepsilon)$ is larger than certain percent (i.e., δ) of the maximum probability, we regard that the arrival t_i is successfully estimated by C_{c_i}. For any $t_i \in A$, there is at most one cluster in $\{C_c | C_c = \{t_1^c, t_2^c, \ldots, t_{n_c}^c\}\}_{c=1}^m$ that can successfully estimate t_i.

Obviously, the more arrivals in A being successfully estimated, the better performance of the estimation. Thus, we define a success rate as

$$SR = \frac{n_m}{n} \times 100\% \tag{3.29}$$

where n_m is the number of arrivals in A being successfully estimated. The best value of SR is 100%, which means that all arrivals are successfully estimated.

If a cluster C_c cannot estimate any $t_i \in A$, we regard C_c is *wasted*. Obviously, the less number of wasted clusters, the better performance of the estimation. Therefore, we define a waste rate as

$$WR = \frac{|m - n_m|}{m} \times 100\% \tag{3.30}$$

The best value of WR is 0%, which means that all clustered groups can be used to estimate a truck arrival.

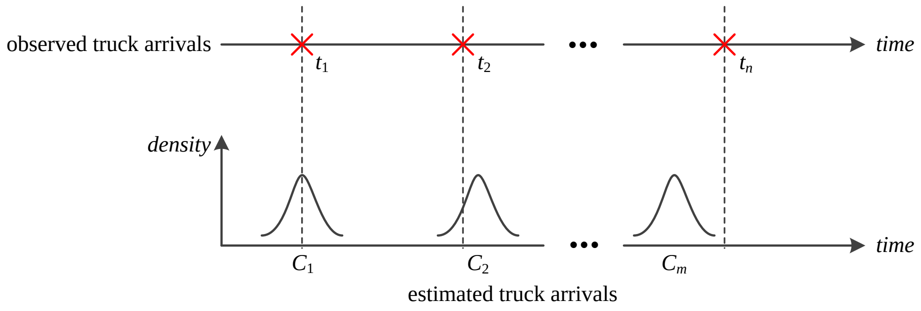


Figure 3.14: Format of the ground truth data and estimated data

Suppose that $t_i \in A$ is estimated by the cluster C_{c_i}; as shown in Figure 3.14, we certainly want t_i to be as close as possible to the time instant when the probability distribution is peaked. Therefore we define two measures to quantify such *closeness*:

- *average distance* to the time instant when the probability density function is peaked:

$$\bar{d} = \frac{1}{n_m} \sum_{j=i_1}^{i_{n_m}} |t_j - t^*_{c_j}| \tag{3.31}$$

  where $t^*_{c_j}$ is the time instant when $f_{c_j}(t)$ is peaked.

- *average percentage* that $P_{c_j}(t_j, \varepsilon)$ accounts for $P_{c_j}(t^*_{c_j}, \varepsilon)$:

$$\bar{P} = 100\% \times \frac{1}{n_m} \sum_{j=i_1}^{i_{n_m}} \frac{P_{c_j}(t_j, \varepsilon)}{max\{P_{c_j}(t, \varepsilon)\}} = 100\% \times \frac{1}{n_m} \sum_{j=i_1}^{i_{n_m}} \frac{P_{c_j}(t_j, \varepsilon)}{P_{c_j}(t^*_{c_j}, \varepsilon)} \tag{3.32}$$

3.5.3 Results

In this section, we present the estimation results of assimilating the noisy dataset ($\sigma_e = 3.0, \sigma_t = 3.0$) with $N_p = 2000$ particles. The model into which we assimilate the noisy data is the same with that we used to generate the simulated data, which means that we use a perfect model of the gold mine system; when retrieving the simulation state at any time t, we use linear interpolation which is introduced in section 3.3.3 to obtain the updated state value.

3.5.3.1 The estimated dimension of the state trajectory

The estimated dimension of state trajectory $x_{0:\mathcal{N}^+_k}$ is shown in Figure 3.15. The average absolute estimation error is

$$\bar{E} = \frac{1}{16} \sum_{k=1}^{16} |\mathcal{N}^+_k - \hat{\mathcal{N}}^+_k| = 0.19.$$

Considering that the dimension is an integer, this error is negligible.

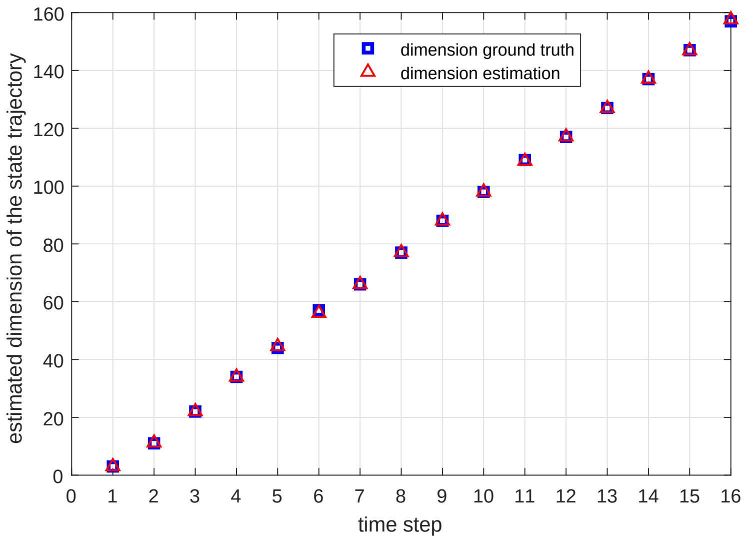


Figure 3.15: The estimated dimension of the state trajectory

3.5.3.2 The estimated truck arrivals

The raw estimation results shown in Figure 3.10b is clustered using the k-means clustering algorithm (Kanungo et al., 2002), and the results are shown in Table 3.3. The k-means clustering algorithm outputs 20 clusters, i.e., $\{C_c\}_{c=1}^{20}$, as shown in the first column of the table; the second column gives the time instant (t_c^*) where the fitted probability distribution is peaked; while the third column computes the probability ($P_c(t_c^*, \varepsilon)$) that a truck arrives at the bottom of the vertical shaft during $[t_c^* - \varepsilon, t_c^* + \varepsilon]$. In this dataset, there are 20 arrivals during the simulation period, i.e., $A = \{t_1, t_2, \ldots, t_{20}\}$. The probability $P_c(t_i, \varepsilon), c = 1, 2, \ldots, 20; i = 1, 2, \ldots, 20$ is computed and presented from the 4th column to the 23rd column. The results show that all arrivals lie in certain cluster, i.e, $\forall t_i \in A, \exists C_{c_i} \in \{C_c\}_{c=1}^{20}, s.t.\ t_i \in [min\{C_{c_i}\}, max\{C_{c_i}\}]$.

We compute the match criterion (see equation 3.28) for each truck arrival in A, and the results are depicted in Figure 3.16. With threshold value $\delta = 50\%$, there are 19 truck arrivals in $A = \{t_1, t_2, \ldots, t_{20}\}$ being successfully estimated by clusters $\{C_c\}_{c=1}^{20}$. Therefore, we have

- success rate $SR = \frac{n_m}{n} \times 100\% = \frac{19}{20} \times 100\% = 95.00\%$.
- waste rate $WR = \frac{m-n_m}{m} \times 100\% = \frac{20-19}{20} \times 100\% = 5.00\%$.
- average distance $\bar{d} = \frac{1}{n_m} \sum_{j=i_1}^{i_{n_m}} |t_j - t_{c_j}^*| = 0.53\ minute$.
- average percentage $\bar{P} = 100\% \times \frac{1}{n_m} \sum_{j=i_1}^{i_{n_m}} \frac{P_{c_j}(t_j,\varepsilon)}{P_{c_j}(t_{c_j}^*,\varepsilon)} = 92.66\%$.

In current operation of the gold mine system, the elevator only comes down until it receives a request from the miner, therefore it will always arrive at the bottom of the

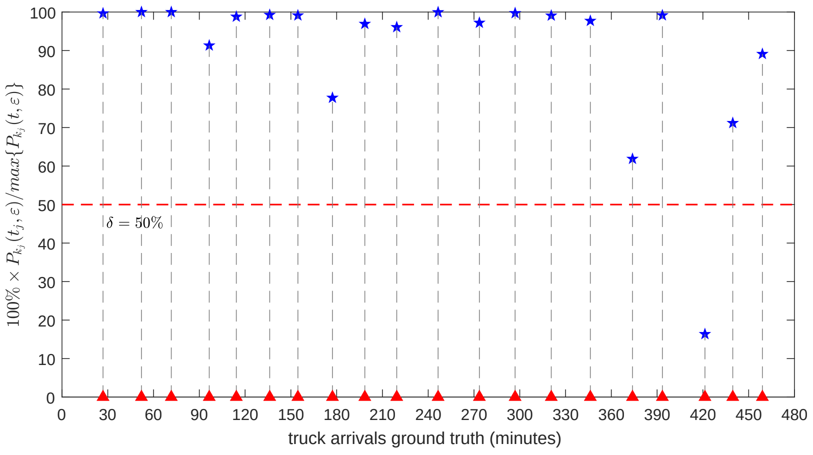


Figure 3.16: The match criterion $100\% \times P_{c_i}(t_i, \varepsilon)/P_{c_i}(t^*_{c_i}, \varepsilon)$ (each red triangle represents a truck arrival in ground truth)

vertical shaft at least 1.8 minutes (the difference between the time of truck traveling full of ore and the time of elevator going down empty) later than the trucks do. In other words, the truck will always wait at least 1.8 minutes until it can be served. However, using data assimilation, we can estimate 95% of all truck arrivals with an average error of 0.53 minute (which is much smaller than 1.8 minutes). If these estimation results can be combined in the operation of the gold mine system (especially in the operation of the elevator), the overall performance of the gold mine system should be improved.

Table 3.3: The data assimilation estimation results ($\sigma_e = 3.0, \sigma_t = 3.0; N_p = 2000; \varepsilon = 0.05$ minute)

data processing results			**truck arrivals ground truth**																			
cluster C_c	t_c^*	$P_c(t_c^*, \varepsilon)$	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}
			26.8918	52.0784	71.6300	96.5060	114.2494	136.1225	154.6024	177.2472	198.4608	219.3735	246.4805	273.5331	296.9505	320.6399	346.2515	373.8774	393.4647	421.3536	439.5858	459.0086
C_1	26.9086	0.0819	0.0816	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
C_2	52.0880	0.0228	–	0.0228	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
C_3	71.6378	0.0308	–	–	0.0308	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
C_4	96.2479	0.0324	–	–	–	0.0296	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
C_5	115.9603	0.0287	–	–	–	–	0.0284	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
C_6	136.0023	0.0226	–	–	–	–	–	0.0224	–	–	–	–	–	–	–	–	–	–	–	–	–	–
C_7	154.6680	0.0283	–	–	–	–	–	–	0.0280	–	–	–	–	–	–	–	–	–	–	–	–	–
C_8	178.2611	0.0239	–	–	–	–	–	–	–	0.0186	–	–	–	–	–	–	–	–	–	–	–	–
C_9	198.7183	0.0214	–	–	–	–	–	–	–	–	0.0207	–	–	–	–	–	–	–	–	–	–	–
C_{10}	221.2574	0.0235	–	–	–	–	–	–	–	–	–	0.0226	–	–	–	–	–	–	–	–	–	–
C_{11}	246.4534	0.0233	–	–	–	–	–	–	–	–	–	–	0.0233	–	–	–	–	–	–	–	–	–
C_{12}	273.3759	0.0289	–	–	–	–	–	–	–	–	–	–	–	0.0280	–	–	–	–	–	–	–	–
C_{13}	297.0031	0.0253	–	–	–	–	–	–	–	–	–	–	–	–	0.0252	–	–	–	–	–	–	–
C_{14}	320.7218	0.0289	–	–	–	–	–	–	–	–	–	–	–	–	–	0.0286	–	–	–	–	–	–
C_{15}	346.1563	0.0384	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0.0375	–	–	–	–	–
C_{16}	372.1869	0.0309	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0.0191	–	–	–	–
C_{17}	393.3903	0.0307	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0.0304	–	–	–
C_{18}	420.1895	0.0733	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0.0120	–	–
C_{19}	441.2210	0.0196	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0.0139	–
C_{20}	459.8954	0.0105	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0.0094

3.5.3.3 The effect of the interpolation operation

In this section, we explore the influence of interpolation on the estimation results. To this end, we run the data assimilation experiment 10 times with different random seeds, and draw box plots of the five error measures (i.e., $\bar{E}$ in equation 3.27, SR in equation 3.29, WR in equation 3.30, $\bar{d}$ in equation 3.31, and $\bar{P}$ in equation 3.32) in Figure 3.17. The results show that though the estimation results obtained from data assimilation without interpolation are already accurate, they can be improved significantly (in the statistic sense) if the interpolation operation is used. Though it is not accurate enough to retrieve the model state without interpolation, the retrieved state still reflects reality to a certain degree, therefore, the estimation results are much better than those without data assimilation. With interpolation, the time elapsed since the last state transition is considered, therefore the real-time evolution, which is not captured in the discrete event simulation model but does happen in reality, will be reflected through the measurement model. Consequently, the estimation results obtained from data assimilation with interpolation are more accurate than those obtained without interpolation.

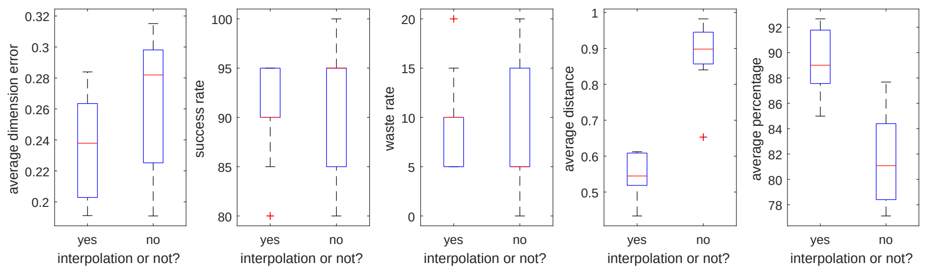


Figure 3.17: The influence of interpolation on the data assimilation results (noisy dataset ($\sigma_e = 3.0, \sigma_t = 3.0$); $N_p = 2000$; 10 independent runs)

3.6 Conclusions

In this chapter, we presented a particle filter based data assimilation framework for discrete event simulations (of closed systems), in which measurements are distributed over the measurement interval (i.e. data fed at time step $k \in \{1, 2, \dots\}$ can contain observations occurring at any time instant during the last measurement interval $[(k-1)\Delta T, k\Delta T]$), implying that the measurements are dependent on the state transitions during that measurement interval. In this framework, two key theoretical problems which hinder the application of particle filtering in discrete event simulations are presented and solved. Both problems are incurred by the mismatch between the discrete event state process and the measurement process, which do not occur in standard discrete time state space models (of closed systems). The problems and corresponding solutions are summarized as follows:

- The first problem is the *state retrieval problem*, which means that the state retrieved from a discrete event simulation model is a combination of sequential states of atomic components that were updated at past time instants, with which inaccurate estimation results will be obtained. The solution is to introduce an interpolation operation which interpolates state values based on the system states updated within a time interval around the time instant when the operation is invoked in order to obtain updated state values. The size of the time interval is determined by the interpolation method employed.

- The second problem is the *variable dimension problem*, which means that the dimension of the state trajectory $s_{0:k}$ (see definition in equation 3.13) is a random variable. This is because the duration between two consecutive state updates in a discrete event simulation is not a constant, but usually a random variable. The variable dimension of $s_{0:k}$ will lead to inapplicability of the standard sequential importance sampling algorithm which updates $p(s_{0:k}|m_{1:k})$. To address the variable dimension problem, we borrow the results from Godsill et al. (2007), in which the problem is solved by extending the state trajectory of interest (with variable dimension) to a sufficient large dimension to make it a fixed dimension state trajectory. The standard sequential importance sampling algorithm can thus be applied to update the joint distribution of the extended state trajectory with fixed dimension. Samples in which the extended state points are discarded will form the samples from the joint distribution of the state trajectory of interest. It is proven that the weight update is independent of these discarded extensions. This result implies that in practice we can safely apply the sequential importance sampling algorithm to update $p(s_{0:k}|m_{1:k})$ where $s_{0:k}$ has a variable dimension.

To illustrate the working of the proposed data assimilation framework, a case in a gold mine system is studied in which we estimate the truck arrival times at the bottom of the vertical shaft. The results show that:

- *The proposed data assimilation framework is able to provide accurate estimation results in discrete event simulations of closed systems.* In the gold mine case study, assimilating (with interpolation) the noisy dataset with Gaussian noise $\mathcal{N}(0, 3^2)$ added on entity positions, the performance indicators of estimating the truck arrivals are 95.00% (success rate), 5.00% (waste rate), 0.53 minute (average distance), and 92.66% (average percentage), respectively (see section 3.5). In contrast, the simulation without data assimilation totally loses its prediction ability from $t = 150$ minutes onwards (see section 3.4).

- *Though the estimation results obtained from data assimilation without interpolation are already accurate, they can be improved significantly (in the statistic sense) if the interpolation operation is used.* If the model state is retrieved without considering the elapsed time (i.e. without interpolation), it can still reflect reality to a certain degree, and the estimation results are much better than those without data assimilation (see section 3.4). However, with interpolation, the elapsed time since the last state transition is considered, and as a result, the real-time evolution, which is not captured in the model but does happen in reality, will be reflected through the measurement model. Consequently, the estimation results obtained from data assimilation with interpolation are the most accurate among the three cases (i.e. estimation without data assimilation, estimation with data assimilation without interpolation, and estimation with data assimilation with interpolation).

- *The variable dimension state trajectory has no tangible effect on weight updating, and particle filtering can approximate the dimension of the state trajectory accurately.* In the gold mine case, the average absolute estimation error for the dimension of the state trajectory is 0.19; considering that the dimension is an integer, this error is negligible (see section 3.5.3.1). The variable dimension problem does not always occur in data assimilation in discrete event simulations. This problem will occur when the measurements are distributed over the measurement interval, i.e., when the measurements are dependent on the (detailed) state transitions within the measurement interval. Otherwise, one can always formalize the discrete event simulation model into a discrete time state space model, similar to the work in Gu and Hu (2008); Hu (2011); Xue et al. (2012), to conduct data assimilation; if more accurate estimation results are required, a proper interpolation operation needs to be developed to get updated state values, as exemplified in section 3.3.3.

In summary, particle filtering is in principle applicable to data assimilation in discrete event simulations. However, to apply particle filtering in discrete event simulations effectively, we need to clearly understand the characteristics of discrete event simulations, which make the state retrieval and the random measure update different from those in discrete time models. Though the random measure update equations finally adopt their standard forms, we should know the reason behind. The standard sequential importance sampling algorithm is used to update the joint distribution of the state trajectory with a fixed dimension, while the desired joint distribution of the state trajectory of interest that has a variable dimension is a marginal. Since weight update is independent of the extensions which extend the variable dimension state trajectory to a fixed dimension, we can in practice directly update this marginal without generating the extensions. As a result, the random measure update equations in discrete event simulations have the same forms with those in discrete time models.

CHAPTER 4

Particle filter based data assimilation in discrete event simulations of open systems[1]

In chapter 3, we presented a particle filter based data assimilation framework for discrete event simulations of closed systems, in which the model components do not change over time. However in reality, we often encounter discrete event systems where entities can continuously flow in and flow out through the system boundaries, such as urban traffic systems, which will incur another form of the variable dimension problem. Therefore in this chapter we investigate the data assimilation problem in discrete event simulations of open systems in which the variable dimension problem arises in a different way. Since in chapter 3 we have elaborated on the key components in the particle filter based data assimilation framework for discrete event simulations, such as the system model, the measurement model (with interpolation), and the particle filtering algorithm, in this chapter we will present the data assimilation framework for discrete event simulations of open systems very briefly, because many components (such as state updates, measurement model, and interpolation) are very similar with those in chapter 3, and we mainly focus on the differences and why the variable dimension problem occurs in discrete event open systems and how we address this problem. Since characteristics summarized in section 2.3.1 are the nature of discrete event systems, they are independent of the modeling formalisms, which means modeling formalisms do not influence the working of the proposed data assimilation framework. Therefore, in this chapter, we do

[1]This chapter is a revised version of a paper submitted to *Transportation Research Part C: Emerging Technologies*: Xie, X., van Lint, J. W. C., Verbraeck, A.: *A generic data assimilation framework for vehicle trajectory reconstruction on signalized urban arterials using particle filters*.

not explicitly mention the formalism that is used to model discrete event open systems, and indeed one can choose any discrete event modeling formalisms which are deemed appropriate to model these systems.

4.1 Particle filtering in discrete event simulations of open systems

In this section, we investigate the data assimilation problem in discrete event simulations of open systems. The system model for discrete event open systems is defined in section 4.1.1, in which we do not put any emphasize on the detailed state transitions (of individual components or of the coupled model) that are similar with those in chapter 3. The measurement model is briefly introduced in section 4.1.2 where we still assume that the measurements are distributed over the last measurement interval, and the interpolation operation invoked at a time instant t is independent of the states beyond that time. Finally the particle filtering algorithm for state estimation in discrete event simulations of open systems is presented in section 4.1.3, in which the variable dimension problem is explained and solved.

4.1.1 System model

A discrete event simulation of an open system can be generalized as

$$S_k = SIM(S_{k-1}, E_{k,\Delta T}) + \nu_{k-1}, k = 1, 2, \ldots \tag{4.1}$$

where S_k defines the system state, $SIM(\cdot)$ is a discrete event simulation model, and ν_{k-1} represents a system noise process. ΔT is the measurement interval, and $E_{k,\Delta T}$ is a list of events that model entity arrivals at the system boundaries during the k-th measurement interval $[(k-1)\Delta T, k\Delta T]$. An entity is an object in the system under study, for example, in an urban traffic system, vehicles are the main entities in the system. Assume that the state of an entity in the system is modeled as s_k^i, where i is the entity index, then the system state can be defined as

$$S_k = \{s_k^i\}_{i=1}^{Q_k} \tag{4.2}$$

where Q_k is the number of entities that have entered the system until time k, which can be defined as

$$Q_k = N_0 + \sum_{j \in Inflow} \int_0^{k\Delta T} q_j(s) ds$$

where N_0 is the number of entities in the system in the beginning, $Inflow$ is a set of system boundaries where entities can enter the system, and $q_j(s)$ is the flow at the boundary $j \in Inflow$.

Since entities enter the system at different time instants which are usually irregularly distributed on the continuous time axis, the local state updates within each entity are asynchronous with each other. As a result, the actual system state updates are still asynchronous with the measurement process, though equation 4.1 has the form of a discrete time state space model. We can interpret S_k in equation 4.1 as an interpolated system state at time instant $k\Delta T$.

The system state S_k can be further separated as

$$S_k = \{s_k^i\}_{i=1}^{Q_k} = \{s_k^i\}_{i=1}^{Q_k-N_k} \cup \{s_k^i\}_{i=Q_k-N_k+1}^{Q_k}, \text{ where}$$
$$N_k = N_0 + \sum_{j\in Inflow} \int_0^{k\Delta T} q_j(s)ds - \sum_{j\in Outflow} \int_0^{k\Delta T} q_j(s)ds \tag{4.3}$$

in which $Outflow$ is a set of boundaries where entities leave the system. $\{s_k^i\}_{i=1}^{Q_k-N_k}$ represents the states of entities that have left the system. Once an entity leaves the system, we stop updating its state in the simulation. N_k is actually the number of (active) entities in the system at time k. Since the arrivals of entities usually obey certain stochastic process (e.g., Poisson process), Q_k is a random variable.

4.1.2 Measurement model

We assume that the measurements available at time k are dependent on the (detailed) state transitions during the last measurement interval $[(k-1)\Delta T, k\Delta T]$; the interpolation operation invoked at any time instant is independent of the states beyond that time. Therefore, the measurement model can be formalized as

$$m_k \sim p(m_k|S_{k-1:k}), k = 1, 2, \ldots \tag{4.4}$$

where $S_{k-1:k}$ represents the detailed state trajectory during $[(k-1)\Delta T, k\Delta T]$.

4.1.3 State estimation using particle filters

The state estimation using particle filters essentially approximates the conditional distribution $p(S_{0:k}|m_{1:k})$ given all measurements until time k, where $S_{0:k} = \{S_i, i = 0, 1, \ldots, k\}, m_{1:k} = \{m_i, i = 1, 2, \ldots, k\}$. To highlight the dimension of S_k, we put Q_k in its subscript, therefore the objective of particle filtering is to approximate $p(S_{0,Q_0:k,Q_k}|m_{1:k})$. Based on Bayes' theorem, we have

$$p(S_{0,Q_0:k,Q_k}|m_{1:k}) = \frac{p(S_{0,Q_0:k,Q_k})p(m_{1:k}|S_{0,Q_0:k,Q_k})}{p(m_{1:k})}$$

A recursive update equation can thus be obtained:

$$p(S_{0,Q_0:k,Q_k}|m_{1:k}) = p(S_{0,Q_0:k-1,Q_{k-1}}|m_{1:k-1}) \times \frac{p(m_k|S_{k,Q_k})p(S_{k,Q_k}|S_{k-1,Q_{k-1}})}{p(m_k|m_{1:k-1})} \tag{4.5}$$

Since Q_k is a random variable, the dimension of $S_{0,Q_0:k,Q_k}$ is also random. Therefore, we cannot use the standard sequential importance sampling algorithm to update the posterior distribution in equation 4.5 due to the variable dimension of $S_{0,Q_0:k,Q_k}$. This variable dimension problem is addressed in a similar way with that in Godsill et al. (2007), i.e., we first make $S_{0,Q_0:k,Q_k}$ to have a fixed dimension with certain extensions, and then we prove that the extensions have no influence on the weight update. To this end, we define a state with fixed dimension as follows

$$S_{k,K} = S_{k,Q_k} \cup V_{k,K-Q_k}$$

where S_{k,Q_k} is defined in equation 4.3; $V_{k,K-Q_k} = \{s_k^j\}_{j=Q_k+1}^{K}$ defines a set of *virtual* entities that make $S_{k,K}$ to have fixed dimension K, which is a sufficient large constant such that $K \geq Q_k, k = 0, 1, 2, \ldots, span/\Delta T$, where $span$ is the length of the simulation study period. These virtual entities have no influence on the evolution of S_{k,Q_k}. Since $S_{k,K}$ has a fixed dimension, the state trajectory $S_{0,K:k,K}$ will have a fixed dimension, and as a result, we can use the sequential importance sampling algorithm to update $p(S_{0,K:k,K}|m_{1:k})$, which is first factorized as

$$p(S_{0,K:k,K}|m_{1:k}) = p(S_{0,Q_0:k,Q_k}|m_{1:k})\pi(V_{0,K-Q_0:k,K-Q_k}|S_{0,Q_0:k,Q_k})$$

where the conditional distribution π complements the state trajectory with variable dimension to a fixed dimension. π can be chosen arbitrarily.

Suppose we have a set of random samples χ_{k-1} to approximate $p(S_{0,Q_0:k-1,Q_{k-1}}|m_{1:k-1})$:

$$\chi_{k-1} = \{S^i_{0,Q_0:k-1,Q_{k-1}}, w^i_{k-1}\}_{i=1}^{N_p} \tag{4.6}$$

We then update the samples based on certain importance density $q(\cdot)$. We can assume that $q(\cdot)$ can be factorized as

$$q(S_{0,Q_0:k,Q_k}|m_{1:k}) = q(S_{k,Q_k}|S_{0,Q_0:k-1,Q_{k-1}}, m_{1:k})q(S_{0,Q_0:k-1,Q_{k-1}}|m_{1:k-1})$$

Then the particles in equation 4.6 are updated in two steps. First, these particles are updated to time k by sampling according to

$$q(S_{k,Q_k}|S_{0,Q_0:k-1,Q_{k-1}}, m_{1:k})$$

Second, the updated particles are complemented to a fixed dimension by drawing samples from

$$q(V_{0,K-Q_0:k,K-Q_k}|S_{0,Q_0:k,Q_k}) = \pi(V_{0,K-Q_0:k,K-Q_k}|S_{0,Q_0:k,Q_k})$$

The weights in equation 4.6 are updated as follows:

$$\begin{aligned}
w_k &= \frac{p(S_{0,K:k,K}|m_{1:k})}{q(S_{0,K:k,K}|m_{1:k})} \\
&= \frac{p(m_k|S_{k,Q_k})p(S_{k,Q_k}|S_{k-1,Q_{k-1}})p(S_{0,Q_0:k-1,Q_{k-1}}|m_{1:k-1})}{p(m_k|m_{1:k-1})q(S_{k,Q_k}|S_{0,Q_0:k-1,Q_{k-1}}, m_{1:k})q(S_{0,Q_0:k-1,Q_{k-1}}|m_{1:k-1})} \\
&\quad \times \frac{\pi(V_{0,K-Q_0:k,K-Q_k}|S_{0,Q_0:k,Q_k})}{\pi(V_{0,K-Q_0:k,K-Q_k}|S_{0,Q_0:k,Q_k})} \\
&\propto \frac{p(m_k|S_{k,Q_k})p(S_{k,Q_k}|S_{k-1,Q_{k-1}})}{q(S_{k,Q_k}|S_{0,Q_0:k-1,Q_{k-1}}, m_{1:k})} \frac{p(S_{0,Q_0:k-1,Q_{k-1}}|m_{1:k-1})}{q(S_{0,Q_0:k-1,Q_{k-1}}|m_{1:k-1})} \\
&= \frac{p(m_k|S_{k,Q_k})p(S_{k,Q_k}|S_{k-1,Q_{k-1}})}{q(S_{k,Q_k}|S_{0,Q_0:k-1,Q_{k-1}}, m_{1:k})} \times w_{k-1}
\end{aligned}$$

which is again independent of the states of the virtual entities.

Notice that we assume the measurement is dependent on the (detailed) state transitions during the last measurement interval (see equation 4.4), the weight update equation can thus be modified to

$$w_k \propto \frac{p(m_k|S_{k-1,Q_{k-1}:k,Q_k})p(S_{k-1,Q_{k-1}:k,Q_k}|S_{k-1,Q_{k-1}})}{q(S_{k-1,Q_{k-1}:k,Q_k}|S_{0,Q_0:k-1,Q_{k-1}}, m_{1:k})} \times w_{k-1}$$

This formal proof implies that the variable dimension state trajectory has no tangible effect on weight updating, we therefore can safely apply the sequential importance sampling algorithm in discrete event simulations of open systems. Since the sequential importance sampling algorithm has been introduced several times in previous chapters, we do not repeat it here any more.

4.2 Case study – reconstructing vehicle trajectories on signalized urban arterials

In most microscopic traffic simulations, the simulation state is updated in a stepwise fashion, such as FOSIM [2] and MovSim [3]. However, urban traffic systems also show clear discrete event features. First, vehicles can flow in and flow out of the study area at any time instants on a continuous time axis. Second, traffic flow is usually regulated by traffic signals. Therefore, we regard urban traffic systems as discrete event open systems. As a result, the discrete event approach is a good choice to model urban traffic systems, in which vehicle arrivals/departures and signal control logic can be modeled more realistically and more conveniently; the stepwise state update mechanism of vehicles can be easily implemented using a discrete event approach (the duration between two consecutive state updates is constant), but not in a reverse way. Therefore, in this chapter, we study a case in urban traffic systems to demonstrate the working of the proposed data assimilation framework in open systems. In this case study, we aim to reconstruct vehicle trajectories on signalized urban arterials.

In order not to distract readers from the storyline of trajectory reconstruction in subsequent sections, we introduce the interpolation operation here, and will not mention this concept in the rest of this chapter. Since vehicles enter the simulation at different time instants, the times when vehicles update their states are different from vehicle to vehicle. As a result, an interpolation operation is required when retrieving the system state. As will be explained in subsequent sections, the state of a vehicle at a time instant $t \in \mathbb{R}^+_{0,\infty} = \{r \in \mathbb{R} | r \geq 0\}$ can be defined as

$$s(t) = \{x(t), v(t), a(t)\} \tag{4.7}$$

where x, v, and a are the vehicle's longitudinal location, speed, and acceleration, respectively. Then the state of the vehicle at any time t can be computed as follows:

$$x(t) = x(t_0) + \int_{t_0}^{t} v(s)ds$$
$$v(t) = v(t_0) + \int_{t_0}^{t} a(s)ds$$

where t_0 is the time instant when the vehicle enters the system, and $x(t_0)$ indicates the location of the system boundary. In many microscopic traffic simulation model, the state

[2]FOSIM (Freeway Operation SIMulation), developed at Delft University of Technology, is a microscopic traffic simulation package which has been extensively calibrated and validated for traffic operations on Dutch freeways. More information can be found at `https://www.fosim.nl/`.

