TU Delft

Delft University of Technology

A particle filter-based data assimilation framework for discrete event simulations

Xie, Xu; Verbraeck, Alexander

DOI
10.1177/0037549718798466

Publication date
2018

Document Version
Final published version

Published in
Simulation

Citation (APA)
Xie, X., & Verbraeck, A. (2018). A particle filter-based data assimilation framework for discrete event simulations. *Simulation*. https://doi.org/10.1177/0037549718798466

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Check for updates

Simulation

Methodology

Simulation: Transactions of the Society for Modeling and Simulation International
1–27
© The Author(s) 2018

DOI: 10.1177/0037549718798466
journals.sagepub.com/home/sim
SAGE SCS

A particle filter-based data assimilation framework for discrete event simulations

Xu Xie[1,2] and Alexander Verbraeck[1]

Abstract

With the advent of new sensor technologies and communication solutions, the availability of data for discrete event systems has greatly increased. This motivates research on data assimilation for discrete event simulations that has not yet fully matured. This paper presents a particle filter-based data assimilation framework for discrete event simulations. The framework is formally defined based on the Discrete Event System Specification formalism. To effectively apply particle filtering in discrete event simulations, we introduce an interpolation operation that considers the elapsed time (i.e., the time elapsed since the last state transition) when retrieving the model state (which was ignored in related work) in order to obtain updated state values. The data assimilation problem finally boils down to estimating the posterior distribution of a state trajectory with variable dimension. This seems to be problematic; however, it is proven that in practice we can safely apply the sequential importance sampling algorithm to update the random measure (i.e., a set of particles and their importance weights) that approximates this posterior distribution of the state trajectory with variable dimension. To illustrate the working of the proposed data assimilation framework, a case is studied in a gold mine system to estimate truck arrival times at the bottom of the vertical shaft. The results show that the framework is able to provide accurate estimation results in discrete event simulations; it is also shown that the framework is robust to errors both in the simulation model and in the data.

Keywords

Data assimilation, discrete event simulations, particle filters, state interpolation

1 Introduction

Enabled by the increased availability of data, the data assimilation technique,[1] which incorporates measured observations into a dynamical system model to produce a time sequence of estimated system states, is gaining popularity. The main reason is that it can produce more accurate estimation results than using a single source of information from either the 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,[2] the availability of data for discrete event systems has increased as well, such as data from machines and processes,[3] or high-resolution event data in traffic.[4] The increased data availability for discrete event systems but the lack of related data assimilation techniques thus motivates this work on data assimilation in discrete event simulations.

1.1 Characteristics of discrete event simulations

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

[1]Department of Multi Actor Systems, Faculty of Technology, Policy, and Management, Delft University of Technology, Netherlands
[2]Department of Modeling and Simulation, College of System Engineering, National University of Defense Technology, China

Corresponding author:
Xu Xie, Department of Multi Actor Systems, Faculty of Technology, Policy, and Management, Delft University of Technology, Jaffalaan 5, 2628 BX Delft, Netherlands.
Email: x.xie@hotmail.com

by the discrete event approach.[5] 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*.[6] State changes are only captured at discrete, but possibly random, instants,[7] where such a change in state occurring at an instant is called an *event*.[6] Since the discrete event approach jumps from one event to the next, omitting the behavior in between, it can be very efficient.[8]

The key characteristics of discrete event simulations can be summarized as follows. Firstly, 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 Section 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, that is, busy or idle, is also a discrete state variable. Secondly, 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. Finally, 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 of a discrete event simulation model is thus piecewise constant, as shown in Figure 1, which only captures changes of interest in the real state evolution.

1.2 Data assimilation in discrete event simulations

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.[9] Therefore, data assimilation relies on the following three elements to work, namely the *system model* that describes the evolution of the state over time, the *measurement model* that relates noisy observations to the state, and the *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.[1] In the literature, many data assimilation techniques exist, such as the Kalman filter,[10] the extended Kalman filter,[11] and the ensemble Kalman filter.[12] However, their working relies on certain assumptions, such as the liner model assumption or the Gaussian error assumption.[13] Another powerful data assimilation technique is particle filters.[10,14] The particle filters approximate a probability density function by a set of particles and their associated importance weights, and therefore they put no assumption on the properties of the system model. As a result, they can effectively deal with nonlinear and/or non-Gaussian applications.[15–17]

As explained in Section 1.1, discrete event simulations are highly nonlinear, non-Gaussian systems, and therefore 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. 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 components has a state update. Consequently, the time between two consecutive state updates is usually not fixed, that is, the discrete event state process is asynchronous with the measurement process, which usually feeds data at fixed times. The mismatch

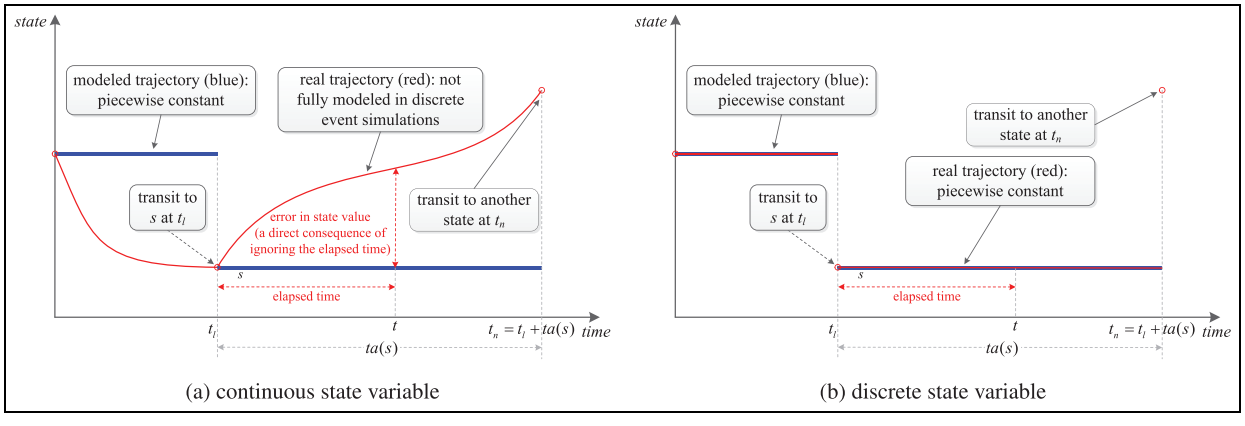


Figure 1. Discrete event simulation of continuous and discrete state variables. (Color online only.)

between the two processes incurs two problems that hinder the application of particle filtering in discrete event simulations. 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 (without the elapsed time; see Figure 1) of atomic components that were updated at past time instants. The consequence of ignoring the elapsed time is that the particles will be evaluated inaccurately, since the measurements are wrongly related to the states that were updated at past time instants. This effect is evident for continuous states (see Figure 1(a)); for discrete states (see Figure 1(b)), in order to compute the weight of a particle, 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*. The ''dimension'' refers to the dimension of a discrete event state trajectory during a fixed time interval, which is defined as the number of state points contained in the discrete event state trajectory during that time interval. Since the duration between two consecutive state updates in a discrete event simulation is not fixed, the dimension of a discrete event state trajectory during a fixed time interval is a random variable. This will lead to inapplicability of the standard sequential importance sampling algorithm.[18,19] Other practical problems, which mainly relate to data issues, such as non-numerical data, for example, event sequences, also make particle filtering in discrete event simulations highly problematic.

The research closely related to the topic of this paper is the work on data assimilation in wildfire spread simulations.[15,16,20,21] However, the two problems explained above were not explicitly considered. In their work, the simulation model for wildfire spread is a cellular automaton-based discrete event simulation model called DEVS-FIRE[22,23]; 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 that 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 1.

1.3 Contribution and outline of this paper

In this paper, we propose a particle filter-based data assimilation framework for discrete event simulations, in which we assume that model components do not change over time (i.e., closed systems). The measurements fed at time step $k \in \{1, 2, \ldots\}$ are assumed to be distributed over the last measurement interval (i.e., data fed at time step 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 interval. To define the data assimilation framework formally, we adopt the Discrete Event System Specification (DEVS) formalism[8]; in this framework, we solve the state retrieval problem and the variable dimension problem explained in Section 1.2. To illustrate the working of the proposed data assimilation framework, we study a case in a gold mine system in which noisy data (partial event sequences, entity positions with Gaussian errors) is assimilated into the discrete event gold mine simulation model in order to estimate 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; it is also shown that the proposed framework is robust to errors both in the simulation model and in the data.

The rest of this paper is organized as follows. Section 2 presents the particle filter-based data assimilation framework, which includes the system model, the measurement model, and the particle filtering algorithm for discrete event simulations. The case in the gold mine system is studied in Section 3 (tailoring the generic data assimilation framework to the specific estimation problem), Section 4 (qualitative analysis), and Section 5 (quantitative analysis). Finally, the paper is concluded in Section 6.

