DATA ASSIMILATION FOR SIMULATION-BASED REAL-TIME PREDICTION/ANALYSIS

Xiaolin Hu

Department of Computer Science
Georgia State University
Atlanta, GA, USA
xhu@gsu.edu

ABSTRACT

There is growing interest in combining simulation and real-time data to support real-time prediction/analysis for complex dynamic systems. This paper presents a framework of using data assimilation to enable simulation-based real-time prediction/analysis for dynamic systems in operation. The different activities of the framework, including dynamic state estimation and online model calibration, are described. A demonstrative example is presented to show how these activities work together for a discrete event simulation application. Experiments results show the effectiveness of making data assimilation work with discrete simulations to support simulation-based real-time prediction/analysis.

Keywords: data assimilation, simulation-based prediction/analysis, real-time data, particle filters.

1 INTRODUCTION

The increasing amount of data collected from dynamic systems in real time poses a fundamental question to the modeling and simulation community: how to make simulation models and real-time data work together to enable real-time prediction/analysis for a dynamic system under study? While simulations have long been used to study dynamic systems, traditional simulation practices have mainly viewed simulation as an offline tool to support analysis for system design or planning. In these simulations, the simulation runs are based on historical data or hypothetic scenarios – they are not informed by real-time data collected from dynamic systems. More recently, there is growing interest in using simulation to support real-time decision making for dynamic systems in operation. The operational usage of simulation models in a real-time context requires simulation models to have the capability of providing real-time prediction/analysis for a system under study. Systematic ways of incorporating real-time data into simulation models are essential for achieving that goal.

Two important issues need to be addressed in order to achieve accurate simulation-based prediction/analysis for a dynamic system in operation. First, a simulation needs to start from an initial state that matches the actual state of the dynamic system. Since a dynamic system constantly changes its state, a simulation-based prediction should initialize itself with the real-time system state at the time of the simulation run. While this idea is straightforward to understand, it is not always easy to implement. This is because for many dynamic systems the real-time states (or some portion of the real-time states) are unknown and cannot be directly derived from the observation data collected from the dynamic systems. To give an example, a wildfire spread simulation needs to start from an initial fire front perimeter. Nevertheless, the observation data (e.g., images collected from a drone) at any moment may only cover a (small) portion the fire perimeter due to the large size of the fire. This brings the issue of dynamic state estimation that estimates the dynamically changing state of a system based on real-time observation data so that a simulation run can be

correctly initialized. Second, to predict how a dynamic system works a simulation model needs to calibrate its model parameters to accurately reflect the real-time characteristics of the system. Such calibration often needs to be done in an online fashion due to the following two reasons: 1) it is common for some characteristics of a system to be known only after the system actually operates in the field. This means one cannot simply assign some "typical" or "average" values for the corresponding model parameters. Instead, the parameters need to be estimated in real time based on how the system actually works; 2) complex dynamic systems may dynamically shift their characteristics due to changes of operating environment or working conditions. For these systems, the corresponding model parameters are not static and need to dynamically estimated based on real-time data from the system. This brings the issue of online model calibration that dynamically calibrates the parameters of a simulation model based on real-time data from a dynamic system.

Both issues of dynamic state estimation and online model calibration can be framed as data assimilation problems and addressed by data assimilation methods. Data assimilation is a methodology that combines observation data with a dynamic model of a system to optimally estimate the evolving state of the system. Data assimilation was originally developed in the field of meteorology to provide weather forecasts using numerical weather models and measurement data of weather conditions. It has since gained popularity in many other science and engineering fields, such as geosciences, oceanography, hydrology, and robotics. Data assimilation addresses the dynamic state estimation issue as it provides a systematic way to estimate a dynamic system's state based on real-time observation data. To address the online model calibration issue, a common approach is to formulate it as a joint state-parameter estimation problem. With this approach, the parameters that need to be calibrated are treated as part of the state vector that needs to be estimated. Then the same data assimilation method developed for state estimation can be applied to estimate the state and parameter values at the same time. The data assimilation works developed in other fields mainly dealt with numerical models (e.g., differential equation models) that have continuous state variables. They do not work well for simulation applications using discrete simulation models, such as discrete event models, discrete time models, and agent-based models.

In previous work we developed data assimilation for discrete simulations and applied it to wildfire spread simulation (Xue, Gu, and Hu 2012, Long and Hu 2017). The developed data assimilation methods are based on particle filters (Arnaud, Freitas, and Gordon 2001, Arulampalam et al. 2002). Particle filters are a set of sample-based methods that use Bayesian inference and stochastic sampling techniques to recursively estimate the state of dynamic systems from some given observations. Compared to Kalman filter-based methods, particle filters have the advantage that they are non-parametric filters that can represent arbitrary probability densities and work well with discrete simulation models. These features make particle filters effective methods to support data assimilation for discrete simulations. Our previous work (Hu and Wu 2019) focused on the data assimilation method for discrete simulations. This paper extends previous work by focusing on simulation-based real-time prediction/analysis enabled by data assimilation. A framework of using data assimilation for simulation-based real-time prediction/analysis is presented. A demonstrative example is developed to show how data assimilation and simulation can work together to support real-time prediction/analysis for a discrete event simulation application.