[3]MovSim is a microscopic lane-based traffic simulator, and more information can be found at `http://www.movsim.org/`.

of a vehicle is updated every δt time units, and the movement of the vehicle during the δt length period is usually modeled as a constant speed or constant acceleration process. Suppose that the last state update was at time t_l, at any time $t \in (t_l, t_l + \delta t)$, the state of the vehicle can be obtained through interpolation (the implementation details can be found in section A.2.5.1) as follows:

$$\hat{s}(t) = interpolate(s(t_l), e)$$

$$\text{where } \begin{cases} x(t) = \begin{cases} x(t_l) + v(t_l) \times (t - t_l) + \frac{1}{2} \times a(t_l) \times (t - t_l)^2 & \text{if } v(t_l) + a(t_l) \times (t - t_l) \geq 0 \\ x(t_l) - \frac{1}{2} \times \frac{v(t_l) \times v(t_l)}{a(t_l)} & \text{if } v(t_l) + a(t_l) \times (t - t_l) < 0 \end{cases} \\ v(t) = max\{0, v(t_l) + a(t_l) \times (t - t_l)\} \\ a(t) = \begin{cases} a(t_l) & \text{if } v(t_l) + a(t_l) \times (t - t_l) \geq 0 \\ 0 & \text{if } v(t_l) + a(t_l) \times (t - t_l) < 0 \end{cases} \end{cases}$$

In the rest of this section, the data assimilation framework presented in section 4.1 is tailored to solve the trajectory reconstruction problem, and the solution is a generic data assimilation framework based on particle filters to reconstruct (plausible) vehicle trajectories on signalized urban arterials. The solution is generic in the sense that any (ensemble of) microscopic simulation models (that implement car following, lane changing, crossing, etc.) can be used. In section 4.2.1, we introduce the trajectory reconstruction problem, present related work, and overview the main notation used in the data assimilation framework. Section 4.2.2 then overviews the overall trajectory reconstruction methodology, after which section 4.2.3 details the particle filter based data assimilation framework and section 4.2.4 discusses data preprocessing methods and assumptions related to error models associated with the different data sources used.

4.2.1 The trajectory reconstruction problem

Vehicle trajectory data provide critically important information for many application areas, ranging from calibration and validation of microscopic traffic flow models (Kesting and Treiber, 2008; Punzo and Montanino, 2016), traffic state reconstruction (van Lint and Hoogendoorn, 2010; Wang et al., 2006), travel time estimation (van Lint, 2010; Coifman, 2002), vehicle energy/emissions estimation (Sun et al., 2015; da Rocha et al., 2015), to name just a few. With trajectory data, a complete picture of traffic flow operations can be obtained, both microscopically and macroscopically. Trajectory data can be collected using a wide range of sensing technologies, such as aerial photography, video, and mobile traffic sensors based on GPS and/or GSM (Montanino and Punzo, 2015; Sun and Ban, 2013). Whereas reconstructing trajectories from microscopic information (e.g., aerial images, GSM traces) requires considerable methodological effort in itself (Montanino and Punzo, 2015), there are only a few cases for which comprehensive data that cover 100% of all vehicle trajectories are available. Collecting such comprehensive trajectory data over large distances (routes, networks) and long time periods is expensive. With infrastructure based sensors or aerial imaging, the collected data can only cover a limited spatiotemporal region due to the high installation and maintenance costs, and it will take a few years (if not longer) before 100% of all vehicles/drivers are equipped with location tracking systems or apps that continuously communicate these data for use in modeling, control or other applications. The next best alternative is to *estimate* vehicle trajectories from whatever data *are* available.

4.2.1.1 Estimating vehicle trajectories

Many methods for estimating vehicle trajectories from data have been proposed, which in general terms (should) combine three ingredients: data (from various types of sensors); assumptions (models, equations) that describe the relation between the data and the underlying vehicle trajectories; and data assimilation techniques that combine these ingredients and in the process address measurement and modeling errors. For example, Coifman (2002) reconstructs vehicle trajectories in order to estimate travel times on freeways using traffic data from a single dual loop detector. The proposed method exploits basic traffic flow theory to extrapolate local traffic conditions to an extended freeway link. However, this method will fail when a queue covers the link, which is very common in signalized arterials. An obvious remedy is to reconstruct vehicle trajectories along a route using multiple loop detectors (van Lint, 2010), so that the trajectory reconstruction process is based on information from both up- and downstream locations. The resulting vehicle trajectories are essentially idealized average vehicle trajectories, similar to Lagrangian solutions of kinematic wave theory. However, the adaptive smoothing method used in van Lint (2010) is not suitable for urban trajectory reconstruction in the original form (Treiber and Helbing, 2002; van Lint and Hoogendoorn, 2010), because it would smooth speeds over intersections. Adjusting the method for use in urban settings seems doable in principle, but has not been reported yet. Mehran et al. (2012) propose a data fusion framework to reconstruct vehicle trajectories on urban arterials by incorporating probe and fixed sensor data and the signal timing parameters. The proposed method is also based on kinematic wave theory and employs the variational formulation (VF) (Daganzo, 2005a,b) to solve a dense solution network which is constructed by discretizing the time-space plane. The key principle of the VF method is that the cumulative number of vehicles at each node in this solution network can be computed by a shortest path search from nodes where the cumulative numbers are known (boundary conditions). As a result, any curve which connects nodes with the same cumulative number in the solution network represents an individual vehicle trajectory. Sun and Ban (2013) also apply the VF method to estimate trajectories by fusing probe vehicle trajectories and the signal timing data. However, in both papers using the VF method, sensor errors such as miss-counting and over-counting are not considered; whereas these pose common and difficult problems when using loop detector data (Lu et al., 2008). With these errors, the cumulative numbers at boundaries will be inaccurate. Worse still, the error resulting from using such erroneous counts in the estimation of the number of vehicles between these cumulative count stations becomes unbounded (van Lint and Hoogendoorn, 2015). Another problem is that, due to the use of first order traffic flow theory, the speeds (trajectory slopes) between nodes are piecewise constant (no acceleration) yielding piecewise linear vehicle trajectories. As shown by

Sun et al. (2015) and da Rocha et al. (2015), in case such piecewise linear trajectories are used to estimate energy consumption or emissions, large errors result, since energy consumption and emissions are influenced largely by the acceleration/deceleration process. This point holds for any traffic state estimation method based on first order traffic flow theory (e.g., Nantes et al. (2016); Yuan et al. (2012); van Hinsbergen et al. (2012)), with which (indirect) also vehicle trajectories can be estimated.

In general, when the (behavioral) assumptions of these trajectory estimation methods are insufficient for the application at hand, more elaborate assumptions are required. This could involve estimation methods using higher order macroscopic models (that include speed dynamics as in Wang et al. (2006)), or methods using microscopic models for driving behavior. Goodall et al. (2016) present such a microscopic estimation method for vehicle trajectories on freeways. The objective here is to use trajectory estimation to artificially increase the penetration rate (in sample size as well as frequency) of connected vehicles. The method proposes a strategy about when and where to add or remove simulated vehicles (called *estimated vehicles*) in the microscopic simulation in order to make the actual behavior of the probe vehicles align with their expected behavior predicted by their car-following models. The results show that the effective penetration rate can be increased by around $20\% \sim 30\%$ using this method, which turned out beneficial for a ramp metering application. However, in this approach no principled method to deal with data or modeling errors is discussed.

4.2.1.2 The proposed solution

In this case study, we propose a generic data assimilation framework based on particle filters to reconstruct (plausible) vehicle trajectories on signalized urban arterials, that *does systematically address errors both in the measurements and in the model*. Like Goodall et al. (2016), our framework uses microscopic models of driving behavior and it assimilates *noisy* data from different sensors using particle filters (Arulampalam et al., 2002; Djurić et al., 2003). The framework does not impose restrictions on the type of microscopic models used; however, to illustrate the working, we consider the longitudinal movements of vehicles only. In terms of data, the method takes in noisy vehicle passages of individual vehicles (and as a result noisy vehicle counts) from loop detectors; signal timing parameters, and coarsely available travel time observations. By 'noisy', we mean that the passage data contains miss- and over-counts, resulting in counting errors.

Before elaborating the proposed data assimilation framework for trajectory reconstruction in subsequent sections, we finish this section by introducing the main notation used in this proposed framework.

Notation

name	**unit**	**description**
ΔT	s	time between two consecutive data feeds
$N(t)$	veh	number of vehicles at time instant t
N_k	veh	number of vehicles at time step k, $N_k \equiv N(k\Delta T)$
$Q(t)$	veh	number of vehicles that have entered the simulation till time instant t

Q_k	veh	number of vehicles that have entered the simulation till time step k, $Q_k \equiv Q(k\Delta T)$
$x^i(t)$	m	location of the i-th vehicle at time instant t
x^i_k	m	location of the i-th vehicle at time step k, $x^i_k \equiv x^i(k\Delta T)$
$v^i(t)$	m/s	speed of the i-th vehicle at time instant t
v^i_k	m/s	speed of the i-th vehicle at time step k, $v^i_k \equiv v^i(k\Delta T)$
$s^i(t)$	$-$	state of the i-th vehicle at time instant t
s^i_k	$-$	state of the i-th vehicle at time step k, $s^i_k \equiv s^i(k\Delta T)$
S_k	$-$	system state at time step k
$E_{k,\Delta T}$	$-$	vehicle arrivals during the k-th measurement interval $[(k-1)\Delta T, k\Delta T]$
p	$-$	detection accuracy of a sensor
λ	s^{-1}	occurrence rate of over-count
m^o_k	$-$	measurement at time step k
N_s	$-$	number of sensors
$\mathcal{L}$	$-$	set of road stretches in the system
E^j_k	$-$	observed vehicle passing times from the j-th ($1 \leq j \leq N_s$) sensor during $[(k-1)\Delta T, k\Delta T]$
N^l_k	veh	observed number of vehicles on road stretch l ($l \in \mathcal{L}$) at time step k
S^i_k	$-$	system state represented by the i-th particle at time step k
$E^i_{k,\Delta T}$	$-$	reconstructed vehicle arrivals for the i-th particle during the k-th measurement interval $[(k-1)\Delta T, k\Delta T]$
w^i_k	$-$	weight of the i-th particle at time step k
N^i_k	veh	number of vehicles in the i-th particle at time step k
N_p	$-$	number of particles
$x^{i,j}(t)$	m	location of the j-th vehicle in the i-th particle at time instant t
$x^{i,j}_k$	m	location of the j-th vehicle in the i-th particle at time step k, $x^{i,j}_k \equiv x^{i,j}(k\Delta T)$
$v^{i,j}(t)$	m/s	speed of the j-th vehicle in the i-th particle at time instant t
$v^{i,j}_k$	m/s	speed of the j-th vehicle in the i-th particle at time step k, $v^{i,j}_k \equiv v^{i,j}(k\Delta T)$
$E^{i,j}_k$	$-$	vehicle passing times from the j-th ($1 \leq j \leq N_s$) sensor during the k-th measurement interval $[(k-1)\Delta T, k\Delta T]$ when generating the i-th particle
$N^{i,l}_k$	veh	number of vehicles on road stretch l ($l \in \mathcal{L}$) at time step k in the i-th particle
N_m	$-$	number (random variable) of miss-counts
N_o	$-$	number (random variable) of over-counts
$Q_i(t)$	veh	cumulative counts from the detector at cross-section x_i at time instant t
$t_i(n)$	s	time instant when the n-th vehicle passes cross-section x_i
$T_r(n)$	s	travel time estimated from the n-th vehicle
$T^{obs}_r(t)$	s	travel time observation observed at time instant t
$P(X = x)$	$-$	discrete probability distribution of random variable X
$P(X \vert Y)$	$-$	conditional probability
$\hat{X}$	$-$	estimation of variable X

4.2.2 Overview: a generic data assimilation framework for trajectory reconstruction

In this section we describe the objective and overall ideas of the data assimilation framework. The objective of the framework is to estimate the most probable system state trajectory (in our case: the most probable set of vehicle trajectories on a roadstretch $[x_1, x_2]$ over a time period T), given the available data. The particle filter does this by estimating the posterior distribution with a sufficient set of support points (particles) spanning this distribution, each of which represents one possible set of vehicle trajectories that is computed by an arbitrary microscopic simulation model. One can interpret this posterior as a "belief histogram" of the system state trajectory, that describes a sample of probable sets of vehicle trajectories, given the data. This interpretation, however, is not very informative in the context of trajectory reconstruction. We do not want many probable sets of vehicle trajectories, but *one* set that is most likely given the data. Therefore we output the set of trajectories with the highest weight, although this is not strictly correct (since we obtain the estimation of interest not by Monte Carlo integration, but by the particle with the highest weight). Notice that the objective of particle filtering is to estimate $p(s_{0:k}|m_{1:k})$ (see section 2.2.3), therefore, as measurements accumulate, the reconstructed vehicle trajectories will span the time-space region $[x_1, x_2] \times [0, T]$.

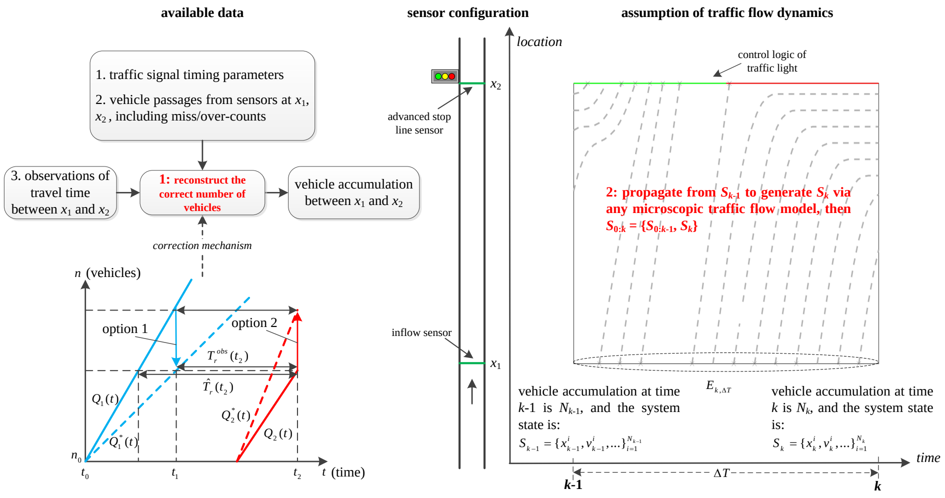

available data
sensor configuration
assumption of traffic flow dynamics
1. traffic signal timing parameters
2. vehicle passages from sensors at x_1, x_2, including miss/over-counts
3. observations of travel time between x_1 and x_2
1: reconstruct the correct number of vehicles
vehicle accumulation between x_1 and x_2
correction mechanism
n (vehicles)
option 1
option 2
$T_r^{obs}(t_2)$
$\hat{T}_r(t_2)$
$Q_1(t)$
$Q_1^*(t)$
$Q_2^*(t)$
$Q_2(t)$
n_0
t_0
t_1
t_2
t (time)
advanced stop line sensor
inflow sensor
x_2
x_1
location
control logic of traffic light
2: propagate from S_{k-1} to generate S_k via any microscopic traffic flow model, then $S_{0:k} = \{S_{0:k-1}, S_k\}$
vehicle accumulation at time k-1 is N_{k-1}, and the system state is:
$S_{k-1} = \{x_{k-1}^i, v_{k-1}^i, ...\}_{i=1}^{N_{k-1}}$
$E_{k,\Delta T}$
vehicle accumulation at time k is N_k, and the system state is:
$S_k = \{x_k^i, v_k^i, ...\}_{i=1}^{N_k}$
time
ΔT
k-1
k

Figure 4.1: The main idea of the data assimilation framework for vehicle trajectory reconstruction

The main idea and ingredients of our approach are depicted in Figure 4.1. Reconstructing vehicle trajectories with this approach thus implies attempting to reconstruct the *correct* number of *plausible* vehicle locations, speeds (and possibly other dynamic attributes). The *italics* indicate that this reconstruction involves two aspects:

1. We need to estimate the correct number of vehicles with the correct departure time on the basis of observed vehicle passages. The key problem to overcome here is that vehicle counts contain errors. Counting errors (caused by miss- or over-counting

passages) have a large effect on reconstructing trajectories. One miss- or over-count may yield errors in *all* subsequent trajectories. Our approach to address this problem is to first make the number of vehicle passages (by extension the flows) between two detectors consistent. To this end we use the method proposed by van Lint and Hoogendoorn (2015), which—similarly to the method described in Bhaskar et al. (2014)— uses coarsely available individual travel times to periodically correct cumulative curves (at location x_1 and x_2 respectively) and as a consequence compute a correct number of vehicles between the detectors.

2. Using these corrected vehicle counts we can now reconstruct plausible vehicle trajectories—potentially aided by either macroscopic speed data or individual trajectory data. As touched upon in the introduction, this can be done by using macroscopic (first or higher order) traffic flow theory, or by using microscopic traffic flow simulation models (or anything in between). We choose the latter. In fact, our framework allows the analyst to use whatever microscopic model deemed appropriate (for the application at hand); that is, a model with longitudinal behavior only, or a full fletched model that includes lane changing, crossing, etc. *Clearly, more degrees of freedom, without associated evidence in the form of microscopic data, may yield more sophisticated estimations but not necessarily more plausible vehicle trajectories.*

Since many of the underlying steps are interrelated, we discuss these aspects in a specific order. In section 4.2.3 we first explain the particle filtering rationale (aspect 2). This involves the basic particle filtering principle (please see section 2.2.3 for more detail), resampling and the procedure to go from partial vehicle trajectories (reconstructed whenever new measurements are available) to full trajectories (over consecutive measurement time intervals). Thereafter, in section 4.2.4, we discuss the measurements (including the vehicle accumulation correction method), and the assumptions related to the errors in each used data source (aspect 1). In section 4.2.4 we then conclude with the computation of particle weights (part of aspect 2), that depends on these error models.

4.2.3 Particle filter design for trajectory reconstruction

4.2.3.1 The resampling scheme

As explained in section 2.2.3, a major problem of particle filters is that the discrete random measure degenerates quickly, therefore we need to resample the particles on a regular basis after they are updated. There exist different resampling algorithms and methods (Douc et al., 2005), and we propose a hierarchical strategy that is tailored to our specific problem.

Recall that the overall objective of our assimilation framework (Figure 4.1) is to estimate the most probable number of vehicles; and—given this number of vehicles—estimate the most probable vehicle trajectories. The rationale of our resampling strategy is based on these two aspects. We first categorize particles (simulation instances) into groups with similar vehicle accumulations, and resample within each group. This will ensure there is a sufficient number of particles selected over a range of possible vehicle accumulations. It is important to note that the weights of particles within such a group do not differ as much compared to when all particles are considered as a whole. This implies that by sampling within each group, the particles with small weights will have a higher probability to be selected than in case all particles are resampled together. As a result, this hierarchical resampling method will ensure that many likely particles are kept for future iterations, whereas a sufficient number of less likely particles are not discarded. The consequence is a better coverage of the entire state space. Suppose that the particles generated from the sampling step are $\{S^i_{0:k}, w^i_k\}^{N_p}_{i=1}$, the hierarchical resampling method now works as follows:

- Sample based on the vehicle accumulations (number of vehicles) at time k. Suppose that the vehicle accumulation at time k is N_k, and the corresponding value in the i-th particle is N^i_k. We categorize all particles into groups, such that any two particles within the same group have the same value of vehicle accumulation. Assume that there are C such groups, and the corresponding value of the vehicle accumulation in the c-th group is $N^c, c = 1, 2, \ldots, C$. We again assign a probability value $p_c = P(E = N^c - N_k)$ to the c-th group, where $P(E)$ corresponds to the error model for vehicle accumulation, which will be discussed in section 4.2.4.2. In simple terms, we favor those particles with a low error. Finally, $p_c, c = 1, 2, \ldots, C$ are normalized and N_p samples are drawn from $\{N^1, N^2, \ldots, N^C\}$ with replacement, and N^c has a probability of p_c to be chosen. Suppose that N^c is selected $Nr(N^c)$ times, then we have $\sum_{c=1}^{C} Nr(N^c) = N_p$.
- Sample based on the weights. In the c-th group, we draw $Nr(N^c)$ particles with probabilities proportional to their weights (the weights are temporarily normalized within the group for sampling). In this process, if a particle is selected, its original weight is also associated with that particle. When all groups are sampled, the particles selected from each group form the resampled particles $\{S^i_{0:k}\}^{N_p}_{i=1}$, and their associated weights are $\{w^i_k\}^{N_p}_{i=1}$.

To actually compute a quantitative value for each weight, we require a mechanism that takes into account the errors associated with each observation. We therefore return to weight computation in section 4.2.4.4 after discussing the data error models. In this section we further focus on the trajectory reconstruction method.

4.2.3.2 State dynamics: individual vehicle movements

In this section we specify the generic state dynamics in equation 4.1. In principle we can choose whatever traffic flow model deemed appropriate for the estimation task at hand, as long as it (endogenously) computes the system state from which measurements (equation 4.4) can be constructed. Without loss of generality, in this case, we consider the longitudinal movements of vehicles only, that is, we consider models for free-driving and car-following (CF) that describe the acceleration behavior of driver-vehicle units as a function of stimuli such as headway and speed difference with respect to a leading vehicle, subject to physical considerations, and a wide range of (behavioral) assumptions. In general terms, (discretized) CF models take on the following form

$$a(t + T^r) = f(\eta(t), \zeta(t)),$$

in which $T^r \geq 0$ represents a reaction time, $\eta(t)$ is a set of stimuli (headway, speed difference with leader, etc.), and $\zeta(t)$ is a set of parameters specifying above mentioned behavioral and physical assumptions. For comprehensive reviews of longitudinal models for driving behavior see e.g., van Wageningen-Kessels et al. (2015); Saifuzzaman and Zheng (2014); Brackstone and McDonald (1999). In our case, the precise form of the CF models (or the models for lane changing, crossing, etc.) does not matter. Clearly, applying a microscopic simulation model for state estimation, requires such a model to be well-calibrated. We refer the reader to Punzo and Montanino (2016); da Rocha et al. (2015); Punzo et al. (2015) for an in-depth treatment of microscopic model calibration, which is beyond the scope of this case study. For our purposes, it is sufficient to describe the resulting vehicle trajectory by time series of two variables: longitudinal location $x^i(t)$ and speed $v^i(t)$, where i is index of the vehicle, that is

$$s^i(t) = \{x^i(t), v^i(t)\}, t \in \mathbb{R}^+_{0,\infty} = \{r \in \mathbb{R} | r \geq 0\}. \tag{4.8}$$

During the simulation, new vehicles continuously arrive at the system boundary and enter the simulation. These arrivals during a ΔT length period $[(k-1)\Delta T, k\Delta T], k = 1, 2, \ldots$, are modeled as a list of arrival events, that is,

$$E_{k,\Delta T} = \{e_1, e_2, \ldots, e_n\}, \tag{4.9}$$

where $e_i = \{t_{e_i}, A_{e_i}\}(i = 1, 2, \ldots, n)$ represents an arrival event of a new vehicle, where t_{e_i} is the relative arrival time to the beginning of the period (i.e. $(k-1)\Delta T$), and A_{e_i} is a set of attributes of the new vehicle, such as speed, destination, route, and vehicle type. Note that in principle, such additional attributes could be part of the state vector, given observations are available to estimate these. This is beyond the scope of this case study.

Now, we can define the simulation state at time k as

$$S_k = \{s^i_k\}^{Q_k}_{i=1}, k = 0, 1, 2, \ldots, \tag{4.10}$$

where $s^i_k \equiv s^i(k\Delta T)$ (see equation 4.8), and $Q_k = N_0 + \sum_{j \in Inflow} \int_0^{k\Delta T} q_j(s)ds$ is the number of vehicles that have entered the simulation till time k; N_0 is the number of vehicles in the simulation initialization; $Inflow$ is a set of system boundaries that vehicles can flow in, and $q_j(s)$ is the the corresponding flow. Notice that S_k still includes the states of vehicles that have left the system. However, once a vehicle leaves the system (i.e. the simulation study area), we stop updating its state in the simulation. The evolution of the total state of the simulation (and thereby equation 4.1) can now be expressed as

$$S_k = SIM(S_{k-1}, E_{k,\Delta T}) + \nu_{k-1}, k = 1, 2, \ldots, \tag{4.11}$$

where SIM is the microscopic urban traffic simulation, that contains models for driving behavior and traffic control logic, and S_k, $E_{k,\Delta T}$ are defined in equation 4.10 and equation 4.9, respectively.

4.2.3.3 Available data

Suppose in an urban traffic network sensors are deployed at each inflow boundary of a lane and each stop line at signalized junctions, and these sensors can provide noisy vehicle passages. Besides, we assume that traffic signal timing parameters and periodic travel time observations are also available. In summary, the proposed data assimilation framework assumes the availability of minimally the following data:

- Vehicle passages. We assume that the data provided by sensors is event-based data, i.e., a sequence of individual vehicle's passage times. We assume these event-based data contain errors such as miss-counts (a vehicle passes by, but the sensor fails to detect its passage) and over-counts (no vehicles pass by, but the sensor reports a passage).

- Traffic signal timing parameters. These data are input when running the traffic simulation model.

- Periodic travel time observations. These data are used in the correction method to estimate vehicle accumulation.

Notice that we assume sensors are deployed at both the inflow boundary and the advanced stop line in this case study. However, this assumption might not hold in real-life applications, e.g., some inflow boundaries are not measured, or some roadstretch does not have both inflow sensor and the advanced stop line sensor installed. In these cases, we need to design a novel method in our future studies to estimate vehicle accumulations probably aided by other types of data. This is certainly possible (i.e. incorporating additional data) in our framework as will be shown in section 5.2.2.2.

We assume data are available every ΔT time units. The size of the measurement interval ΔT has large influence on the data assimilation results. If ΔT is too large, the behavior predicted by the simulation model might diverge too much from the real behavior, and as a result, large errors will be obtained; however, if ΔT is too small, the computation burden will increase, but the estimation results will not necessarily be improved (Gu, 2010); therefore, a proper value of ΔT should be adopted. The effect of the measurement interval needs to be verified in the future research. In this case study, we choose ΔT as the cycle time of the traffic signal. The measurements at time k are denoted by

$$m_k^o = \{\{E_k^j\}_{j=1}^{N_s}, \{N_k^l\}_{l=1}^{\mathcal{L}}\}, k = 1, 2, \dots, \tag{4.12}$$

where N_s is the number of sensors, and E_k^j depicts the passage times obtained from sensor j during time interval $[(k-1)\Delta T, k\Delta T]$. Note that vehicle accumulation is not measured directly but estimated from vehicle counts, which will be explained in section 4.2.4.1. The symbol $\mathcal{L}$ denotes a set of road stretches for which such vehicle accumulation is available, and N_k^l depicts the vehicle accumulation on road stretch l at time k. Traffic signal timing data and travel time observations are not used in weight computation, therefore we omit them in equation 4.12.

4.2.3.4 Partial trajectory reconstruction using particle filters

Recall that particle filtering essentially approximates the posterior distribution $p(S_{0:k}|m_{1:k}^o)$, in which S_k is defined in equation 4.10. However, Q_k (see definition in equation 4.10) is random, and therefore the dimension $S_{0:k}$ is also random. This randomness results in the so-called variable dimension problem (Godsill et al., 2007), which *in principle* makes the sequential importance sampling algorithm introduced in section 2.2.3 not applicable to our problem. However, it turns out that, in line with Godsill et al. (2007), we formally proved that the variable dimension sequence has no tangible effect on weight updating in our case (see section 4.1.3), and that we therefore can safely apply the sequential importance sampling algorithm.

Since the measurements (equation 4.12) are not directly related to a system state at one specific time instant, but are related with a sequence of system states over a period of time, the measurement model is formalized as

$$m_k^o = g_k(S_{k-1:k}) + \varepsilon_k, k = 1, 2, \dots,$$

where $S_{k-1:k}$ represents a sequence of (intermediate) system states from time $k-1$ to time k (sampled every $\delta t << \Delta T$ time units during $[(k-1)\Delta T, k\Delta T]$). Note that $S_{k-1:k}$ essentially defines a set of (sampled) vehicle trajectories during $[(k-1)\Delta T, k\Delta T]$. Accordingly, this also implies that the particle weights should be updated as

$$w_k \propto p(m_k^o | S_{k-1:k}) w_{k-1}.$$

Note that although the state evolution after time k depends on S_k only, the application at hand requires us to store the system state trajectory for as many time periods as needed to reconstruct trajectories. Algorithm 3 describes in detail how a particle filter is applied to fulfill the trajectory reconstruction task. The main steps of the proposed algorithm are summarized as below.

- **Initialization**. In the initialization step (line 2 $\sim$ 5 in Algorithm 3), the i-th sample S_0^i is actually a guess of vehicles' locations and speeds in the network, i.e., $S_0^i = \{x_0^{i,j}, v_0^{i,j}\}_{j=1}^{N_0^i}$, where N_0^i is the vehicle accumulation (i.e. the number of vehicles) in the i-th sample at time $k = 0$.

- **Sampling**. After initialization, the microscopic traffic simulation model is run for one time step ΔT (i.e. until new measurements become available) to obtain a sample (line 8 in Algorithm 3). This is done for every particle. To run the simulation, inputs (vehicle arrivals) are reconstructed from the (noisy) passage times observed by sensors deployed at the inflow boundary. During the run, intermediate states $S_{k-1:k}^i$ are recorded, which represent a set of (sampled) vehicle trajectories that will be used in weight computation and subsequent trajectory reconstruction. Once a sample is generated, its weight is updated based on the newly available measurements (line 9 in Algorithm 3).

- **Resampling**. To solve the degeneracy problem, we resample the particles on a regular basis (see section 4.2.3.1).

- **Estimation**. We reconstruct partial trajectories from the particle with the highest weight. Note that we elaborate in detail in section 4.2.3.5 how these partial trajectories are subsequently concatenated to full trajectories.

4.2.3.5 Full trajectory reconstruction

The final step is now to concatenate partial trajectories over multiple measurement intervals into full trajectories over the desired time-space window. This full trajectory reconstruction procedure is graphically illustrated in Figure 4.2, in which for illustration purposes, only

Algorithm 3: The particle filter for vehicle trajectory reconstruction

```
1  % initialization of particles at $k = 0$
2  for $i = 1 : N_p$ do
3      generate the $i$-th sample $S_0^i$
4      set weight $w_0^i = 1/N_p$
5  end
6  % the sampling step for any time $k \geq 1$
7  for $i = 1 : N_p$ do
8      prediction step: reconstruct a sequence of vehicle arrivals $E_{k,\Delta T}^i$, and run the
         simulation for one time step ($\Delta T$) with initial state $S_{k-1}^i$, then $S_k^i$ is generated.
         The $i$-th particle $S_{0:k-1}^i$ is thus expanded to $S_{0:k}^i = \{S_{0:k-1}^i, S_k^i\}$; meanwhile,
         the intermediate states $S_{k-1:k}^i$ are also recorded for weight computation and
         trajectory reconstruction
9      update step: assign $S_{0:k}^i$ a weight: $w_k^i = p(m_k^o|S_{k-1:k}^i)w_{k-1}^i$
10 end
11 normalize the weights, denote them as $\{S_{0:k}^i, w_k^i\}_{i=1}^{N_p}$
12 % the resampling step (more details are explained in section 4.2.3.1)
13 $\{S_{0:k}^i, w_k^i\}_{i=1}^{N_p} = hierarchical\_resampling(\{S_{0:k}^i, w_k^i\}_{i=1}^{N_p})$
14 % output latest trajectories
15 find the particle with the maximum weight: $w_k^{i^*} = max\{w_k^i|i = 1, \ldots, N_p\}$
16 output trajectories recorded in the simulation which generated the $i^*$-th particle, and
     update previous reconstruction results; more details are presented in section 4.2.3.5
17 for $i = 1 : N_p$ do
18     $w_k^i = 1/N_p$
19 end
```

two particles are considered. The evolutions of particle 1 and particle 2 are depicted respectively at the top and in the middle of Figure 4.2, whereas the reconstructed vehicle trajectories are placed at the bottom part.

At time k, the first particle has the highest weight. Therefore, $\{T_{1,1}, T_{1,2}, T_{1,3}, T_{1,4}, T_{1,5}\}$ from particle 1 now constitute the reconstructed trajectories in time-space region $[(k-1)\Delta T, k\Delta T] \times [x_1, x_2]$, as shown in the bottom-left plot in Figure 4.2. Among these trajectories, $T_{1,1}$, $T_{1,2}$, and $T_{1,3}$ have passed cross-section x_2, and therefore, they are fully evaluated given all available evidence. This implies that information estimated by particle 1 at boundaries

$$\begin{aligned} &\{distance \in [x_1, x_2] \cap time = (k-1)\Delta T\} \\ &\{distance = x_1 \cap time \in [(k-1)\Delta T, t_1(T_{1,3})]\} \end{aligned} \tag{4.13}$$

is already the best estimation of the true boundary information, with $t_1(T_{1,3})$ denoting the time instant when the last full trajectory $T_{1,3}$ crosses x_1. We temporarily keep $T_{1,4}$ and $T_{1,5}$ in our reconstructed results. In case data assimilation stops at time k, the trajectory set $\{T_{1,1}, T_{1,2}, T_{1,3}, T_{1,4}, T_{1,5}\}$ is the best estimation based on the observed evidence; whereas in case new data (passages) are available in the next iteration, information at boundary $\{distance \in [x_1, x_2] \cap time = k\Delta T\}$ will be re-evaluated.