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 2.1, we show how to achieve such formalization. In Section 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 2.3, in which the variable dimension problem is addressed. Finally some practical remarks that can help simplify the application of the data assimilation framework are given in Section 2.4.

2.1 System model

In order to describe the discrete event simulations formally, we need to adopt certain discrete event modeling and simulation formalism. Therefore, in Section 2.1.1, we briefly introduce the DEVS formalism,[8] which is adopted widely in the simulation community. Subsequently, in Section 2.1.2, we introduce how the state is evolved in a DEVS model. Finally, in Section 2.1.3, we show how to formalize the state transitions in a DEVS model as an integer indexed state process.

2.1.1 Discrete Event System Specification. DEVS[8] 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, $\delta_{int} : S \to S$ is the internal state transition function, $\delta_{ext} : Q \times X \to S$ is the external state transition function, where $Q = \{(s, e) | s \in S, 0 \leqslant e \leqslant ta(s)\}$ is the total state set, e is the time elapsed since the last transition, $\lambda : S \to Y$ is the output function, and $ta : S \to R^{+}_{0,\infty}$ is the time advance function, where $\mathbb{R}^{+}_{0,\infty}$ is the positive real with 0 and ∞.

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

$$N = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, Select > \quad (1)$$

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 follows:

$$M_i = < X_i, Y_i, S_i, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i >, \forall i \in D$$

- for each $i \in D \cup \{N\}$, I_i is the set of components that 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:

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

- *Select*: $2^D \to D$ is a tie-breaking function with $Select(E) \in E$ to arbitrate the occurrence of simultaneous events.

DEVS models are closed under coupling, that is, the coupling of DEVS models defines an equivalent atomic DEVS model.[24]

2.1.2 State evolution in a coupled DEVS model. Consider a coupled DEVS model N defined in Equation (1). The state evolution of its atomic component M_i is achieved by executing 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 atomic components.

Since DEVS models are closed under coupling,[24] 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[24]). The sequential state of M (equivalent to the coupled DEVS model N) can be represented as follows:

$$s = (\ldots, (s_i, e_i), \ldots) \in S = \times_{i \in D} Q_i \quad (2)$$

where $Q_i = \{(s_i, e_i) | s_i \in S_i, 0 \leqslant e_i \leqslant ta_i(s_i)\}$. The state evolution of the coupled DEVS model is triggered by either an internal state transition of the selected imminent component i^*,[24] which transforms the different parts of the total state as follows:

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

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

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

2.1.3 Formalize discrete event state evolution as an 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:

$$\begin{aligned} x_{\tilde{k}} &= (s_{\tilde{k}}, t_{\tilde{k}}) \\ &= ((\ldots, (s_{i,\tilde{k}_i}, e_{i,\tilde{k}_i}), \ldots), t_{\tilde{k}}), i \in D; \\ &\tilde{k} = 0, 1, 2, \ldots; \tilde{k}_i = 0, 1, 2, \ldots \end{aligned} \quad (3)$$

where $s_{\tilde{k}} \in S$ is a sequential state of a coupled DEVS model as defined in Equation (2), 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

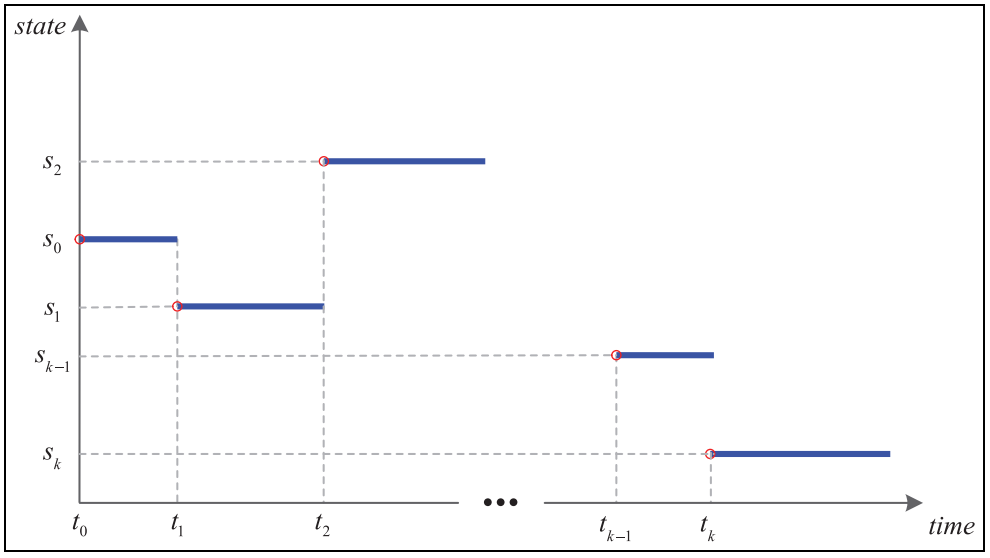


Figure 2. The integer indexed state process (each red circle represents a state point $x_{\tilde{k}} = (s_{\tilde{k}}, t_{\tilde{k}})$). (Color online only.)

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, that is, $\tilde{k}_i$. 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 2.

We denote the input event segment for the coupled DEVS model as $w : (t_{\tilde{k}}, t_{\tilde{k}} + ta(s_{\tilde{k}})] \to X^{\varnothing} = X \cup \{\varnothing\}$, where $ta(s_{\tilde{k}}) = ta_{i^*}(s_{i^*,\tilde{k}_{i^*}}) - e_{i^*,\tilde{k}_{i^*}} = min\{\sigma_{i,\tilde{k}_i} = ta_i(s_{i,\tilde{k}_i}) - e_{i,\tilde{k}_i} | i \in D\}$, that is, 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 $(t_{\tilde{k}}, t_{\tilde{k}} + ta(s_{\tilde{k}})]$, that is, $\nexists t \in (t_{\tilde{k}}, t_{\tilde{k}} + ta(s_{\tilde{k}})]$, $s.t.\ w(t) \neq \varnothing$, $x_{\tilde{k}+1} = (s_{\tilde{k}+1}, t_{\tilde{k}+1})$ is determined as follows:

$$\begin{aligned} s_{\tilde{k}+1} &= \delta_{int}(s_{\tilde{k}}) = (\ldots, (s_{i,\tilde{k}'_i}, e_{i,\tilde{k}'_i}), \ldots) \\ t_{\tilde{k}+1} &= t_{\tilde{k}} + ta(s_{\tilde{k}}) \end{aligned} \tag{4}$$

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

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

- if there exist external events during $(t_{\tilde{k}}, t_{\tilde{k}} + ta(s_{\tilde{k}})]$, that is, $\exists t \in (t_{\tilde{k}}, t_{\tilde{k}} + ta(s_{\tilde{k}})]$, $s.t.\ w(t) \neq \varnothing \cap \nexists t' \in (t_{\tilde{k}}, t)$, $s.t.\ w(t') \neq \varnothing$, $x_{\tilde{k}+1} = (s_{\tilde{k}+1}, t_{\tilde{k}+1})$ is determined as follows:

$$\begin{aligned} x_{\tilde{k}+1} &= \delta_{ext}(s_{\tilde{k}}, t - t_{\tilde{k}}, w(t)) = (\ldots, (s_{i,\tilde{k}'_i}, e_{i,\tilde{k}'_i}), \ldots) \\ t_{\tilde{k}+1} &= t \end{aligned} \tag{5}$$

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

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

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

$$x_{\tilde{k}+1} = SIM(x_{\tilde{k}}, w) + \nu_{\tilde{k}}, \tilde{k} = 0, 1, 2, \ldots \tag{6}$$

where w is the input event segment and SIM is a discrete event simulation model that transfers state $x_{\tilde{k}}$ to $x_{\tilde{k}+1}$ based on Equations (4) and (5); $\nu_{\tilde{k}}$ is the process noise. Notice that the time duration between two consecutive state points, that is, $t_{\tilde{k}+1} - t_{\tilde{k}}$, is not a constant, but a random variable. In this paper, we focus on closed systems; therefore, $w = \varnothing$.

2.2 Measurement model

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

$$m_k = g_k(s_k) + \varepsilon_k, k = 1, 2, \ldots \quad (7)$$

where ε_k is the measurement noise. Notice that the measurement process is assumed to feed data at fixed times, that is, every ΔT time units, and 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). The state points in the discrete event state process can also be indexed by an integer $\tilde{k}$ (see section 2.1.3), 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.

In a discrete event simulation, the state values are only updated when events happen. As shown in Figure 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. If these retrieved states (updated at past time instants) were used for estimation, inaccurate estimation results would be obtained. This will incur the state retrieval problem, as introduced in Section 1.2. To get an 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 the 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 $x_{0:\infty} = \{x_i, i = 0, 1, 2, \ldots\}$ is a sequence of state points defined in Equation (3); $\mathcal{N}_k(x_{0:\infty})$ 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, if we use 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:

$$m_k \sim p(m_k|\hat{s}_k) = p(m_k|x_{\mathcal{N}_k}) \quad (8)$$

which is just a reformulation of Equation (7).