2 RELATED WORK

Using simulation to provide decision support for complex systems is one of the major applications of simulation. Simulations enable what-if analysis to study the impact of different decision choices. They also support sensitivity analysis and other analyses to provide useful information for decision making. A survey of using simulation for decision support in manufacturing systems is provided in (Kasie, Bright, and Walker 2017). According to the survey, using simulation to provide decision support can be classified into two broad categories: 1) system design and analysis simulation, and 2) system operational simulation. System design and analysis simulation is to analyze and evaluate a new system design prior to its final implementation. It is usually applied in long-term decision-making activities. System operational simulation is applicable for short-term planning, scheduling and control of manufacturing systems. An

example of system operational simulation can be found in (Heilala et al. 2010), which uses discrete event simulation to support customer-driven manufacturing system design and operations planning.

The need of integrating real-time or near real-time data with simulation is increasingly recognized in recent years. For example, Galan et al. (2021) proposed a method named as real-time reconciled simulation, which reconcile both historical and real-time data, and analyzes its usefulness as decision-making tool for process operators. Turker et al. (2019) developed a decision support system for dynamic job-shop scheduling using real-time data with simulation, which has been shown to increase the performance of dispatching rules in dynamic job scheduling. The integration of real-time data with a digital model is one of the key features of the emerging trend of *digital twin* (Rasheed, San, and Kvamsdal 2020). Most of the digital twin models are simulation models that model the dynamic behavior of the corresponding physical systems. A digital twin-enhanced dynamic job-shop scheduling methodology is proposed in Zhang, Tao, and Nee (2020). The need of using data assimilation to update digital twin models is discussed in Wright and Davidson (2020).

Data assimilation has been used in many fields to support real-time prediction from a dynamic model. The most well-known works happen in the weather forecast domain, where data assimilation is widely used to support weather prediction based on numerical weather models (see, e.g., Navon (2009), Houtekamer and Zhang (2016)). In other fields, for example, data assimilation is used to support rainfall-runoff prediction based on coupled atmospheric-hydrologic models (Wang et al. 2021). Data assimilation is also used to improve the predictability of COVID-19 spreading, see (e.g., Nadler et al. (2020)).

The problem of real-time prediction from data has also been studied by other approaches. In particular, the field of machine learning has achieved major advances in supporting real-time prediction from data (Sarker 2021). The machine learning approach involves using historical data to train a model (e.g., an artificial neural network model) that maps the input-output relationship for a dynamic system. The trained model is then used to predict the output for any given inputs based on real-time data. The machine learning approach has the advantage that it is applicable to a system, even when there is lack of knowledge about how the system works. Nevertheless, it heavily relies on the availability and quality of data. Furthermore, the trained models are largely black-box models that make it difficult to explain the prediction results. On the other hand, a simulation model explicitly models the mechanisms of how a dynamic system works. It utilizes the knowledge about a dynamic system, and thus makes it possible to predict a system's behavior even historical data does not exist. Furthermore, a simulation model can support real-time analysis, e.g., what-if analysis based on real-time conditions of a dynamic system. Kim et al. (2017) discussed the differences between the simulation approach and machine learning approach for studying dynamic systems. In this paper, we focus on real-time prediction/analysis using simulation models.

3 FRAMEWORK OF USING DATA ASSIMILATION FOR SIMULATION-BASED PREDICTION/ANALYSIS

We present a framework of using data assimilation for simulation-based real-time prediction/analysis. The framework includes multiple activities that are organized in a layered architecture, as shown in Figure 1. The ultimate goal of the framework is to support real-time decision making for a system under study (the top layer). To achieve this goal, simulation-based prediction/analysis for the system's future behavior is needed (the prediction/analysis layer). The simulation-based prediction/analysis would rely on an accurate assessment of the real-time situation of the system under study. This asks for dynamic state estimation so that a simulation-based prediction can be initialized from the estimated system state. Furthermore, simulation-based prediction/analysis would need the simulation model to accurately characterize the system in operation. This asks for online model calibration for the model parameters based on real-time data collected from the system. Both the dynamic state estimation and online model calibration activities are addressed by the data assimilation method (the data assimilation layer). Data assimilation combines information from both the real-time data and simulation model. The former is the observation data reflecting the system's real-time condition; the latter is the dynamic model describing how the system changes its

state over time. The real-time data and simulation model are part of the model & data layer, which is the foundation of the overall framework. Below we elaborate each component of the framework.

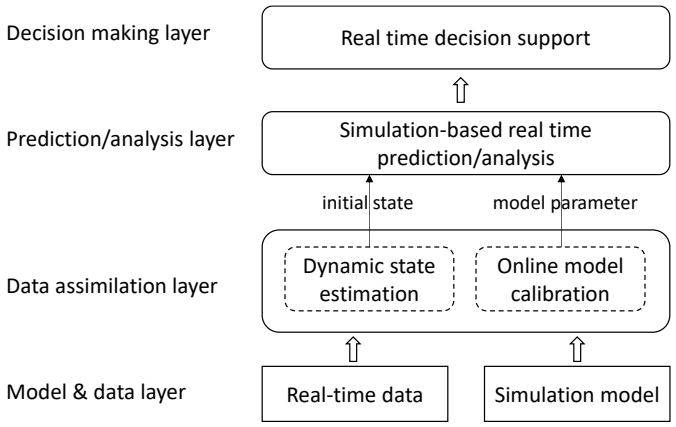


Figure 1: Framework of using Data Assimilation for Simulation-based Prediction/Analysis.

Real-time Data – Real-time data are real-time observation data collected from the dynamic system in operation. These observation data are collected by sensors deployed within the dynamic system that collect data with a certain frequency, e.g., every 1 minute or every 30 minutes. Typically, real-time data are noisy data subject to the accuracy of sensors.