Since information at boundaries defined in equation 4.13 has been determined, we introduce an intermediate step as shown in the middle column in Figure 4.2. In this step, we check in each particle ($\neq 1$) whether there is a trajectory meeting the following two conditions:

- does it cross with boundary $\{distance \in [x_1, x_2] \cap time = k\Delta T\}$?
- does it originate from boundaries defined in equation 4.13?

If any trajectory meets these conditions, we delete it. As shown in Figure 4.2, we delete $T_{2,2}$ and $T_{2,3}$ from particle 2. This step ensures that in future iterations, newly reconstructed trajectories will not influence trajectories which were reconstructed in past iterations.

Now consider that at time $k+1$, the second particle has the highest weight. Therefore, we now populate time-space region $[k\Delta T, (k+1)\Delta T] \times [x_1, x_2]$ with trajectories extracted from particle 2, i.e., $\{T_{2,4}, T_{2,5}, T_{2,6}\}$. Note that we trace back the trajectories at boundary $\{distance \in [x_1, x_2] \cap time = k\Delta T\}$ to boundary $\{distance = x_1 \cap time = [t_1(T_{1,3}) + h_{min}, k\Delta T]\}$[4]; update the reconstruction results in the last iteration by deleting $T_{1,4}$ and $T_{1,5}$, and then concatenate the updated trajectories with the newly reconstructed trajectories (in this case $T_{2,4}$), as shown in the plot at the bottom-right of Figure 4.2. In this case one full trajectory (passing x_2) has been reconstructed, and we wait for new evidence in the next iteration.

4.2.4 Vehicle count correction method, specification of error models and weight computation

In this section, we discuss the method to estimate vehicle accumulation and we specify our assumptions related to the error models for the various data sources. On the basis thereof, we then return to the computation of the particle weights.

4.2.4.1 Estimating vehicle accumulation

Consider two detectors installed at x_1 (upstream) and x_2 (downstream) on a closed road-stretch (no entry or exits), both measuring flow $q_i(t), i = 1, 2$. Vehicle accumulation between x_1 and x_2 at any time instant t is equal to

$$N(t) = Q_1(t) - Q_2(t), \tag{4.14}$$

where $Q_i(t) = \int_0^t q_i(s)ds$ is the cumulative curve of the detector installed at $x_i, i = 1, 2$. The travel time from x_1 to x_2 of the n^{th} vehicle equals $T_r(n) = t_2(n) - t_1(n)$, where $t_i(n) = Q_i^{-1}(n), i = 1, 2$. In case of FIFO (first in, first out), T_r is exact, otherwise an approximation; $N(t)$ is exact in both cases. Unfortunately, if the detectors miss or over-count vehicle passages, that is $q_i^{true}(t) = q_i^{detector}(t) + \beta_{q_i}(t)$, with $\beta_{q_i}(t)$ an arbitrary noise term, both $N(t)$ and T_r will contain errors. Since $N(t)$ integrates consecutive flow counts (equation 4.14) this is an additive error that—regardless of the precise form of $\beta_{q_i}(t)$—yields an unbounded estimation bias for $N(t)$. In van Lint and Hoogendoorn

[4]We add h_{min} to $t_1(T_{1,3})$ to prevent collisions.

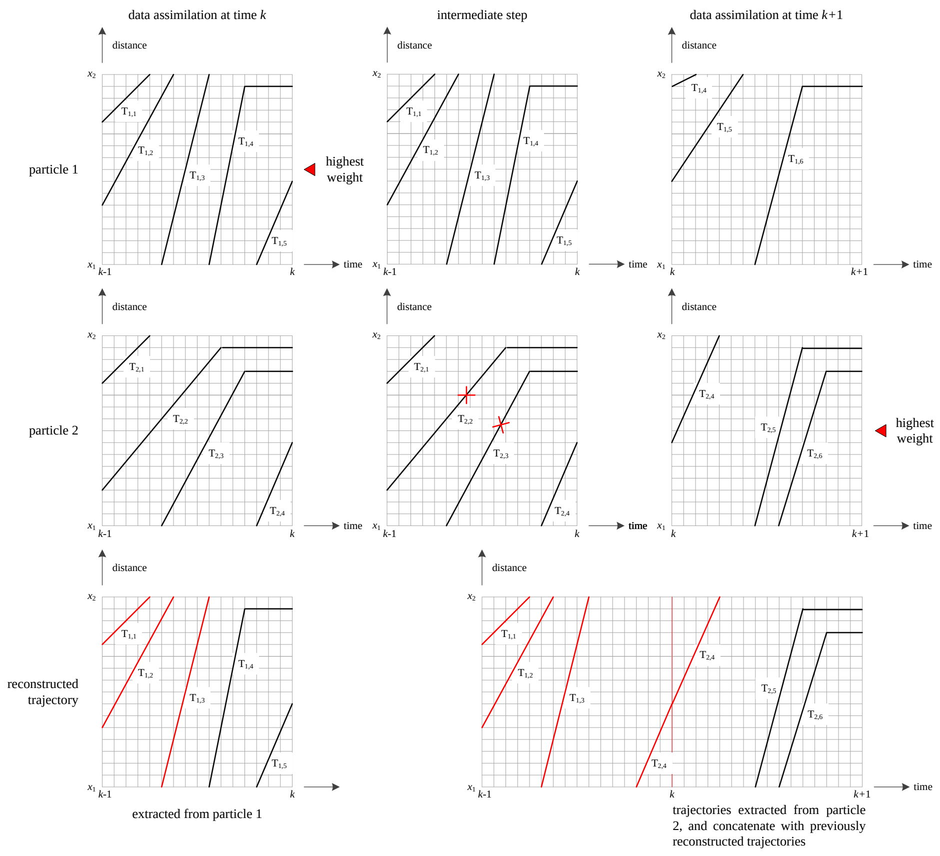


Figure 4.2: Illustration of full vehicle trajectory reconstruction (the duration between two consecutive time steps is ΔT)

(2015) a method is proposed which uses travel time observations to correct the flows and thereby the cumulative counts to estimate the vehicle accumulation. This method works along the same lines as the CUPRITE method proposed in Bhaskar et al. (2014). The basic idea is shown in Figure 4.3. Suppose the n_2^{th} vehicle passes x_2 at time t_2, the estimated travel time given by the cumulative curves is

$$\hat{T}_r(t_2) = T_r(n_2) = t_2 - t^* = t_2 - Q_1^{-1}(n_2)$$

Assume at t_2, we observe a travel time $T_r^{obs}(t_2)$. The travel time error will be

$$\varepsilon_T(t_2) = T_r^{obs}(t_2) - \hat{T}_r(t_2)$$

We can now use this travel time error to correct the error in the flows (and thus cumulative curves). We have two equivalent options (one error and two degrees of freedom), as shown in Figure 4.3. The first is to rotate the cumulative curve $Q_1(t)$ such that $Q_1(t) \rightarrow Q_1^*(t)$ and the second is to rotate the cumulative curve $Q_2(t)$ such that $Q_2(t) \rightarrow Q_2^*(t)$, in

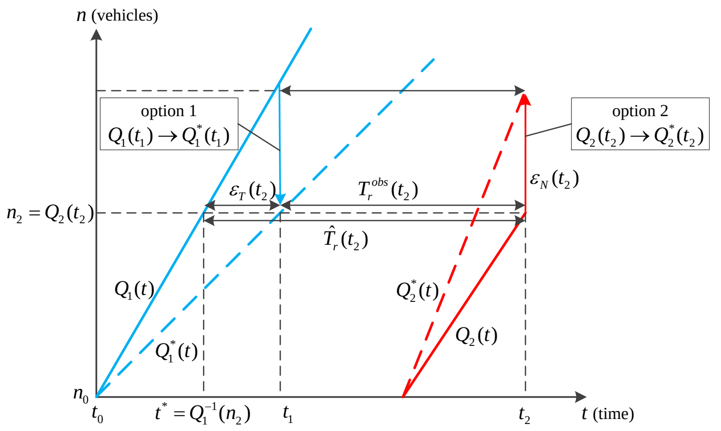


Figure 4.3: Vehicle accumulation estimation using the correction mechanism

both cases by an amount $\varepsilon_N(t_2)$. Let us assume we correct $Q_1(t)$ and that the previous correction took place for cumulative count $Q_1(t_0) = n_0$. Some straightforward algebra leads to

$$Q_1^*(t) = Q_1(t) + \frac{t - t_0}{t_1 - t_0}\varepsilon_N(t_2) \tag{4.15}$$

with

$$\varepsilon_N(t_2) = \frac{n_2 - n_0}{t^* - t_0}\varepsilon_T(t_2)$$

substituting (4.15) in (4.14) then gives

$$\hat{N}(t) = Q_1^*(t) - Q_2(t)$$

This method yields very good results in terms of estimating vehicle accumulation, even when limited travel time measurements are available. Note that since $q(t) = \frac{dQ}{dt}$, this correction implies that all flow measurements $q(t)$, with $t \in [t_0, t_1]$ are effectively adjusted by this correction. However, since we have two degrees of freedom (up- and downstream flows) to adjust for one error, these adjusted flows may not be more accurate than before the correction. To generate vehicles on the roadstretch we therefore use the estimated vehicle accumulation $\hat{N}(t)$ *only*, and *not* these corrected flows. Further below we detail how vehicles are generated and how the corrected cumulative counts of vehicles is used in our framework.

4.2.4.2 Error models for the different data sources

Vehicle accumulation

Although the method proposed above is able to remove the bias due to the additive errors in consecutive flow measurements, the resulting vehicle accumulations will still contain errors, which we can now assume to be zero mean (van Lint and Hoogendoorn, 2015). For the simulation case later on we approximate the discrete probability distribution of this estimation error by the histogram:

$$P(E = e) = \frac{Nr(e(t) = e)}{N}$$

in which $e(t)$ is the vehicle accumulation estimation error defined as

$$e(t) = N(t) - \hat{N}(t), t = t_0, t_1, \ldots, t_{N-1}$$

where $N(t)$ is the ground truth value (obtained in the simulation) of the vehicle accumulation at time instant t, and $\hat{N}(t)$ is its corresponding estimation. $Nr(e(t) = e)$ is the number of time instants when $e(t) = e$, and N is the number of time instants when the vehicle accumulation is estimated. Clearly, when applied in practice we need to assume a parameterized (discrete) distribution, for example fitted to this histogram.

Vehicle passages

There are two types of errors in vehicle passages, i.e., miss-counts and over-counts, and we model these two types of errors using two parameters:

- *detection accuracy* $p \in [0, 1]$, depicting the probability that a sensor successfully detects a vehicle passage. Conversely, the probability that the sensor misses the passage equals $1 - p$.
- *occurrence rate of over-count* λ. The number of over-counts during a time interval can be regarded as a Poisson distribution, and we define its occurrence rate as λ. As a result, the time between two consecutive over-count events is an Exponential distribution, with mean $\frac{1}{\lambda}$.

4.2.4.3 Arrival reconstruction

Assume that at time k, the available (noisy!) passage times from an inflow sensor are $E_k^j = \{t_1, t_2, \ldots, t_m\}$, where j is the index of the inflow sensor (see equation 4.12). The problem we are going to solve in this section is to reconstruct a possible passage sequence based on the observation and the error model. We first apply a few simple rules to 'clean' the data from obvious errors. For example, if the time headway between two consecutive passage times on a single lane is (much) smaller than the minimum time headway, one of them is likely an over-count. A passage observed at the stop-line of an intersection during the red phase is over likely an over-count. In both cases, we can clean the data by deleting these unlikely passage times. After this preprocessing step, we reconstruct a possible passage sequence E_k^j that is probable under the assumed error models for miss- and over-counts.

To this end, we first define a two-dimensional random variable: $X = (N_m, N_o)$, where N_m represents the number of miss-counts, and N_o the number of over-counts. We compute the joint probability distribution $p(X)$ using Algorithm 4. Once $p(X)$ is available, we draw (n_m, n_o) from $p(X)$ to reconstruct a possible passage sequence. Then we reconstruct passage times as follows:

- randomly delete n_o elements in E_k^j;
- randomly generate n_m time instants between $[(k-1)\Delta T, k\Delta T]$, until the time headways meet some predefined requirements (i.e., resultant time headway is larger than the minimum time headway); insert them into E_k^j.

Algorithm 4: Compute the joint probability distribution of miss-count and over-count

```
Input: noisy passage times: $E_k^j$; sensor error model parameters: $p, \lambda$
Output: joint probability distribution: $p(X)$
% determine the probability distribution of the number of over-count $p(N_o)$
for $n_o \in \mathbb{N} \wedge n_o \leq |E_k^j|$ do
    compute $p_{n_o} = \frac{(\lambda \Delta T)^{n_o} e^{-\lambda \Delta T}}{n_o!}$
    if $p_{n_o} \geq 10^{-3}$ then
        $P(N_o = n_o) = p_{n_o}$
    end
end
normalize $p(N_o)$ to make it a probability distribution
for every possible $n_o$ do
    determine the maximum number of miss-counts:
      $M = floor(\Delta T/h_m) - |E_k^j| + n_o$, where $\Delta T$ is the duration of the period
      during which $E_k^j$ was collected, $h_m$ is the minimum time headway, and
      $floor(x)$ returns the greatest integer less than or equal to $x$
    for $n_m \in \mathbb{N} \wedge n_m \leq M$ do
        compute $p_{n_m} = C_{|E_k^j|-n_o+n_m}^{|E_k^j|-n_o} p^{|E_k^j|-n_o}(1-p)^{n_m}$
        if $p_{n_m} \geq 10^{-3}$ then
            $P(N_m = n_m) = p_{n_m}$
        end
    end
    normalize $p(N_m)$ to make it a probability distribution
    for every possible $n_m$ do
        $P(X = (n_m, n_o)) = p_{n_m} \times p_{n_o}$
    end
end
normalize $p(X)$ to make it a probability distribution
```

4.2.4.4 Weight computation: utilizing the error models

We can now return to the computation of the particle filter weights, which depends on the error models we introduced above. Recall that when a sample is generated, its weight is updated based on newly available measurement $m_k^o = \{\{E_k^j\}_{j=1}^{N_s}, \{N_k^l\}_{l=1}^{\mathcal{L}}\}$, where N_s is the number of sensors, and E_k^j depicts the passage times obtained from sensor j; $\mathcal{L}$ denotes the set of road stretches, and N_k^l is the vehicle accumulation estimated by the correction method. Since the two types of data are conditionally independent given $S_{k-1:k}$, we have $p(m_k^o|S_{k-1:k}) = p(\{E_k^j\}_{j=1}^{N_s}|S_{k-1:k})p(\{N_k^l\}_{l=1}^{\mathcal{L}}|S_{k-1:k})$.

Assume that at time k, the available observation from the j-th sensor is $E_k^j = \{t_1, t_2, \ldots, t_m\}$; at the same time, the simulated value from the i-th particle is $E_k^{i,j} = \{t'_1, t'_2, \ldots, t'_{m_i}\}$. $E_k^{i,j}$ can be obtained by linear interpolation based on $S_{k-1:k}^i$ or by simply recording passage times in the sampling process. Now we show how to compute $p(E_k^j|E_k^{i,j})$.

Before the computation, we first need to figure out which element in E_k^j should be compared with which one in $E_k^{i,j}$. To solve this, a match procedure is developed, as shown in Algorithm 5. This match procedure is essentially the same with that in Wang and Hu (2015), which is used by the authors to match two groups of agents (without identity information) based on their locations. When applying the match procedure on E_k^j and $E_k^{i,j}$, the distance function is defined as $d(t, t') = |t - t'|$, where $t \in E_k^j, t' \in E_k^{i,j}$, and the threshold value is chosen as the minimum time headway. After the match, $p(E_k^j|E_k^{i,j})$ is calculated as follows:

- initialize $p(E_k^j|E_k^{i,j}) = 1.0$;
- assume $d_m = max\{d(t,t')|(t,t') \in MS = match(E_k^j, E_k^{i,j})\}$, we update $p(E_k^j|E_k^{i,j})$ by $p(E_k^j|E_k^{i,j}) = p(E_k^j|E_k^{i,j}) \times e^{-d_m}$;
- if we denote $T_{E_k^{i,j}}$ as the set of time instants in $E_k^{i,j}$ that are matched with some time instants in E_k^j, $E_k^{i,j} \setminus T_{E_k^{i,j}}$ will be the set of time instants that fail to be matched. Any time instant in $E_k^{i,j} \setminus T_{E_k^{i,j}}$ is regarded as a miss-count. Suppose the number of miss-count is n_m, we update $p(E_k^j|E_k^{i,j})$ by $p(E_k^j|E_k^{i,j}) = p(E_k^j|E_k^{i,j}) \times p^{|MS|} \times (1-p)^{n_m}$. Note that $|E_k^{i,j} \setminus T_{E_k^{i,j}}| = n_m$, $|MS| = |T_{E_k^{i,j}}|$;
- similarly, we denote $T_{E_k^j}$ as the set of time instants in E_k^j that are matched with some time instants in $E_k^{i,j}$, then any time instant in $E_k^j \setminus T_{E_k^j}$ is regarded as an over-count. Suppose the number of over-count is n_o, we update $p(E_k^j|E_k^{i,j})$ by $p(E_k^j|E_k^{i,j}) = p(E_k^j|E_k^{i,j}) \times p_{n_o}$, where $p_{n_o} = \frac{(\lambda \Delta T)^{n_o} e^{-\lambda \Delta T}}{n_o!}$.

When information from all sensors is computed, we have $p(\{E_k^j\}_{j=1}^{N_s}|S_{k-1:k}^i) = \prod_{j=1}^{N_s} p(E_k^j|E_k^{i,j})$, since measurements from different sensors are conditionally independent given $S_{k-1:k}^i$.

For vehicle accumulations, we have

$$p(\{N_k^l\}_{l=1}^{\mathcal{L}}|S_{k-1:k}^i) = \prod_{l \in \mathcal{L}} P(E^l = N_k^{i,l} - N_k^l) \tag{4.16}$$

where N_k^l is the vehicle accumulation on road stretch l at time k estimated by the correction method, while $N_k^{i,l}$ is the corresponding value in the i-th particle. $P(E^l)$ is the discrete probability distribution of the estimation error of the vehicle accumulation on road stretch l. Equation 4.16 holds since vehicle accumulations on different road segments are conditionally independent given $S^i_{k-1:k}$.

Algorithm 5: The $match$ procedure

```
Input: two sets: $X = \{x_1, x_2, \ldots, x_m\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, and a threshold
       $threshold$
Output: a matched set: $MS = \{(x, y) | x \in X, y \in Y\}$
% initialization
$MS \leftarrow \emptyset$
% compute distance between $x_i$ and $y_j$
foreach $x_i \in X, y_j \in Y$ do
    compute the distance between $x_i$ and $y_j$: $d(x_i, y_j)$
    put $d(x_i, y_j)$ in a list $L$
end
sort $L$ in ascending order
% select matched pairs
while $L$ is not empty do
    remove the first element in $L$, denoted by $d(x_{i_1}, y_{j_1})$
    if $d(x_{i_1}, y_{j_1}) \leq threshold$ then
        put $(x_{i_1}, y_{j_1})$ in $MS$
        for $(x_i, y_j) \in L$ do
            if $x_i = x_{i_1} \vee y_j = y_{j_1}$ then
                remove $(x_i, y_j)$ from $L$
            end
        end
    end
    else
        break
    end
end
```

4.3 Case study in the urban traffic system – experiment and results

4.3.1 Experimental setup

To test the framework we consider the area shown in Figure 4.4, which consists of two signalized intersections and six sensors (represented by green bars) that can detect individual vehicle passages. Each traffic light (*TLB* and *TLD*) has a fixed cycle time of 60 seconds, and the splits for red and green are 30 seconds each (as shown in the right part of Figure 4.4). The vehicle arrivals obey a Poisson process, and the mean of inter-arrival time is 6 seconds (undersaturated condition). When generating a vehicle, the probabilities that it goes to Sink 1, Sink 2 and Sink 3 are 60%, 20% and 20%, respectively. The reasons why we conduct the experiment in undersaturated conditions are as follows. First, the travel time measurements can be fed with little delay, and then they can be used to correct the cumulative counts timely before errors (in cumulative counts) accumulate to be very large. Second, observations from the inflow sensors will less likely to aggregate into groups in a cycle, and consequently, this incurs less uncertainty in the arrival reconstruction process.

In this chapter, we use the Intelligent Driver Model (IDM) (Treiber et al., 2000) as our car-following model. The IDM defines a vehicle's acceleration as

$$\dot{v}_{IDM}(s, v, \Delta v) = a\left\{1 - \left(\frac{v}{v_0}\right)^4 - \left(\frac{s^*(v, \Delta v)}{s}\right)^2\right\}$$

$$s^*(v, \Delta v) = s_0 + vT + \frac{v\Delta v}{2\sqrt{ab}}$$

where the parameters are assigned with typical values for city traffic (Treiber and Kesting, 2013), which are listed in Table 4.1.

Table 4.1: The parameters for the IDM

name	maximum acceleration $a\ (m/s^2)$	comfortable deceleration $b\ (m/s^2)$	minimum distance $s_0\ (m)$	safe time headway $T\ (s)$	desired speed $v_0\ (m/s)$
value	1.0	1.5	2.0	1.0	15.0

The objective is to apply the proposed data assimilation framework to reconstruct the trajectories of vehicles entering from A and leaving from D. To this end, we run the simulation for 1800 seconds (30 cycles), and record (ground truth) vehicle trajectories at 1 Hz along with (ground truth) passage times across each sensor. We allow for a warm up time of 240 seconds, and use 14 cycles (from $t = 240\ s$ to $t = 1080\ s$) for study, in which the passages from each sensor were processed into different datasets with different detection accuracy (p) and occurrence rate of over-count (λ), in order to imitate different sensor qualities. We set the measurement interval to $\Delta T = 60$ seconds, i.e. passage times become available every 60 seconds.

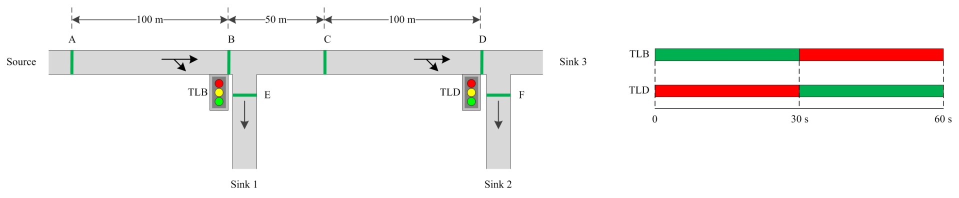


Figure 4.4: A road stretch with two signalized intersections

4.3.2 Evaluation criteria

We assess the data assimilation framework in line with the two aspects shown in Figure 4.1: correctness of the reconstructed vehicle passages at each sensor; and agreement of the reconstructed vehicle trajectories with the ground truth.

4.3.2.1 Vehicle passages and departure times per cycle and over multiple cycles

Let $E_i^j = \{t_1, t_2, \dots, t_{n_i}\}$ and $\hat{E}_i^j = \{\hat{t}_1, \hat{t}_2, \dots, \hat{t}_{\hat{n}_i}\}$ depict the ground truth and estimated passing times at sensor j during the i-th cycle respectively. We define the estimation error of the number of vehicle passages as

$$E^j_{number,i} = 100\% \times (n_i - \hat{n}_i)/n_i,$$

and the departure time error as

$$E^j_{match,i} = 100\% \times (n_i - \hat{n}_i^m)/n_i,$$

in which $\hat{n}_i^m$ is the number of time instants in E_i^j that are *matched* with certain time instants in $\hat{E}_i^j$ using the match procedure in Algorithm 5. In this procedure we use a threshold value for the minimum time headway h_{min} to determine which ground truth and estimated departure times most likely coincide. $E^j_{match,i}$ essentially defines the percentage of passages in E_i^j that are *not accurately* reconstructed.

Over multiple cycles we take the mean absolute percent error, that is,

$$\bar{E}^j_{number} = \frac{1}{n_{cycles}} \sum_{i=1}^{n_{cycles}} |E^j_{number,i}|, \bar{E}^j_{match} = \frac{1}{n_{cycles}} \sum_{i=1}^{n_{cycles}} |E^j_{match,i}| \quad (4.17)$$

4.3.2.2 Generalized flow and density

To assess the quality of vehicle trajectory reconstruction at the macroscopic level, we compare the generalized flow q and density k for a time-space region using Edie's well known definitions (Edie, 1963):

$$q = \frac{\sum_i d_i}{XT}, k = \frac{\sum_i r_i}{XT},$$

which, using the variables in Figure 4.5a, can be computed as

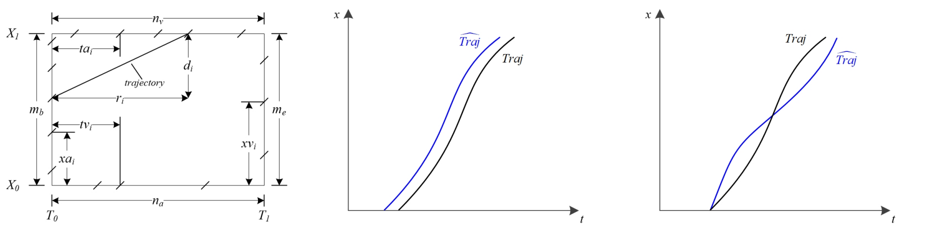
$$\begin{aligned} q &= \frac{n_v X - \sum_{i=1}^{m_b} xa_i + \sum_{i=1}^{m_e} xv_i}{XT} \\ k &= \frac{m_e T - \sum_{i=1}^{n_a} ta_i + \sum_{i=1}^{n_v} tv_i}{XT}, \end{aligned} \quad (4.18)$$

where n_a is the number of arrivals at cross-section X_0, n_v is the number of departures at cross-section X_1, m_b is the vehicle accumulation at T_0, and m_e is the vehicle accumulation at T_1. The corresponding estimation errors for flow and density in the i-th cycle are defined as percent errors:

$$E_{flow,i} = 100\% \times (q_i - \hat{q}_i)/q_i$$
$$E_{density,i} = 100\% \times (k_i - \hat{k}_i)/k_i$$

where q_i, k_i are the ground truth values of flow and density in the i-th cycle, respectively, and $\hat{q}_i$, $\hat{k}_i$ are their corresponding estimated values. Finally, again over multiple cycles we take the mean absolute percent error, that is,

$$\bar{E}_{flow} = \frac{1}{n_{cycles}} \sum_{i=1}^{n_{cycles}} |E_{flow,i}|, \bar{E}_{density} = \frac{1}{n_{cycles}} \sum_{i=1}^{n_{cycles}} |E_{density,i}| \quad (4.19)$$



(a) Macroscopic agreement (b) Translation error only (c) Translation and distortion error

Figure 4.5: Evaluation of vehicle trajectory reconstruction

4.3.2.3 Agreement between estimated and ground-truth trajectories

To assess how well *a single* reconstructed vehicle trajectory $\widehat{Traj}$ fits to the ground truth $Traj$, we consider two error measures that capture the error in location and the error in (local) speed, that we term translation and distortion error respectively. The so called *translation error* quantifies how much effort is needed to shift $Traj$ along the time axis to obtain $\widehat{Traj}$. The less effort, the better the estimation. We define it as

$$E^t_{tr}(Traj, \widehat{Traj}) = \frac{1}{N_{\delta x}} \sum_{i=1}^{N_{\delta x}} \frac{|t(x_i) - \hat{t}(x_i)|}{TT(Traj)} \times 100\% \tag{4.20}$$

where $t(x)$, $\hat{t}(x)$ are the time instants when trajectory $Traj$ and its estimation $\widehat{Traj}$ cross location x, respectively; $N_{\delta x}$ is the number of discretized locations that $Traj$ and $\widehat{Traj}$ crosses; $TT(Traj)$ is the total travel time from the upstream to the downstream, measured from the ground truth trajectory $Traj$.

The so-called *distortion error* quantifies how much effort is needed to distort $Traj$ to obtain $\widehat{Traj}$. The less effort, the better the estimation. This distortion is the result of errors in speed along the trajectory. We define it as

$$E^v_{tr}(Traj, \widehat{Traj}) = \frac{1}{N_{\delta x}} \sum_{i=1}^{N_{\delta x}} \frac{|v(x_i) - \hat{v}(x_i)|}{\bar{v}(Traj)} \times 100\% \tag{4.21}$$

where $v(x)$, $\hat{v}(x)$ are the speeds when trajectory $Traj$ and its estimation $\widehat{Traj}$ cross location x, respectively; $\bar{v}(Traj)$ is the mean speed measured from the ground truth trajectory $Traj$ (i.e., total traveled distance divided by total travel time within the time-space region).

Figure 4.5b illustrates a case where a trajectory is perfectly estimated safe for a translation error. Figure 4.5c shows a case in which there is a distortion error (and by implication also a translation error). We finally also consider an overall error which combines the two effects (we arbitrarily give both error components equal weight):

$$E^{t,v}_{tr}(Traj, \widehat{Traj}) = \frac{E^t_{tr}(Traj, \widehat{Traj}) + E^v_{tr}(Traj, \widehat{Traj})}{2} \tag{4.22}$$

Finally, we define error measures for two *sets of trajectories*. Prior to definition, we need to figure out which trajectory in the set of ground truth trajectories $TS = \{Traj_i\}$ is estimated by which trajectory in the set of reconstructed trajectories $\widehat{TS} = \{\widehat{Traj}_j\}$. To this end, we denote the start point of trajectory $Traj_i \in TS$ as $(t_{0,i}, x_{0,i})$, and for a trajectory $\widehat{Traj}_j \in \widehat{TS}$, as $(\hat{t}_{0,j}, \hat{x}_{0,j})$. If the two start points are *close enough*, $\widehat{Traj}_j$ is regarded as the estimation of $Traj_i$. By 'close enough', we mean that one of the following conditions is met:

- $x_{0,i} = \hat{x}_{0,j} = X_0 \cap |t_{0,i} - \hat{t}_{0,j}| < h_{min}$ or
- $t_{0,i} = \hat{t}_{0,j} \cap |x_{0,i} - \hat{x}_{0,j}| < s_{min}$,

where h_{min} and s_{min} are the minimum time headway and the minimum space headway, respectively. As a result we now define the following three error measures for set $TS = \{Traj_i\}$ being estimated by set $\widehat{TS} = \{\widehat{Traj}_j\}$:

$$\begin{aligned}\bar{E}^t_{tr}(TS,\widehat{TS}) &= \frac{1}{n}\sum E^t_{tr}(Traj_i, \widehat{Traj}_{j_i}) \\ \bar{E}^v_{tr}(TS,\widehat{TS}) &= \frac{1}{n}\sum E^v_{tr}(Traj_i, \widehat{Traj}_{j_i}) \\ \bar{E}^{t,v}_{tr}(TS,\widehat{TS}) &= \frac{1}{n}\sum E^{t,v}_{tr}(Traj_i, \widehat{Traj}_{j_i})\end{aligned} \tag{4.23}$$

where $\widehat{Traj}_{j_i} \in \widehat{TS}$ is the estimation of $Traj_i \in TS$, and n is the number of vehicle trajectories in TS which have corresponding estimations in $\widehat{TS}$.

4.3.3 Results

In this section, we use a fixed detection accuracy of $p = 0.9$; an occurrence rate of over-counts $\lambda = 1/300\ s^{-1}$; and we assume that travel time observations are available every 3 minutes. Due to space limitations, we restrict the presentation to road stretch *AB* in Figure 4.4.

4.3.3.1 Reconstructed vehicle passages per cycle

The vehicle accumulation estimation results for the roadstretch between sensors *A* and *B* are shown in Figure 4.6, where the left plot shows the estimated vehicle accumulation, while the right plot depicts the associated histogram of—indeed approximately zero-mean!—estimation errors. Figure 4.7 shows the ground-truth, observed, estimated and matched numbers of vehicle passages at upstream sensor (*A*) and downstream sensor (*B*) for a sequence of 14 cycles, in blue, red, green, and yellow, respectively.

Table 4.2 lists the average errors (equation 4.17) over 14 cycles, which are 7.43% and 5.60% for sensor *A*; and 2.70% and 5.63% for sensor *B*, respectively. This implies that our method is able to reconstruct ca. 95% of the vehicle passages, which is a promising result, considering our dataset contains around 10% miss-counts and over-counts ($p = 0.9, \lambda = 1/300\ s^{-1}$).