In this research, we want to generalize the measurement model to include situations where measurements are dependent on the state trajectory, that is, 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 1 minute.[4] 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 the measurement model:

$$m_k \sim p(m_k|x_{\mathcal{N}^+_{k-1}+1:\mathcal{N}^+_k}) \quad (9)$$

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 that are indexed from $\mathcal{N}^+_{k-1} + 1$ to $\mathcal{N}^+_k$.

2.3 State estimation using particle filters

2.3.1 Principles of particle filters. A general discrete time state evolution can be expressed by the following:

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

where f_k is a possibly nonlinear function of the state s_{k-1} and ν_{k-1} is the process noise. The measurement at time k is given by the following:

$$m_k = g_k(s_k) + \varepsilon_k, k = 1, 2, \ldots$$

where g_k is possibly a nonlinear function that maps the state to the measurement and ε_k is the measurement noise.

The objective of the particle filter is to estimate the conditional distribution of all states up to time k given all available measurements up to k, that is, $p(s_{0:k}|m_{1:k})$, where

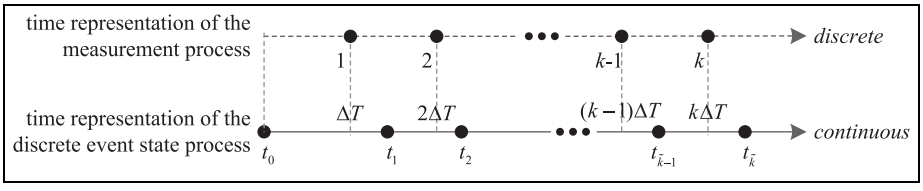


Figure 3. Time representation of the discrete event state process (each black dot indicates a state update) and the (discrete time) measurement process (each black dot represents the arrival of a measurement).

$s_{0:k} = \{s_i, i = 0, 1, 2, \ldots, k\}$, $m_{1:k} = \{m_j, j = 1, 2, \ldots, k\}$. Since an analytic solution of $p(s_{0:k}|m_{1:k})$ is usually intractable, we generate a set of Monte Carlo samples (particles) with their associated weights to approximate this posterior distribution. If the number of particles is sufficiently large, the posterior can be approximated to an arbitrary accuracy.[10,14] With this sample of particles all relevant statistical moments can be obtained using standard Monte Carlo integration techniques.

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

$$p(s_{0:k}|m_{1:k}) \approx \sum_{i=1}^{N_p} w^i_k \delta(s_{0:k} - s^i_{0:k}) \tag{10}$$

where $\delta(\cdot)$ is the Dirac delta function. A very important concept in particle filtering is the principle of importance sampling. If we can generate the particles $\{s^i_{0:k}\}^{N_p}_{i=1}$ from $p(s_{0:k}|m_{1:k})$, each of them will be assigned a weight equal to $1/N_p$. However, direct sampling from $p(s_{0:k}|m_{1:k})$ is usually intractable. An alternative (i.e., importance sampling) is to generate the particles from a distribution $q(s_{0:k}|m_{1:k})$, known as *importance density*,[10,14] and assign weights according to the following:

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

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 follows:

$$\begin{aligned} p(s_{0:k}|m_{1:k}) &= \frac{p(m_k|s_k)p(s_k|s_{k-1})p(s_{0:k-1}|m_{1:k-1})}{p(m_k|m_{1:k-1})} \\ &\propto p(m_k|s_k)p(s_k|s_{k-1})p(s_{0:k-1}|m_{1:k-1}). \end{aligned} \tag{11}$$

In the case that the importance density is chosen to factorize such that $q(s_{0:k}|m_{1:k}) = q(s_k|s_{0:k-1}, m_{1:k})\, q(s_{0:k-1}|m_{1:k-1})$, the random measure $\chi_{k-1} = \{s^i_{0:k-1}, w^i_{k-1}\}^{N_p}_{i=1}$ can be updated sequentially whenever new measurements m_k become available. The procedure then becomes the following:

- obtain samples $s^i_{0:k} \sim q(s_{0:k}|m_{1:k})$ 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 $s^i_k \sim q(s_k|s^i_{0:k-1}, m_{1:k})$;
- update weights by the following:

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

If we assume that $q(s_k|s_{0:k-1}, m_{1:k}) = q(s_k|s_{k-1}, m_k)$, that is, the importance density depends on s_{k-1} and m_k *only*, we have the following:

$$w^i_k \propto \frac{p(m_k|s^i_k)p(s^i_k|s^i_{k-1})}{q(s^i_k|s^i_{k-1}, m_k)} w^i_{k-1} \tag{12}$$

A pragmatic choice for the importance density is the system transition density, that is, $q(s_k|s_{k-1}, m_k) = p(s_k|s_{k-1})$. As a result, Equation (12) simplifies to the following:

$$w^i_k \propto p(m_k|s^i_k) w^i_{k-1} \tag{13}$$

A major problem of particle filters is that the discrete random measure degenerates quickly.[10,14] 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.[10,14,25] 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.[10,14]

2.3.2 Application in discrete event simulations. Consider a discrete event system with sensors deployed to monitor its operation. The measurement fed at time k, that is, 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, that is, $p(x_{0:\mathcal{N}^+_k}|m_{1:k})$. Based on Bayes' theorem, $p(x_{0:\mathcal{N}^+_k}|m_{1:k})$ can be expressed as $p(x_{0:\mathcal{N}^+_k}|m_{1:k}) = \frac{p(x_{0:\mathcal{N}^+_k})p(m_{1:k}|x_{0:\mathcal{N}^+_k})}{p(m_{1:k})}$. Similarly, we have $p(x_{0:\mathcal{N}^+_{k-1}}|m_{1:k-1}) = \frac{p(x_{0:\mathcal{N}^+_{k-1}})p(m_{1:k-1}|x_{0:\mathcal{N}^+_{k-1}})}{p(m_{1:k-1})}$. Therefore, we have the following:

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

Consequently, we can obtain a sequential update equation:

$$
\begin{aligned}
p(x_{0:\mathcal{N}_k^+}|m_{1:k}) &= \frac{p(m_k|x_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+})p(x_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+}|x_{\mathcal{N}_{k-1}^+})}{p(m_k|m_{1:k-1})} \\
&\quad \times p(x_{0:\mathcal{N}_{k-1}^+}|m_{1:k-1}) \\
&\propto p(m_k|x_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+})p(x_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+}|x_{\mathcal{N}_{k-1}^+}) \\
&\quad \times p(x_{0:\mathcal{N}_{k-1}^+}|m_{1:k-1})
\end{aligned} \tag{14}
$$

This sequential update equation is similar in form to that in Equation (11), but an important difference here is that $\mathcal{N}_k^+$ is a random variable, which means that the dimension of $x_{0:\mathcal{N}_k^+}$, that is, the number of state points in $x_{0:\mathcal{N}_k^+}$, is also random. The variable dimension problem will lead to inapplicability of the standard sequential importance sampling algorithm (see section 2.3.1).[18,19]

In Godsill et al.,[19] 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 sufficiently 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 the following:

$$
\begin{aligned}
w_k &= \frac{p(x_{0:K}|m_{1:k})}{q(x_{0:K}|m_{1:k})} \\
&\propto \frac{p(m_k|x_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+})p(x_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+}|x_{\mathcal{N}_{k-1}^+})}{q(x_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+}|x_{0:\mathcal{N}_{k-1}^+}, m_{1:k})} \times w_{k-1}
\end{aligned} \tag{15}
$$

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 et al.[19] and Godsill and Vermaak.[18]

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 the 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.

2.4 Practical remarks

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, for example, the equation-based model. However, in discrete event simulations that involve a large number of interacting components, this stopping condition is not easy to capture since the model is

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 $x^i_0 = (s^i_0, t^i_0)$, where $s^i_0 \sim p(s_0)$ ($p(s_0)$ is the probability distribution of the initial state), and $t^i_0 = 0$
4. set weight $w^i_0 = 1/N_p$
5. **end**
6. % *the sampling step for any time* $k \geqslant 1$
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 $x^i_j \sim q(x_j|x^i_{0:j-1}, m_{1:k})$

   set $\mathcal{N}_k^{+^i} = j$, and append the newly generated states to particle: $x^i_{0:\mathcal{N}_k^+} = (x^i_{0:\mathcal{N}_{k-1}^+}, x^i_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+})$, where $\mathcal{N}_0^+ \equiv 0$

   update weight:

$$
w^i_k \propto \frac{p(m_k|x^i_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+})p(x^i_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+}|x^i_{\mathcal{N}_{k-1}^+})}{q(x^i_{\mathcal{N}_{k-1}^+ + 1:\mathcal{N}_k^+}|x^i_{0:\mathcal{N}_{k-1}^+}, m_{1:k})} \times w^i_{k-1}
$$

9. **end**
10. normalize the weights, such that $\sum_{i=1}^{N_p} w^i_k = 1$
11. % *the resampling step*
12. resample particles $\{x^i_{0:\mathcal{N}_k^+}, w^i_k\}_{i=1}^{N_p}$ based on the chosen resampling method, which can be found in Douc et al.[25]

separated from its simulator. One possible solution is to put a little more effort into modeling by adding certain attributes that 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 that will happen in the future. For example, in the gold mine case that will be studied in subsequent sections, we have a speed attribute for moving entities; as a consequence, when we need to get an entity position at a time instant, we only need the last updated state (which contains speed and location) and the elapsed time to fulfill linear interpolation in order to get updated entity positions.