Simulation Model **–** The simulation model is a dynamic model modeling how the dynamic system works. The operational usage of simulation models in a real-time context requires simulation models to be developed at the abstraction level corresponding to actual operations of dynamic systems. The simulation model serves two roles in the framework. First, during data assimilation, it serves as the state transition model of the state-space formulation for the dynamic system under study. The state transition model is used to predict a prior distribution of the system state, representing an estimation of the state without incorporating information from the observation data yet. Second, during simulation-based prediction/analysis, the simulation model is used to carry out simulation-based prediction/analysis. More details about each role of the simulation model are given below.

Data Assimilation Algorithm **–** The data assimilation algorithm is the specific algorithm for carrying out data assimilation. In this work, we consider only sequential data assimilation, where each step of the data assimilation utilizes the state estimate from the previous step to produce a new state estimate, resulting in a recursive process that can be applied whenever a new observation data becomes available. The sequential data assimilation employs the Bayesian filtering process. It adopts a state space formulation for a dynamic system under study that includes a state transition model and a measurement model. The former defines the evolution of the state and the latter defines the mapping from the state to the observation data. Based on these two models, each step of the data assimilation involves running the state transition model forward starting from the previous state estimate to predict a prior distribution of the state at the current step. The prior distribution is then updated by the likelihood probability computed from the measurement model and the observation data from the current step. The result of the update is the posterior distribution that represents the new state estimate after incorporating information from both the state transition model (i.e., the simulation model) and the observation data.

The two most widely used Bayesian filtering approaches are based on Kalman filter and particle filters. Due to the non-linear non-gaussian behavior and discrete or hybrid states (i.e., having both discrete and continuous state variables) of discrete simulations, we choose to use particle filters to carry out data assimilation for discrete simulations. Particle filters approximate the probability distribution of a state under estimation using a large set of random samples, named particles. These particles are propagated over time using importance sampling and resampling mechanisms. In each data assimilation step, the particle filtering algorithm receives a set of particles representing the previous belief of the system state, and an observation

data. During importance sampling, each particle is used to predict the next state by running the state transition model (the simulation model). The importance weight of each particle is then computed based on the measurement model and observation data. These weights are normalized after generating all the particles. During resampling, a new set of offspring particles are drawn with probability proportional to the normalized weights. More details about particle filters can be found in Arnaud, Freitas, and Gordon (2001) and Arulampalam et al. (2002). How to carry out particle-filter based data assimilation for discrete simulations can be found in our previous works (Xue, Gu, and Hu 2012, Hu and Wu 2019).

Dynamic State Estimation – The dynamic state estimation refers to the activity of estimating the dynamically changing state of a system in operation based on real-time observation data collected from the system. The estimated state is then used to initialize simulation runs for the simulation-based prediction/analysis. Dynamic state estimation is necessary because in most cases the observation data collected from a dynamic system are noisy and partial, which make it infeasible to derive the actual system state directly from the observation data. The dynamic state estimation is achieved through data assimilation.

Online Model Calibration – The online model calibration is the other activity at the data assimilation layer. It refers to calibrating the parameters of a simulation model in an online fashion based on real-time observation data collected from a dynamic system. The calibrated simulation model is then used to support more accurate simulation-based prediction/analysis. Online model calibration differs from the calibration work that happens during the modeling process before the operational usage of the model (we refer to this type of calibration as offline model calibration). The goal of offline model calibration is to make simulation results match the historical data or certain desired outputs. It is often formulated as an optimization problem to minimize the error between simulated results and the desired outputs. This compares to online model calibration that is formulated as a sequential filtering problem to estimate the model parameter based on real-time observation data. Despite their differences, offline calibration and online calibration are related. Offline calibration provides the baseline model parameters. Given the baseline model parameters, online calibration focuses on adjusting the parameter values to make them more accurately reflect the prevailing conditions of a system in operation.

Online model calibration is achieved through data assimilation too, where the parameters under estimation are considered as part of the states to be estimated. Since the online model calibration problem can be treated in a similar way as the dynamic state estimation problem, in the demonstrative example in Section 4 we focus only on the dynamic state estimation problem.

Simulation-based Real-time Prediction/Analysis – The simulation-based prediction/analysis refers to running simulations in a real-time fashion to predict or analyze a dynamic systems' future behavior. Note that we treat real-time prediction and real-time analysis as two slightly different activities. Real-time prediction is to predict a system's behavior based on the most likely path the dynamic system will take for the future. Real-time analysis, on the other hand, analyze how the system will behave under various hypothetic scenarios that are intentionally introduced for testing/evaluating the system. For example, a real-time what-if analysis would introduce a specific change to the system and then simulate. Despite this difference, both real-time prediction and real-time analysis must pertain to the actual operation of the system and thus need to start from the real-time state of the system.

To improve the accuracy of prediction/analysis, simulation-based prediction/analysis utilizes the dynamic state estimation and online model calibration from the data assimilation layer. In particle filter-based data assimilation, the state/parameter estimation results are represented by the set of particles. Each particle has a specific instance of the state/parameter estimation. When carrying out prediction/analysis, a simulation would be run for each particle using the initial state/model parameter represented by the particle. The prediction results from all the particles together form the distribution of the prediction results, from which statistical properties such as average and standard deviation can be computed.

Because data assimilation is carried out in a stepwise fashion, the simulation-based prediction/analysis is not a one-time activity. Instead, it may be invoked in each data assimilation step (or in every few steps).

Whenever it is invoked, it always uses the real-time data assimilation results from the current step to predict/analyze the future.

Real-time Decision Support – The real-time decision support is to use information from the simulation-based prediction/analysis to support real-time decision making for the dynamic system under study. How to carry out real-time decision support is application specific and not the focus of this paper.