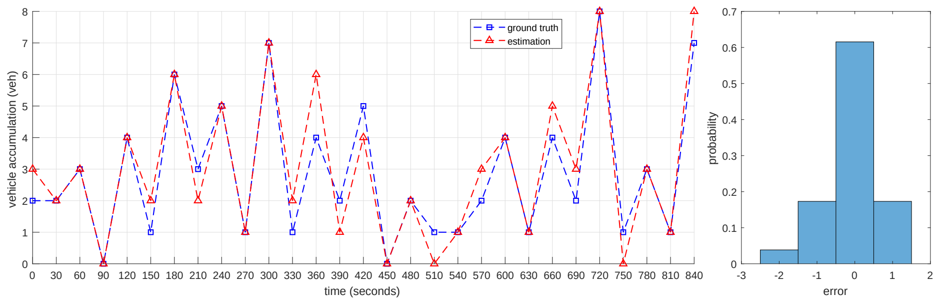


Figure 4.6: The vehicle accumulation estimation results for the roadstretch between sensors *A* and *B* using the correction method (the left plot shows the estimated vehicle accumulation, while the right plot depicts the histogram of the estimation errors)

Table 4.2: The estimation error of the number of vehicle passages and the percentage of passages that are not accurately reconstructed as defined in equation 4.17 computed over 14 cycles at sensors *A* and *B* respectively

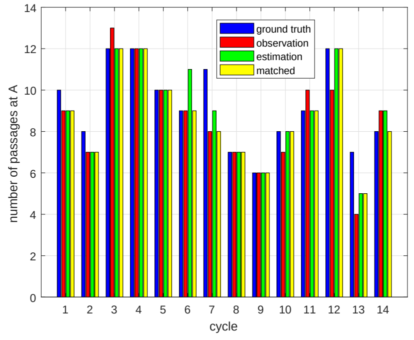
	$\bar{E}^{A}_{number}$	$\bar{E}^{A}_{match}$	$\bar{E}^{B}_{number}$	$\bar{E}^{B}_{match}$
performance	7.43%	5.60%	2.70%	5.63%

4.3.3.2 Generalized flow and density

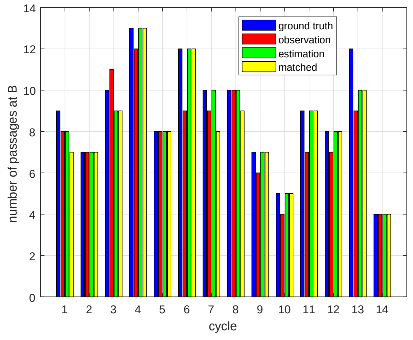
The estimation results for flow q and density k on roadstretch *AB* are shown in Figure 4.8a and Figure 4.8b, respectively. The average estimation error over 14 cycles (computed with equation 4.19) for flow q and density k are 5.26% and 5.23%, respectively.

It is insightful to analyze a few results in detail. For example, flow q and density k are overestimated in the 6-th cycle (300 s $\sim$ 360 s) while underestimated in the 13-th cycle (720 s $\sim$ 780 s). The (large) error in the 6-th cycle is mainly a result of the errors in the vehicle accumulation. As shown in the left plot in Figure 4.6, vehicle accumulation is overestimated in the end of the cycle ($t = 360$ s). Based on the definitions of q and k in equation 4.18, these errors will lead to overestimations of q and k. In the 13-th cycle, the (large) error is probably due to the miss-counts of sensor *A*, as shown in Figure 4.9. The vehicles missed at sensor *A* are also not fully observed at downstream sensors. In this case, the chance to successfully reconstruct their trajectories is very low. Consequently, the flow q and the density k will be underestimated because the missed vehicles do not show up at sensor *B* (n_v thus has a smaller value in the computation of q) and their travel times are not included in the computation of k as shown in equation 4.18.

In the 7-th cycle (360 s $\sim$ 420 s), the flow q is largely underestimated, which is mainly due to the overestimation of vehicle accumulation at $t = 360$ s. This overestimation in vehicle accumulation leads to a smaller number of vehicle arrivals upstream (sensor *A*) in order for the simulated passages to match the observed passages downstream (sensor *B*). As a consequence, q is assigned a smaller value (see the definition in equation 4.18).

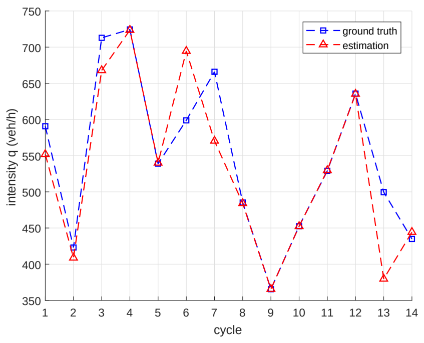


(a) Vehicle passages at *A*

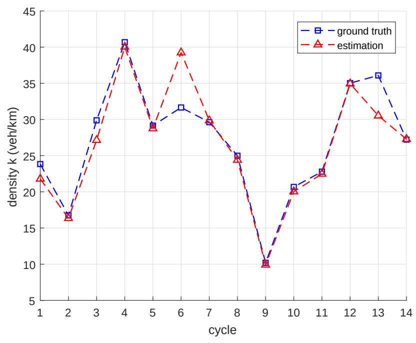


(b) Vehicle passages at *B*

Figure 4.7: Number of vehicle passages per cycle



(a) The estimated flow q



(b) The estimated density k

Figure 4.8: The estimated flow q and density k on roadstretch *AB*

4.3.3.3 Translation error and distortion error

In the time-space region on roadstretch *AB*, i.e., $[0\ s, 840\ s] \times [0\ m, 100\ m]$, there are 131 vehicle trajectories in the ground truth dataset; our method reconstructs 128 trajectories in the same region, and 123 of them can be matched with trajectories in the ground truth dataset. The translation error E^t_{tr} (equation 4.20), distortion error E^v_{tr} (equation 4.21), and overall error $E^{t,v}_{tr}$ (equation 4.22) for each pair of trajectories (123 pairs) are computed, and the histograms of these errors are depicted in Figure 4.10(a), Figure 4.10(b), and Figure 4.10(c), respectively. The average values for the translation error, the distortion error, and the overall error are $\bar{E}^t_{tr} = 2.56\%$, $\bar{E}^v_{tr} = 6.98\%$, and $\bar{E}^{t,v}_{tr} = 4.77\%$, respectively. Again these averages of about 5% are promising given the degree of sensor errors we imposed. Since the overall error $E^{t,v}_{tr}$ is a linear combination of E^t_{tr} and E^v_{tr}, its histogram follows the shape of the histogram of E^v_{tr}.

We make two general observations based on the results. First, the histograms in Figure 4.10 do illustrate that some pairs of trajectories have large errors (15% and more).

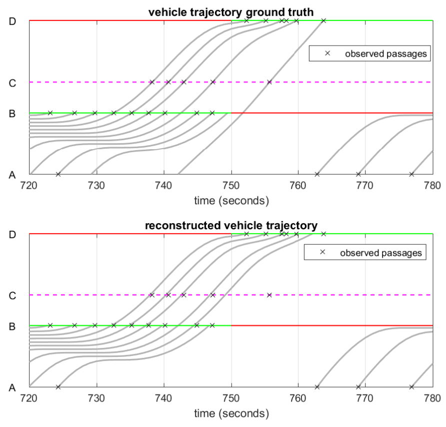


Figure 4.9: Vehicle trajectories in the 13-th cycle

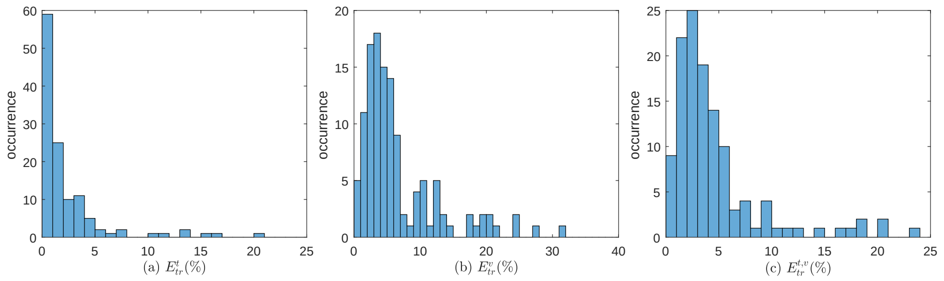
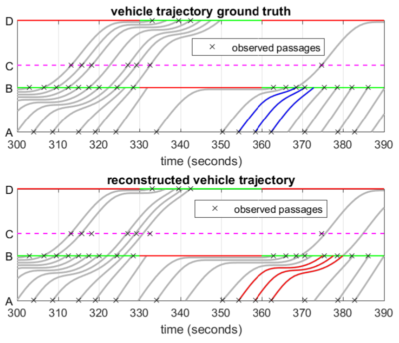


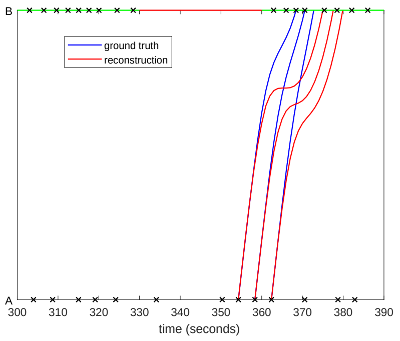
Figure 4.10: The histogram of translation error E^t_{tr}, distortion error E^v_{tr}, and overall error $E^{t,v}_{tr} = (E^t_{tr} + E^v_{tr})/2$ (123 pairs of trajectories in total; the width of each bin is 1.0%)

These pairs of trajectories mainly lie in the 6-th cycle ($300\ s \sim 360\ s$) and the 7-th cycle ($360\ s \sim 420\ s$), which are depicted in Figure 4.11. As shown in the left plot in Figure 4.6, the corresponding vehicle accumulation is overestimated in the end of the 6-th cycle ($t = 360\ s$). This results in the (false) reconstruction of two trajectories (originating from $t \approx 330\ s$ and $t \approx 340\ s$) in order to match the vehicle accumulation at $t = 360\ s$. Essentially the method falsely reconstructs a queue, yielding strongly sloped trajectories after $t \approx 350\ s$ until that queue is resolved. This discrepancy in trajectory shape leads to a large translation and distortion error.

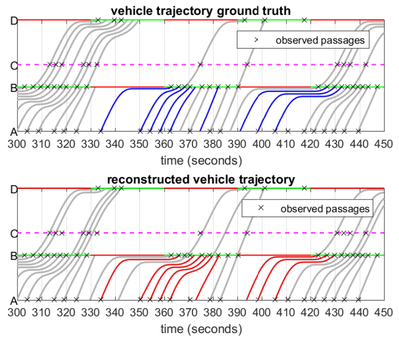
Second, and related to this point, the results indicate that the distortion errors are in general larger than the translation errors. This makes intuitively sense: the data (passages and accumulation) gives direct information on how much vehicles we expect where and when. This information, however, can be reconstructed with an—in principle—infinite number of individual speed profiles. The simulation models applied constrain this large solution space to those trajectories that are plausible. We take the last pair of trajectories (originating from $t \approx 405\ s$) in Figure 4.11d as an example. We redraw this pair of trajectories in the left plot of Figure 4.12, and we show their time differences (see



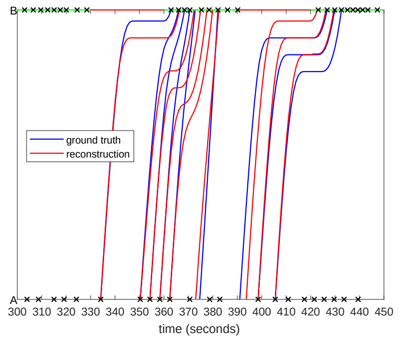
(a) General view of trajectories whose $E^t_{tr} > 15\%$



(b) Enlarged view of trajectories whose $E^t_{tr} > 15\%$



(c) General view of trajectories whose $E^v_{tr} > 15\%$



(d) Enlarged view of trajectories whose $E^v_{tr} > 15\%$

Figure 4.11: The trajectories whose translation/distortion error is larger than 15%

equation 4.20) in the middle of Figure 4.12, and we depict their speed differences (see equation 4.21) in the right plot of Figure 4.12. The translation error and distortion error for this pair are $E^t_{tr} = 5.97\%$ and $E^v_{tr} = 31.16\%$, respectively. From Figure 4.12, we can see that $|v(x) - \hat{v}(x)|/\bar{v}$ is less robust to a small trajectory discrepancy than $|t(x) - \hat{t}(x)|/TT$. For example, at $x \in [0, 50]$ where the estimated trajectory is very close to the ground truth trajectory, the speed difference $|v(x) - \hat{v}(x)|/\bar{v}$ can reach above 20%, while time difference $|t(x) - \hat{t}(x)|/TT$ is only around 1%. Therefore, the distortion errors are in general larger than the translation errors. This also explains why the translation errors are not as large as the distortion errors for the last three pairs of trajectories in Figure 4.11d.

4.4 Conclusions

In an open system, since entities can flow in and flow out through the system boundaries, the number of entities in the system is a random variable, thus the dimension of the system state is random, so is the dimension of the state trajectory. This incurs the so-called variable dimension problem in particle filtering, which leads to inapplicability of the sequential importance sampling algorithm in discrete event simulations of open systems. In this

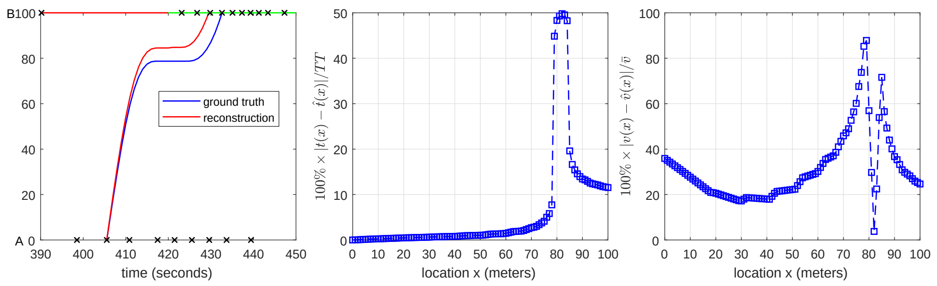


Figure 4.12: Time differences and speed differences of a pair of trajectories (the plot in the left shows the pair of trajectories, the plot in the middle depicts the time series of $100\% \times |t(x) - \hat{t}(x)|/TT$, and the plot in the right depicts the time series of $100\% \times |v(x) - \hat{v}(x)|/\bar{v}$; for this pair of trajectories, $E^t_{tr} = 5.97\%$, $E^v_{tr} = 31.16\%$)

chapter, we investigated the data assimilation problem in discrete event simulations of open systems, and solved the variable dimension problem as follows. The variable dimension system state is complemented with certain number of virtual entities in order to make it a fixed dimension state, and as a result, the state trajectory will have a fixed dimension. The standard sequential importance sampling algorithm can thus be applied to update the joint distribution of the state trajectory with fixed dimension. Samples in which the states of virtual entities are discarded will form the samples from the joint distribution of the state trajectory of interest that has a variable dimension. We proved that the weight update is again independent of these discarded states of virtual entities. Notice that the proof implicitly assumes that every entity has identical dimension of state, but it does not matter too much, since we can always augment each entity's state to the same dimension, and then apply the proof similarly. Though the proof finally implies that we can safely apply the sequential importance sampling algorithm in discrete event simulations of open systems, we should have a clear understanding of the problem and why it does not influence the application. The solution of the variable dimension problem proposed in this chapter is essentially applicable to simulations of general open systems with few modifications, since the proof in section 4.1.3 did not utilize any properties of discrete event systems.

To illustrate the working of the proposed data assimilation framework, a case in an urban traffic system is studied in which we reconstruct plausible vehicle trajectories on signalized urban arterials. The solution contributes to a generic (in the sense that any (ensemble of) microscopic simulation models (that implement car following, lane changing, crossing, etc.) can be used) particle filter based data assimilation framework for trajectory reconstruction on signalized urban arterials, in which relevant macroscopic variables (flow, density) can also be inferred. The experiment results show that the proposed data assimilation framework is indeed able to provide accurate estimation results in discrete event simulations of open systems. In the urban traffic case, the framework is able to reconstruct plausible vehicle trajectories under realistic error assumptions (detection accuracy $p = 0.9$, occurrence rate of over-count $\lambda = 1/300\ s^{-1}$) yielding good performance on both macro- and microscopic error measures (see section 4.3 for more details). The overall absolute percent errors on reconstructing passage counts are around 7%; whereas reconstructing the departure sequences over stop lines has an error around 5%. Given the error assumptions this is a very promising result—recall that without assimilation, these errors are unbounded! The errors in density and flow (using Edie's definitions (Edie, 1963)) are around 5% as well. Also the quality of the reconstructed trajectories in terms of matching locations and speeds is satisfactory, although here we observe a (small) subset of large(r) errors.

The solution proposed in this chapter finally completes the data assimilation framework for discrete event simulations, which means that the data assimilation framework proposed in chapter 3 can be applied to both closed and open systems. However, to effectively conduct data assimilation in open systems, two extra steps (compared with the data assimilation procedure in closed systems) are required. First, we need to estimate (directly from data or indirectly using an estimation method) the number of entities in the system, otherwise we will have an underdetermined problem, since the number of entities in an open system is unknown in advance. Second, we need to reconstruct entity arrivals at system boundaries based on (noisy) observations. One can also randomly generate a sequence of arrivals, but this would require a much larger number of particles to achieve similar performance.

CHAPTER 5

The particle filter based data assimilation framework – sensitivity analysis

In chapter 3 and chapter 4, we presented a particle filter based data assimilation framework for discrete event simulations of both closed and open systems. In this chapter, we use the data generated from the two cases studied in chapter 3 and chapter 4 to analyze the characteristics of the proposed data assimilation framework and its sensitivity to a number of important parameters that relate to the errors in the data and in the simulation model used, as well as to the number of particles employed. For each set of parameters, we run the data assimilation experiment 10 times with different random seeds.

5.1 Case study in the gold mine system[1]

5.1.1 Revisiting the performance indicators

As explained in section 3.5.2 in chapter 3, a set of performance indicators are defined to quantify the quality of the estimated truck arrival times. These performance indicators are summarized in Table 5.1.

[1]This section is part of a paper submitted to *Simulation: Transactions of the Society for Modeling and Simulation International*: Xie, X., Verbraeck, A.: *A particle filter based data assimilation framework for discrete event simulations*.

Table 5.1: Summary of the performance indicators for truck arrival estimation

symbol	unit	description
$\bar{E}$	–	average estimation error of the dimension of the state trajectory; see equation 3.27
SR	%	success rate, i.e., the percentage of truck arrivals been successfully estimated; see equation 3.29
WR	%	waste rate, i.e., percentage of clusters that fails to estimate an arrival; see equation 3.30
$\bar{d}$	min	average distance to the time instant when the probability density function is peaked; see equation 3.31
$\bar{P}$	%	average percentage that $P_{c_j}(t_j, \varepsilon)$ accounts for $P_{c_j}(t^*_{c_j}, \varepsilon)$; see equation 3.32

5.1.2 Effect of the data quality

In the gold mine case, only position data of entities (Elevator and Truck) is noisy, and the quality of the noisy position data is characterized by the standard deviation of the zero mean Gaussian noise, i.e. σ_e (for Elevator) and σ_t (for Truck). We vary σ_e and σ_t from 3.0 to 20.0; when retrieving the model state, we retrieve states through interpolation; for all experiments, we set $N_p = 2000$. The results are shown in Table 5.2. The results are in line with our expectations that the performance improves as the data becomes more accurate. We can conclude that the proposed method is quite robust to data errors. Even with a 20-meter standard deviation on entity positions, the performance does not degenerate too much. Specifically, the error of estimating the dimension of the state trajectory is 0.24, and the performance indicators of estimating the truck arrivals are 85.00% (success rate), 15.00% (waste rate), 0.83 minute (average distance), and 85.82% (average percentage), respectively.

Table 5.2: The influence of data quality, i.e. σ_e, σ_t, on the data assimilation results (states are retrieved through interpolation; $N_p = 2000$). In each table cell the median error over the 10 simulations is shown along with (in brackets underneath) the 25th and 75th percentiles

σ_e, σ_t	average dimension error $\bar{E}\,(-)$	success rate $SR\,(\%)$	waste rate $WR\,(\%)$	average distance $\bar{d}\,(min)$	average percentage $\bar{P}\,(\%)$
$\sigma_e = 3.0, \sigma_t = 3.0$	0.24 (0.20, 0.26)	90.00 (90.00, 95.00)	10.00 (5.00, 10.00)	0.54 (0.52, 0.61)	89.00 (87.56, 91.77)
$\sigma_e = 5.0, \sigma_t = 5.0$	0.22 (0.21, 0.32)	90.00 (90.00, 90.00)	10.00 (10.00, 10.00)	0.62 (0.59, 0.75)	88.53 (86.91, 90.48)
$\sigma_e = 10.0, \sigma_t = 10.0$	0.22 (0.21, 0.23)	90.00 (90.00, 95.00)	10.00 (5.00, 10.00)	0.73 (0.70, 0.85)	86.40 (85.31, 88.60)
$\sigma_e = 15.0, \sigma_t = 15.0$	0.24 (0.24, 0.26)	85.00 (85.00, 90.00)	15.00 (10.00, 15.00)	0.79 (0.77, 0.93)	86.50 (83.77, 87.98)
$\sigma_e = 20.0, \sigma_t = 20.0$	0.24 (0.22, 0.26)	85.00 (85.00, 90.00)	15.00 (10.00, 15.00)	0.83 (0.76, 0.93)	85.82 (83.61, 87.04)

5.1.3 Effect of the model errors

In the experiment in section 3.5, the model we used to carry out data assimilation is the same with that we used to generate the ground truth data, which implies that we have a perfect model of the reality. This is a very strong assumption. In this section, we investigate the data assimilation results in case that the model has errors. We build an imperfect model by simply changing the distribution of the drilling time of the miner from Triangular distribution with varying mode (i.e. perfect model) to a standard Triangular distribution with lower bound 15 minutes, upper bound 30 minutes, and mode 20 minutes (acting as the imperfect model). For all experiments, we set $\sigma_e = 3.0, \sigma_t = 3.0$, and $N_p = 2000$; states are retrieved through interpolation. The results are shown in Table 5.3.

Table 5.3: The influence of model quality on the data assimilation results (states are retrieved through interpolation; $\sigma_e = 3.0, \sigma_t = 3.0$; $N_p = 2000$). In each table cell the median error over the 10 simulations is shown along with (in brackets underneath) the 25^{th} and 75^{th} percentiles

model	**average dimension error** $\bar{E}\,(-)$	**success rate** $SR\,(\%)$	**waste rate** $WR\,(\%)$	**average distance** $\bar{d}\,(min)$	**average percentage** $\bar{P}\,(\%)$
perfect model (drilling time: Triangular distribution with varying mode)	0.24 (0.20, 0.26)	90.00 (90.00, 95.00)	10.00 (5.00, 10.00)	0.54 (0.52, 0.61)	89.00 (87.56, 91.77)
imperfect model (drilling time: standard Triangular distribution)	0.28 (0.25, 0.33)	90.00 (85.00, 90.00)	12.14 (10.00, 15.00)	0.63 (0.57, 0.69)	88.83 (86.65, 90.95)

The results in Table 5.3 reveal that the proposed method is robust with respect to model errors, although with the case involved, we cannot claim to have tested this exhaustively. In the case that we model one component incorrectly (i.e. with a different distribution), the overall performance is not significantly different with that we use a perfect model. Clearly, the accuracy of the data assimilation results largely depends on the validity of the simulation models used. In our case, this validity is evident, since the ground-truth data is produced by a similar model.

5.1.4 Effect of the number of particles

The influence of the number of particles (N_p) used on the data assimilation results is summarized in Table 5.4. As expected, the overall performance has an upward tendency as the number of particles increases. With more particles, components can explore more possibilities on their time advance values, and thus result in different event sequences and entity positions/phases, which will lead to a better coverage of the system state space.

Figure 5.1 depicts the error measures relative to those at $N_p = 2000$ (the ensemble size chosen in the gold mine case). The plot shows that the upward tendency in performance by increasing the number of particles is not proportional. A reduction in ensemble size from $N_p = 2000$ to $N_p = 100$ (i.e. 2000%) leads to an increase in error metrics ranging from around 7% (average percentage $\bar{P}$) to around 200% (waste rate WR); it seems that we could have safely decreased the number of particles in the gold mine case from $N_p = 2000$ to $N_p = 1000$ without a significant loss of accuracy in terms of all error measures.

Table 5.4: The influence of number of particles on the data assimilation results (states are retrieved through interpolation; $\sigma_e = 3.0, \sigma_t = 3.0$). In each table cell the median error over the 10 simulations is shown along with (in brackets underneath) the 25^{th} and 75^{th} percentiles

N_p	**average dimension error** $\bar{E}\,(-)$	**success rate** $SR\,(\%)$	**waste rate** $WR\,(\%)$	**average distance** $\bar{d}\,(min)$	**average percentage** $\bar{P}\,(\%)$
100	0.31 (0.28, 0.38)	70.00 (60.00, 70.00)	30.00 (30.00, 40.00)	0.81 (0.75, 1.08)	82.29 (80.76, 83.64)
400	0.28 (0.19, 0.33)	80.00 (75.00, 80.00)	20.00 (20.00, 25.00)	0.58 (0.50, 0.74)	85.96 (82.52, 89.62)
700	0.26 (0.22, 0.29)	82.50 (80.00, 90.00)	17.50 (10.00, 20.00)	0.62 (0.47, 0.76)	87.62 (86.82, 88.65)
1000	0.23 (0.18, 0.28)	87.50 (85.00, 90.00)	12.50 (10.00, 15.00)	0.67 (0.58, 0.80)	88.33 (87.99, 89.38)
1500	0.25 (0.20, 0.27)	87.50 (85.00, 90.00)	12.50 (10.00, 15.00)	0.64 (0.56, 0.75)	88.17 (87.36, 89.93)
2000	0.24 (0.20, 0.26)	90.00 (90.00, 95.00)	10.00 (5.00, 10.00)	0.54 (0.52, 0.61)	89.00 (87.56, 91.77)

5.2 Case study in the urban traffic system[2]

5.2.1 Revisiting the performance indicators

As explained in section 4.3.2 in chapter 4, a set of performance indicators are defined to quantify the quality of the reconstructed trajectories. These performance indicators are summarized in Table 5.5.

Table 5.5: Summary of the performance indicators for trajectory reconstruction

symbol	**unit**	**description**
$\bar{E}^j_{number}$	%	average estimation error of the number of vehicle passages at sensor $j \in \{A, B\}$; see equation 4.17
$\bar{E}^j_{match}$	%	average percentage of passages that are not accurately reconstructed at sensor $j \in \{A, B\}$; see equation 4.17
$\bar{E}_{flow}$	%	average estimation error of the generalized flow; see equation 4.19
$\bar{E}_{density}$	%	average estimation error of the generalized density; see equation 4.19
$\bar{E}^t_{tr}$	%	average translation error; defined between two sets of trajectories; see equation 4.23
$\bar{E}^v_{tr}$	%	average distortion error; defined between two sets of trajectories; see equation 4.23
$\bar{E}^{t,v}_{tr}$	%	average overall error; defined between two sets of trajectories; see equation 4.23

5.2.2 Effect of the data quality

In this section, we explore how the data affects the data assimilation results. Specifically, in section 5.2.2.1 we test the effect of data accuracy on the data assimilation results; while in section 5.2.2.2, we focus the effect of data quantity on the data assimilation results.

[2]This section is part of a paper submitted to *Transportation Research Part C: Emerging Technologies*: Xie, X., van Lint, J. W. C., Verbraeck, A.: *A generic data assimilation framework for vehicle trajectory reconstruction on signalized urban arterials using particle filters.*

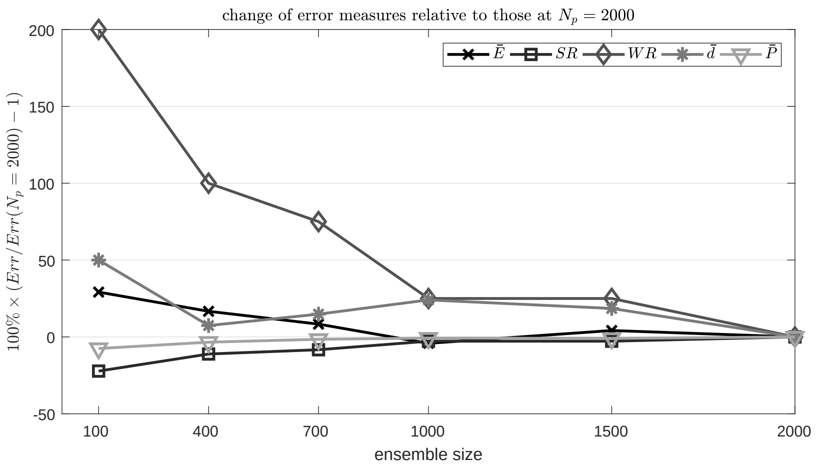


Figure 5.1: The influence of N_p on the data assimilation results (states are retrieved through interpolation; $\sigma_e = 3.0, \sigma_t = 3.0$); the performance indicators are relative to those at $N_p = 2000$

As explained in chapter 4, vehicle accumulations are key in the trajectory reconstruction method, but they are obtained indirectly via the correction method (van Lint and Hoogendoorn, 2015), therefore in section 5.2.2.3, we look into the effect of vehicle accumulations on the data assimilation results by assuming that we have error-free vehicle accumulations. Finally in section 5.2.2.4, we explore how the sampling rate of the travel time observations affect the data assimilation results.

5.2.2.1 Effect of the sensor quality

In the traffic case study, the sensor quality is characterized by two parameters: the detection accuracy p and the occurrence rate of over-count λ. When varying one of the two, the other is assigned a perfect value, i.e. when varying p, we set λ to $1/\infty\ s^{-1}$ (no over-counts); when varying λ, we set $p = 1$ (no miss-counts). For all experiments we set $N_p = 1000$. The results are shown in Table 5.6 and Table 5.7, respectively. The results are in line with our expectations that the performance improves as the sensors become more accurate. We can conclude that the proposed data assimilation framework is quite robust to data errors. Even with 40% miss-counts or one over-count every two minutes, all estimation errors are (well) within a 20% range.

5.2.2.2 Incorporating probe vehicle data into the framework

The proposed data assimilation framework relies on the vehicle accumulation and vehicle passages to reconstruct vehicle trajectories, but it is also open to other types of data if they are available. In this section, we briefly sketch how, for example, we can assimilate probe vehicle data (sampled vehicle trajectories) in the framework. To demonstrate this we consider the case where one such a sampled vehicle trajectory in each cycle is available,

Table 5.6: The influence of p on the data assimilation results ($\lambda = 1/\infty\ s^{-1}, N_p = 1000$). In each table cell the median error over the 10 simulations is shown along with (in brackets underneath) the 25^{th} and 75^{th} percentiles

	reconstructed vehicle passages				**generalized flow and density**		**translation/distortion errors**		
p	$\bar{E}^A_{number}$ (%)	$\bar{E}^A_{match}$ (%)	$\bar{E}^B_{number}$ (%)	$\bar{E}^B_{match}$ (%)	$\bar{E}_{flow}$ (%)	$\bar{E}_{density}$ (%)	$\bar{E}^t_{tr}$ (%)	$\bar{E}^v_{tr}$ (%)	$\bar{E}^{t,v}_{tr}$ (%)
0.6	14.15 (12.85, 15.01)	17.67 (16.89, 18.49)	6.65 (6.51, 7.69)	18.32 (16.34, 19.39)	11.96 (10.49, 13.31)	19.77 (17.88, 21.43)	7.49 (6.79, 8.75)	16.05 (15.52, 17.64)	11.94 (11.33, 12.97)
0.7	10.64 (8.94, 12.11)	12.12 (10.48, 15.49)	7.93 (5.25, 10.86)	13.21 (10.60, 14.54)	8.00 (7.39, 8.70)	12.22 (10.96, 16.42)	5.80 (5.37, 6.91)	16.10 (14.01, 17.64)	11.15 (9.59, 11.89)
0.8	7.99 (6.45, 9.45)	7.99 (7.71, 9.09)	5.55 (4.19, 7.36)	7.85 (7.37, 9.37)	7.10 (5.60, 7.53)	12.54 (10.00, 15.30)	4.72 (4.27, 5.06)	13.19 (11.97, 14.49)	8.72 (8.36, 9.78)
0.9	5.71 (4.42, 7.64)	2.96 (2.10, 3.35)	6.87 (6.44, 7.20)	3.73 (2.28, 4.12)	6.29 (4.88, 7.09)	12.11 (11.09, 14.32)	3.20 (2.37, 3.97)	12.18 (10.99, 12.86)	7.67 (6.68, 8.68)
1.0	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.79 (0.00, 0.79)	0.30 (0.13, 0.68)	1.38 (1.31, 1.43)	1.24 (1.21, 1.27)	4.95 (4.84, 5.28)	3.09 (3.02, 3.28)

Table 5.7: The influence of λ on the data assimilation results ($p = 1.0, N_p = 1000$). In each table cell the median error over the 10 simulations is shown along with (in brackets underneath) the 25^{th} and 75^{th} percentiles

	reconstructed vehicle passages				**generalized flow and density**		**translation/distortion errors**		
$\lambda(s^{-1})$	$\bar{E}^A_{number}$ (%)	$\bar{E}^A_{match}$ (%)	$\bar{E}^B_{number}$ (%)	$\bar{E}^B_{match}$ (%)	$\bar{E}_{flow}$ (%)	$\bar{E}_{density}$ (%)	$\bar{E}^t_{tr}$ (%)	$\bar{E}^v_{tr}$ (%)	$\bar{E}^{t,v}_{tr}$ (%)
$1/120$	10.33 (9.21, 13.14)	12.17 (11.05, 14.34)	10.08 (8.93, 12.61)	12.60 (10.76, 14.26)	10.64 (8.89, 12.44)	17.83 (16.59, 19.78)	3.78 (3.52, 4.00)	13.25 (12.73, 15.28)	8.42 (8.25, 9.79)
$1/180$	7.94 (6.39, 9.33)	8.57 (7.17, 11.14)	6.81 (5.58, 8.66)	8.24 (6.90, 9.99)	7.49 (5.85, 9.09)	13.25 (11.18, 17.10)	3.45 (2.97, 3.80)	12.08 (9.67, 13.37)	7.81 (6.32, 8.51)
$1/240$	6.79 (5.88, 7.31)	6.79 (5.88, 7.31)	6.19 (5.73, 6.92)	6.83 (6.23, 7.44)	6.91 (6.36, 7.78)	14.92 (14.38, 18.16)	3.11 (2.94, 3.24)	11.99 (10.85, 12.76)	7.56 (6.89, 8.15)
$1/300$	4.59 (4.37, 6.58)	4.19 (3.67, 6.00)	5.05 (4.52, 5.91)	6.01 (4.79, 6.90)	5.03 (4.56, 6.09)	8.11 (6.29, 10.21)	2.35 (2.05, 2.52)	9.21 (7.51, 9.98)	5.83 (4.82, 6.22)
$1/\infty$	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.79 (0.00, 0.79)	0.30 (0.13, 0.68)	1.38 (1.31, 1.43)	1.24 (1.21, 1.27)	4.95 (4.84, 5.28)	3.09 (3.02, 3.28)

see Figure 5.2. For each sampled trajectory, we assume we have available its position every second. Accordingly, we need to adjust the weight computation presented in section 4.2.4.4 by adding a process in which we consider three facts: 1) is there a trajectory in the particle which has a *similar* entering time (i.e. passing time at sensor *A*) with that of the sampled trajectory? By 'similar', we mean that the difference between the two entering times is smaller than the minimum time headway; 2) if the first condition is met, we check if the corresponding trajectory in the particle passes the same sequence of sensors as the sampled trajectory does; 3) if the first two conditions are both met, we finally look at their average position difference over all sampled time instants. Combining the three facts (differences), we assign a reasonable weight for each particle; since the vehicle passages and the probe vehicle data are independent with each other, the final weight is updated by simply multiplying the weight computed in section 4.2.4.4 by the weight obtained from the probe vehicle data.