The two state generation processes are compared in Figure 4. The blue and red dots represent state points in a discrete event state process. Specifically, the blue dots represent state points generated in the $(k-1)$-th data assimilation iteration, while the red dots are generated in the k-th iteration. Suppose now we need to obtain the state value at time instant $k\Delta T$ using linear interpolation. In the state generation process in Algorithm 1 (Figure 4(a)), the discrete event simulation needs to generate one more state point beyond time instant $k\Delta T$ to apply linear interpolation. In contrast, if the interpolation operation at a time instant is independent of state points beyond that time (Figure 4(b)), we can simply stop the simulation at time instant $k\Delta T$, since we only need one state point that lies on the left-hand side of $k\Delta T$ and the elapsed time to fulfill linear interpolation. The benefit of the state generation process in Figure 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 instant $k\Delta T$ 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 (14) will be simplified to the following:

$$\begin{aligned} p(x_{0:\mathcal{N}_k^+}|m_{1:k}) &= p(s_{0:k}|m_{1:k}) \\ &\propto p(m_k|s_{k-1:k})p(s_{k-1:k}|\hat{s}_{k-1})p(s_{0:k-1}|m_{1:k-1}) \end{aligned} \tag{16}$$

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

$$\begin{aligned} s_{k-1:k} &= \{s_{\tilde{k}}|x_{\tilde{k}} = (s_{\tilde{k}}, t_{\tilde{k}}) \cap (k-1)\Delta T \leqslant t_{\tilde{k}} \leqslant k\Delta T\} \cup \{\hat{s}_k\} \\ s_{0:k} &= s_{0:k-1} \cup s_{k-1:k} \end{aligned} \tag{17}$$

The weight update in Equation (15) will thus be modified to the following:

$$w_k = \frac{p(x_{0:K}|m_{1:k})}{q(x_{0:K}|m_{1:k})} \propto \frac{p(m_k|s_{k-1:k})p(s_{k-1:k}|\hat{s}_{k-1})}{q(s_{k-1:k}|s_{0:k-1}, m_{1:k})} \times w_{k-1} \tag{18}$$

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., we 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 that 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. Suppose the state of an atomic model can be represented as $s = \{p, \theta\}$, where p denotes the phase and θ indicates the corresponding parameters (state variables). Note that θ can be a combination of discrete and continuous variables. Let $FS \subseteq \times_{i \in D} P_i$ denote a set of feasible combinations of phases of atomic components, where D is the set of names of components of the discrete event model (i.e., a coupled DEVS model) and P_i is the set of possible phases of component i. We denote the combination of initial phases of all atomic components as a random variable P_0, and it should take value from FS. Since P_0 is a discrete random variable, we formalize its probability distribution as follows:

$$P(P_0 = p_0^j) = p_j, p_0^j \in FS, p_j \in (0, 1) \tag{19}$$

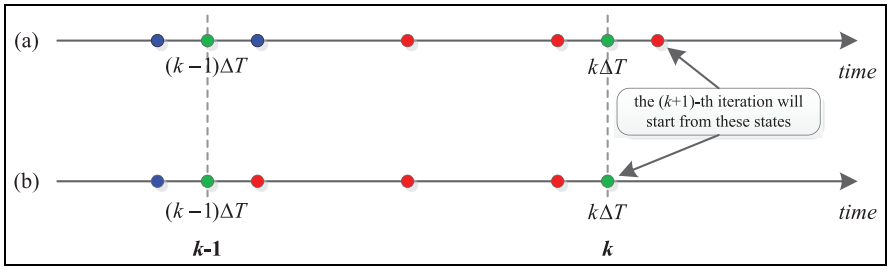


Figure 4. The state points generation process (the blue (generated in the $(k-1)$-th iteration) and red (generated in the k-th iteration) dots represent state points in a discrete event state process, while the green dots represent interpolated state points; color online only).

Table 1. State variables of key components in the Discrete Event System Specification 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

and $\sum_{j=1}^{|FS|} p_j = 1$. Notice that $p_0^j = (\ldots, p_{0,i}^j, \ldots)$, $i \in D, p_{0,i}^j \in P_i$. Based on this discrete probability distribution, generating an initial model state is done as follows.

- Generate a feasible combination of initial phases of all atomic components, $p_0^j = (\ldots, p_{0,i}^j, \ldots) \in FS, i \in D, p_{0,i}^j \in P_i$, by sampling the discrete probability distribution $P(P_0)$.
- For each atomic component $i \in D$, its initial phase is $p_{0,i}^j$, and we now need to generate values for its corresponding parameters $\theta_{0,i}^j$:
  - For a discrete variable in $\theta_{0,i}^j$, its value can be generated by sampling certain discrete probability distribution.
  - For a continuous variable in $\theta_{0,i}^j$, its value can be generated by sampling certain continuous probability distribution. For example, in the gold mine case that will be studied in Section 3, suppose the initial phase of the elevator is GO_DOWN_EMPTY, one of its continuous parameters *pos* (the position of the elevator; see the more detailed definition in Table 1) can be generated by sampling a Uniform distribution $U(bottom, top)$, where *bottom* and *top* represent the bottom position and top position of the vertical elevator shaft, respectively.

  Then the initial state of atomic component i can be represented as $s_{0,i}^j = \{p_{0,i}^j, \theta_{0,i}^j\}$ and the initial state of the coupled model can be represented as $s_0^j = (\ldots, s_{0,i}^j, \ldots), i \in D$.

- Once we have generated the initial state $s_{0,i}^j$ for atomic component $i \in D$, we can compute its time advance, which is denoted as $ta(s_{0,i}^j)$. For each atomic component i, we can simply set its elapsed time $e_i = 0$.
- Finally, combining the generated 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 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 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.

3.1 Scenario description

A gold mine system is shown in Figure 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 250/3 m/min when full through the mine shaft, and 500/3 m/min when empty. The current mine shaft is 400 m long.
- An elevator can take a batch of gold ore up. The depth of the elevator shaft is 100 m; it takes the elevator 8 min to go up with ore and 3 min 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