4 THE ONE-WAY TRAFFIC CONTROL SYSTEM

As a demonstrative example, we consider data assimilation and simulation-based prediction/analysis for a one-way traffic control system that is often used during road construction. The one-way traffic control system is illustrated in Figure 2. During a road construction, the one-way traffic control is managed by two persons deployed to the west and east ends of the one-way traffic road segment. Each person carries a STOP/SLOW hand-held traffic paddle to control the traffic, where the STOP sign means vehicles should stop and wait, and the SLOW sign means vehicles can slowly move ahead to pass the road segment. It is assumed that the two persons coordinate and always use the STOP/SLOW signs in opposite directions: when one uses the STOP sign the other would use the SLOW sign. In the following description, we refer to the STOP sign as the red traffic light and the SLOW sign as the green traffic light, and refer to vehicles' moving directions as east-moving (moving towards east) and west-moving (moving towards west). During the time when the traffic light is green on a specific direction (east-moving or west-moving), the arrival vehicles moving in the opposite direction would be queued. The queues at the west side and east side of the road segment are named as the west-side queue and east-side queue, respectively.

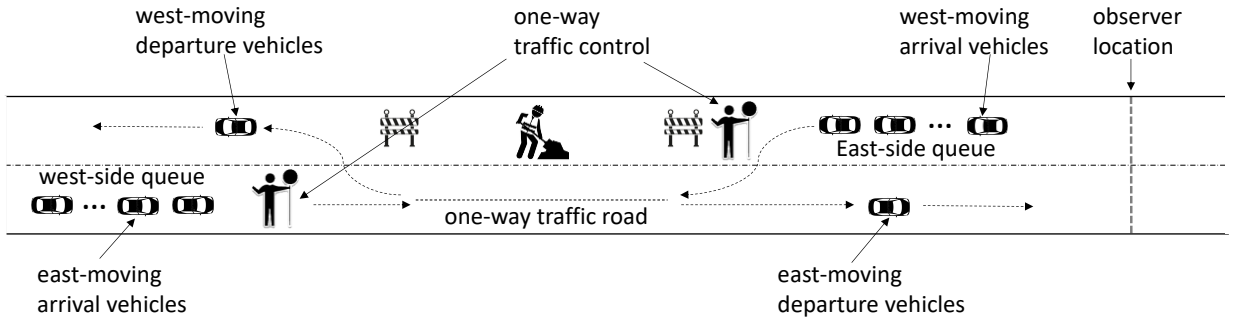


Figure 2: The one-way traffic control system.

To ensure construction workers' safety, only one vehicle is allowed to move on the one-way traffic road at any time. During a green light period, the traffic-control person on the corresponding side would signal a vehicle to move ahead only when the previous vehicle has finished crossing the road segment. The one-way traffic control system uses the following rules to switch traffic lights:

1. If the elapsed time for the current moving direction (east-moving or west-moving) has reached a pre-defined threshold (120 seconds in this example) and the opposite moving direction has cars waiting, switch the traffic light. Note: the traffic light switches only after the road segment is cleared if there is a car already moving on the road.
2. If the current moving direction has no car waiting and the opposite moving direction has cars waiting in the queue, switch the traffic light even if the 120-second threshold is not reached.
3. If the current moving direction has cars waiting in the queue and the opposite moving direction has no car waiting, keep the traffic light unswitched even after the 120-second threshold. In this case, the cars on the current moving direction keeps moving forward.
4. If none of the current moving direction and the opposite moving direction has any car needing to cross the road segment, keep the traffic light unswitched.

The vehicles at both sides of the road segment arrive randomly and independently, modeled by Poisson distributions. For the east-moving vehicles, the Poisson distribution has arriving rate $\lambda = 1/7$ (1 car per 7 seconds). For the west-moving vehicles, the arriving rate $\lambda = 1/10$ (1 car per 10 seconds). This means there are more east-moving vehicles than west-moving vehicles. The time it takes for a vehicle to cross the one-way traffic road segment is also a random number, modeled by a truncated normal distribution that has mean $\mu = 4.0$ (seconds) variance $\sigma^2 = 0.5^2$ and lies within the range of [3.0, 5.0].

To collect real-time data from the system, an observer (sensor) is deployed at the location on the east end of the one-way traffic road that is marked as the "observer location" in Figure 2. The observer is able to count the number of vehicles moving crossing its location for both the east-moving departure vehicles (named as *eastMoving_departure*) and west-moving arrival vehicles (named as *westMoving_arrival*). The observer reports data in every 30 seconds. It does not record the specific time that a vehicle crosses the observation location – all it reports is the number of vehicles that have departed and arrived in the past time interval. The data reported by the observer is noisy. In this example, a 10% noise is added to the actual number of vehicles crossing the observer location for both the *eastMoving_departure* and *westMoving_arrival* observation data (rounded to the closest integer values).

The one-way traffic control system is modeled by a discrete event simulation model based on the Discrete Event System Specification (DEVS) formalism (Zeigler, Praehofer, and Kim 2000). The discrete event simulation model is a DEVS coupled model that includes three atomic models: *eastMovCarGenr*, *westMovCarGenr*, and *oneWayTrafficRoad*. The three atomic models are described below. Specific code of these models as well as their couplings are omitted to save space.