The results are shown in Table 5.8. Clearly, probe vehicle data improves the performance in terms of most metrics, and expectedly, the most significant improvements relate to the translation & distortion errors, and number of reconstructed vehicle passages. Sampled trajectories are direct samples of what happens between two locations, the accuracy of individual vehicle trajectory reconstruction can thus be improved; furthermore, sampled trajectories contain information of passing times by any locations, which can help to detect

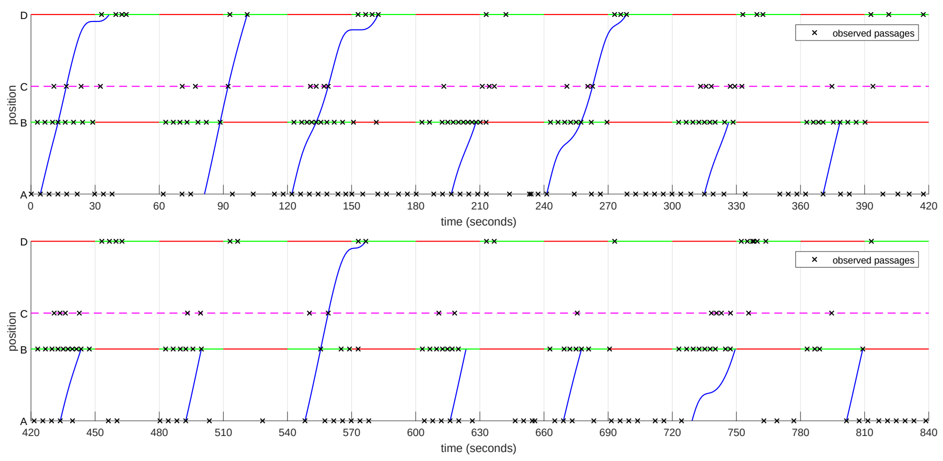


Figure 5.2: The sampled trajectories

miss-counts and remove over-counts, therefore we can reconstruct more accurate number of vehicle passages.

Table 5.8: The data assimilation results when sampled trajectories are available ($p = 0.9, \lambda = 1/300\ s^{-1}, N_p = 1000$). In each table cell the median error over the 10 simulations is shown along with (in brackets underneath) the 25th and 75th percentiles

	reconstructed vehicle passages				**generalized flow and density**		**translation/distortion errors**		
Assimilate sampled trajectories?	$\bar{E}^A_{number}$ (%)	$\bar{E}^A_{match}$ (%)	$\bar{E}^B_{number}$ (%)	$\bar{E}^B_{match}$ (%)	$\bar{E}_{flow}$ (%)	$\bar{E}_{density}$ (%)	$\bar{E}^t_{tr}$ (%)	$\bar{E}^v_{tr}$ (%)	$\bar{E}^{t,v}_{tr}$ (%)
no	9.37 (7.70, 11.18)	10.47 (7.81, 11.02)	5.90 (5.09, 6.64)	9.97 (7.15, 10.18)	8.37 (7.17, 9.65)	9.48 (7.64, 11.84)	3.57 (3.14, 3.73)	10.49 (10.16, 11.05)	7.04 (6.77, 7.37)
yes	7.33 (7.12, 8.91)	8.43 (8.03, 8.94)	4.45 (3.17, 5.83)	7.18 (6.92, 8.36)	6.90 (5.88, 8.48)	8.25 (6.73, 9.56)	2.79 (2.49, 2.91)	9.11 (8.39, 10.02)	6.08 (5.44, 6.42)

5.2.2.3 The difference between estimated and error-free vehicle accumulations

Vehicle accumulations are key in the trajectory reconstruction method, but difficult to come by. In the traffic study case, we used a correction method to estimate vehicle accumulations through cumulative counts and coarsely available travel time observations. Although this method yields good results, the estimates are certainly not error free. Table 5.9 summarizes the results when error-free vehicle accumulations are assimilated. Although the original results (top row in Table 5.9) are acceptable and around 10% or less, these results confirm that indeed all error measures (number of passages, departure times, flow, density and the three trajectory errors, respectively) decrease significantly, also in the statistical sense. By investing in vehicle counting (better loops, or via radar or other sensing devices), around 40% more accurate cumulative counts ($\bar{E}^j_{number}$, $\bar{E}^j_{match}, j \in \{A, B\}$); around 25-30% more accurate macroscopic estimates ($\bar{E}_{flow}$, $\bar{E}_{density}$); and also more accurate microscopic vehicle dynamics (trajectories) can be estimated, particularly when it comes to the correct location of the trajectories (38% improvement in $\bar{E}^t_{tr}$ versus 10% improvement in $\bar{E}^v_{tr}$ respectively).

Table 5.9: Data assimilation results when accurate vehicle accumulations are available. For all simulations we have $p = 0.9, \lambda = 1/300\ s^{-1}, N_p = 1000$ (i.e. 1000 particles). In each table cell the median error over the 10 simulations is shown along with (in brackets underneath) the 25^{th} and 75^{th} percentiles

	reconstructed vehicle passages				**generalized flow and density**		**translation/distortion errors**		
Estimated accumulations	$\bar{E}^A_{number}$ (%)	$\bar{E}^A_{match}$ (%)	$\bar{E}^B_{number}$ (%)	$\bar{E}^B_{match}$ (%)	$\bar{E}_{flow}$ (%)	$\bar{E}_{density}$ (%)	$\bar{E}^t_{tr}$ (%)	$\bar{E}^v_{tr}$ (%)	$\bar{E}^{t,v}_{tr}$ (%)
inaccurate accumulations	9.37	10.47	5.90	9.97	8.37	9.48	3.57	10.49	7.04
	(7.70, 11.18)	(7.81, 11.02)	(5.09, 6.64)	(7.15, 10.18)	(7.17, 9.65)	(7.64, 11.84)	(3.14, 3.73)	(10.16, 11.05)	(6.77, 7.37)
error-free accumulations	5.84	7.71	5.09	6.62	6.65	7.36	2.19	9.39	5.76
	(5.20, 6.86)	(6.56, 8.45)	(4.43, 6.41)	(5.74, 7.14)	(6.22, 7.23)	(6.95, 8.10)	(2.01, 2.27)	(8.75, 9.84)	(5.48, 6.20)

5.2.2.4 Effect of sampling rate of travel time observations

The experiment in chapter 4 assumes that the travel time observations are available every 3 minutes. In this section, we explore how the sampling rate of the travel time observations affect the data assimilation results. This effect is essentially predictable. On the one hand, van Lint and Hoogendoorn (2015) shows that more frequent travel time observations leads to more accurate vehicle accumulation estimations. On the other hand, the results in Table 5.9 reveal that with more accurate vehicle accumulations, we can reconstruct more accurate vehicle trajectories. Therefore, we can predict that with more frequent travel time observations, the data assimilation results will be improved. This prediction is confirmed by the experiment results shown in Table 5.10, which tell that the trajectory reconstruction results become more accurate as the travel time observations are fed more frequently, but this trend is below proportional.

Table 5.10: The influence of the sampling rate of the travel time observations. For all simulations we have $p = 0.9, \lambda = 1/300\ s^{-1}, N_p = 1000$ (i.e. 1000 particles). In each table cell the median error over the 10 simulations is shown along with (in brackets underneath) the 25^{th} and 75^{th} percentiles

sampling rate (minutes)	**reconstructed vehicle passages**				**generalized flow and density**		**translation/distortion errors**		
	$\bar{E}^A_{number}$ (%)	$\bar{E}^A_{match}$ (%)	$\bar{E}^B_{number}$ (%)	$\bar{E}^B_{match}$ (%)	$\bar{E}_{flow}$ (%)	$\bar{E}_{density}$ (%)	$\bar{E}^t_{tr}$ (%)	$\bar{E}^v_{tr}$ (%)	$\bar{E}^{t,v}_{tr}$ (%)
9	11.21	11.48	7.43	11.06	10.40	14.11	4.49	14.37	9.52
	(10.35, 11.94)	(10.31, 13.98)	(6.16, 8.13)	(9.87, 11.86)	(9.65, 10.88)	(11.21, 17.03)	(3.81, 4.82)	(12.25, 15.50)	(8.00, 10.16)
6	10.00	11.50	7.91	10.51	9.50	15.19	3.92	12.45	8.22
	(9.27, 11.62)	(10.75, 11.92)	(6.14, 9.20)	(9.81, 12.58)	(8.22, 10.33)	(10.85, 18.44)	(3.56, 4.20)	(11.90, 12.91)	(7.60, 8.56)
3	9.37	10.47	5.90	9.97	8.37	9.48	3.57	10.49	7.04
	(7.70, 11.18)	(7.81, 11.02)	(5.09, 6.64)	(7.15, 10.18)	(7.17, 9.65)	(7.64, 11.84)	(3.14, 3.73)	(10.16, 11.05)	(6.77, 7.37)

5.2.3 Effect of the model errors

In the experiment in chapter 4, the model we used to carry out data assimilation is the same with that we used to generate the ground truth data, which implies that we have a perfect model of the reality. This is a very strong assumption. In this section, we investigate the data assimilation results in case the model has errors. Specifically, we test two cases: 1) assimilating data generated by the IDM using the IDM with 5% calibration errors on the minimum spacing s_0 and the desired time headway T; 2) assimilating data generated by the IDM using a different car-following model, the Improved Full Velocity Difference

Model (IFVDM) (Treiber and Kesting, 2013), which defines a vehicle's acceleration as

$$v_{opt}(s) = max\{0, v_0 \frac{\tanh(\frac{s}{\Delta s} - \beta) + \tanh \beta}{1 + \tanh \beta}\}$$

$$\dot{v}_{IFVDM} = \frac{v_{opt}(s) - v}{\tau} - \frac{\gamma \Delta v}{max\{1, s/(v_0 T)\}}$$

where Δs is the transition width, β is the form factor, v_0 is the desired speed, τ is the adaption time, T is the time gap, and γ is the speed difference sensitivity. These parameters are assigned with the typical values of city traffic (Treiber and Kesting, 2013) as shown in the last row in Table 5.11. The parameters of the IDM which generates the ground truth data are $a = 1.0\,m/s^2,\ b = 1.5\,m/s^2,\ s_0 = 2.0\,m,\ T = 1.0\,s,\ v_0 = 15.0\,m/s$. The results are summarized in Table 5.12.

Table 5.11: The models used to assimilate the data generated by the IDM ($a = 1.0\,m/s^2,\ b = 1.5\,m/s^2,\ s_0 = 2.0\,m,\ T = 1.0\,s,\ v_0 = 15.0\,m/s$)

model	**parameters**
IDM (benchmark)	$a = 1.0\,m/s^2,\ b = 1.5\,m/s^2,\ s_0 = 2.0\,m,\ T = 1.0\,s,\ v_0 = 15.0\,m/s$
IDM ($s_0 - 5\%$)	$a = 1.0\,m/s^2,\ b = 1.5\,m/s^2,\ s_0 = 1.9\,m,\ T = 1.0\,s,\ v_0 = 15.0\,m/s$
IDM ($s_0 + 5\%$)	$a = 1.0\,m/s^2,\ b = 1.5\,m/s^2,\ s_0 = 2.1\,m,\ T = 1.0\,s,\ v_0 = 15.0\,m/s$
IDM ($T - 5\%$)	$a = 1.0\,m/s^2,\ b = 1.5\,m/s^2,\ s_0 = 2.0\,m,\ T = 0.95\,s,\ v_0 = 15.0\,m/s$
IDM ($T + 5\%$)	$a = 1.0\,m/s^2,\ b = 1.5\,m/s^2,\ s_0 = 2.0\,m,\ T = 1.05\,s,\ v_0 = 15.0\,m/s$
IFVDM	$\Delta s = 8.0\,m,\ \beta = 1.5,\ v_0 = 15.0\,m/s,\ \tau = 5.0\,s,\ T = 1.2\,s,\ \gamma = 0.6\,s^{-1}$

Table 5.12: The influence of model errors on the data assimilation results ($p = 0.9, \lambda = 1/300\ s^{-1}, N_p = 1000$). In each table cell the median error over the 10 simulations is shown along with (in brackets underneath) the 25th and 75th percentiles

	reconstructed vehicle passages				**generalized flow and density**		**translation/distortion errors**		
model type	$\bar{E}^A_{number}$ (%)	$\bar{E}^A_{match}$ (%)	$\bar{E}^B_{number}$ (%)	$\bar{E}^B_{match}$ (%)	$\bar{E}_{flow}$ (%)	$\bar{E}_{density}$ (%)	$\bar{E}^t_{tr}$ (%)	$\bar{E}^v_{tr}$ (%)	$\bar{E}^{t,v}_{tr}$ (%)
IDM (benchmark)	9.37 (7.70, 11.18)	10.47 (7.81, 11.02)	5.90 (5.09, 6.64)	9.97 (7.15, 10.18)	8.37 (7.17, 9.65)	9.48 (7.64, 11.84)	3.57 (3.14, 3.73)	10.49 (10.16, 11.05)	7.04 (6.77, 7.37)
IDM ($s_0 - 5\%$)	8.86 (8.57, 10.24)	10.06 (9.50, 11.05)	6.20 (5.41, 6.41)	9.52 (8.02, 10.92)	7.65 (6.96, 8.86)	10.67 (9.22, 12.08)	3.69 (3.27, 3.84)	11.29 (10.53, 12.34)	7.51 (6.98, 7.81)
IDM ($s_0 + 5\%$)	9.31 (8.74, 10.45)	9.49 (7.38, 10.41)	5.67 (4.94, 6.42)	8.94 (7.66, 9.63)	8.42 (7.14, 8.91)	8.67 (7.67, 9.85)	2.93 (2.51, 3.65)	10.40 (9.54, 11.45)	6.77 (6.09, 7.37)
IDM ($T - 5\%$)	9.45 (8.69, 10.05)	10.31 (8.35, 10.60)	5.69 (5.36, 7.41)	9.29 (8.10, 9.47)	8.01 (7.48, 9.29)	9.83 (8.23, 10.90)	3.41 (2.95, 3.63)	10.93 (9.34, 12.57)	7.27 (6.24, 8.10)
IDM ($T + 5\%$)	9.64 (8.53, 11.38)	9.98 (9.56, 11.73)	6.82 (5.65, 7.61)	9.69 (8.88, 11.11)	8.32 (7.90, 9.48)	10.77 (10.20, 13.09)	3.60 (3.38, 4.09)	11.14 (10.70, 11.77)	7.46 (7.33, 7.74)
IFVDM	13.04 (10.53, 13.57)	11.79 (9.89, 13.24)	9.46 (8.99, 12.06)	12.49 (9.98, 13.65)	10.42 (9.00, 11.86)	14.06 (12.57, 15.62)	5.72 (5.59, 5.82)	30.66 (28.57, 31.20)	18.14 (17.12, 18.54)

The results in Table 5.12 reveal that the proposed method is robust with respect to model errors, although with the two cases involved, we cannot claim to have tested this exhaustively. In the case that the model has 5% calibration errors, the overall performance is not significantly different with a perfect model; in the case that a different model is applied, the performance does not degenerate except for the distortion error $\bar{E}^v_{tr}$, which is likely a result of different acceleration processes between the two models (see Figure 5.3). Clearly, this will result in larger differences in individual speed dynamics and thus larger distortion errors. Since the overall error $E_{tr}^{t,v}$ is a linear combination of E_{tr}^{t} and E_{tr}^{v}, we also observe large error in the last column ($\bar{E}_{tr}^{t,v}$) of the last row in Table 5.12. This result does emphasize an important underlying point. Clearly, unless we have actual evidence (data), either macroscopically in the form of queue dynamics, or (even better) microscopically in the form of sample trajectories, the quality of the reconstruction completely depends on the validity of the microscopic models used in the framework for the specific case. As demonstrated in section 5.2.2.2, such evidence (probe vehicle data) can be naturally incorporated in the framework as well, and that it indeed improves the results.

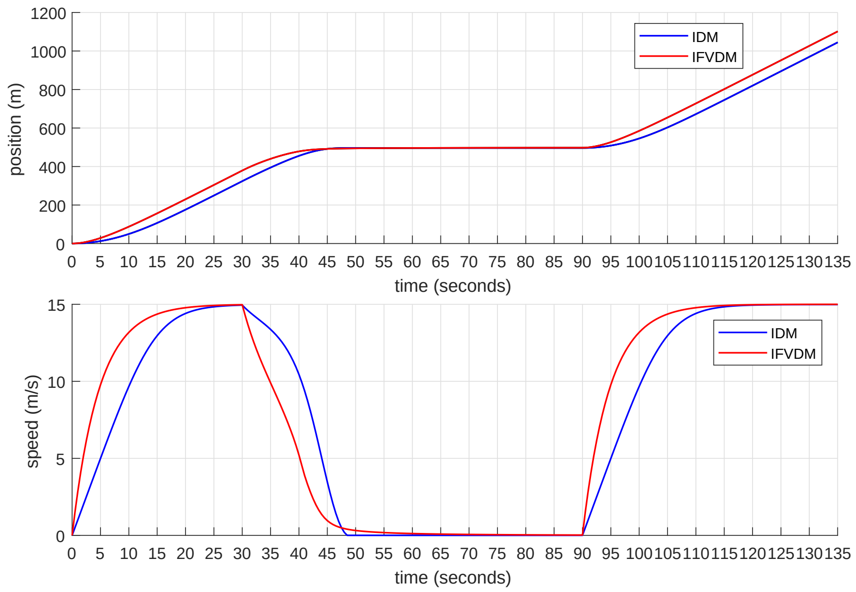


Figure 5.3: The position and speed time-series of approaching a traffic light (model parameters are given in Table 5.11). The initial speed and the initial position are both zero; the position of the traffic light is 500 m, and it switches to red at time $t = 30$ s, and then lasts for 60 seconds

5.2.4 Effect of the number of particles

The influence of the number of particles (N_p) used on the data assimilation results is summarized in Table 5.13. As expected, the overall performance has an upward tendency as the number of particles increases. With more particles, the proposed method can explore more possibilities on the entering times & speeds of individual vehicles and vehicle dispersion in the network, which leads to a better coverage of the system state space.

The good news, however, is that the trend is not proportional, which becomes apparent when looking at results in a graph, as presented in Figure 5.4(a) by means of absolute numbers (similar to those in Table 5.13) and in Figure 5.4(b) by dividing all the error measures by the corresponding performance at $N_p = 1000$ (the ensemble size chosen

Table 5.13: The influence of N_p on the data assimilation results ($p = 0.9$, $\lambda = 1/300\ s^{-1}$). In each table cell the median error over the 10 simulations is shown along with (in brackets underneath) the 25th and 75th percentiles

	reconstructed vehicle passages				generalized flow and density		translation/distortion errors		
N_p	$\bar{E}^A_{number}$ (%)	$\bar{E}^A_{match}$ (%)	$\bar{E}^B_{number}$ (%)	$\bar{E}^B_{match}$ (%)	$\bar{E}_{flow}$ (%)	$\bar{E}_{density}$ (%)	$\bar{E}^t_{tr}$ (%)	$\bar{E}^v_{tr}$ (%)	$\bar{E}^{t,v}_{tr}$ (%)
100	12.03 (11.18, 12.64)	11.78 (10.58, 12.37)	7.83 (7.42, 9.28)	10.87 (10.51, 11.60)	10.60 (10.45, 10.71)	14.54 (12.79, 15.10)	4.17 (4.02, 4.37)	13.46 (12.83, 13.84)	8.78 (8.47, 9.12)
400	10.41 (10.13, 10.74)	10.35 (9.76, 11.21)	6.67 (6.32, 6.90)	9.61 (9.31, 10.12)	9.06 (8.79, 9.69)	11.73 (10.09, 13.49)	3.65 (3.44, 3.93)	11.99 (11.31, 12.19)	7.85 (7.26, 7.99)
700	9.52 (8.23, 10.42)	9.23 (8.47, 9.82)	5.79 (5.13, 6.41)	8.57 (7.47, 8.99)	7.81 (7.35, 8.67)	10.36 (9.37, 11.41)	3.38 (3.15, 3.51)	10.82 (10.34, 11.60)	6.94 (6.70, 7.44)
1000	9.37 (7.70, 11.18)	10.47 (7.81, 11.02)	5.90 (5.09, 6.64)	9.97 (7.15, 10.18)	8.37 (7.17, 9.65)	9.48 (7.64, 11.84)	3.57 (3.14, 3.73)	10.49 (10.16, 11.05)	7.04 (6.77, 7.37)
2000	7.87 (6.61, 9.10)	7.24 (6.52, 8.70)	4.84 (3.47, 5.88)	6.70 (6.21, 7.72)	7.09 (6.14, 7.55)	7.64 (6.25, 9.17)	3.03 (2.72, 3.24)	9.89 (9.52, 10.24)	6.57 (6.36, 6.73)

in the urban traffic case). What Figure 5.4(b) shows, is that a reduction in ensemble size from $N_p = 1000$ to $N_p = 100$ (i.e. 1000%) leads to an increase in error metrics ranging from less than 10% (cumulative count at location *B*) to slightly over 50% (error in density); whereas doubling the ensemble size (to $N_p = 2000$) improves the performance no more than between 5 and 25%. Although relatively such gains may seem worthwhile, Figure 5.4(a) shows the actual gains are small. In our view, the increased error measures in the case of $N_p = 100$ (Table 5.13 top row) are well within acceptable bounds; it seems that we could have safely decreased the number of particles in our experiment from $N_p = 1000$ to $N_p = 700$ without a significant loss of accuracy in terms of all error measures.

5.3 Conclusions

In this chapter, we tested the particle filter based data assimilation framework on the studied cases that allowed us to analyze the characteristics of the framework and its sensitivity to a number of important parameters that relate to the errors in the data and in the simulation model used, as well as to the number of particles employed.

Sensitivity analysis with respect to data quality shows that the framework is quite robust to error assumptions since particle filters are assumption-free. In the gold mine case, even with a 20-meter standard deviation on entity positions, the performance does not degenerate too much. Specifically, the error of estimating the dimension of the state trajectory is 0.24, and the performance indicators of estimating the truck arrivals are 85.00% (success rate), 15.00% (waste rate), 0.83 minute (average distance), and 85.82% (average percentage), respectively (see Table 5.2). In the urban traffic case, the (macroscopic) estimation errors are consistently about half the percentages of miss- and over-counts; even with 40% miss-counts or one over-count every two minutes, all estimation errors are (well) within a 20% range (see Table 5.6 and Table 5.7). Similarly, the framework is robust to model errors (i.e. differences between the models generating the ground-truth data and the models used in the case studies), although we cannot claim to have tested this exhaustively. The result shows that using models with errors does not significantly affect the estimation results (see Table 5.3 and Table 5.12). This result does, however, emphasize an important underlying point. Clearly, unless we have actual evidence (data), the accuracy of the estimation results depends on the validity of the simulation models used in the framework

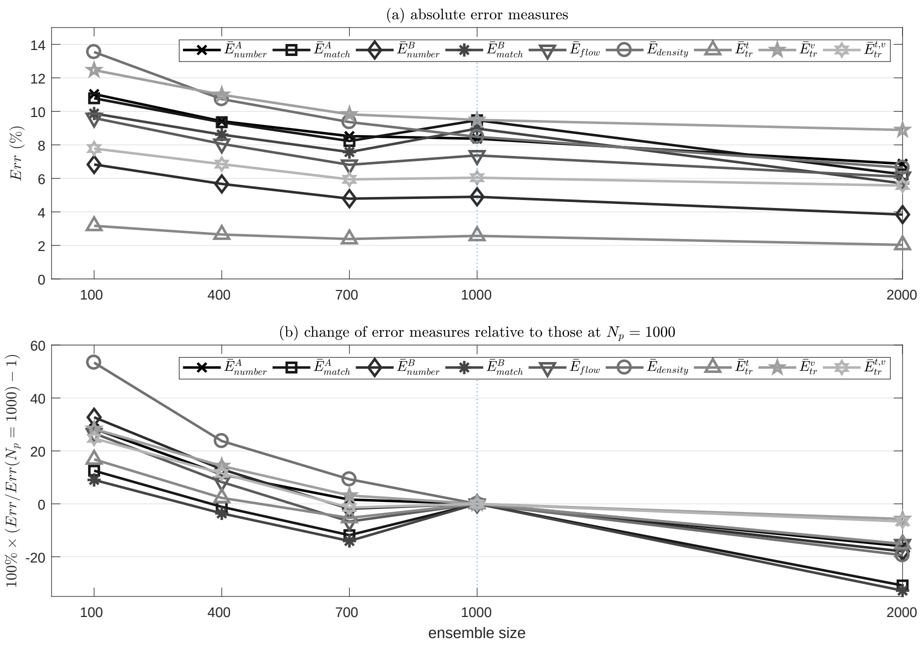


Figure 5.4: The influence of N_p on the data assimilation results ($p = 0.9, \lambda = 1/300\ s^{-1}$); the bottom plot shows error measures relative to those at $N_p = 1000$

for the specific case at hand. In our case, this validity is evident, since the ground-truth data is produced by a similar model. In real life, when the predictions given by the simulation model diverge too much from the real behavior of the system, it stands to reason that the estimation results will be farther away from the ground truth. In such a situation, we need to develop a more effective importance density, which can combine model predictions and real-time measurements together to propose new samples (remember that the importance density is defined as $q(s_k|s_{0:k-1}, m_{1:k})$). As demonstrated by Xue and Hu (2013), a new fire ignited during the simulation cannot be estimated if the system transition density is chosen as the importance density ((i.e. $s_k \sim q(s_k|s_{0:k-1}, m_{1:k}) = p(s_k|s_{k-1})$)), since the wildfire spread simulation does not model igniting new fires during the simulation process; however, after real-time measurements are properly incorporated to propose samples (i.e. $s_k \sim q(s_k|s_{0:k-1}, m_{1:k})$), the newly ignited fire can be successfully estimated.

Sensitivity analysis in terms of the number of particles employed reveals that with fixed data/model quality, the estimation errors decrease when the number of particles increases (see Figure 5.1 and Figure 5.4). With more particles, we can have a better coverage of the system state space, leading to more accurate estimation results. However, the trend is not proportional, which means that it is impossible to achieve error-free estimation results by continuously increasing the number of particles, since the quality of the estimation results is restricted by the quality of the data and the model involved in the data assimilation process. Increasing the number of particles also increases the computational complexity, therefore a balance between computational complexity and estimation accuracy should be achieved.

In summary, the quality of the data assimilation results is affected by the validity of the system model, the data quality, and the number of particles used. The more valid the simulation models, and the more accurate the sensors, and the higher the number of the particles, the better the estimation results. This implies several important research lines, i.e., developing simulation models that can make more valid predictions of the real system behavior, and developing more advanced sensor technologies that can provide more accurate measurement data of the real systems, and developing a parallel and distributed version of the proposed data assimilation framework to effectively deal with more complex scenarios.

CHAPTER 6

Conclusions and Future Research

Enabled by the increased availability of data, the data assimilation technique (Bouttier and Courtier, 1999; Nichols, 2003), which incorporates measured observations into a dynamical system model to produce a time sequence of estimated system states, gains popularity. The main reason is that it can produce more accurate estimation results than using either a simulation model or the measurements. Due to this benefit, the data assimilation technique has been applied in many continuous systems applications, but very little data assimilation research has been found for discrete event simulations. With the application of new sensor technologies and communication solutions, such as smart sensors, or Internet of Things (Atzori et al., 2010), the availability of data for discrete event systems has increased as well, such as data from machines and processes (Lee et al., 2013), or high-resolution event data in traffic (Wu and Liu, 2014). The increased data availability for discrete event systems but the lack of related data assimilation techniques thus motivated this work on data assimilation in discrete event simulations.

Since discrete event simulations are highly nonlinear, non-Gaussian systems, particle filters are used to conduct data assimilation in discrete event simulations. However, applying particle filtering in discrete event simulations still encounters several theoretical and practical problems, such as the state retrieval problem (discrete event simulation models have a piecewise constant state trajectory, so the retrieved state was updated at a past time instant, with which inaccurate estimation results will be obtained), the variable dimension problem (the dimension of the state trajectory during a fixed time interval is a random variable, leading to inapplicability of the standard sequential importance sampling algorithm), and the processing of non-numerical data. In this research, we presented a particle filter based data assimilation framework for discrete event simulations, in which the aforementioned problems are addressed. Besides, we analyzed the characteristics of the proposed data assimilation framework and its sensitivity to a number of important parameters in the framework, such as model errors, data quality, and the number of particles employed.

6.1 Research findings

6.1.1 Answers to research questions

In this section, we summarize the main research findings to answer the two research questions proposed in chapter 1.

RQ 1*: What existing or adapted data assimilation technique is suitable for discrete event simulations?*

Since particle filters are able to approximate arbitrary probability densities and have no assumption about the properties of the system model, they are *in principle* applicable to discrete event simulations, which show highly nonlinear, non-Gaussian behavior. However, applying particle filtering in discrete event simulations still encounters several problems, such as the state retrieval problem, and the variable dimension problem, as explained in chapter 2. In this research, we proposed a particle filter based data assimilation framework for discrete event simulations, in which these problems can be solved. In this data assimilation framework, we assume that the measurements available at (discrete) time k are distributed over the last measurement interval (i.e. data fed at time k can contain observations occurring at any time instant during $[(k-1)\Delta T, k\Delta T]$, where ΔT is the measurement interval), implying that the measurements are dependent on the state transitions during that measurement interval. The problems and corresponding solutions are summarized below:

- The first problem is the *state retrieval problem*, which means that the model state retrieved from a discrete event simulation model is a combination of sequential states of atomic components that were updated at past time instants. This problem is a result of the mismatch between the discrete event state transition process and the measurement process. In a discrete event simulation, state updates happen locally and asynchronously within each atomic model component; the system state takes a new value when one of its components has a state update. Therefore the duration between two consecutive state updates (at both the coupled level and the atomic level) is usually not fixed, which is thus asynchronous with the measurement process that usually feeds data at fixed times (although random arrivals of data are also possible). With outdated states, inaccurate estimation results will be obtained. Our solution is to introduce an interpolation operation which interpolates state values based on the system states updated within a time interval around the time instant when the operation is invoked in order to obtain updated state values (see sections 3.2.2 and 3.3.3, and also the beginning of section 4.2). The size of the time interval is determined by the interpolation method employed.
- The second problem is the *variable dimension problem*, which means that the dimension of the state trajectory $s_{0:k}$ (see definition in equation 3.13) is a random

variable. This problem arises due to two reasons. The first reason is that the duration between two consecutive (system) state updates is not fixed, therefore the number of state points during a fixed time interval $[0, k\Delta T]$ is random; the second reason is that in open systems, where entities can flow in and flow out through the system boundaries, the number of entities in the system is random, and as a result, the dimension of the system state is random, so is the state trajectory. The variable dimension of $s_{0:k}$ will lead to inapplicability of the standard sequential importance sampling algorithm which updates $p(s_{0:k}|m_{1:k})$. In this research, we solved this problem in the following way. We first make the variable dimension state trajectory to have a fixed dimension with certain extensions. The standard sequential importance sampling algorithm can thus be applied to update the joint distribution of the extended state trajectory with fixed dimension. Samples in which the extensions are discarded will form the samples from the joint distribution of the state trajectory of interest that has a variable dimension. In section 3.2.3 and section 4.1.3, we proved that the discarded extensions have no tangible effect on the weight update. This result implies that in practice we can safely apply the sequential importance sampling algorithm to update $p(s_{0:k}|m_{1:k})$ where $s_{0:k}$ has a variable dimension (no matter the discrete event system is open or closed). The proposed data assimilation framework can therefore be applied to discrete event simulations of both closed and open systems.