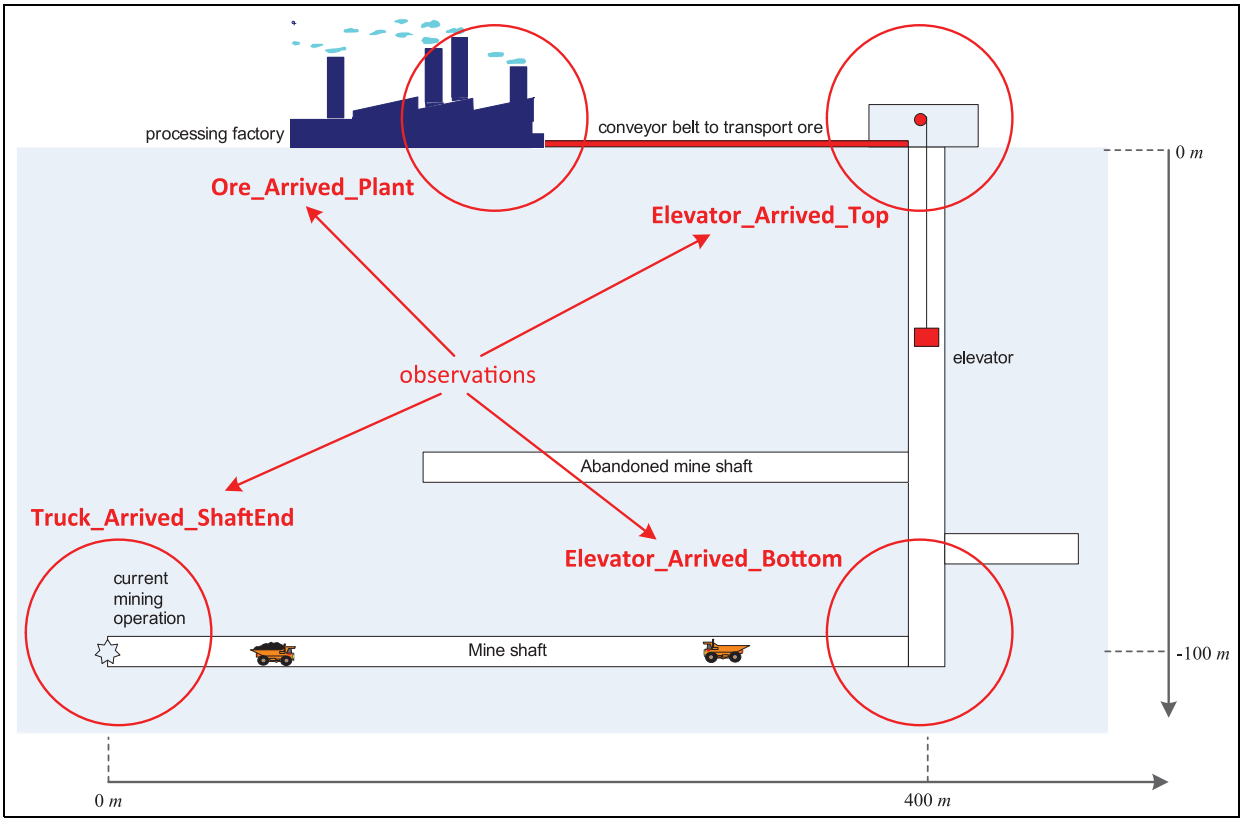


Figure 5. The gold mine system. (Color online only.)

the truck before the elevator can go up. Unloading takes between 5 and 10 min. 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 min before the load can be put on a 100-m long conveyor belt that transports the gold ore to a processing plant. The conveyor belt has a speed of 10 m/min.

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 3.4). 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.

3.2 Modeling the gold mine system in the DEVS formalism

The scenario described in Section 3.1 is a typical discrete event system, and therefore we model it using the DEVS formalism,[8] as shown in Figure 6. Notice that the gold mine simulation model has no external inputs. We model each component into different phases,[26] 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 1, while other components (such as Queue, Conveyor, Observer) are quite simple, and therefore we do not describe them in detail due to space limitations.

As shown in Table 1, each component has a transient phase, that is, 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 say that Miner is idle and can drill and load other trucks if there are 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 (although 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

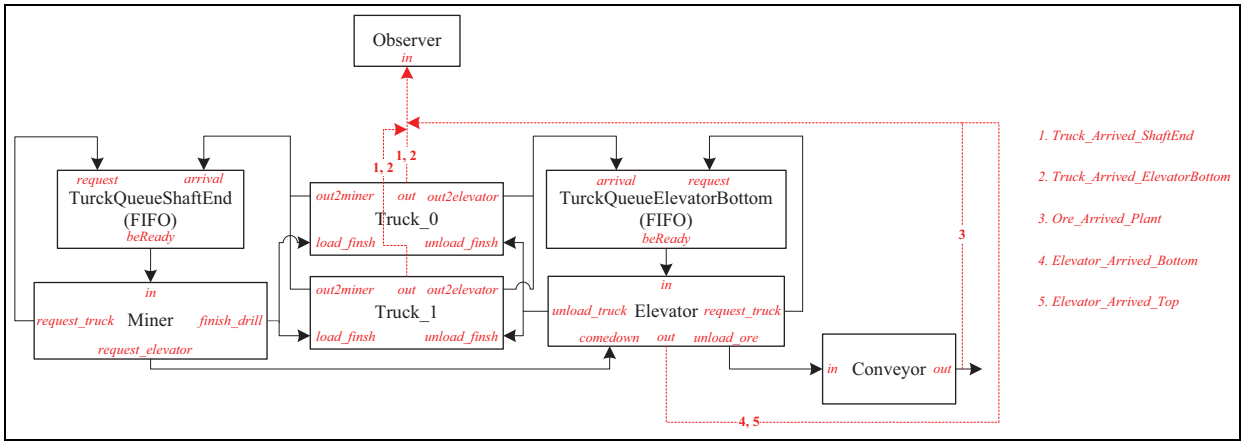


Figure 6. The Discrete Event System Specification model of the gold mine system. FIFO: first-in, first-out. (Color online only.)

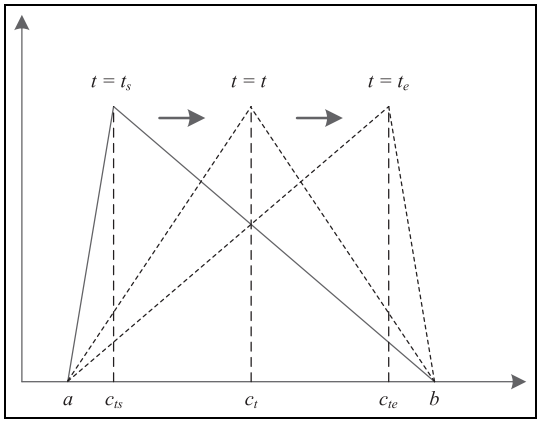


Figure 7. Triangular distribution with varying modes.

Triangular distribution with varying modes (shown in Figure 7). The purpose of varying modes is to simulate miners' tiredness, which means that miners can become tired, that is, 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 minutes.

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 follows:

$$s = (\ldots, (s_i, e_i), \ldots) \in S = \times_{i \in D} Q_i \tag{20}$$

where $Q_i = \{(s_i, e_i) | s_i \in S_i, 0 \leqslant e_i \leqslant ta_i(s_i)\}$. Based on the derivation shown in Section 2.1.3, 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):

$$\begin{aligned} x_{\tilde{k}} &= ((\ldots, (s_{i,\tilde{k}_i}, e_{i,\tilde{k}_i}), \ldots), t_{\tilde{k}}), i \in D \\ x_{\tilde{k}+1} &= GoldMineSim(x_{\tilde{k}}) + \nu_{\tilde{k}}, \tilde{k} = 0, 1, 2, \ldots \end{aligned} \tag{21}$$

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

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 1.

3.3.1 Continuous state. Continuous states can be interpolated. We take the elevator as an example, whose (sequential) state is represented as $s = (phase, pos, v)$ (the component index is omitted here), where $phase$ is the phase name and pos and v are its position and velocity, respectively. Although 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 3.2, 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:

$$\hat{s}(t) = interpolate(s(t_l), e) \quad \text{where} \begin{cases} phase_t & = phase_l \\ pos_t & = pos_l + v_l \times e = pos_l + v_l \times (t - t_l) \\ v_t & = v_l \end{cases} \tag{22}$$

which is independent of the states beyond time t.

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 that 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 the following:

$$\hat{s}(t) = interpolate(s(t_l), e) = (s(t_l), e) \tag{23}$$

where the elapsed time $e = t - t_l$. We still denote $(s(t_l), e)$ as $\hat{s}(t)$, that is, $(s(t_l), e)$ is equivalent to those continuous states that can be interpolated (e.g., Equation (22)). 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 Interpolated state. Suppose that the (sequential) state of the coupled model at time instant t_l is $s(t_l) = (\ldots, (s_i, e_i), \ldots), i \in D$, 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 follows:

$$\hat{s}(t) = interpolate(s(t_l), e) = (\ldots, (s_i', e_i'), \ldots), \text{where}$$
$$(s_i', e_i') = \begin{cases} (interpolate(s_i, e_i), 0) \\ \quad \text{if } s_i \text{ can be interpolated (see Equation(22))} \\ (s_i, e_i + t - t_l) \\ \quad \text{if } s_i \text{ can not be interpolated (see Equation(23))} \end{cases} \tag{24}$$

Notice that the time advance of state $interpolate(s_i, e_i)$ will be $ta(s_i) - e_i$. In Section 2.2, when computing $\hat{s}_k, k = 1, 2, \ldots$, we essentially compute $\hat{s}_k = \hat{s}(k\Delta T)$ based on Equation (24).

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 8, the positions of the elevator in the discrete event simulation are 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. Note that the piecewise constant segments between the elevator top and the elevator bottom in Figure 8 are the result of the elevator processing external events, for example, miners ask the elevator to come down.

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. In this specific case, the resulted state trajectory by interpolation is equivalent to that if we simulate the continuous state variable (the position of the elevator) using Generalized Discrete Event Specification (GDEVS)[27] with the degree of the polynomial equal to 1. 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 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 5.