- *eastMovCarGenr*: generates the east-moving traffic arriving at the west side of the road segment. Traffic are generated based on a Poisson distribution with rate $\lambda = 1/7$.
- *westMovCarGenr*: generates the west-moving traffic arriving at the east side of the road segment. Traffic are generated based on a Poisson distribution with rate $\lambda = 1/10$.
- *oneWayTrafficRoad*: models the one-way traffic road segment, including the traffic control logic as well as the time for vehicles to cross the road segment. This model has two input ports *east-MovArival* and *westMovArrival* that receive vehicles generated from the *eastMovCarGenr* and *westMovCarGenr*, respectively. The vehicles finishing crossing the road segment are sent out as outputs through the *eastMovDeparture* and *westMovDeparture* output ports of the model.

A fourth atomic model called *dataSavingTransducer* is used to save data for data assimilation. This model is coupled to the above three models so that it monitors all the input and output messages of the three models. The corresponding data are saved and used in data assimilation.

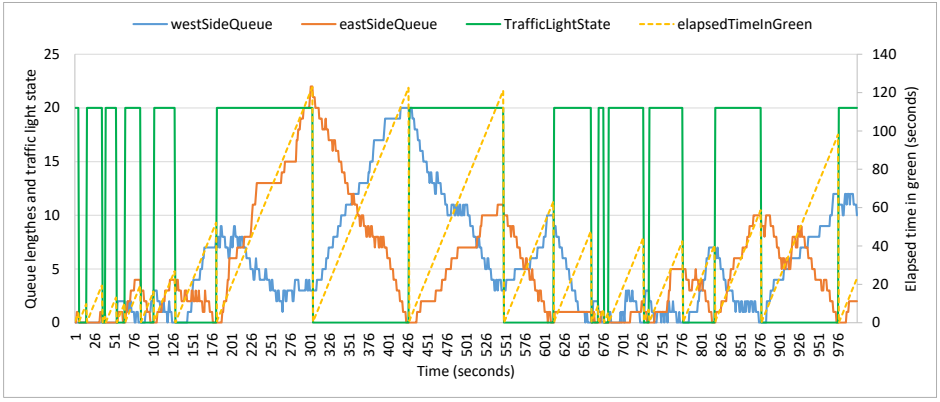


Figure 3: Simulation results of the one-way traffic control model.

Figure 3 shows the results of a specific simulation that runs for 1000 seconds of simulation time. The values of four state variables *westSideQueue* size, *eastSideQueue* size, *TrafficLightState*, and *elapsedTimeInGreen*

are displayed over time. The *trafficLightState* has two possible values: "east-moving green" or "west-moving green". The former is displayed as value 20, and the latter is displayed as value 0. The *elapsedTimeInGreen* is the elapsed time in the "east-moving green" or "west-moving green" state. It is a continuous variable with non-negative values. The *elapsedTimeInGreen* is reset to zero whenever the traffic light switches.

One can see that the time it takes for the traffic light to switch is irregular, which is governed by the traffic control rules and influenced by the number of vehicles waiting on the east side and west side queues. When both queues have vehicles waiting, the traffic light switches after *elapsedTimeInGreen* bypasses 120 seconds. When the queues on both sides are small, the traffic light may switch frequently because once all the vehicles on one side pass the road segment the traffic light switches immediately to allow vehicles on the other side to use the road. One can also see that when the *TrafficLightState* is "east-moving green", the *westSideQueue* decreases because vehicles on the west side can move forward (note that the queue may also increase occasionally because new vehicles may arrive and join the queue). A similar pattern can be observed for the *eastSideQueue*.

5 DATA ASSIMILATION

The goal of data assimilation is to dynamically estimate the state of the one-way traffic control system based on real-time observation data collected by the observer. In this example, the observation data are *eastMoving_departure* and *westMoving_arrival*. The system state incudes four state variables *westSideQueue, eastSideQueue, TrafficLightState, and elapsedTimeInGreen*. None of them can be directly derived from the observation data.

We use the identical twin experiment design to set up the data assimilation experiments. In this design, a simulation is first run to act as the ground truth model. The specific configuration of the ground truth model is unknown to the data assimilation. The data assimilation only knows the observation data collected from the ground truth model, and needs to assimilate the observation data to estimate the ground truth model's state (referred to as the ground truth state). For this example, the system behavior is driven by the east-moving and west-moving traffic as well as the time for vehicles to cross the road segment, all of which are modeled by random number generation. By changing the seeds of random number generation, we can produce different simulation results. Based on this idea, in the identical twin experiment design we set up a ground truth model using a specific set of random number seeds that are unknown to the data assimilation. We consider two ground truth models named as Ground truth case I and Ground truth case II. The observation data from the two ground truth cases are displayed in Figure 4. In both cases, the observation data are collected every 30 seconds for a total period of 3000 seconds.

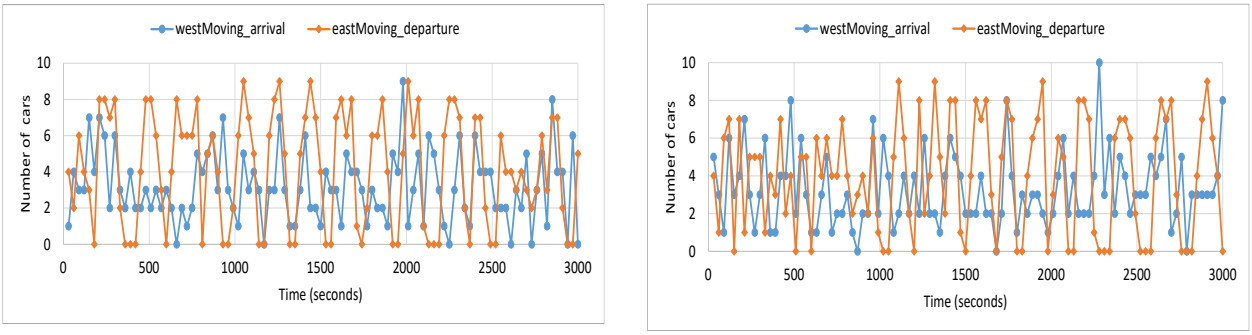


Figure 4: Observation data: ground truth case I (left); ground truth case II (right).