We use two case studies to demonstrate the working of the proposed data assimilation framework. The first case studied a gold mine system (closed system), in which noisy data (event sequences, entity positions) was assimilated into a gold mine simulation to estimate the truck arrival times at the bottom of the vertical shaft; the second case studied an urban traffic system (open system), in which noisy data (vehicle passages, traffic signal timings, travel time observations) was assimilated into a microscopic traffic simulation model to reconstruct vehicle trajectories on signalized urban arterials. The characteristics of the two case studies are compared in Table 6.1. The second case also contributes to a generic (in the sense that any (ensemble of) microscopic simulation models can be used) data assimilation framework for vehicle trajectory reconstruction on signalized urban arterials. The results from the case studies imply that:

- The proposed data assimilation framework is indeed able to provide accurate estimation results in both closed and open discrete event systems. As shown in section 3.5, after we assimilate (with interpolation) the noisy dataset with Gaussian noise $\mathcal{N}(0, 3^2)$ added on entity positions, the performance indicators of estimating the truck arrivals are 95.00$\%$ (success rate), 5.00$\%$ (waste rate), 0.53 minute (average distance), and 92.66$\%$ (average percentage), respectively (see section 3.5.3.2); in contrast, the simulation without data assimilation totally loses its prediction ability after 150 minutes (see section 3.4). In the urban traffic case, the framework is able to reconstruct plausible vehicle trajectories under realistic error assumptions (detection accuracy $p = 0.9$, occurrence rate of over-count $\lambda = 1/300\ s^{-1}$) yielding good performance on both macro- and microscopic error measures (see section 4.3 for more details). The overall absolute percent errors on reconstructing passage counts are around 7$\%$; whereas reconstructing the departure sequences over stop lines has an error around 5$\%$. Given the error assumptions this is a very promising result—recall that without assimilation, these errors are unbounded! The errors in density and flow (using Edie's definitions (Edie, 1963)) are around 5% as well. Also the quality of the reconstructed trajectories in terms of matching locations and speeds is satisfactory, although here we observe a (small) subset of large(r) errors.

- Though the estimation results obtained from data assimilation without interpolation are already accurate, they can be improved significantly (in the statistic sense) if a proper interpolation operation is used. If the model state is retrieved without considering the elapsed time (i.e. without interpolation), it can still reflect reality to a certain degree, and the estimation results are much better than those without data assimilation (see section 3.4). However, with interpolation, the time elapsed since the last state transition is considered, and as a result, the real-time evolution, which is not captured in the model but does happen in reality, will be reflected through the measurement model. Consequently, the estimation results obtained from data assimilation with interpolation are more accurate than those obtained without interpolation, which is proven in section 3.5.3.3.
- The variable dimension state trajectory has no tangible effect on weight updating, and particle filtering can approximate the dimension of the state trajectory accurately. In the gold mine case, the average absolute estimation error for the dimension of the state trajectory is 0.19, which is negligible since the dimension is an integer (see section 3.5.3.1). In the urban traffic case, the dimension of the state trajectory is directly related to the passage counts at the system boundaries (see equation 4.3). We can reconstruct these counts with an overall absolute percent error ranging from 5% to 7% depending on sensor locations (see section 4.3.3.1).

To conclude, particle filtering is in principle applicable to data assimilation in discrete event simulations. However, to apply particle filtering in discrete event simulations effectively, one needs to realize the characteristics of discrete event simulations, and understand their influence on the data assimilation procedure. First, a proper interpolation operation should be defined to obtain updated state values, since the discrete event state process is asynchronous with the measurement process. The interpolation operation can capture the real-time evolution that is not depicted in the model but does happen in reality. Second, though in practice the standard sequential importance sampling algorithm can be safely applied to update the random measure which approximates the posterior distribution of a state trajectory with variable dimension, we should realize the theoretical reason behind. To effectively conduct data assimilation in open systems, two extra steps (compared with the data assimilation procedure in closed systems) are required. First, we need to estimate (directly from data or indirectly using an estimation method) the number of entities in the system, otherwise we will have an underdetermined problem, since the number of entities in an open system is unknown in advance. Second, we need to reconstruct entity arrivals at system boundaries based on (noisy) observations. One can also randomly generate a sequence of arrivals, but this would require a much larger number of particles to achieve similar performance.

RQ 2*: How do the parameters of key components in the data assimilation framework affect the data assimilation results?*

Table 6.1: Comparison of the case studies

	the gold mine case	**the urban traffic case**
system properties	consist of miners, trucks, elevator, etc.; drilling time of the miners and unloading time of the elevator are random; closed discrete event system	consist of roads, traffic signals, vehicles, etc.; vehicles can enter and leave the system at any time through the system boundaries; open discrete event systems
model	coupled DEVS model	microscopic traffic flow model (focus on car-following behavior; based on event scheduling)
data	partial event sequences, entity positions (with Gaussian errors)	vehicle passages (with miss- and over-counts), traffic signal timings, travel time observations
variable of interest	truck arrival times at the bottom of the vertical shaft	vehicle trajectories over a time-space region
state retrieval problem	time advance of atomic components is random, therefore the duration between two consecutive system state updates is not fixed; use linear interpolation to obtain entity positions	vehicles enter the system at different times (irregularly distributed), and consequently system state updates happen randomly on a continuous time base; the interpolation operation boils down to integrating acceleration and speed over the elapsed time to obtain updated vehicle state (location, speed, acceleration, etc.)
variable dimension problem	the number of state points during a fixed time interval is random	vehicles entering and leaving the system lead to a dynamic (random) number of vehicles in the system
notes for the data assimilation procedure	–	need to estimate the number of vehicles to avoid an underdetermined problem; need to reconstruct vehicle arrivals at system boundaries in order to run the simulation model of the open system

The key components in the proposed particle filter based data assimilation framework are the system model, the noisy data, and the particle filtering algorithm. Based on the data generated from the two case studies, we analyzed the characteristics of the proposed data assimilation framework and its sensitivity to a number of important parameters that relate to the errors in the data and in the simulation model, as well as to the number of particles employed.

Sensitivity analysis with respect to data quality shows that the framework is quite robust to error assumptions since particle filters are assumption-free. In the gold mine case, even with a 20-meter standard deviation on entity positions, the performance does not degenerate too much. Specifically, the error of estimating the dimension of the state trajectory is 0.24, and the performance indicators of estimating the truck arrivals are 85.00% (success rate), 15.00% (waste rate), 0.83 minute (average distance), and 85.82% (average percentage), respectively (see Table 5.2). In the urban traffic case, the (macroscopic) estimation errors are consistently about half the percentages of miss- and over-counts; even with 40% miss-counts or one over-count every two minutes, all estimation errors are (well) within a 20% range (see Table 5.6 and Table 5.7). Similarly, the framework is robust to model errors (i.e. differences between the models generating the ground-truth data and the models used in the case studies), although we cannot claim to have tested this exhaustively. The result shows that using models with errors does not significantly affect the estimation results (see Table 5.3 and Table 5.12). This result does, however, emphasize an important underlying point. Clearly, unless we have actual evidence (data), the accuracy of the estimation results depends on the validity of the simulation models used in the framework for the specific case at hand. In our case, this validity is evident, since the ground-truth data is produced by a similar model. In real life, when the predictions given by the simulation model diverge too much from the real behavior of the system, it stands to reason that the estimation results will be farther away from the ground truth. In such a situation, we need to develop a more effective importance density, which can combine model predictions and real-time measurements together to propose new samples (remember that the importance density is defined as $q(s_k|s_{0:k-1}, m_{1:k})$).

Sensitivity analysis in terms of the number of particles employed reveals that with fixed data/model quality, the estimation errors decrease when the number of particles increases (see Figure 5.1 and Figure 5.4). With more particles, we can have a better coverage of the system state space, leading to more accurate estimation results. However, the trend is not proportional, which means that it is impossible to achieve error-free estimation results by continuously increasing the number of particles, since the quality of the estimation results is restricted by the quality of the data and the model involved in the data assimilation process. Increasing the number of particles also increases the computational complexity, therefore a balance between computational complexity and estimation accuracy should be achieved.

To conclude, the quality of the data assimilation results is affected by the validity of the system model, the data quality, and the number of particles used. The more valid the simulation models, and the more accurate the sensors, and the higher the number of the particles, the better the estimation results. The sensitivity analysis also implies several future research directions, which will be detailed in section 6.2.

6.1.2 Main contributions

To summarize, the main contributions of this thesis are listed below:

- We formally propose a particle filter based data assimilation framework for discrete event simulations of both closed and open systems (see chapters 3 and 4). In this framework, characteristics of discrete event simulations are given full consideration in order to effectively apply the particle filtering algorithm in discrete event simulations. Additionally, the results for discrete event open systems can be easily extended to general open systems.

- We conduct extensive simulation studies to analyze the characteristics of the particle filter based data assimilation framework (see chapter 5).
- We study two case studies that are representative in discrete event simulations of both closed and open systems (see chapters 3 and 4). In these case studies, we exemplify several effective solutions for computing weight with discrete type of observations, e.g., event sequences, phases, and vehicle passages. The urban traffic case also contributes to a generic (in the sense that any (ensemble of) microscopic simulation models can be used) data assimilation framework for vehicle trajectory reconstruction on urban arterials.
- We present a conceptual framework for implementing the particle filter based data assimilation framework (see Appendix A). This conceptual framework is fully object-oriented, therefore it is very easy to tailor, extend and maintain. For illustration purpose, a reference implementation in DSOL (**D**istributed **S**imulation **O**bject **L**ibrary, which is the simulation environment chosen for this research) is also provided.

6.2 Future research directions

There are many directions for further research. Some possible options are listed below:

- It needs to be investigated how the size of the measurement interval affects the data assimilation results, and the existence of an optimal measurement interval should be explored. If an optimal measurement interval does exist, a balance between estimation accuracy and frequency of sensor data would be achieved.
- In the gold mine case, it was assumed that the partial event sequence is accurate. This assumption needs to be relaxed by considering event sequence (containing multiple types of events) with errors, such as missing events, and false detection. The sensitivity of the data assimilation results to the types of observations needs to be explored; the results can be utilized to optimize the measurement process.
- If the simulation model is not very accurate, samples drawn from the system transition density (as the importance density) will fail to effectively represent the true state space. Therefore, another research direction is to develop more effective importance density, which can combine model predictions and real-time measurements together to propose new samples.
- With increased complexity of the simulation model, the number of particles required will likely increase exponentially. Therefore, a parallel and distributed version of our framework could be developed to effectively deal with more complex scenarios.
- The sensitivity analysis tells that the more valid the simulation models, and the more accurate the sensors, the better the estimation results. This implies a general research

direction for all research communities, that is developing simulation models that can make more valid predictions of the real system behavior, and developing more advanced sensor technologies that can provide more accurate measurement data of the real systems.

- Future work will also be oriented to looking for more discrete event real world applications to apply the proposed data assimilation framework to solve real world problems.

Appendix A

Implementation of the particle filter based data assimilation framework

Since very little research has reported implementation issues of particle filters in complex discrete event simulations, and implementing particle filtering in complex discrete event simulations is not trivial, in this chapter, we address the implementation issues of the proposed particle filter based data assimilation framework in discrete event simulations. This chapter contributes to an object-oriented software conceptual framework, based on which a concrete data assimilation library can be implemented to support the working of the proposed data assimilation framework.

A.1 Key components in a particle filter based data assimilation system

As explained in previous chapters, a generic particle filter mainly consists of three computational steps (as shown in Figure A.1) (Saha et al., 2010):

- sampling: in this step samples (particles) of the unknown state are generated based on the given sampling function. These samples provide an estimate of the current state of the system and also propagate the particles from previous time instant to the current time instant.
- weight calculation: based on the observations an importance weight is assigned to each particle.

- resampling: this step involves the act of redrawing particles from the same probability density based on some function of the particle weights such that the weight of each new particle is approximately equal. Resampling is a very important step in a particle filter and without this step a particle filter is highly likely to degenerate, i.e., after a few iterations all the weights will go to zero except the weight of one particle.

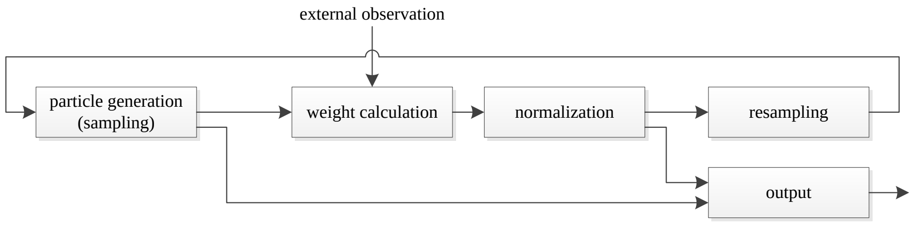


Figure A.1: The computation steps in a generic particle filter

To support the computation steps shown in Figure A.1, following components are required to be implemented as shown in Figure A.2:

- *representation of particles and weights*. A particle is essentially a specific model state (represented by the data structure `ModelState`), and its weight is a numerical value which reflects the likelihood that the system is in that state. `ModelState` should be capable of replicating itself, since some particles will be replicated in the resampling step.

- *representation of observations*. The representation should indicate the period during which the observation is collected, and the corresponding observed value.

- *strategies for sampling, resampling, and weight updating*. The `SamplingStrategy` implements a sampling action which generates samples based on the importance density. One of the most frequently used importance density is the system transition density (i.e. the prior), which generates a sample by simply running the simulation model for one time step. The `ResamplingStrategy` implements a resampling action which selects particles for the next iteration on a regular basis. The selection is based on a particle-dependent weighting function v_k, with $v_k^i = w_k^i$ we have standard resampling according to the particle weights. Different weighting function yields different resampling strategy, and possible candidates can be found in Douc et al. (2005). The `WeightUpdatingStrategy` implements a weight computation function which updates a particle's weight based on the newly available observation. A particle's weight can be updated by the likelihood if the system transition density is chosen as the importance density, or by other methods based on the chosen importance density, for example, the kernel method presented in Xue and Hu (2013).

- *communication with the simulation model*, which aims to get/set simulation model state, perturb model state, and generate predicted measurements, etc.

- the *central control logic*, which organizes these key components to conduct particle filtering *properly*. By ‘properly’, we mean that every method invocation should be at the right time and in the right order. Notice that the sampling action is usually asynchronous, i.e., we cannot obtain the expected sample just after we invoke the sampling method. Therefore, we must have a control logic to make sure that the main computation steps are executed in the correct way.

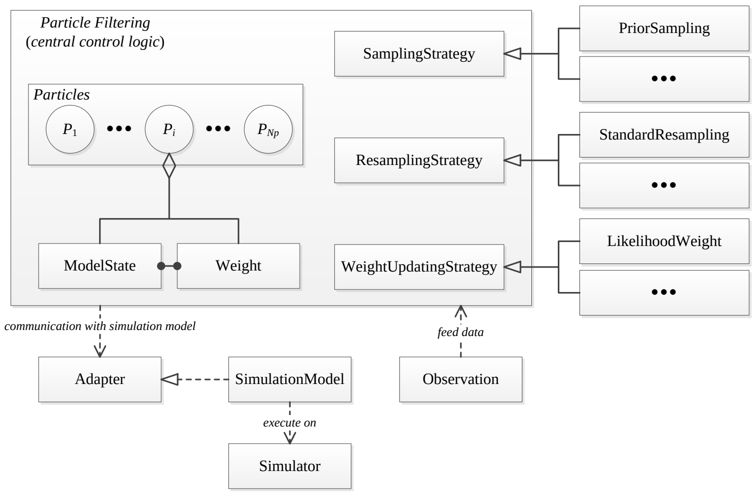


Figure A.2: Key components in a particle filter based data assimilation system (Xue, 2014)

A.2 The conceptual framework to implement the particle filter based data assimilation system

In this section, we present an object-oriented software conceptual framework, based on which a concrete data assimilation library can be implemented to support the working of the proposed data assimilation framework. Firstly, we give a general view of the structure of the conceptual framework, and explain the relations between the interfaces/classes in the conceptual framework and the key components in section A.1. Subsequently, we elaborate how these interfaces/classes are designed to support particle filtering. Notice that the conceptual framework uses concepts in Java, however we argue that this framework can be easily transferred to other object-oriented programming languages by replacing these concepts in Java with equivalent ones in target programming languages.

A.2.1 General view of the conceptual framework

The bottom part of Figure A.3 shows the key interfaces/classes in the conceptual framework:

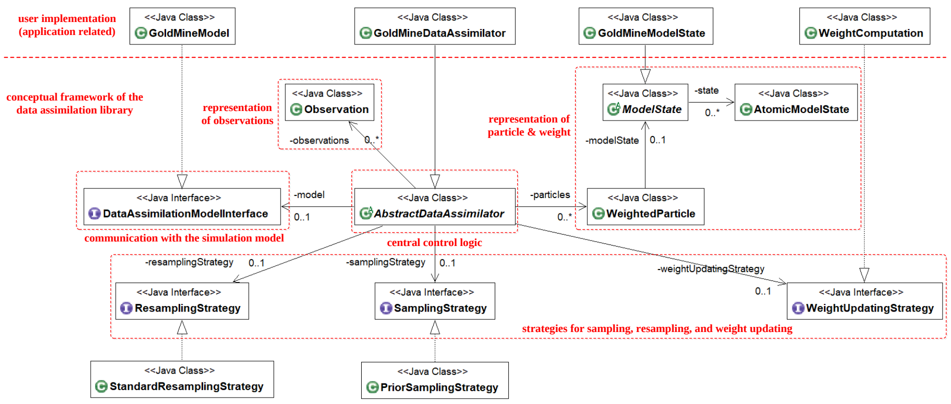

user implementation (application related)
<<Java Class>> GoldMineModel
<<Java Class>> GoldMineDataAssimilator
<<Java Class>> GoldMineModelState
<<Java Class>> WeightComputation
conceptual framework of the data assimilation library
representation of observations
<<Java Class>> Observation
-observations 0..*
representation of particle & weight
<<Java Class>> ModelState
-state 0..*
<<Java Class>> AtomicModelState
-modelState 0..1
<<Java Interface>> DataAssimilationModelInterface
-model 0..1
<<Java Class>> AbstractDataAssimilator
-particles 0..*
<<Java Class>> WeightedParticle
communication with the simulation model
central control logic
-resamplingStrategy 0..1
-samplingStrategy 0..1
-weightUpdatingStrategy 0..1
<<Java Interface>> ResamplingStrategy
<<Java Interface>> SamplingStrategy
<<Java Interface>> WeightUpdatingStrategy
strategies for sampling, resampling, and weight updating
<<Java Class>> StandardResamplingStrategy
<<Java Class>> PriorSamplingStrategy

Figure A.3: Key interfaces/classes in the conceptual framework

- `WeightedParticle`, which represents a particle (i.e. a specific system state represented by `ModelState`) and its associated weight. The definition of `Weighted Particle` will be introduced in detail in section A.2.2.

- `Observation`, which encapsulates observed data collected during a specified time interval, and will be explained in section A.2.3.

- `SamplingStrategy`, `ResamplingStrategy`, and `WeightUpdatingStrategy`, which defines a set of functions (name, arguments, and return value) that should be implemented in order to fulfill the computation steps in Figure A.1. Their definition will be given in section A.2.4. The most commonly used sampling strategy and resampling strategy are to sample based on system transition density (i.e. `PriorSamplingStrategy`), and to resample particles in proportion to their weights (i.e. `StandardResamplingStrategy`). Notice that weight computation is closely related to simulation applications (more specifically, to assumptions on error models of the simulation model and the data), therefore it should be implemented by users based on their applications.

- `AbstractDataAssimilator`, which implements the central control logic of the particle filtering by properly organizing the execution of functions, and will be explained in section A.2.6.

- `DataAssimilationModelInterface`, which is an adapter between the central control logic (`AbstractDataAssimilator`) and the simulation model in order for the data assimilation process to communicate with the simulation model easily. More details will be presented in section A.2.5.

A.2.2 Representation of particles and weights

A particle is essentially a specific model state, and its weight is a numerical value which reflects the likelihood that the system is in that state. Therefore, we design the `WeightedParticle` class (shown in Figure A.4) to represent a particle and its weight.

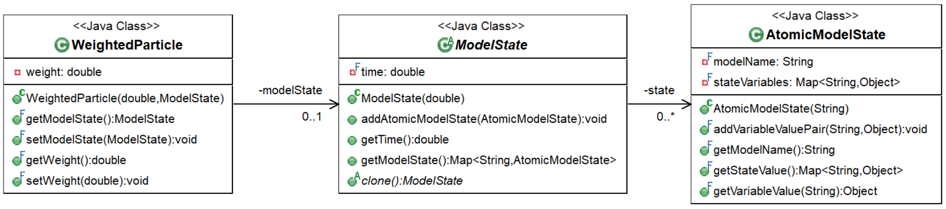

<<Java Class>>
WeightedParticle
weight: double
WeightedParticle(double,ModelState)
getModelState():ModelState
setModelState(ModelState):void
getWeight():double
setWeight(double):void
-modelState
0..1
<<Java Class>>
ModelState
time: double
ModelState(double)
addAtomicModelState(AtomicModelState):void
getTime():double
getModelState():Map<String,AtomicModelState>
clone():ModelState
-state
0..*
<<Java Class>>
AtomicModelState
modelName: String
stateVariables: Map<String,Object>
AtomicModelState(String)
addVariableValuePair(String,Object):void
getModelName():String
getStateValue():Map<String,Object>
getVariableValue(String):Object

Figure A.4: The particle and its weight

The model state is defined by the `ModelState` class, where the *time* attribute indicates when the state is retrieved, and the state value is represented as the product of states of all its atomic components. Essentially, the state of an atomic component can be represented as a collection of name-value pairs where each pair consists of a `String` type representing the name of a state variable and an `Object` type representing corresponding value of the state variable. As shown in Figure A.4, `AtomicModelState` defines the state of an atomic model component, where attribute *modelName* indicates the name of the model component, and attribute *stateVariables* records name-value pairs for each of its state variable.

The `clone` method produces a deep copy of the model state itself. Since this operation is closely dependent on applications, it is declared as an abstract method which should be implemented by users based on specific applications.

A.2.3 Representation of observations

Measurements reflect real-time information from the real system, which are defined by the `Observation` class, as shown in Figure A.5. The *interval* field indicates the period during which the observation is collected; while the observation data is represented by a set of tuples where each tuple consists of the name of the data source and the observed data from that source.

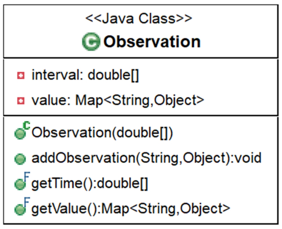

<<Java Class>>
Observation
interval: double[]
value: Map<String,Object>
Observation(double[])
addObservation(String,Object):void
getTime():double[]
getValue():Map<String,Object>

Figure A.5: The `Observation` class

A.2.4 Strategies for sampling, resampling, and weight updating

The particle filtering works relying on strategies of sampling, resampling, and weight updating, which are defined in Figure A.6. The arguments and return values are explained as follows:

- The arguments of the `sample` method include a weighted particle (`WeightedPart icle`), observation (`Observation`), and the simulation model (`DataAssimila tionModelInterface`). The definition of arguments is an one-by-one translation of the formal sampling function $s_k^i \sim q(s_k|s_{k-1}^i, m_k)$. The time information (i.e., $k-1$ and k) can be obtained from the observation (i.e., the *interval* attribute in `Observation`). Since sampling action is usually asynchronous, we cannot obtain the new sample s_k^i directly, therefore the return value is defined as `void`. To obtain s_k^i, it is required to implement the `scheduleExecutionOfonSampleGenerat ed` method, which tells the central control logic (implemented in `AbstractDataAs similator` which will be introduced in section A.2.6) when to retrieve the generated sample based on the sampling strategy.

- The argument of the `resample` method is a list of weighted particles, and its return value indicates which particles are selected, and further how many times each of the selected particles is selected.

- The two arguments of the `evaluateWeight` method are the predicted observation (generated from the measurement model) and the real observation, respectively; the return value is a collection of weights, each of which is associated with a specified data source. The `combineWeight` method is defined in order to combine the weights from different data sources into one. This definition makes it easy to incorporate multiple sources of data.

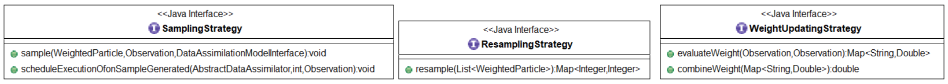

<<Java Interface>>
SamplingStrategy
sample(WeightedParticle,Observation,DataAssimilationModelInterface):void
scheduleExecutionOfonSampleGenerated(AbstractDataAssimilator,int,Observation):void
<<Java Interface>>
ResamplingStrategy
resample(List<WeightedParticle>):Map<Integer,Integer>
<<Java Interface>>
WeightUpdatingStrategy
evaluateWeight(Observation,Observation):Map<String,Double>
combineWeight(Map<String,Double>):double

Figure A.6: The sampling, resampling, and weight updating strategy

A.2.5 Communication with the simulation model

During the data assimilation process, frequent communication with the simulation model is required. The purposes of these communications are mainly:

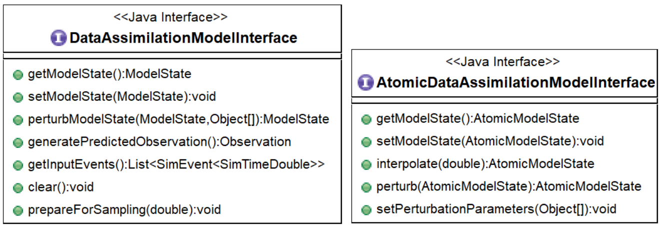
- *getting/setting simulation model state*. When generating a sample, we need to initialize the simulation model with a given initial model state (i.e., setting model state); once the running is finished, we need to retrieve the model state (i.e., getting model state) to obtain the generated sample.

- *perturbing model state*. In some cases, it is required to add noise to the generated model state to imitate a stochastic setting (i.e., perturbation of model state), for example, adding graph noise to the fire front in Xue et al. (2012).

- *generating predicted measurements*, which implements the measurement model.

To this end, we require the simulation model to implement the `DataAssimilatio nModelInterface` shown in Figure A.7(a). The meaning of the specified methods is explained as follows:

- The `getModelState` method and the `setModelState` method implement state getting and setting.
- The `perturbModelState` method implements the state perturbation. The two arguments are the original model state (`ModelState`) and required parameters (`Object[]`) for perturbation. The method returns a perturbed version of the original model state.
- The `generatePredictedObservation` method implements the measurement model by returning a predicted measurement (`Observation`).
- The `clear` method deletes all transient atomic components in the simulation model. This method is used for discrete event simulations of open systems.
- The `prepareForSampling` method gets the simulation model prepared for sampling, and its use will be explained in detail in section A.2.6.


<<Java Interface>>
DataAssimilationModelInterface
getModelState():ModelState
setModelState(ModelState):void
perturbModelState(ModelState,Object[]):ModelState
generatePredictedObservation():Observation
getInputEvents():List<SimEvent<SimTimeDouble>>
clear():void
prepareForSampling(double):void
<<Java Interface>>
AtomicDataAssimilationModelInterface
getModelState():AtomicModelState
setModelState(AtomicModelState):void
interpolate(double):AtomicModelState
perturb(AtomicModelState):AtomicModelState
setPerturbationParameters(Object[]):void

(a) DataAssimilationModelInterface (b) AtomicDataAssimilationModelInterface

Figure A.7: The interfaces that the simulation model should implement

These methods are defined at the coupled model level, and we require its atomic components to implement the `AtomicDataAssimilationModelInterface` shown in Figure A.7(b) which defines methods with similar functionalities but at the atomic model level. As a result, a method defined in `DataAssimilationModelInterface` can be implemented by invoking corresponding method of all its atomic components. Among these methods defined in `AtomicDataAssimilationModelInterface`, we particularly explain how to get and set state of an atomic model component in a discrete event simulation.

A.2.5.1 Getting atomic model state by interpolation

As introduced in chapter 3, the state of an atomic component in a discrete event simulation should be obtained through interpolation. This process is illustrated in Figure A.8, which consists of three steps. First, the atomic component needs to request time information related to itself, i.e., the time of the last event, t_l, the time of the next event, t_n, and the current time, t, because only the simulator manages the time advance in a discrete event simulation. Second, the interpolation operation is conducted to obtain an interpolated state at time t. As explained in section 3.3.3, we need to distinguish continuous states (can be interpolated) from discrete states (cannot be interpolated). If state s can be interpolated, the interpolated state will be $\hat{s}(t) = interpolate(s, t - t_l)$; otherwise, the interpolated state will be $\hat{s}(t) = (s, t - t_l)$. Finally, both the interpolated state $\hat{s}(t)$ and its related time information are encapsulated in data structure `AtomicModelState` which is introduced in section A.2.2. If s can be interpolated, we record the time advance of $\hat{s}(t) = interpolate(s, t - t_l)$ as $ta(\hat{s}(t)) = t_n - t$; otherwise, we record the remaining time of $\hat{s}(t) = (s, e)$ as $\sigma = t_n - t$.

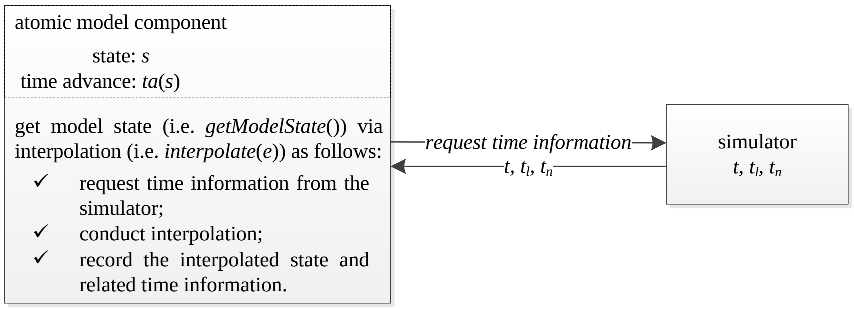

atomic model component
state: s
time advance: ta(s)
get model state (i.e. getModelState()) via interpolation (i.e. interpolate(e)) as follows:
✓ request time information from the simulator;
✓ conduct interpolation;
✓ record the interpolated state and related time information.
request time information
t, tl, tn
simulator
t, tl, tn

Figure A.8: Get state from an atomic model component in a discrete event simulation

A.2.5.2 Setting an interpolated state to an atomic model

In discrete event simulations of closed systems, setting an interpolated state (encapsulated in `AtomicModelState`) to an atomic model is quite simple. If the state can be interpolated, i.e. $\hat{s}(t) = interpolate(s, e)$, we assign each state variable in the model with corresponding value stored in the data structure `AtomicModelState`, and then schedule an internal state transition on the simulator at $t_n = t + ta(\hat{s}(t))$; otherwise, i.e. $\hat{s}(t) = (s, e)$, after assigning value for each state variable, we schedule an internal state transition at $t_n = t + \sigma$.

For discrete event simulations of open systems, if the target atomic model is deleted from the simulation model, we need to instantiate this atomic model with the interpolated state (encapsulated in `AtomicModelState`) first, and then schedule an internal state transition for the new component similarly.

A.2.6 Central control logic

Given all those functionalities designed in section A.2.2 $\sim$ A.2.5, the next task is to organize them in a right order to conduct particle filtering. To this end, we propose one feasible control logic, which is implemented in `AbstractDataAssimilator` (see Figure A.9).