3.4 Available data and measurement model

The simulated data is generated by running the gold mine simulation (Section 3.2) for 480 min. 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 min) 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 the following types of events (as shown in Figure 5): trucks arriving at the shaft end (*Truck_Arrived_ShaftEnd*); the elevator arriving at

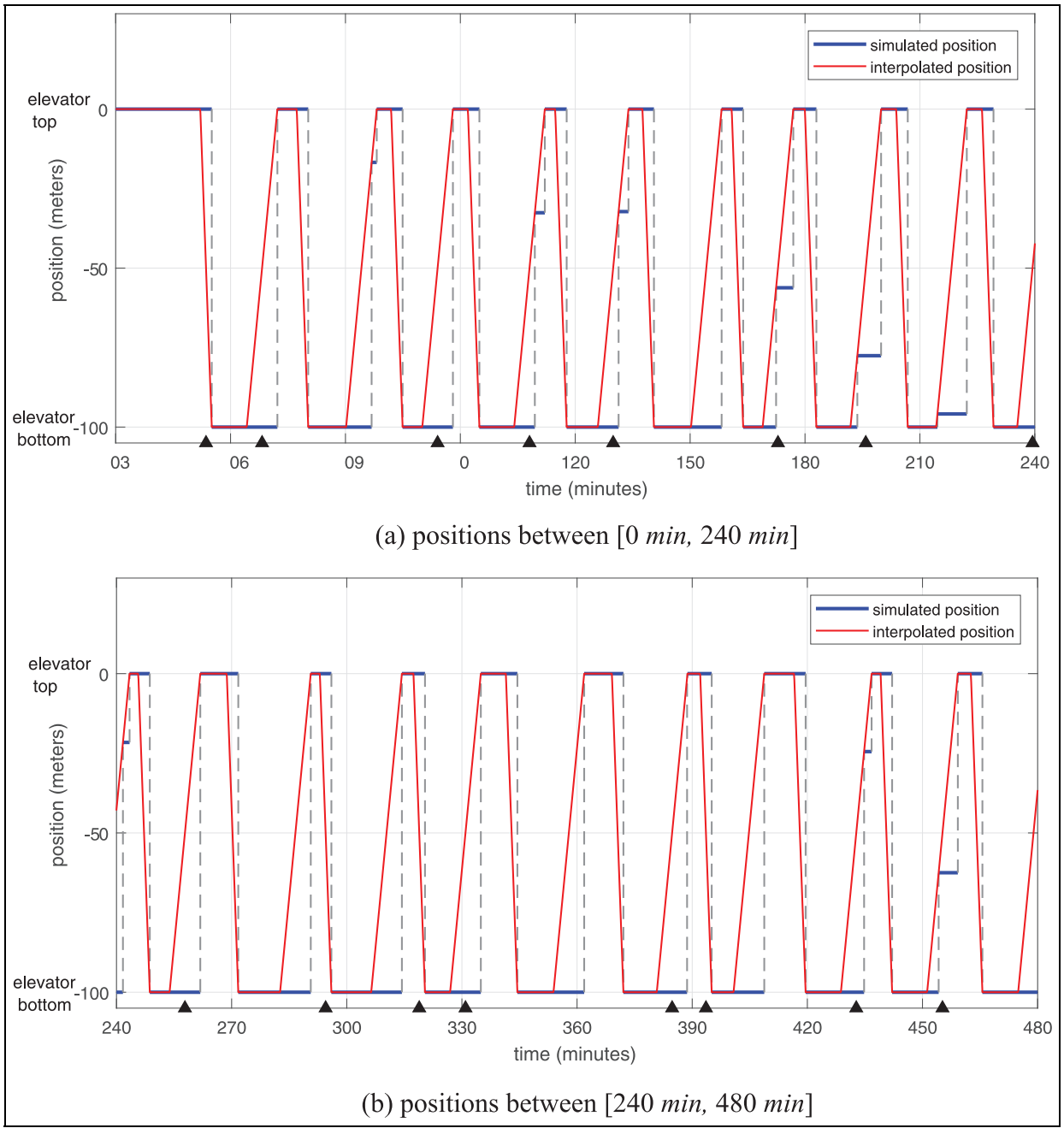


Figure 8. The state trajectory of the elevator in terms of position (the piecewise constant segments between the elevator top and the elevator bottom are the result of the elevator processing external events; each black triangle represents a time instant when a noisy observation of the elevator position is available; color online only).

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).

- 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 $\Delta T = 30$ min. The measurement at time k is denoted as m_k^o, which contains the following noisy data collected during $[(k-1)\Delta T, k\Delta T]$:

- Event sequence $E_k = \{(t_1, e_1), (t_2, e_2), \ldots, (t_n, e_n)\}, (k-1)\Delta T \leqslant t_1 \leqslant t_2 \leqslant \ldots \leqslant t_n \leqslant k\Delta T$; $e_i \in \{Truck\_Arrived\_ShaftEnd, Elevator\_Arrived\_Top, Elevator\_Arrived\_Bottom, Ore\_Arrived\_Plant\}$.
- $PX_k = \{(phase^j(t_j), pos^j(t_j)) | j \in \{Elevator, Truck\_0, Truck\_1\}, t_j \in [(k-1)\Delta T, k\Delta T]\}$, 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 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 follows:

$$m_k^o = \{E_k, PX_k\} \tag{25}$$

and the measurement model can be formalized as follows:

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

where $x_{\tilde{k}} = (s_{\tilde{k}}, t_{\tilde{k}}), \tilde{k} = 0, 1, 2, \ldots$ is defined in Equation (21). As introduced in Section 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 the following:

$$m_k^o \sim p(m_k^o | s_{k-1:k}) \tag{26}$$

where $s_{k-1:k} = \{s_{\tilde{k}} | x_{\tilde{k}} = (s_{\tilde{k}}, t_{\tilde{k}}) \cap (k-1)\Delta T \leqslant t_{\tilde{k}} \leqslant k\Delta T\} \cup \{\hat{s}_k\}$, and $\hat{s}_k$ is computed based on Equation (24) ($\hat{s}_k = \hat{s}(k\Delta T)$).

3.5 Estimating truck arrivals using particle filters

Having formalized the system model (Section 3.2) and the measurement model (Section 3.4), in this section, we implement (on the algorithmic level) the particle filtering framework (Section 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.

3.5.1 Particle filtering for truck arrivals estimation. Algorithm 2 describes in detail how the generic particle filter shown in Algorithm 1 is 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

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* $k \geqslant 1$
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 (24) ($t = (k-1)\Delta T$); the newly generated partial state trajectory is $s_{k-1:k}^i = \{s_k^i | x_k^i = (s_k^i, t_k^i), (k-1)\Delta T \leq t_k^i \leq k\Delta T\} \cup \{\hat{s}_k^i\}$; 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**

Table 2. Initial states of the gold mine simulation model.

Elements in *FS*	Component name	Phase	Parameters	Value	Time advance (min)
p_0^1	Miner	TRANSIENT_PHASE	serving_truck	NULL	0
	Truck_0	TRANSIENT_PHASE	*pos*	0 m	0
			v	0 m/min	
	Truck_1	TRANSIENT_PHASE	*pos*	0 m	0
			v	0 m/min	
	Elevator	IDLE_AT_TOP	*pos*	0 m	$+\infty$
			v	0 m/min	
			serving_truck	NULL	
			hasUnprocessedRequest	NULL	
p_0^2	Miner	TRANSIENT_PHASE	serving_truck	NULL	0
	Truck_0	TRANSIENT_PHASE	*pos*	0 m	0
			v	0 m/min	
	Truck_1	TRANSIENT_PHASE	*pos*	0 m	0
			v	0 m/min	
	Elevator	GO_DOWN_EMPTY	*pos*	$pos \sim U(-100,0)$ m	$(100+pos)/v$
			v	$v \sim U(0,200/3)$ m/min	
			serving_truck	NULL	
			hasUnprocessedRequest	NULL	

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 2–5 in Algorithm 1), 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 3.5.2.
- **Sampling**. In this case, we adopt the system transition density (a reformulation of $GoldMineSim(\cdot)$ in Equation (21)) 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), 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 3.5.3.
- **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 5.

3.5.2 Generating initial particles. In this case study, initial particles are generated based on the procedure introduced in Section 2.4.2. For illustration purpose, we only enumerate two feasible combinations of phases, which are listed in Table 2, although there are many more feasible choices. We assume $P(p_0^1) = P(p_0^2) = 0.5$. Note that we assume the maximum speed of the elevator is 200/3 m/min; we generate values of *pos* (the position) and *v* (the speed) for the elevator by sampling Uniform distributions, and the time advance can thus be computed as $(100 + pos)/v\, min$ (see the last row in Table 2). For other atomic components in the gold mine model, that is, *TruckQueueShaftEnd, TruckQueueElevatorBottom, Conveyor, Observer*, we initialize them as passive (i.e., time advance is $+\infty$).

3.5.3 Weight computation. In this section, we detail how the weight is computed, that is, we 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 $[(k-1)\Delta T, k\Delta T]$ and $PX_k = \{(phase^j(t_j), pos^j(t_j)) | j \in \{Elevator, Truck\_0, Truck\_1\}, t_j \in [(k-1)\Delta T, k\Delta T]\}$ 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 $p(m_k^o|s_{k-1:k}^i) = p(E_k|s_{k-1:k}^i)p(PX_k|s_{k-1:k}^i)$.