We carry out data assimilation following the standard particle filter-based data assimilation procedure described in Hu and Wu (2019). The data assimilation is carried out for 100 steps, with the time interval of each step to be 30 seconds. The particle filter algorithm uses 1000 particles. During initialization, all the particles are initialized with randomly generated state values. Each data assimilation step goes through an

importance sampling procedure and a resampling procedure. During the importance sampling, each particle evolves its state to a new state by running the simulation model for 30 seconds (corresponding to the time interval of one data assimilation step). Once a new particle is generated, its importance weight is computed based on the likelihood of observing the ground truth *eastMoving_departure* and *westMoving_arrival* data. Then the importance weights of all the particles are normalized, and resampling is invoked to generate a new set of particles that represent the state estimate for that data assimilation step.

Figure 5 shows the data assimilation results for estimating the *westSideQueue* and *eastSideQueue* for both the ground truth model case I and case II. The state estimation in each data assimilation step is computed as the average from all the particles (after the resampling). The corresponding "true" states from the two ground truth models are also displayed in the figure.

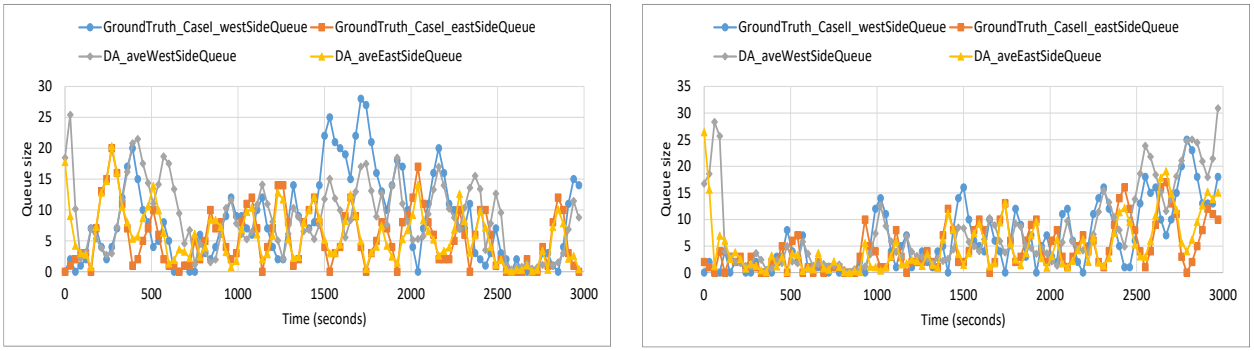


Figure 5: State estimation results: case I (left); case II (right).

One can see that in both cases, after first several steps the state estimation for *westSideQueue* and *eastSideQueue* begin to track the corresponding true states. Taking case II as an example: before time 1000s when the true *westSideQueue* and *eastSideQueue* are small, the estimated states are also small; after time 2000s when the true *westSideQueue* and *eastSideQueue* become large, the estimated states become large too. Furthermore, the cyclic increase and decrease of the estimated *westSideQueue* and *eastSideQueue* generally match those of the corresponding true states. The *eastSideQueue* has smaller estimation errors because the observation data *westMoving_arrival* is a direct observation of the number of vehicles arriving at the east side of the road segment. Since the focus of this paper is on simulation-based prediction/analysis as opposed to the data assimilation itself, we omit the quantitative analysis of the data assimilation results.

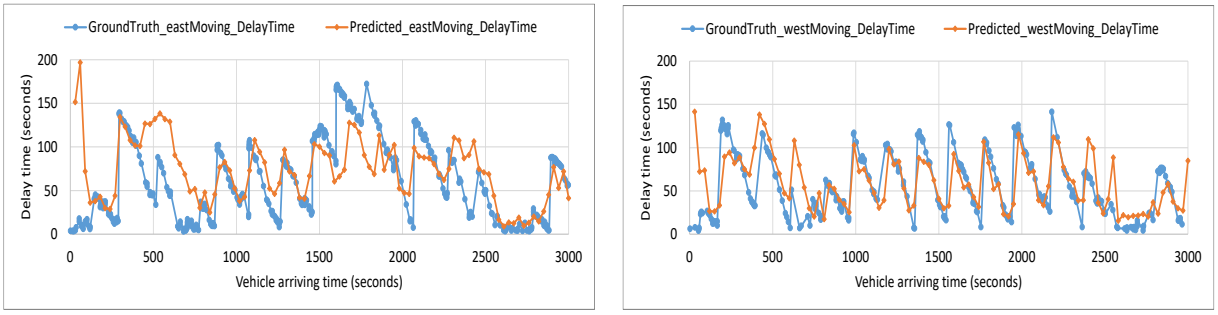
6 SIMULATION-BASED REAL-TIME PREDICTION/ANALYSIS

The data assimilation results presented above allow us to carry out simulation-based prediction/analysis for the one-way traffic control system. Multiple types of prediction/analysis may be carried out. In this work we are interested in the real-time traffic delay time. The real-time traffic delay time at time t is defined as the time for a vehicle newly arriving at time t to finish crossing the road segment. This includes the time for the vehicle to wait in the queue and the time for the vehicle to move through the road segment when the traffic light is green. The real-time traffic delay time changes dynamically, influenced by the number of vehicles waiting in the queue at time t, the traffic light state at time t, as well as how the one-way traffic control system works after time t. Due to the asymmetric traffic at the two sides of the road segment, the real-time delay time for east-moving and west-moving vehicles would be different.