When new measurements are available, the `assimilate` method is triggered, in which the data assimilation procedure shown in Algorithm 6 is implemented. The first step is to generate samples based on the sampling strategy, in which we must guarantee that *weights normalization and resampling can only be executed after all particles are updated.* Since sampling usually involves running a complex simulation model, a callback function (`onSampleGenerated`) is designed to process the particle as soon as it is generated. The two arguments of the callback function indicate the index of the particle

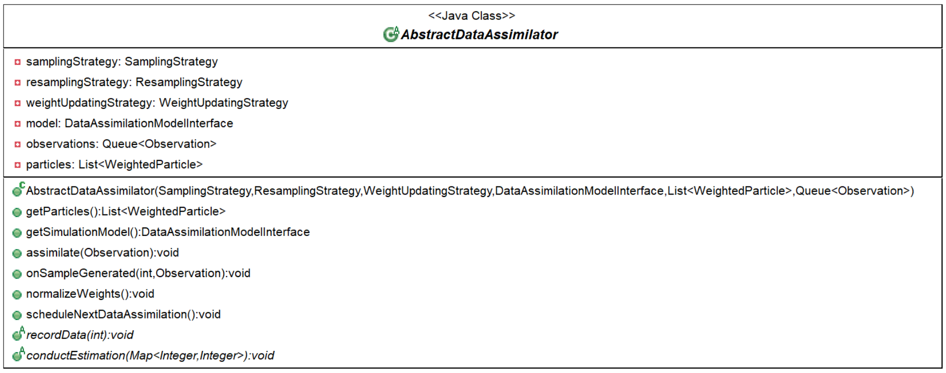

<<Java Class>>
AbstractDataAssimilator
samplingStrategy: SamplingStrategy
resamplingStrategy: ResamplingStrategy
weightUpdatingStrategy: WeightUpdatingStrategy
model: DataAssimilationModelInterface
observations: Queue<Observation>
particles: List<WeightedParticle>
AbstractDataAssimilator(SamplingStrategy,ResamplingStrategy,WeightUpdatingStrategy,DataAssimilationModelInterface,List<WeightedParticle>,Queue<Observation>)
getParticles():List<WeightedParticle>
getSimulationModel():DataAssimilationModelInterface
assimilate(Observation):void
onSampleGenerated(int,Observation):void
normalizeWeights():void
scheduleNextDataAssimilation():void
recordData(int):void
conductEstimation(Map<Integer,Integer>):void

Figure A.9: The `AbstractDataAssimilator` class

and the current observation. When to trigger the callback function is determined by the sampling strategy (see the `scheduleExecutionOfonSampleGenerated` method in the `SamplingStrategy` interface). In the callback function (line 11∼46 in Algorithm 6), one needs to process the generated sample, update the weight, and if there are any particles not updated, we continue sampling; otherwise, we move to the next step of weights normalization, resampling, and estimation, etc. Since data recording and state estimation are closely related to the applications, the corresponding methods (i.e. `recordData` and `conductEstimation`) are designed as abstract. Users can implement a data assimilator tailored to their specific application by inheriting `AbstractDataAssimilator` and overriding certain methods if needed.

Notice that in the data assimilation procedure presented in this research, only one simulation model instance is employed, so is the corresponding discrete event simulator. Therefore, when generating a new sample, it should be guaranteed that the model instance is prepared for sampling, see the `prepareForSampling` method in line 6 and line 33 in Algorithm 6. In the `prepareForSampling` method, one should clean the discrete event simulator by deleting events scheduled in the last run; if the simulation model describes an open system, one needs to delete all transient model components (by invoking the `clear` method) as well. The `prepareForSampling` method will get the simulation model (also the discrete event simulator) prepared to be assigned a new model state for sampling. The implementation of the `prepareForSampling` method is closely related to the discrete event simulation environment, and we will exemplify its implementation in DSOL in section A.3.2.

A.2.7 Memory Consumption & Speed

Notice that in the data assimilation procedure presented in section A.2.6, only one simulation model instance is employed, so is the corresponding discrete event simulator; samples are generated by recursively assigning an initial state to the model instance, and then running the model based on the sampling strategy to generate a sample. In this section, we briefly analyze the memory consumption and speed of the proposed implementation compared with the implementation (for convenience, we refer it as *standard implementation*)

Algorithm 6: The data assimilation procedure (`assimilate(Observation obs)`)

```
Input: current observation: $obs$
% generate samples
$index \leftarrow 0$ % Java uses zero-based indexing
$particle \leftarrow getParticles().get(index)$ % i.e. $s^i_{k-1}$
$model \leftarrow getSimulationModel()$
% prepare the simulation model for sampling
$model.prepareForSampling(particle.getModelState().getTime())$
% generate sample $s^i_k \sim p(s_k|s^i_{k-1}, m_k)$
$samplingStrategy.sample(particle, obs, model)$
% schedule execution of the callback function (onSampleGenerated) based on the sampling strategy
  with arguments: the data assimilator ($this$), $index$ and $obs$
$samplingStrategy.scheduleExecutionOfonSampleGenerated(this, index, obs)$

% callback function: onSampleGenerated(int index, Observation obs)
% retrieve the generated state
$state \leftarrow getSimulationModel.getModelState()$
% perturb the state if needed
$state \leftarrow getSimulationModel.perturbModelState(state)$
% generate predicted observation
$predictedObs \leftarrow getSimulationModel().generatePredictedObservation()$
% compute weights in terms of data sources, where $s$ indicates the name of data source, $w$ is the
  corresponding weight
$\{(s, w)\} \leftarrow weightUpdatingStrategy.evaluateWeight(predictedObs, obs)$
% combine weights from different data sources together
$weight \leftarrow weightUpdatingStrategy.combineWeight(\{(s, w)\}) \times particle.getWeight()$
% update state of the particle
$getParticles.get(index).setModelState(state)$
% update weight of the particle
$getParticles.get(index).setWeight(weight)$
% record data that is associated with this particle for estimation
$recordData(index)$
$index \leftarrow index + 1$
if $index < getParticles().size()$ then
    $particle \leftarrow getParticles().get(index)$
    $model \leftarrow getSimulationModel()$
    % generate the next particle
    $model.prepareForSampling(particle.getModelState().getTime())$
    $samplingStrategy.sample(particle, obs, model)$
    $samplingStrategy.scheduleExecutionOfonSampleGenerated(this, index, obs)$
end
else
    % all particles have been updated
    % normalize weights
    $normalizeWeights()$
    % resampling, $i$ represent the index of the selected particles, while $n_i$ indicates how many times the
      $i$-th particle is selected
    $\{(i, n_i)\} \leftarrow resamplingStrategy.resample(getParticles())$
    % state estimation based on the weighted particles
    $conductEstimation(\{(i, n_i)\})$
    based on $\{(i, n_i)\}$, replicate the $i$-th particle $n_i$ times, while discard those that are not selected; set
      the weight of each particle as $1/N_p$, where $N_p$ is the number of particles
end
```

in which samples are generated by running multiple model instances. The analysis shows that the proposed implementation consumes much less memory, but is almost as fast as the standard implementation.

A.2.7.1 Memory Consumption

Assume that the memory consumption of a discrete event model is M_m, while the proportion of the memory consumption of the state in the whole memory consumption of the model is $p(0 < p \leq 1)$. The memory consumption of a discrete event simulator with an empty event list is M_s. During the simulation, we assume that there are on average N_e events stored in the event list, and the average memory consumption of an event is M_e. In the standard implementation with N_p particles, the memory consumption at time t is

$$MC_{standard} = N_p(M_m + M_s + N_e M_e)$$

, while in the proposed implementation with the same number of particles, the memory consumption is

$$MC_{proposed} = M_m + M_s + N_e M_e + (N_p - 1)M_m p$$

Obviously, $MC_{proposed} \ll MC_{standard}$, i.e., the proposed implementation consumes much less memory compared with the standard implementation.

A.2.7.2 Speed

In order to accelerate the execution of the particle filters, parallel/distributed implementation are commonly applied. If there is only one processing unit (PU) available, N_p particles in the standard implementation can only be executed sequentially, therefore, in the single-PU case, the proposed implementation is almost as fast as the standard implementation.

If there are multiple PUs available (assume the number of PUs is N_{pu}, usually we have $N_{pu} < N_p$), the proposed implementation can be parallelized or distributed as follows: deploy one simulation and n_i particles on the processing unit $PU_i(i = 1, \ldots, N_{pu})$, such that $\sum_{i=1}^{N_{pu}} n_i = N_p$; on PU_i, particles will be generated as described in section A.2.6; when all particles are generated, resampling will be done in a centralized way. Therefore, in the multiple-PU case, there is no obvious speed drop in the proposed implementation compared with the standard implementation if proper parallelizing/distributing schemes are employed.

A.3 Reference implementation in DSOL

A.3.1 DSOL & its support for discrete event simulations

DSOL, which stands for **D**istributed **S**imulation **O**bject **L**ibrary, is a Java-based, open-source, object-oriented, distributed and extendible research test-bed for simulation (Jacobs, 2005). The basic entities in Modeling & Simulation (M&S) are implemented in DSOL as shown in Figure A.10. The *experiment frame* (`ExperimentalFrame`) mainly defines a list of experiments that will be conducted in order to fulfill certain objectives. Each *experiment* (`Experiment`) consists of a simulation model (`ModelInterface`) on which experiments will be conducted, a *simulator* (`SimulatorInterface`) which executes the model, a list of *replications* (`Replication`) which mainly specify random number streams in each run, and a *treatment* (`Treatment`) which defines a set of parameters to conduct the experiment, such as warm-up period, run length, start time, and end time. DSOL represents the simulation time in a flexible way. In DSOL, the simulation time can be represented in *various types of data format*, such as double, integer, and calendar, and *with various types of time units*, such as seconds, minutes, or days. This is implemented by parameterizing interfaces/classes with A, R and T, which stands for *absolute time type*, *relative time type*, and *simulation time type*, respectively.

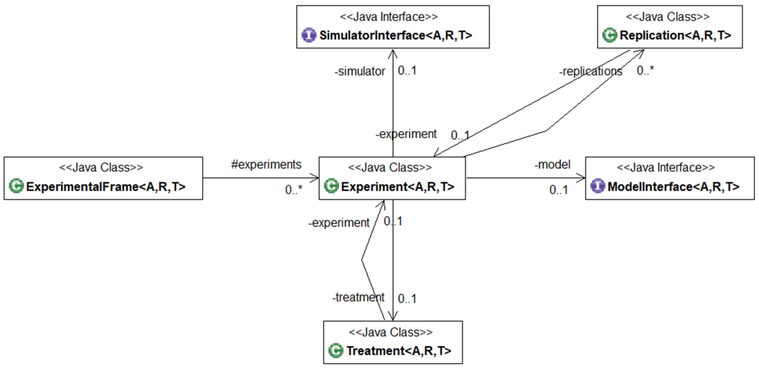


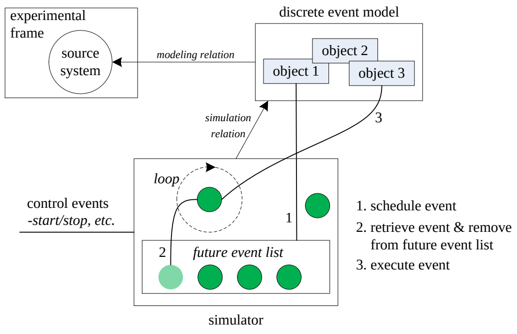
Figure A.10: The basic entities in the Modeling & Simulation (M&S) framework implemented in DSOL

DSOL supports multi-formalism simulations, but since the focus of this thesis is on discrete event simulations, here we only introduce how discrete event simulations are supported in DSOL. Interested readers can refer to Jacobs et al. (2002); Jacobs (2005); Lang et al. (2003); Seck and Verbraeck (2009) for more details on other formalisms. In DSOL, discrete event simulations are implemented based on the event scheduling world view as shown in Figure A.11 (Jacobs et al., 2002). The discrete event model is built as a collection of objects that interact in some fashion, but the interaction is achieved not by a direct method invocation, but by scheduling them via constructing a simulation event. To execute discrete event models, DSOL provides a future event list based discrete event simulator (`DEVSSimulator` in Figure A.11).

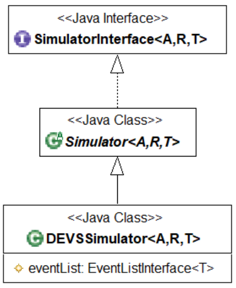
DSOL also provides full support for Zeigler's DEVS formalism, but the hierarchical DEVS model is simulated using a flattened simulator (`DEVSSimulator`) (Seck and Verbraeck, 2009). More information on DSOL can be found in Jacobs et al. (2002); Jacobs (2005); Lang et al. (2003), and also at `http://www.simulation.tudelft.nl`.

A.3.2 Implement the data assimilation procedure in DSOL

As explained in section A.2.6, since only one simulation model instance (thus one discrete event simulator) is involved in the data assimilation procedure, we need to guarantee that



(a) The object-oriented implementation of the event scheduling world view



(b) The future event list based discrete event simulator in DSOL

Figure A.11: Discrete event simulations in DSOL; the left figure is from Jacobs et al. (2002)

the model instance is prepared for sampling when we start to generate a new sample. This is done by implementing the `prepareForSampling` method, in which one should clean the discrete event simulator by deleting events scheduled in the last run; if the simulation model describes an open system, one needs to delete all transient model components (by invoking the `clear` method) as well. To this end, we modify the discrete event simulator to make it have flexible time management, as shown in Figure A.12 and Algorithm 7. Then the `prepareForSampling` method can be implemented in DSOL as shown in Algorithm 8.

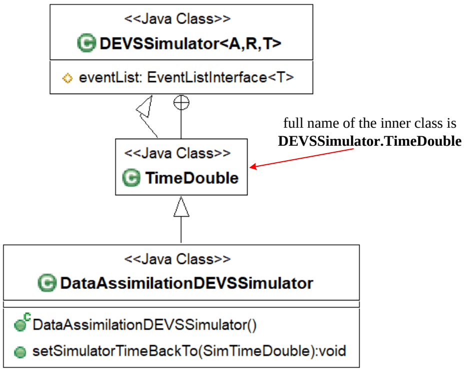

<<Java Class>>
DEVSSimulator<A,R,T>
eventList: EventListInterface<T>
full name of the inner class is DEVSSimulator.TimeDouble
<<Java Class>>
TimeDouble
<<Java Class>>
DataAssimilationDEVSSimulator
DataAssimilationDEVSSimulator()
setSimulatorTimeBackTo(SimTimeDouble):void

Figure A.12: The `DataAssimilationDEVSSimulator` class

After the model instance is prepared for sampling, we can then set the (interpolated) model state (i.e. s^i_{k-1}) to the model instance (see section A.2.5.2), and start to generate samples based on the sampling strategy.

As an example, Algorithm 9 and 10 illustrate the implementation of `PriorSampling Strategy` in DSOL. Notice that `PriorSamplingStrategy` generates samples based on the system transition density, therefore the observation is not considered in

Algorithm 7: $setSimulatorTimeBackTo(t)$

```
Input: a past time instant: $t$
if $t \leq simulator.currentTime$ then
    foreach $event \in simulator.eventList$ do
        if event is an internal event then
            % an internal event is scheduled by the model components for state updating
            delete $event$ from $simulator.eventList$
        end
    end
    $simulator.time \leftarrow t$
end
```

Algorithm 8: $prepareForSampling(t)$

```
Input: a past time instant: $t$
% get the simulator which executes the model
$simulator \leftarrow model.getSimulator()$
% clean the future event list
$simulator.setSimulatorTimeBackTo(t)$
% if the model describes an open system, delete all transient model components
$model.clear()$
```

the `sample` method.

Algorithm 9: Implementing the `sample` method of `PriorSamplingStrategy` in DSOL

```
Input: particle, obs, model % parameters are from the sample method
% set initial state to the simulation model
$model.setModelState(particle.getModelState())$
% for open systems, schedule arrivals of entities
for event ∈ model.getInputEvents() do
    $simulator.scheduleEvent(event)$
end
```

A.4 Conclusion

The work of particle filtering relies on a sequence of computation steps, i.e., sampling, weight computation, resampling, etc. To support these computation steps, at least following components need to be implemented: 1) representation of particles and weights; 2) representation of observations; 3) strategies for sampling, resampling, and weight updating; 4) communication with the simulation model to access the model state; and 5) central

Algorithm 10: Implementing the `scheduleExecutionOfonSampleGenerated` method of `PriorSamplingStrategy` in DSOL

```
1 % parameters are from the scheduleExecutionOfonSampleGenerated
    method
  Input: $assimilator$, $index$, $observation$
2 % the time instant when the new sample is due
3 $t \leftarrow observation.getTime().getIntervalEnd$
4 schedule the execution of the onSampleGenerated method belonging to
    $assimilator$ at time $t$, with arguments $index$ and $observation$
```

control logic, which organizes these components to conduct particle filtering *properly*. By 'properly', we mean that every method invocation should be at the right time and in the right order.

In this chapter, we proposed an object-oriented software conceptual framework, based on which a concrete data assimilation library can be implemented. In the proposed conceptual framework, a particle together with its weight are represented by the `WeightedParticle` class, in which the model state is defined by the `ModelState` class and its weight is indicated by a numerical field. The `ModelState` class contains a collection of atomic model states, each of which is represented by the `AtomicModelState` class (defining name-value pairs for each state variable). The `ModelState` class is abstract, and it requires users to implement a `clone` method which enables model state to replicate itself. Observations are represented by the `Observation` class, in which data from different sources can be easily incorporated. Strategies for sampling, resampling, and weight updating are defined by three interfaces respectively, i.e. `SamplingStrategy`, `ResamplingStrategy`, and `WeightUpdatingStrategy`, which allow users to implement different strategies that best fit their applications. The `DataAssimilationModelInterface` defines methods that the simulation model should implement in order to facilitate communications with the simulation model during the data assimilation process, such as setting/getting model state, perturbing model state, and generating predicted observation (implementing the measurement model). The implementation of `DataAssimilationModelInterface` requires each atomic model component to implement the `AtomicDataAssimilationModelInterface`, which defines methods with similar functionalities but at the atomic model level. As a result, a method defined in `DataAssimilationModelInterface` can be implemented by invoking corresponding method of all its atomic components. Finally, the abstract class `AbstractDataAssimilator` properly organizes these components to fulfill the data assimilation task. In `AbstractDataAssimilator`, only one simulation model instance is employed, so is the corresponding discrete event simulator. Samples are generated by recursively assigning an initial state to the model instance, and then running the model based on the sampling strategy to generate a sample. Therefore, this requires the model instance gets prepared (implementing the `prepareForSampling` method) when generating a new sample, which means that before assigning a state to the model instance, one should clean the discrete event simulator by deleting events scheduled in the last run; if the simulation model describes an open system, one needs to delete all transient atomic model components (by invoking the `clear` method) as well. In this chapter, a reference implementation in DSOL (**D**istributed **S**imulation **O**bject **L**ibrary) is provided to exemplify the use of the proposed conceptual framework.

The proposed conceptual framework is fully object-oriented, therefore it is very easy to tailor, extend and maintain. Notice that the proposed conceptual framework is definitely not optimal in terms of performance, ease of implementation, etc. The key focus is to organize the computation steps in particle filtering in a correct way to obtain correct estimation results. But future work will be oriented to address the performance issues.

Bibliography

Arulampalam, M. S., Maskell, S., Gordon, N., Clapp, T., 2002. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. IEEE Transactions on Signal Processing 50 (2), 174–188.

Atzori, L., Iera, A., Morabito, G., 2010. The internet of things: A survey. Computer Networks 54 (15), 2787–2805.

Bai, F., Gu, F., Hu, X., Guo, S., 2016. Particle routing in distributed particle filters for large-scale spatial temporal systems. IEEE Transactions on Parallel and Distributed Systems 27 (2), 481–493.

Bai, F., Guo, S., Hu, X., 2011. Towards parameter estimation in wildfire spread simulation based on sequential Monte Carlo methods. In: Proceedings of the 44th Annual Simulation Symposium. Boston, MA, USA, pp. 159–166.

Banks, J., 1998. Handbook of Simulation - Principles, Methodology, Advances, Applications, and Practice. John Wiley & Sons.

Bhaskar, A., Tsubota, T., Kieu, L. M., Chung, E., 2014. Urban traffic state estimation: fusing point and zone based data. Transportation Research Part C: Emerging Technologies 48, 120–142.

Bouttier, F., Courtier, P., 1999. Data assimilation concepts and methods. Meteorological Training Course Lecture Series, ECMWF (European Centre for Medium-Range Weather Forecasts).

Brackstone, M., McDonald, M., 1999. Car-following: a historical review. Transportation Research Part F: Traffic Psychology and Behaviour 2 (4), 181–196.

Bryman, A., 2015. Social Research Methods, 5th Edition. Oxford University Press.

Carton, J. A., Giese, B. S., 2008. A reanalysis of ocean climate using simple ocean data assimilation (SODA). Monthly Weather Review 136 (8), 2999–3017.

Ciuffo, B., Punzo, V., Montanino, M., 2012. The Calibration of Traffic Simulation Models. Report on the Assessment of Different Goodness of Fit Measures and Optimization Algorithms. MULTITUDE Project-COST Action TU0903. Tech. rep., European Commission-Joint Research Centre.

Coifman, B., 2002. Estimating travel times and vehicle trajectories on freeways using dual loop detectors. Transportation Research Part A: Policy and Practice 36 (4), 351–364.

Constantinescu, E. M., Sandu, A., Chai, T., Carmichael, G. R., 2007. Ensemble-based chemical data assimilation. I: General approach. Quarterly Journal of the Royal Meteorological Society 133 (626), 1229–1243.

da Rocha, T. V., Leclercq, L., Montanino, M., Parzani, C., Punzo, V., Ciuffo, B., Villegas, D., 2015. Does traffic-related calibration of car-following models provide accurate estimations of vehicle emissions? Transportation Research Part D: Transport and Environment 34, 267–280.

Daganzo, C. F., 2005a. A variational formulation of kinematic waves: basic theory and complex boundary conditions. Transportation Research Part B: Methodological 39 (2), 187–196.

Daganzo, C. F., 2005b. A variational formulation of kinematic waves: solution methods. Transportation Research Part B: Methodological 39 (10), 934–950.

Darema, F., 2004. Dynamic data driven applications systems: A new paradigm for application simulations and measurements. In: Bubak, M., van Albada, G. D., Sloot, P. M. A., Dongarra, J. (Eds.), Computational Science - ICCS 2004. Springer Berlin Heidelberg, pp. 662–669.

Darema, F., 2005. Dynamic data driven applications systems: New capabilities for application simulations and measurements. In: Sunderam, V. S., van Albada, G. D., Sloot, P. M. A., Dongarra, J. J. (Eds.), Computational Science - ICCS 2005. Springer Berlin Heidelberg, pp. 610–615.

De Schutter, B., Van den Boom, T., 2008. Max-plus algebra and max-plus linear discrete event systems: An introduction. In: Proceedings of the 9th International Workshop on Discrete Event Systems. Göteborg, Sweden, pp. 36–42.

DiCesare, F., Kulp, P. T., Gile, M., List, G., 1994. The application of Petri nets to the modeling, analysis and control of intelligent urban traffic networks. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 2–15.

Djurić, P. M., Kotecha, J. H., Zhang, J., Huang, Y., Ghirmai, T., Bugallo, M. F., Miguez, J., 2003. Particle filtering. IEEE Signal Processing Magazine 20 (5), 19–38.

Douc, R., Cappé, O., Moulines, E., 2005. Comparison of resampling schemes for particle filtering. In: Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis. Zagreb, Croatia, pp. 64–69.

Edie, L. C., 1963. Discussion of traffic stream measurements and definitions. In: Proceedings of the 2nd International Symposium on the Theory of Traffic Flow. Paris, France, pp. 139–154.

Evensen, G., 2003. The ensemble Kalman filter: theoretical formulation and practical implementation. Ocean Dynamics 53 (4), 343–367.

Fujimoto, R. M., 2000. Parallel and Distributed Simulation Systems. Wiley New York.

Gillijns, S., Mendoza, O., Chandrasekar, J., De Moor, B. L. R., Bernstein, D., Ridley, A., 2006. What is the ensemble Kalman filter and how well does it work? In: Proceedings of the 2006 American Control Conference. Minneapolis, MN, USA, pp. 4448–4453.

Godsill, S., Vermaak, J., 2005. Variable rate particle filters for tracking applications. In: IEEE/SP 13th Workshop on Statistical Signal Processing. Bordeaux, France, pp. 1280–1285.

Godsill, S., Vermaak, J., Ng, W., Li, J., 2007. Models and algorithms for tracking of maneuvering objects using variable rate particle filters. Proceedings of the IEEE 95 (5), 925–952.

Goodall, N. J., Smith, B. L., Park, B. B., 2016. Microscopic estimation of freeway vehicle positions from the behavior of connected vehicles. Journal of Intelligent Transportation Systems 20 (1), 45–54.

Gu, F., 2010. Dynamic data driven application system for wildfire spread simulation. Ph.D. thesis, Georgia State University.

Gu, F., Hu, X., 2008. Towards applications of particle filters in wildfire spread simulation. In: Proceedings of the 2008 Winter Simulation Conference. Miami, FL, USA, pp. 2852–2860.

Gu, F., Yan, X., Hu, X., 2009. State estimation using particle filters in wildfire spread simulation. In: Proceedings of the 2009 Spring Simulation Multiconference. San Diego, CA, USA, pp. 34:1–34:8.

Heaney, K., Gawarkiewicz, G., Duda, T., Lermusiaux, P., 2007. Nonlinear optimization of autonomous undersea vehicle sampling strategies for oceanographic data-assimilation. Journal of Field Robotics 24 (6), 437–448.

Ho, Y.-C., 1989. Introduction to special issue on dynamics of discrete event systems. Proceedings of the IEEE 77 (1), 3–6.

Honig, H. J., Seck, M. D., 2012. ϕDEVS: Phase based discrete event modeling. In: Proceedings of the 2012 Symposium on Theory of Modeling and Simulation. Orlando, FL, USA, pp. 39:1–39:8.

Hu, X., 2011. Dynamic Data Driven Simulation. SCS M&S Magazine II (1), 16–22.

Hu, X., Sun, Y., Ntaimo, L., 2012. DEVS-FIRE: design and application of formal discrete event wildfire spread and suppression models. SIMULATION: Transactions of The Society for Modeling and Simulation International 88 (3), 259–279.

Huang, X.-Y., Xiao, Q., Barker, D. M., Zhang, X., Michalakes, J., Huang, W., Henderson, T., Bray, J., Chen, Y., Ma, Z., Dudhia, J., Guo, Y., Zhang, X., Won, D.-J., Lin, H.-C., Kuo, Y.-H., 2009. Four-dimensional variational data assimilation for WRF: Formulation and preliminary results. Monthly Weather Review 137 (1), 299–314.

Huang, Y., 2013. Automated simulation model generation. Ph.D. thesis, Delft University of Technology.

Ide, K., Courtier, P., Ghil, M., Lorenc, A. C., 1997. Unified notation for data assimilation : Operational, sequential and variational. Journal of the Meteorological Society of Japan, Special Issue on “Data Assimilation in Meteology and Oceanography: Theory and Practice” 75 (1B), 181–189.

Jacobs, P., 2005. The DSOL simulation suite. Ph.D. thesis, Delft University of Technology.

Jacobs, P., Lang, N., Verbraeck, A., 2002. D-SOL; a distributed Java based discrete event simulation architecture. In: Proceedings of the 2002 Winter Simulation Conference. San Diego, CA, USA, pp. 793–800.

Joines, J. A., Roberts, S. D., 1999. Simulation in an object-oriented world. In: Proceedings of the 1999 Winter Simulation Conference. Phoenix, AZ, USA, pp. 132–140.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., Wu, A. Y., 2002. An efficient k-means clustering algorithm: analysis and implementation. IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (7), 881–892.

Kesting, A., Treiber, M., 2008. Calibrating car-following models by using trajectory data: methodological study. Transportation Research Record: Journal of the Transportation Research Board 2088, 148–156.

Kothari, C. R., 2004. Research Methodology: Methods and Techniques, 2nd Edition. New Delhi: New Age International Publishers.

Kouichi, H., Turbelin, G., Ngae, P., Feiz, A. A., Barbosa, E., Chpoun, A., 2016. Optimization of sensor networks for the estimation of atmospheric pollutants sources. WIT Transactions on Ecology and the Environment 207, 11–21.

Lahoz, W. A., Khattatov, B., Menard, R., 2010. Data Assimilation: Making Sense of Observations, 1st Edition. Springer-Verlag Berlin Heidelberg.

Lahoz, W. A., Schneider, P., 2014. Data assimilation: Making sense of earth observation. Frontiers in Environmental Science 2 (16), 1–28.

Lang, N., Jacobs, P., Verbraeck, A., 2003. Distributed, open simulation model development with DSOL services. In: Proceedings 15th European Simulation Symposium. Delft, the Netherlands, pp. 210–218.

Lee, J., Lapira, E., Bagheri, B., Kao, H., 2013. Recent advances and trends in predictive manufacturing systems in big data environment. Manufacturing Letters 1 (1), 38–41.

Liu, H. X., Wu, X., Ma, W., Hu, H., 2009. Real-time queue length estimation for congested signalized intersections. Transportation Research Part C: Emerging Technologies 17 (4), 412–427.

Lorenc, A. C., Rawlins, F., 2005. Why does 4D-Var beat 3D-Var? Quarterly Journal of the Royal Meteorological Society 131 (613), 3247–3257.

Lu, X., Varaiya, P., Horowitz, R., Palen, J., November 2008. Faulty loop data analysis/correction and loop fault detection. In: 15th World Congress on Intelligent Transport Systems and ITS America's 2008 Annual Meeting. New York, NY, USA, pp. 1–12.

Mannila, H., Ronkainen, P., 1997. Similarity of event sequences. In: Fourth International Workshop on Temporal Representation and Reasoning. Daytona Beach, FL, USA, pp. 136–139.

Marinică, N. E., Sarlette, A., Boel, R. K., 2013. Distributed particle filter for urban traffic networks using a platoon-based model. IEEE Transactions on Intelligent Transportation Systems 14 (4), 1918–1929.

Mehran, B., Kuwahara, M., Naznin, F., 2012. Implementing kinematic wave theory to reconstruct vehicle trajectories from fixed and probe sensor data. Transportation Research Part C: Emerging Technologies 20 (1), 144–163.

Montanino, M., Punzo, V., 2015. Trajectory data reconstruction and simulation-based validation against macroscopic traffic patterns. Transportation Research Part B: Methodological 80, 82–106.

Nance, R. E., 1981. The time and state relationships in simulation modeling. Communications of the ACM - Special issue on simulation modeling and statistical computing 24 (4), 173–179.

Nantes, A., Ngoduy, D., Bhaskar, A., Miska, M., Chung, E., 2016. Real-time traffic state estimation in urban corridors from heterogeneous data. Transportation Research Part C: Emerging Technologies 66, 99–118.

Nichols, N. K., 2003. Data assimilation: Aims and basic concepts. In: Swinbank, R., Shutyaev, V., Lahoz, W. A. (Eds.), Data Assimilation for the Earth System. Springer Netherlands, pp. 9–20.

Ntaimo, L., Hu, X., Sun, Y., 2008. DEVS-FIRE: Towards an integrated simulation environment for surface wildfire spread and containment. SIMULATION: Transactions of The Society for Modeling and Simulation International 84 (4), 137–155.

Ören, T. I., Zeigler, B. P., 2012. System theoretic foundations of modeling and simulation: a historic perspective and the legacy of A Wayne Wymore. SIMULATION: Transactions of The Society for Modeling and Simulation International 88 (9), 1033–1046.

Overstreet, C. M., Nance, R. E., 2004. Characterizations and relationships of world views. In: Proceedings of the 2004 Winter Simulation Conference. Washington, D.C., USA, pp. 279–287.

Pegden, C. D., 2010. Advanced tutorial: Overview of simulation world views. In: Proceedings of the 2010 Winter Simulation Conference. Baltimore, MD, USA, pp. 210–215.

Pelc, J. S., 2013. Data assimilation for marine ecosystem models. Ph.D. thesis, Delft University of Technology.

Punzo, V., Montanino, M., 2016. Speed or spacing? Cumulative variables, and convolution of model errors and time in traffic flow models validation and calibration. Transportation Research Part B: Methodological 91, 21–33.

Punzo, V., Montanino, M., Ciuffo, B., 2015. Do we really need to calibrate all the parameters? Variance-based sensitivity analysis to simplify microscopic traffic flow models. IEEE Transactions on Intelligent Transportation Systems 16 (1), 184–193.

Saha, S., Bambha, N. K., Bhattacharyya, S. S., 2010. Design and implementation of embedded computer vision systems based on particle filters. Computer Vision and Image Understanding 114 (11), 1203–1214.

Saifuzzaman, M., Zheng, Z., 2014. Incorporating human-factors in car-following models: a review of recent developments and research needs. Transportation Research Part C: Emerging Technologies 48, 379–403.

Sargent, R. G., 2011. Verification and validation of simulation models. In: Proceedings of the 2011 Winter Simulation Conference. Phoenix, AZ, USA, pp. 183–198.

Saunders, M., Lewis, P., Thornhill, A., 2009. Research Methods for Business Students, 5th Edition. London: Prentice Hall.

Schriber, T. J., Brunner, D. T., Smith, J. S., 2012. How discrete-event simulation software works and why it matters. In: Proceedings of the 2012 Winter Simulation Conference. Berlin, Germany, pp. 1–15.

Seck, M., Verbraeck, A., 2009. DEVS in DSOL: Adding DEVS operational semantics to a generic event-scheduling simulation environment. In: Proceedings of the 2009 Summer Computer Simulation Conference. Istanbul, Turkey, pp. 261–266.

Shannon, R. E., 1975. Systems Simulation: The art and science. New Jersey: Prentice Hall.

Sun, Z., Ban, X. J., 2013. Vehicle trajectory reconstruction for signalized intersections using mobile traffic sensors. Transportation Research Part C: Emerging Technologies 36, 268–283.

Sun, Z., Hao, P., Ban, X. J., Yang, D., 2015. Trajectory-based vehicle energy/emissions estimation for signalized arterials using mobile sensing data. Transportation Research Part D: Transport and Environment 34, 27–40.

Treiber, M., Helbing, D., 2002. Reconstructing the spatio-temporal traffic dynamics from stationary detector data. Cooperative Transportation Dynamics 1, 3.1–3.24.

Treiber, M., Hennecke, A., Helbing, D., 2000. Congested traffic states in empirical observations and microscopic simulations. Physical Review E 62 (2), 1805–1824.