3.5.3.1 Event sequences. Given state points $s_{k-1:k}^i$, it is very easy to retrieve an event sequence that only contains the four types of events shown in Figure 5 (i.e., types of observed events). We denote such event sequences

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:

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

We adopt the *edit distance*[28] to define the ''distance'' between two event sequences. The edit distance is defined as ''the amount of work that has to be done to convert one sequence to another,'' and the amount of work is quantified by a set of transformation operations and their associated costs (more details can be found in Mannila and Ronkainen[28]). 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 follows:

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

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, that is:

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

where O_j is an arbitrary operation sequence that transforms S to T.

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

$$p(E_k|s_{k-1:k}^i) = p(E_k|E_k^i) = e^{-\frac{d(E_k, E_k^i)}{d_m}} \quad (27)$$

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

3.5.3.2 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. 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.

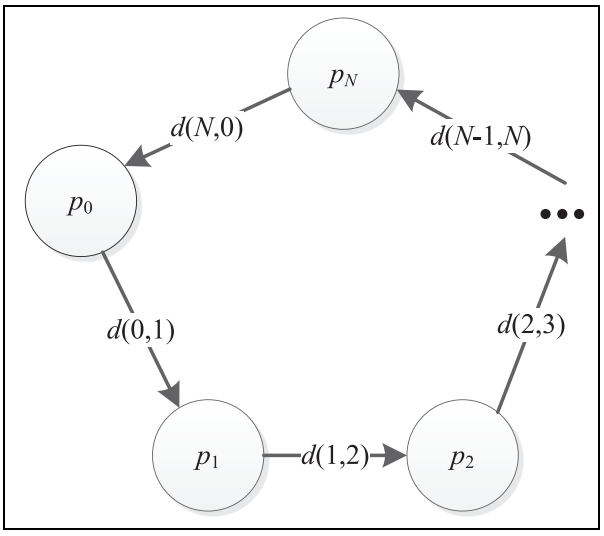


Figure 9. The phase transition graph.

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 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, and their distance is defined as follows:

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

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 $d(i, i+1) = 1$.

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

$$\begin{aligned} p(PX_k|s_{k-1:k}^i) &= p(PX_k|PX_k^i) \\ &= \prod_{j \in D_c} p(\{phase^j, pos^j\}|\{phase^{i,j}, pos^{i,j}\}) \end{aligned} \quad (28)$$

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

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

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

4 Case study of 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 10(a). 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 $t = 150$ min 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 $\Delta T = 30$ min with 2000 particles to estimate truck arrival times. The estimation results are depicted in Figure 10(b). 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 min). 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 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 11. Since the minimal drilling time is 15 min, there are at most two arrivals during one time step of duration $\Delta T = 30$ min. Notice that the estimation results actually give a distribution of truck arrival times. In order to know how accurate the estimation results are and also to explore the influences of factors, such as data errors, model errors, and the number of particles employed, in Section 5.2 we define a set of performance indicators and conduct quantitative analysis accordingly.

5 Case study of 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 10(b). In this section, we show how these raw data are processed in order to conduct a more informative analysis; based on the processed data, a set of performance indicators is proposed to quantify how accurate the estimation results are; finally, the results computed based on these performance indicators are presented and analyzed.

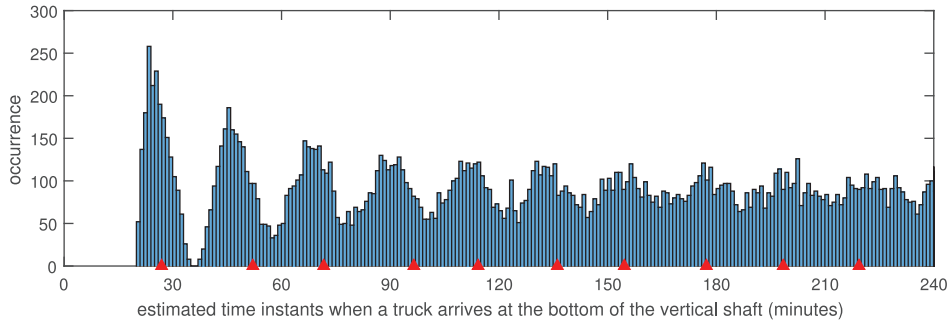
5.1 Data processing for estimating truck arrival times

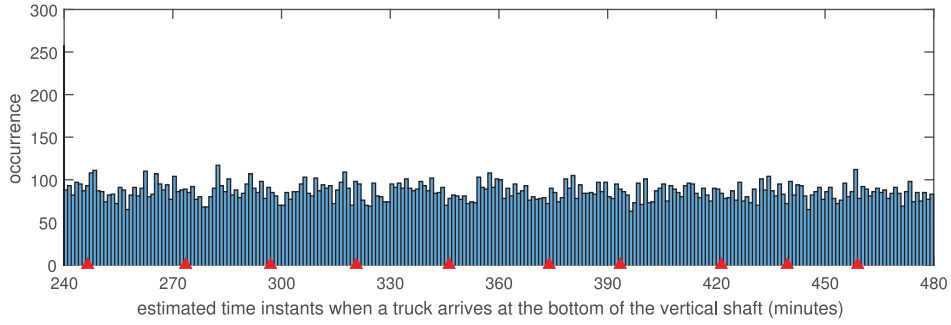
As shown in Figure 11(b), the estimated truck arrival times obviously belong to two groups, each of which approximates the distribution of a truck arrival. Therefore, we cluster the estimated arrival times into groups (for example, using the k-means clustering algorithm[29]), and each group estimates 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 12, we show the obtained kernel distribution fitted to the data belonging to the cluster on the right-hand side in Figure 11(b).

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 follows:

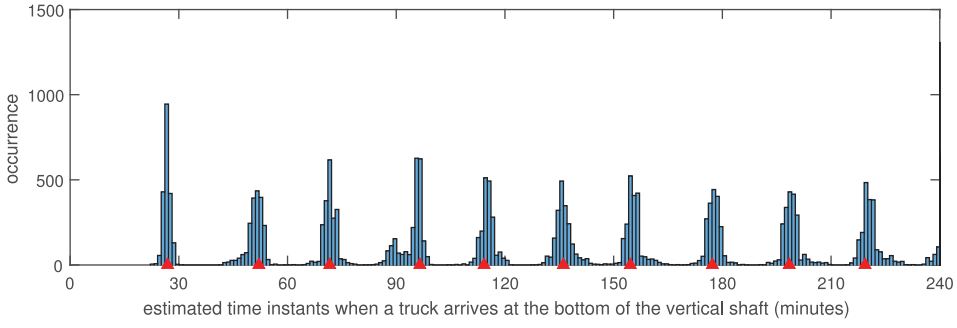
$$Prob(\text{arriving during } [t-\varepsilon, t+\varepsilon]) = F_c(t+\varepsilon) - F_c(t-\varepsilon)$$

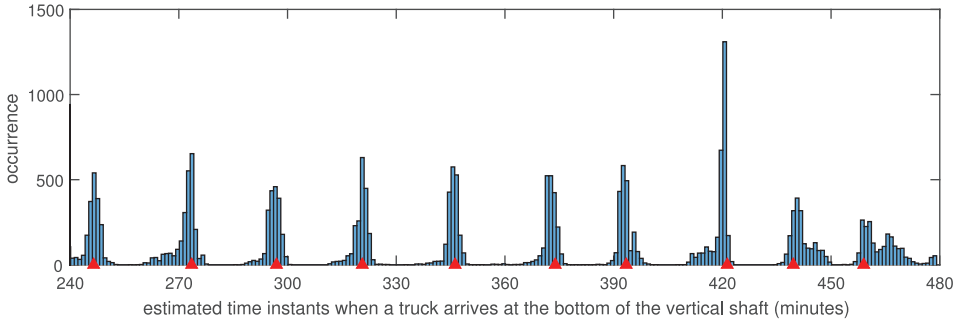
and for convenience, we denote this probability as $P_c(t, \varepsilon)$. $P_c(t, \varepsilon)$ thus represents the probability of a truck arriving at the bottom of the vertical shaft during $[t-\varepsilon, t+\varepsilon]$; the subscript c indicates that the probability is computed from the probability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





(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 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; color online only).