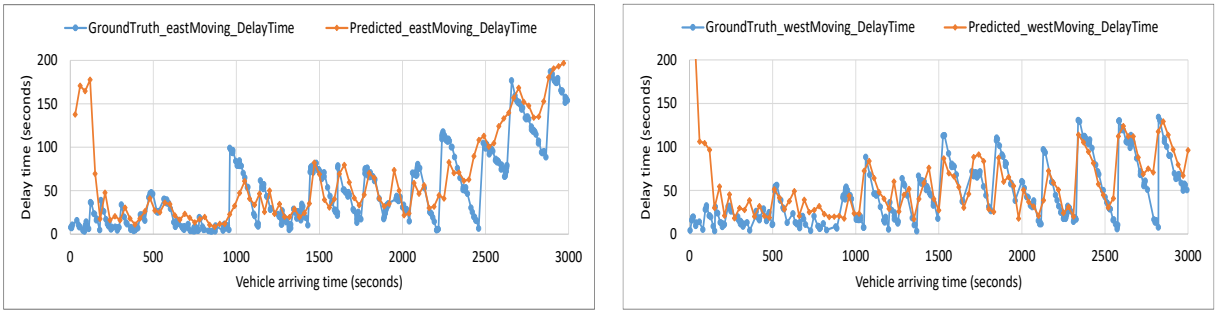
To predict the real-time delay time at each data assimilation step, after each data assimilation we run simulations to simulate the dynamics of the one-way traffic control system starting from the "current" system state estimated by the data assimilation. The simulations need to take into consideration the future vehicles that will arrive at the road segment too, because the one-way traffic control rules are dynamically influenced by the number of vehicles arriving at both sides of the road segment. Since the estimated state is represented by a set of particles, a simulation is created for each particle with the initial state defined by

the particle's value. More specifically, at time t_k of data assimilation step k, each particle in the resampled particle set would create a simulation including the *oneWayTrafficRoad* model as well as the *eastMovCarGenr* and *westMovCarGenr* models. Let $x_k^{(i)}, i = 1, 2, \ldots, N$ be the set of particles at step k. For each particle $x_k^{(i)}$, the *oneWayTrafficRoad* model is initialized using the particle state $x_k^{(i)}$. To simulate the delay time for vehicles arriving at t_k, we add a special vehicle at the end of the *westSideQueue* and the *eastSideQueue* during the initialization of the simulation model. The duration for the two special vehicles to finish crossing the road segment is monitored and used as the predicted east-moving delay time and west-moving delay time from the corresponding particle.

Figure 6 shows the predicted east-moving and west-moving delay time and the actual delay time at each data assimilation step for the ground truth case I and case II. The predicted delay time is computed as the average from all the particles. In the figure, the horizontal axis is the vehicle arriving time and the vertical axis is the traffic delay time. A data point (t_x, t_y) means that at time t_x the predicted (or actual) delay time is t_y. Since the predictions happen only at the time corresponding to the data assimilation steps, the predicted delay time has data points only in every 30 seconds. To evaluate the prediction results, the actual east-moving and west-moving delay time for each vehicle in the ground truth models is measured and displayed in the figure too. Note that for the actual delay time data points, their t_x values do not match exactly with those from the predicted delay time data points. This is because the ground truth models may not have a vehicle arriving at exactly the data assimilation time.



(a) Predicted traffic delay time for case I



(b) Predicted traffic delay time for case II

Figure 6: Traffic delay time prediction.

Several observations can be made from Figure 6. First, the predicted delay time has large errors during the first several data assimilation steps. This is understandable because during those steps the data assimilation results have not converged. In the below analysis we exclude those steps. Table 1 shows a quantitative analysis of the prediction results. We calculate the prediction Absolute Error (AE) in each data assimilation step as the absolute difference between the actual delay time and the predicted delay time. Table 1 shows the mean absolute error (MAE), which is the mean AE from all the steps, and the percentage of steps whose AE is less than 30.0 seconds. One can see that except for the predicted east-moving delay time of Case I that has MAE of 25.7 seconds, all other predictions have MAE less than 20 seconds. Furthermore, in all the

predictions except for the predicted east-moving delay time of case I, more than 80% of the steps has AE less than 30 seconds, which is considered as a reasonable prediction error for this application. Compared to the east-moving delay time, the predicted west-moving delay time is more accurate because the data assimilation gives more accurate estimation for the *eastSideQueue*.

Table 1: Simulation-based prediction results.

	CASE I		CASE II	
	East-moving delay time	West-moving delay time	East-moving delay time	West-moving delay time
MAE (seconds)	25.7	19.6	17.5	17.1
Percentage of steps with AE < 30	72.9%	83.3%	84.2%	87.3%

Second, a pattern can be observed that the prediction errors fluctuate in a cyclic fashion that coincides with the cyclic switches of the traffic light. The prediction errors are generally larger around the time when the ground truth traffic light switches directions. This is because a switch of the traffic light causes an abrupt increase of the delay time for the vehicles that are stopped by the traffic light. In data assimilation, not all particles precisely estimate the time when the ground truth traffic light switches (particles may estimate the traffic light switch time a bit earlier or later). This means particles do not predict the happening of the abrupt changes of the traffic delay time at exactly the same time. The averaged prediction results are thus smoother around the time when the ground truth delay time abruptly changes. This smoothing effect causes relatively large prediction errors around the time when the traffic light switches directions.