Treiber, M., Kesting, A., 2013. Traffic Flow Dynamics: Data, Models and Simulation. Springer Berlin Heidelberg.

van Hinsbergen, C. P. I. J., Schreiter, T., Zuurbier, F. S., van Lint, J. W. C., van Zuylen, H. J., 2012. Localized extended Kalman filter for scalable real-time traffic state estimation. IEEE Transactions on Intelligent Transportation Systems 13 (1), 385–394.

van Leeuwen, P. J., 2009. Particle filtering in geophysical systems. Monthly Weather Review 137 (12), 4089–4114.

van Lint, J. W. C., 2010. Empirical evaluation of new robust travel time estimation algorithms. Transportation Research Record 2160, 50–59.

van Lint, J. W. C., Hoogendoorn, S. P., 2010. A robust and efficient method for fusing heterogeneous data from traffic sensors on freeways. Computer-Aided Civil and Infrastructure Engineering 25 (8), 596–612.

van Lint, J. W. C., Hoogendoorn, S. P., 2015. A generic methodology to estimate vehicle accumulation on urban arterials by fusing vehicle counts and travel times. In: Transportation Research Board 94th Annual Meeting. No. 15-5134. Washington, D.C., USA.

van Wageningen-Kessels, F., van Lint, J. W. C., Vuik, K., Hoogendoorn, S. P., 2015. Genealogy of traffic flow models. EURO Journal on Transportation and Logistics 4 (4), 445–473.

Vangheluwe, H. L., 2000. DEVS as a common denominator for multi-formalism hybrid systems modelling. In: Proceedings of the 2000 IEEE International Symposium on Computer-Aided Control System Design. Anchorage, AK, USA, pp. 129–134.

Vangheluwe, H. L., 2001. The Discrete EVent System specification (DEVS) formalism. Tech. rep., McGill University, School of Computer Science, Montreal, Quebec, Canada. URL `http://www.cs.mcgill.ca/~hv/classes/MS/DEVS.pdf`

Wainer, G. A., 2009. Discrete-Event Modeling and Simulation: A Practitioner's Approach. CRC Press.

Wang, M., Hu, X., 2015. Data assimilation in agent based simulation of smart environments using particle filters. Simulation Modelling Practice and Theory 56, 36–54.

Wang, Y., Papageorgiou, M., Messmer, A., 2006. RENAISSANCE - A unified macroscopic model-based approach to real-time freeway network traffic surveillance. Transportation Research Part C: Emerging Technologies 14 (3), 190–212.

Weber, R., 2004. Editor's comments: The rhetoric of positivism versus interpretivism: a personal view. MIS Quarterly 28 (1), iii–xii.

Weinberg, G. M., 1971. The Psychology of Computer Programming. Van Nostrand Reinhold, New York.

Wu, P., Xue, H., Hu, X., 2015. Particle filter based traffic data assimilation with sensor informed proposal distribution. In: Proceedings of the 48th Annual Simulation Symposium. Alexandria, VA, USA, pp. 173–180.

Wu, W., Purser, R. J., Parrish, D. F., 2002. Three-dimensional variational analysis with spatially inhomogeneous covariances. Monthly Weather Review 130 (12), 2905–2916.

Wu, X., Liu, H. X., 2014. Using high-resolution event-based data for traffic modeling and control: An overview. Transportation Research Part C: Emerging Technologies 42, 28–43.

Xue, H., 2014. Data assimilation based on sequential Monte Carlo methods for dynamic data driven simulation. Ph.D. thesis, Georgia State University.

Xue, H., Gu, F., Hu, X., 2012. Data assimilation using sequential Monte Carlo methods in wildfire spread simulation. ACM Transactions on Modeling and Computer Simulation 22 (4), 23:1–23:25.

Xue, H., Hu, X., 2012. Exploiting sensor spatial correlation for dynamic data driven simulation of wildfire. In: Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation. Washington, D.C., USA, pp. 243–249.

Xue, H., Hu, X., 2013. An effective proposal distribution for sequential Monte Carlo methods-based wildfire data assimilation. In: Proceedings of the 2013 Winter Simulation Conference. Washington, D.C., USA, pp. 1938–1949.

Yuan, Y., 2013. Lagrangian multi-class traffic state estimation. Ph.D. thesis, Delft University of Technology.

Yuan, Y., van Lint, J. W. C., Wilson, R. E., van Wageningen-Kessels, F., Hoogendoorn, S. P., 2012. Real-time Lagrangian traffic state estimator for freeways. IEEE Transactions on Intelligent Transportation Systems 13 (1), 59–70.

Zeigler, B. P., Praehofer, H., Kim, T. G., 2000. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, 2nd Edition. Academic Press.

Summary

Enabled by the increased availability of data, the data assimilation technique, which incorporates measured observations into a dynamical system model to produce a time sequence of estimated system states, gains popularity. The main reason is that it can produce more accurate estimation results than using either a simulation model or the measurements. Due to this benefit, the data assimilation technique has been applied in many continuous systems applications, but very little data assimilation research has been found for discrete event simulations. With the application of new sensor technologies and communication solutions, the availability of data for discrete event systems has increased as well. The increased data availability for discrete event systems but the lack of related data assimilation techniques thus motivated this work on data assimilation for discrete event simulations.

Since discrete event simulations are highly nonlinear, non-Gaussian systems, particle filters are used to conduct data assimilation in discrete event simulations. However, applying particle filtering in discrete event simulations still encounters several theoretical and practical problems, such as the state retrieval problem (discrete event simulation models have a piecewise constant state trajectory, so the retrieved state was updated at a past time instant, with which inaccurate estimation results will be obtained), the variable dimension problem (the dimension of the state trajectory during a fixed time interval is random, leading to inapplicability of the standard sequential importance sampling algorithm), and the processing of non-numerical data. Therefore, this research aims to develop a particle filter based data assimilation framework for discrete event simulations, in which the aforementioned problems can be addressed.

In this research, we propose a particle filter based data assimilation framework for discrete event simulations, in which we assume that the measurements available at a time instant t are distributed over the last measurement interval (i.e. data fed at time t can contain observations occurring at any time instant during the last measurement interval), implying that the measurements are dependent on the state transitions during that measurement interval. The proposed data assimilation framework solves the aforementioned problems in two ways. First, an interpolation operation is introduced, which takes the elapsed time (i.e. the time elapsed since the last state transition) into account when retrieving the state of a discrete event simulation model. With interpolation, the real-time state evolution, which is not described in the discrete event simulation model but does happen in reality, can be captured. Second, to solve the variable dimension problem, the state trajectory with variable dimension is extended to a fixed dimension with certain extensions, e.g.,

by adding virtual entities. The standard sequential importance sampling algorithm can thus be applied to update the random measure (i.e. a set of particles and their importance weights) which approximates the joint distribution of the extended state trajectory with fixed dimension. Samples in which the extensions are discarded will form the samples from the joint distribution of the state trajectory of interest that has a variable dimension. We prove that the discarded extensions have no tangible effect on the weight update. Therefore in practice we can safely apply the sequential importance sampling algorithm to update the random measure that approximates the posterior distribution of a state trajectory with variable dimension. The proposed data assimilation framework can therefore be applied to discrete event simulations of both closed and open systems.

Two case studies, one in a (closed) gold mine system and the other in an (open) urban traffic system, are used to test and validate the proposed data assimilation framework. The results show that the proposed data assimilation framework is indeed able to provide accurate estimation results. A proper interpolation operation can significantly improve the estimation results in the statistic sense. The variable dimension state trajectory has no tangible effect on weight updating, and particle filtering can approximate the dimension accurately. The case study in the urban traffic system also contributes to a generic (in the sense that any (ensemble of) microscopic simulation models can be used) data assimilation framework for vehicle trajectory reconstruction on signalized urban arterials. The case studied for the urban traffic system implies that in order to effectively conduct data assimilation in open systems, two extra steps (compared with the data assimilation procedure in closed systems) are required. First, we need to estimate (directly from data or indirectly using an estimation method) the number of entities in the system, otherwise we will have an underdetermined problem, since the number of entities in an open system is unknown in advance. Second, we need to reconstruct entity arrivals at system boundaries based on (noisy) observations. One can also randomly generate a sequence of arrivals, but this would require a much larger number of particles to achieve similar performance.

Using data generated from the case studies, we analyze the characteristics of the proposed data assimilation framework and its sensitivity to a number of important parameters that relate to the errors in the data and in the simulation model, as well as to the number of particles employed. Sensitivity analysis with respect to data quality and model errors shows that the framework is quite robust to both errors in the data and the model errors (i.e. differences between the models generating the ground-truth data and the models used in the case studies), although we cannot claim to have tested this exhaustively. Sensitivity analysis for the number of particles employed reveals that with fixed data/model quality, the estimation errors decrease when the number of particles increases. However, the trend is not proportional, which means that it is impossible to achieve error-free estimation results by continuously increasing the number of particles, since the quality of the estimation results is most fundamentally determined by the data quality and the validity of the system model involved in the data assimilation process. The results of the sensitivity analysis also imply several general future research directions in order to improve the quality of the estimation results, such as building simulation models that can make more valid predictions of the real system behavior, and developing more advanced sensor technologies that can provide more accurate measurement data of the real systems.

To conduct controlled experiments in case studies, an implementation of the proposed data assimilation framework is needed. In this thesis, a conceptual software implementation is provided. Based on this, a concrete data assimilation library can be implemented for a specific simulation environment. The conceptual framework is fully object-oriented, therefore it is very easy to tailor, extend and maintain. We have to mention that this conceptual framework is definitely not optimal in terms of performance or ease of implementation. Future research will be oriented toward addressing the performance issues, e.g., by developing a parallel and distributed version of the proposed data assimilation framework in order to deal with more complex scenarios.

Samenvatting

De toegenomen beschikbaarheid van data maakt het mogelijk om data-assimilatietechnieken te gebruiken, die gemeten waarnemingen verwerken in een dynamisch systeemmodel, met als doel om een tijdreeks van geschatte systeemtoestanden te produceren. Data-assimilatie levert nauwkeuriger schattingsresultaten dan het gebruik van alleen het simulatiemodel of alleen de metingen. Vanwege dit voordeel wordt data-assimilatie al veel toegepast in continue systeemtoepassingen, maar data-assimilatieonderzoek binnen discrete-eventsimulaties is schaars. Met de toegenomen populariteit van nieuwe sensortechnologieën en communicatieoplossingen is ook de beschikbaarheid van gegevens voor discrete-eventsystemen toegenomen. De toegenomen beschikbaarheid van gegevens in discrete-eventsystemen, maar het gebrek aan gerelateerde data-assimilatietechnieken, motiveerde dit onderzoek naar data-assimilatie voor discrete-eventsimulaties.

Omdat discrete-eventsimulaties niet-lineaire, niet-Gaussiaanse systemen zijn, worden zogenaamde *particle filters* gebruikt om data-assimilatie uit te voeren voor discrete-eventsimulaties. Het toepassen van *particle filters* in discrete-eventsimulaties stuit echter nog steeds op diverse theoretische en praktische problemen, zoals het probleem van het bepalen van de toestand van het model wanneer een observatie binnenkomt (aangezien de toestand van een discrete-event simulatiemodel stuksgewijs constant is, zal de laatst bekende toestand van het model een toestand zijn die is bijgewerkt op een tijdstip in het verleden, zodat onnauwkeurige schattingsresultaten worden verkregen), het variabele-dimensieprobleem (de dimensie van de toestandsfunctie gedurende een vast tijdsinterval $[0, t]$ is willekeurig, wat leidt tot niet-toepasbaarheid van het standaard toegepaste sequentieel steekproefalgoritme), de verwerking van niet-numerieke gegevens, enz.. Daarom is de focus van dit onderzoek het ontwikkelen van een op *particle filters* gebaseerde data-assimilatiemethode voor discrete-eventsimulaties, waarbij de bovengenoemde problemen kunnen worden ondervangen.

In dit onderzoek ontwikkelen we een op *particle filters* gebaseerde data-assimilatiemethode voor discrete-eventsimulaties, waarbij we veronderstellen dat de observaties die beschikbaar zijn op tijdstip t, waarnemingen bevat die op elk willekeurig tijdstip tijdens het meetinterval kunnen hebben plaatsgevonden. De metingen zijn daarmee afhankelijk van de toestandsovergangen gedurende het meetinterval.

De voorgestelde methode voor data-assimilatie lost de bovengenoemde problemen op twee manieren op. Ten eerste wordt een interpolatiemethode geïntroduceerd, die rekening houdt met de verstreken tijd (dat wil zeggen de tijd sinds de laatste toestandsovergang), bij het bepalen van de toestand van een discrete-eventsimulatiemodel. Met interpolatie

kan het real-time verloop van de toestand, die niet wordt beschreven in het discrete event-simulatiemodel, maar wel in werkelijkheid plaatsvindt, worden vastgelegd. Ten tweede wordt, om het probleem van de variabele dimensie op te lossen, de toestandsfunctie met variabele dimensie uitgebreid tot een vaste dimensie met aanvullingen, bijvoorbeeld door virtuele entiteiten toe te voegen. Het standaard sequentiële algoritme om de gewichten van de *particles* te bepalen kan daarmee worden toegepast om een willekeurige meting te verwerken (dat wil zeggen een reeks *particles* en het bepalen van hun gewichten), die de gezamenlijke verdeling van de toestandsfunctie met vaste dimensie benadert. De gezamenlijke verdeling van de betreffende toestandsfunctie met variabele dimensie wordt gebaseerd op steekproeven waarin deze aanvullingen worden weggelaten. In het proefschrift wordt bewezen dat het weglaten van de aanvullingen voor de dimensie een verwaarloosbaar effect heeft op de update van de gewichten. Daarmee kan in de praktijk het sequentiële algoritme voor het bepalen van de gewichten van de *particles* toegepast worden om de willekeurige meetwaarde te verwerken die de *posterior* verdeling van een toestandstraject met variabele dimensie benadert. De voorgestelde data-assimilatiemethode kan daarmee worden toegepast op discrete-eventsimulaties van zowel gesloten als open systemen.

Twee *case studies*, één in een (gesloten) systeem van een goudmijn en de andere in een (open) stedelijk verkeerssysteem, worden gebruikt om de voorgestelde methode voor data-assimilatie te testen en te valideren. De resultaten tonen aan dat de voorgestelde data-assimilatiemethode inderdaad in staat is om nauwkeurige schattingsresultaten voor de toestand van het systeem te verschaffen. De genoemde interpolatiebewerking kan de schattingsresultaten in statistische zin aanzienlijk verbeteren. Zoals verondersteld, heeft het variabele aantal dimensies nauwelijks effect op het bijwerken van de gewichten, en de methode van *particle filtering* kan de dimensie nauwkeurig benaderen. De casestudie voor het stedelijk verkeerssysteem draagt tevens bij tot een generieke (in de zin dat elk (geheel van) microscopische simulatiemodellen kan worden gebruikt) data-assimilatie-methode voor reconstructie van voertuigtrajecten binnen een stedelijk verkeerssysteem met verkeerslichten. De casestudie in het stedelijk verkeerssysteem laat zien dat om data-assimilatie in open systemen effectief uit te voeren, twee extra stappen nodig zijn (in vergelijking met de procedure voor data-assimilatie in gesloten systemen). Ten eerste moeten we (direct uit de gegevens of indirect via een schattingsmethode) het aantal entiteiten in het systeem schatten, anders hebben we een ondergedimensioneerd probleem omdat het aantal entiteiten in een open systeem van tevoren onbekend is. Ten tweede moeten we de aankomst van entiteiten bij de systeemgrenzen reconstrueren op basis van (fouten bevattende) waarnemingen. Men kan dit ook oplossen door een willekeurige reeks aankomsten te genereren, maar dit zou een veel groter aantal *particles* vereisen om vergelijkbare prestaties te bereiken.

Aan de hand van gegevens uit de casestudies analyseren we de kenmerken van de voorgestelde data-assimilatiemethode en de gevoeligheid voor een aantal belangrijke parameters die betrekking hebben op fouten in de data en in het simulatiemodel, evenals het aantal *particles* dat gebruikt wordt. Gevoeligheidsanalyse met betrekking tot de gegevenskwaliteit en modelfouten toont aan dat het raamwerk vrij robuust is voor zowel fouten in de meetgegevens als modelfouten (verschillen tussen het model dat de *ground truth* data genereert en de modellen die in de casestudies zijn gebruikt), hoewel we niet kunnen beweren dat we dit uitputtend hebben getest. Gevoeligheidsanalyse met betrekking tot het aantal gebruikte *particles* laat zien dat met bij gelijkblijvende observaties en modelkwaliteit de schattingsfouten verminderen als het aantal *particles* toeneemt. De trend is echter niet proportioneel, wat betekent dat het onmogelijk is om foutloze schattingsresultaten te verkrijgen door het aantal *particles* te blijven verhogen, omdat de kwaliteit van de schattingsresultaten voornamelijk wordt bepaald door de gegevenskwaliteit en de validiteit van het model dat gebruikt wordt voor het data-assimilatieproces. De resultaten van de gevoeligheidsanalyse geven ook toekomstige onderzoeksrichtingen aan om de kwaliteit van de schattingsresultaten te verbeteren, zoals het bouwen van simulatiemodellen die het echte systeemgedrag meer valide kunnen voorspellen, en het ontwikkelen van sensortechnologieën die de meetgegevens van de echte systemen nauwkeuriger kunnen bepalen.

Om gecontroleerde experimenten uit te voeren in casestudies, is een implementatie van het voorgestelde data-assimilatiekader nodig. In dit proefschrift wordt een conceptueel software-raamwerk gepresenteerd, op basis waarvan een concrete data-assimilatiebibliotheek kan worden geïmplementeerd in een specifieke simulatieomgeving. Het conceptuele raamwerk is volledig object georiënteerd. Daarmee is het zeer eenvoudig aan te passen, uit te breiden en te onderhouden. Dit conceptuele kader is nog niet optimaal is qua prestaties en implementatiegemak. Toekomstig onderzoek zal gericht zijn op het aanpakken van de prestatieproblemen, bijvoorbeeld door een parallelle en gedistribueerde versie van de voorgestelde data-assimilatiemethode te ontwikkelen om met complexere scenario's om te kunnen gaan.

(The English summary has been translated into Dutch by Prof. dr. ir. A. Verbraeck)

SIKS Dissertation Series

The following list contains the most recent dissertations since 2011 in the SIKS Dissertation Series. For a complete overview, please see the SIKS website: `http://www.siks.nl/dissertations.php`.

201101 Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models

02 Nick Tinnemeier (UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

03 Jan Martijn van der Werf (TUE), Compositional Design and Verification of Component-Based Information Systems

04 Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference

05 Bas van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

06 Yiwen Wang (TUE), Semantically-Enhanced Recommendations in Cultural Heritage

07 Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction

08 Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues

09 Tim de Jong (OU), Contextualised Mobile Media for Learning

10 Bart Bogaert (UvT), Cloud Content Contention

11 Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective

12 Carmen Bratosin (TUE), Grid Architecture for Distributed Process Mining

13 Xiaoyu Mao (UvT), Airport under Control. Multiagent Scheduling for Airport Ground Handling

14 Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets

15 Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval

16 Maarten Schadd (UM), Selective Search in Games of Different Complexity

17 Jiyin He (UVA), Exploring Topic Structure: Coherence, Diversity and Relatedness

18 Mark Ponsen (UM), Strategic Decision-Making in complex games

19 Ellen Rusman (OU), The Mind's Eye on Personal Profiles

20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach

21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems

22 Junte Zhang (UVA), System Evaluation of Archival Description and Access

23 Wouter Weerkamp (UVA), Finding People and their Utterances in Social Media

24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

25 Syed Waqar ul Qounain Jaffry (VU), Analysis and Validation of Models for Trust Dynamics

26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns

28 Rianne Kaptein (UVA), Effective Focused Retrieval by Exploiting Query Context and Document Structure

29 Faisal Kamiran (TUE), Discrimination-aware Classification

30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions

31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science

33 Tom van der Weide (UU), Arguing to Motivate Decisions

34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training

36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach

37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization

39 Joost Westra (UU), Organizing Adaptation using Agents in Serious Games

40 Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development

41 Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control

42 Michal Sindlar (UU), Explaining Behavior through Mental State Attribution

43 Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge

44 Boris Reuderink (UT), Robust Brain-Computer Interfaces

45 Herman Stehouwer (UvT), Statistical Language Models for Alternative Sequence Selection

46 Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

47 Azizi Bin Ab Aziz (VU), Exploring Computational Models for Intelligent Support of Persons with Depression

48 Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

49 Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

---

201201 Terry Kakeeto (UvT), Relationship Marketing for SMEs in Uganda

02 Muhammad Umair (VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

03 Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories

04 Jurriaan Souer (UU), Development of Content Management System-based Web Applications

05 Marijn Plomp (UU), Maturing Interorganisational Information Systems

06 Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks

07 Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

08 Gerben de Vries (UVA), Kernel Methods for Vessel Trajectories

09 Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms

10 David Smits (TUE), Towards a Generic Distributed Adaptive Hypermedia Environment

11 J.C.B. Rantham Prabhakara (TUE), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

12 Kees van der Sluijs (TUE), Model Driven Design and Data Integration in Semantic Web Information Systems

13 Suleman Shahid (UvT), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

14 Evgeny Knutov (TUE), Generic Adaptation Framework for Unifying Adaptive Web-based Systems

15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

17 Amal Elgammal (UvT), Towards a Comprehensive Framework for Business Process Compliance

18 Eltjo Poort (VU), Improving Solution Architecting Practices

19 Helen Schonenberg (TUE), What's Next? Operational Support for Business Process Execution

20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval

22 Thijs Vis (UvT), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval

25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

26 Emile de Maat (UVA), Making Sense of Legal Text

27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

28 Nancy Pascall (UvT), Engendering Technology Empowering Women

29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval

30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making

31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning

33 Rory Sie (OUN), Coalitions in Cooperation Networks (COCOON)

34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications

35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes

37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation

38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms

39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks

40 Agus Gunawan (UvT), Information Access for SMEs in Indonesia

41 Sebastian Kelle (OU), Game Design Patterns for Learning

42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning

43 Withdrawn

44 Anna Tordai (VU), On Combining Alignment Techniques

45 Benedikt Kratz (UvT), A Model and Language for Business-aware Transactions

46 Simon Carter (UVA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

47 Manos Tsagkias (UVA), Mining Social Media: Tracking Content and Predicting Behavior

48 Jorn Bakker (TUE), Handling Abrupt Changes in Evolving Time-series Data

49 Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

50 Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering

51 Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

---

201301 Viorel Milea (EUR), News Analytics for Financial Decision Support

02 Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

03 Szymon Klarman (VU), Reasoning with Contexts in Description Logics

04 Chetan Yadati (TUD), Coordinating autonomous planning and scheduling

05 Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns

06 Romulo Goncalves (CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

07 Giel van Lankveld (UvT), Quantifying Individual Player Differences

08 Robbert-Jan Merk (VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

09 Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications

10 Jeewanie Jayasinghe Arachchige (UvT), A Unified Modeling Framework for Service Design.

11 Evangelos Pournaras (TUD), Multi-level Reconfigurable Self-organization in Overlay Services

12 Marian Razavian (VU), Knowledge-driven Migration to Services

13 Mohammad Safiri (UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly

14 Jafar Tanha (UVA), Ensemble Approaches to Semi-Supervised Learning Learning

15 Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications

16 Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation

17 Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid

18 Jeroen Janssens (UvT), Outlier Selection and One-Class Classification

19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling

20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval

21 Sander Wubben (UvT), Text-to-text generation by monolingual machine translation

22 Tom Claassen (RUN), Causal Discovery and Logic

23 Patricio de Alencar Silva (UvT), Value Activity Monitoring

24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning

25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning

27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance

28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience

29 Iwan de Kok (UT), Listening Heads

30 Joyce Nakatumba (TUE), Resource-Aware Business Process Management: Analysis and Support

31 Dinh Khoa Nguyen (UvT), Blueprint Model and Language for Engineering Cloud Applications

32 Kamakshi Rajagopal (OUN), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development

33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere

34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search

35 Abdallah El Ali (UvA), Minimal Mobile Human Computer Interaction

36 Than Lam Hoang (TUe), Pattern Mining in Data Streams

37 Dirk Börner (OUN), Ambient Learning Displays

38 Eelco den Heijer (VU), Autonomous Evolutionary Art

39 Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems

40 Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games

41 Jochem Liem (UVA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning

42 Léon Planken (TUD), Algorithms for Simple Temporal Reasoning

43 Marc Bron (UVA), Exploration and Contextualization through Interaction and Concepts

---

201401 Nicola Barile (UU), Studies in Learning Monotone Models from Data

02 Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method

03 Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions

04 Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation

05 Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability

06 Damian Tamburri (VU), Supporting Networked Software Development

07 Arya Adriansyah (TUE), Aligning Observed and Modeled Behavior

08 Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints

09 Philip Jackson (UvT), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language

10 Ivan Salvador Razo Zapata (VU), Service Value Networks

11 Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support

12 Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control

13 Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains

14 Yangyang Shi (TUD), Language Models With Meta-information

15 Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare

16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria

17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability

18 Mattijs Ghijsen (UVA), Methods and Models for the Design and Study of Dynamic Agent Organizations

19 Vinicius Ramos (TUE), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support

20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments

22 Marieke Peeters (UU), Personalized Educational Games - Developing agent-supported scenario-based training

23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era

24 Davide Ceolin (VU), Trusting Semi-structured Web Data

25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction

26 Tim Baarslag (TUD), What to Bid and When to Stop

27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty

28 Anna Chmielowiec (VU), Decentralized k-Clique Matching

29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software

30 Peter de Cock (UvT), Anticipating Criminal Behaviour

31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support

32 Naser Ayat (UvA), On Entity Resolution in Probabilistic Data

33 Tesfa Tegegne (RUN), Service Discovery in eHealth

34 Christina Manteli (VU), The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.

35 Joost van Ooijen (UU), Cognitive Agents in Virtual Worlds: A Middleware Design Approach

36 Joos Buijs (TUE), Flexible Evolutionary Algorithms for Mining Structured Process Models

37 Maral Dadvar (UT), Experts and Machines United Against Cyberbullying

38 Danny Plass-Oude Bos (UT), Making brain-computer interfaces better: improving usability through post-processing.

39 Jasmina Maric (UvT), Web Communities, Immigration, and Social Capital

40 Walter Omona (RUN), A Framework for Knowledge Management Using ICT in Higher Education

41 Frederic Hogenboom (EUR), Automated Detection of Financial Events in News Text

42 Carsten Eijckhof (CWI/TUD), Contextual Multidimensional Relevance Models

43 Kevin Vlaanderen (UU), Supporting Process Improvement using Method Increments

44 Paulien Meesters (UvT), Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.

45 Birgit Schmitz (OUN), Mobile Games for Learning: A Pattern-Based Approach

46 Ke Tao (TUD), Social Web Data Analytics: Relevance, Redundancy, Diversity

47 Shangsong Liang (UVA), Fusion and Diversification in Information Retrieval

---

201501 Niels Netten (UvA), Machine Learning for Relevance of Information in Crisis Response

02 Faiza Bukhsh (UvT), Smart auditing: Innovative Compliance Checking in Customs Controls

03 Twan van Laarhoven (RUN), Machine learning for network data

04 Howard Spoelstra (OUN), Collaborations in Open Learning Environments

05 Christoph Bösch (UT), Cryptographically Enforced Search Pattern Hiding

06 Farideh Heidari (TUD), Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes

07 Maria-Hendrike Peetz (UvA), Time-Aware Online Reputation Analysis

08 Jie Jiang (TUD), Organizational Compliance: An agent-based model for designing and evaluating organizational interactions

09 Randy Klaassen (UT), HCI Perspectives on Behavior Change Support Systems

10 Henry Hermans (OUN), OpenU: design of an integrated system to support lifelong learning

11 Yongming Luo (TUE), Designing algorithms for big graph datasets: A study of computing bisimulation and joins

12 Julie M. Birkholz (VU), Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks

13 Giuseppe Procaccianti (VU), Energy-Efficient Software

14 Bart van Straalen (UT), A cognitive approach to modeling bad news conversations

15 Klaas Andries de Graaf (VU), Ontology-based Software Architecture Documentation

16 Changyun Wei (UT), Cognitive Coordination for Cooperative Multi-Robot Teamwork

17 André van Cleeff (UT), Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs

18 Holger Pirk (CWI), Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories

19 Bernardo Tabuenca (OUN), Ubiquitous Technology for Lifelong Learners

20 Lois Vanhée (UU), Using Culture and Values to Support Flexible Coordination

21 Sibren Fetter (OUN), Using Peer-Support to Expand and Stabilize Online Learning

22 Zhemin Zhu (UT), Co-occurrence Rate Networks

23 Luit Gazendam (VU), Cataloguer Support in Cultural Heritage

24 Richard Berendsen (UVA), Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation

25 Steven Woudenberg (UU), Bayesian Tools for Early Disease Detection

26 Alexander Hogenboom (EUR), Sentiment Analysis of Text Guided by Semantics and Structure

27 Sándor Héman (CWI), Updating compressed colomn stores

28 Janet Bagorogoza (TiU), Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO

29 Hendrik Baier (UM), Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains

30 Kiavash Bahreini (OU), Real-time Multimodal Emotion Recognition in E-Learning

31 Yakup Koç (TUD), On the robustness of Power Grids

32 Jerome Gard (UL), Corporate Venture Management in SMEs

33 Frederik Schadd (TUD), Ontology Mapping with Auxiliary Resources

34 Victor de Graaf (UT), Gesocial Recommender Systems

35 Jungxao Xu (TUD), Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

---

201601 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines

02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow

03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support

04 Laurens Rietveld (VU), Publishing and Consuming Linked Data

05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers

06 Michel Wilson (TUD), Robust scheduling in an uncertain environment

07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training

08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data

09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts

10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms

11 Anne Schuth (UVA), Search Engines that Learn from Their Users

12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems

13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach

14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization

15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments

16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward

17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms

18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web

19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data

20 Daan Odijk (UVA), Context & Semantics in News & Web Search

21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground

22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems

23 Fei Cai (UVA), Query Auto Completion in Information Retrieval

24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach

25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior

26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains

27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media

28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control

29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning

30 Ruud Mattheij (UvT), The Eyes Have It

31 Mohammad Khelghati (UT), Deep web content monitoring

32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations

33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example

34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment

35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation

36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies

37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry

38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design

39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect

40 Christian Detweiler (TUD), Accounting for Values in Design

41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance

42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora

43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice

44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration

45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control

46 Jorge Gallego Perez (UT), Robots to Make you Happy

47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks

48 Tanja Buttler (TUD), Collecting Lessons Learned

49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis

50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

---

201701 Jan-Jaap Oerlemans (UL), Investigating Cybercrime

02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation

03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines

04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store

05 Mahdieh Shadi (UVA), Collaboration Behavior

06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search

07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly

08 Rob Konijn (VU) , Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery

09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text

10 Robby van Delden (UT), (Steering) Interactive Play Behavior

11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment

12 Sander Leemans (TUE), Robust Process Mining with Guarantees

13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology

14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior

15 Peter Berck (RUN), Memory-Based Text Correction

16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines

17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution

18 Ridho Reinanda (UVA), Entity Associations for Search

19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval

20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility

21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)

22 Sara Magliacane (VU), Logics for causal inference under uncertainty

23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces

24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning

25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search

26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch

27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors

28 John Klein (VU), Architecture Practices for Complex Contexts

29 Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"

30 Wilma Latuny (UvT), The Power of Facial Expressions

31 Ben Ruijl (UL), Advances in computational methods for QFT calculations

32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives

33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity

34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics

35 Martine de Vos (VU), Interpreting natural science spreadsheets

36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging

37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy

38 Alex Kayal (TUD), Normative Social Applications

39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR

40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems

41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle

42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets

43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval

44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering

45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement

46 Jan Schneider (OU), Sensor-based Learning Support

47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration

48 Angel Suarez (OU), Collaborative inquiry-based learning

201801 Han van der Aa (VUA), Comparing and Aligning Process Representations

02 Felix Mannhardt (TUE), Multi-perspective Process Mining

03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction

04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks

05 Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process

06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems

07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems

08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems

09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations

---

Curriculum Vitae

Xu Xie was born in Hanzhong, Shaanxi, China on 17 October, 1988. He completed his bachelor in Simulation Engineering at College of Mechatronic Engineering and Automation, National University of Defense Technology (NUDT) in June 2010. Then, he was recommended to continue his master study in Control Science and Engineering in the same college at NUDT without sitting the entrance examination, and graduated in December 2012 as Excellent Graduate Student (top 2%). In September 2013, Xu was awarded a four-year scholarship from the China Scholarship Council (CSC) and started his PhD project on data assimilation in discrete event simulations at Delft University of Technology, under the supervision of Prof. Alexander Verbraeck. His main research interests include modeling and simulation, distributed simulation, discrete event simulation, dynamic data driven simulation, data assimilation, and transportation.

ISBN: 978-94-6186-893-0