Third, the prediction accuracy is closely related to the state estimation accuracy from data assimilation. Take case I as an example. During time 400s-700s and 1500s-1800s, the estimated west-side queue have relatively large errors. These estimation errors make the predicted east-moving delay time during the two time periods have relatively large errors too. In general, to achieve more accurate predictions more accurate state estimation would be needed. For example, one may deploy more sensors, e.g., to deploy sensors at the west side of the road segment too, to collect more data, which would improve the state estimation results.

7 CONCLUSION

This paper presents a framework of using data assimilation to enable simulation-based real-time prediction/analysis for discrete simulation applications. A demonstrative example is developed to show how the different activities of the framework work together. Experiment results show the effectiveness of the framework as well as the potential of applying the framework to other discrete simulation applications. Future work includes more comprehensive analysis of the data assimilation and real-time prediction/analysis results, and applying the framework to other discrete simulation applications.

ACKNOWLEDGMENTS

This work is supported in part by the US Department of Agriculture (USDA) National Institute of Food and Agriculture (NIFA) under grant number 2019-67021-29011.

REFERENCES

Arnaud, D., N. D. Freitas, and N. Gordon. 2001. “An Introduction to Sequential Monte Carlo Methods”. In *Sequential Monte Carlo methods in practice*, 3-14. Springer. New York, NY.

Arulampalam, M. S., S. Maskell, N. Gordon, and T. Clapp. 2002. “A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking”. *IEEE Trans. Signal Processing*, Vol.50, No.2, 174-188.

Galan, A., C. D. Prada, G. Gutierrez, D. Sarabia, and R. Gonzalez. 2021. "Real-Time Reconciled Simulation as Decision Support Tool for Process Operation". *Journal of Process Control*, vol. 100, pp. 41-64.

Heilala, J., J. Montonen, P. Jarvinen, and S. Kivikunnas. 2010. "Decision Support using Simulation for Customer-Driven Manufacturing System Design and Operations Planning". In A*dvances in decision support systems*. INTECH, Croatia.

Houtekamer, P. L., and F. Zhang. 2016. "Review of the Ensemble Kalman Filter for Atmospheric Data Assimilation". *Monthly Weather Rev.*, vol. 144(12), pp. 4489–4532.

Hu, X., and P. Wu. 2019. "A Data Assimilation Framework for Discrete Event Simulations". *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 29(3), no. 17.

Kasie, F. M., G. Bright, and A. Walker. 2017. "Decision Support System in Manufacturing: A Survey and Future Trends". *Journal of Modelling in Management* 12(3) 432-454.

Kim, B. S., B. G. Kang., S. H. Choi., and T. G. Kim. 2017. "Data Modeling Versus Simulation Modeling in the Big Data Era: Case Study of a Greenhouse Control System." *Simulation* vol. 93, pp. 579–94.

Long, Y., and X. Hu. 2017. "Spatial Partition-Based Particle Filtering for Data Assimilation in Wildfire Spread Simulation". *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, vol. 3(2), No. 5.

Nadler, P., S. Wang, R. Arcucci, X. Yang, and Y. Guo. 2020. "An Epidemiological Modelling Approach for Covid19 Via Data Assimilation". *European Journal of Epidemiology*, vol. 35, pp.749–761.

Navon, I. M. 2009. "Data Assimilation for Numerical Weather Prediction: A Review", in *Data Assimilation for Atmospheric, Oceanic and Hydrologic Applications*, S. K. Park and L. Xu, Eds. New York, NY, USA: Springer, 2009, pp. 21–65.

Rasheed, A., O. San, and T. Kvamsdal. 2020. "Digital Twin: Values, Challenges and Enablers from a Modeling Perspective". *IEEE Access*, vol. 8, pp. 21980–22012.

Sarker, I. H. 2021. "Machine learning: Algorithms, Real-World Applications and Research Directions". *SN Comput. Sci.* 2021, vol. 2, pp. 1–21.

Turker, A., A. Aktepe, A. Inal, O. Ersoz, G. Das, and B. Birgoren. 2019. "A Decision Support System for Dynamic Job-Shop Scheduling using Real-Time Data with Simulation". *Mathematics* vol. 7, pp. 278.

Wang, W., J. Liu, C. Li, Y. Liu, and F. Yu. 2021. "Data Assimilation for Rainfall-Runoff Prediction Based on Coupled Atmospheric-Hydrologic Systems with Variable Complexity". *Remote Sens*. vol. 13, pp. 595.

Wright L., and S. Davidson. 2020. "How to Tell the Difference between a Model and a Digital Twin". *Adv. Model. Simul. Eng. Sci.*, vol. 7(1), No. 13.

Xue, H., F. Gu, and X. Hu. 2012. "Data Assimilation Using Sequential Monte Carlo Methods in Wildfire Spread Simulation". *The ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 22(4), No. 23.

Zhang, M., F. Tao, and A. Y. C. Nee. 2020. "Digital Twin Enhanced Dynamic Job-Shop Scheduling". *J. Manuf. Syst.*, vol. 58(B), pp. 146-156.

Zeigler, B.P., B. Praehofer, T.G. Kim. 2000. *Theory of Modelling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd edition. Academic Press.

AUTHOR BIOGRAPHIES

XIAOLIN HU is a Professor of Computer Science at Georgia State University. He received his Ph.D. degree from the University of Arizona. His research interests include modeling and simulation theory and application, complex systems science, agent and multi-agent systems. His email address is xhu@gsu.edu.