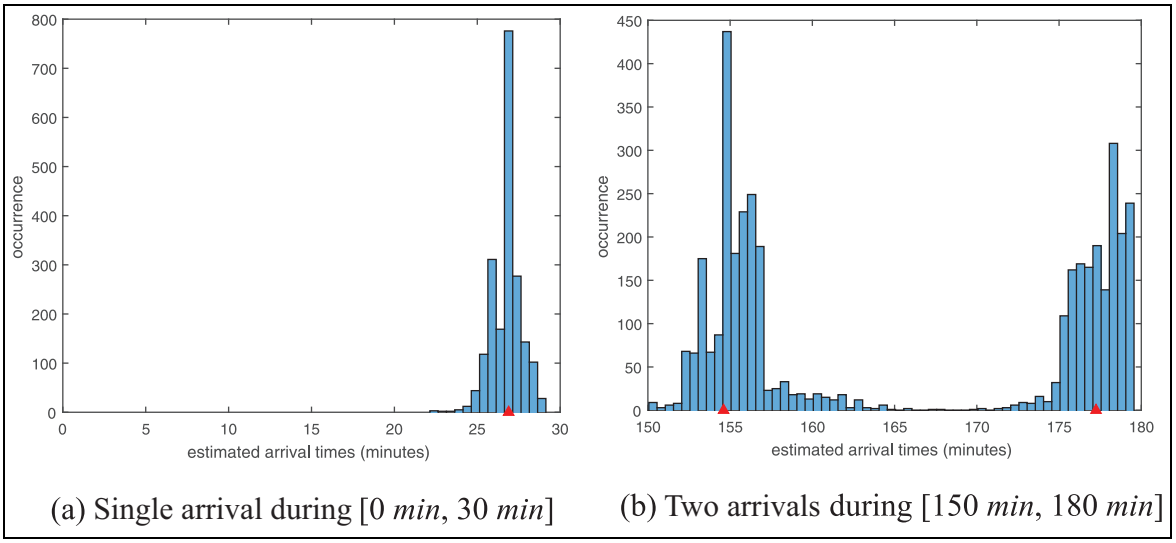


Figure 11. Histogram of estimated truck arrival times at the bottom of the vertical shaft during one step $[(k-1)\Delta T, k\Delta T]$, where $\Delta T = 30$ min (each red triangle represents a truck arrival in ground truth; color online only).

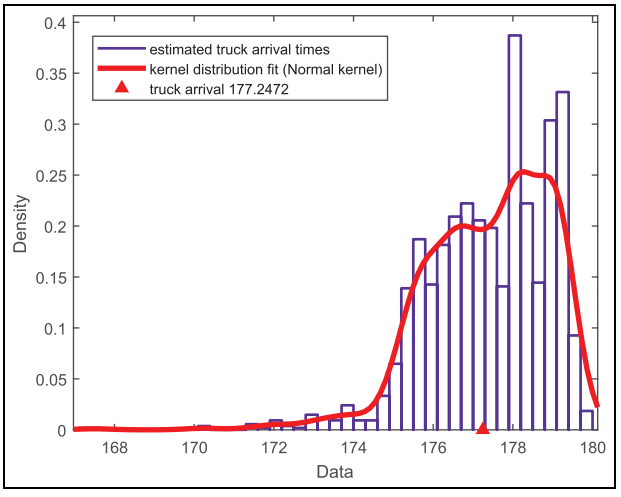


Figure 12. Fitting a kernel probability distribution using the Normal kernel to truck arrival times in one cluster (this group of data belongs to the cluster on the right-hand side in Figure 11(b); the red triangle represents a truck arrival in ground truth; color online only).

5.2 Evaluation criteria

Assume that the ground-truth value of truck arrivals is $A = \{t_1, t_2, \ldots, t_n\}$. After data processing, we obtain m clusters $\{C_c | C_c = \{t_1^c, t_2^c, \ldots, t_{n_c}^c\}\}_{c=1}^{m}$; from each cluster, we have a fitted probability density function. The format of the ground-truth data and the estimated data can thus be shown in Figure 13. The performance indicators are defined as follows.

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

$$\frac{P_{c_i}(t_i, \varepsilon)}{max\{P_{c_i}(t, \varepsilon)\}} > \delta \tag{29}$$

we consider 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, that is, if the probability $P_{c_i}(t_i, \varepsilon)$ is larger than a 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 the success rate as follows:

$$SR = \frac{n_m}{n} \times 100\% \tag{30}$$

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 as *wasted*. Obviously, the lower the number of wasted clusters, the better performance of the estimation. Therefore, we define the waste rate as follows:

$$WR = \frac{|m - n_m|}{m} \times 100\% \tag{31}$$

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

Suppose that $t_i \in A$ is estimated by the cluster C_{c_i}; as shown in Figure 13, we certainly want t_i to be as close as

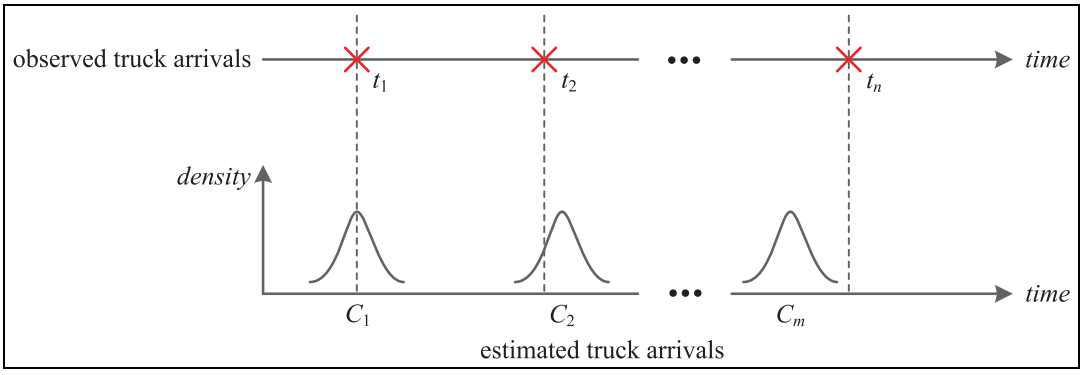


Figure 13. Format of the ground-truth data and estimated data. (Color online only.)

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:

$$\bar{d} = \frac{1}{n_m}\sum_{j=i_1}^{i_{nm}} |t_j - t^*_{c_j}| \tag{32}$$

  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 $P_{c_j}(t^*_{c_j}, \varepsilon)$:

$$\begin{aligned}\bar{P} &= 100\% \times \frac{1}{n_m}\sum_{j=i_1}^{i_{nm}} \frac{P_{c_j}(t_j,\varepsilon)}{max\{P_{c_j}(t,\varepsilon)\}} \\ &= 100\% \times \frac{1}{n_m}\sum_{j=i_1}^{i_{nm}} \frac{P_{c_j}(t_j,\varepsilon)}{P_{c_j}(t^*_{c_j},\varepsilon)}\end{aligned} \tag{33}$$

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 as 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 was introduced in Section 3.3, to obtain the updated state value.

5.3.1 The estimated truck arrival times. The raw estimation results shown in Figure 10(b) are clustered using the *k*-means clustering algorithm,[29] and the results are shown in Table 3. The *k*-means clustering algorithm outputs 20 clusters, that is, $\{C_c\}_{c=1}^{20}$, 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; 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, that is, $A = \{t_1, t_2, \ldots, t_{20}\}$. The probability $P_c(t_i, \varepsilon), c = 1, 2, \ldots, 20; i = 1, 2, \ldots, 20$ is computed and presented from the fourth column to the 23rd column. The results show that all arrivals lie in a certain cluster, that is, $\forall t_i \in A, \exists C_{c_i} \in \{C_c\}_{c=1}^{20}$, *s.t.* $t_i \in [min\{C_{c_i}\}, max\{C_{c_i}\}]$.

We compute the match criterion (see Equation (29)) for each truck arrival in A, and the results are depicted in Figure 14. 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 the following:

- 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 $\bar{d} = \frac{1}{n_m}\sum_{j=i_1}^{i_{nm}} |t_j - t^*_{c_j}| = 0.53$ min;
- average percentage $\bar{P} = 100\% \times \frac{1}{n_m}\sum_{j=i_1}^{i_{nm}} \frac{P_{c_j}(t_j,\varepsilon)}{P_{c_j}(t^*_{c_j},\varepsilon)} = 92.66\%$.

In the current operation of the gold mine system, the elevator only comes down when it receives a request from the miner, and therefore it will always arrive at the bottom of the vertical shaft at least 1.8 min (the difference between the time of the truck traveling full of ore and the time of the elevator going down empty) later than the trucks do. In other words, the truck will always wait at least 1.8 min until it can be served. However, using data assimilation, we can estimate 95% of all truck arrivals with an average error of 0.53 min (which is much smaller than 1.8 min). 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. The data assimilation results ($\sigma_e = 3.0, \sigma_t = 3.0; N_p = 2000; \varepsilon = 0.05$ min).

Data processing results			Truck arrivals ground truth																			
Cluster C_c	t_c^*	$P_c(t_c^*, \varepsilon)$	t_1 26.8918	t_2 52.0784	t_3 71.6300	t_4 96.5060	t_5 114.2494	t_6 136.1225	t_7 154.6024	t_8 177.2472	t_9 198.4608	t_{10} 219.3735	t_{11} 246.4805	t_{12} 273.5331	t_{13} 296.9505	t_{14} 320.6399	t_{15} 346.2515	t_{16} 373.8774	t_{17} 393.4647	t_{18} 421.3536	t_{19} 439.5858	t_{20} 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

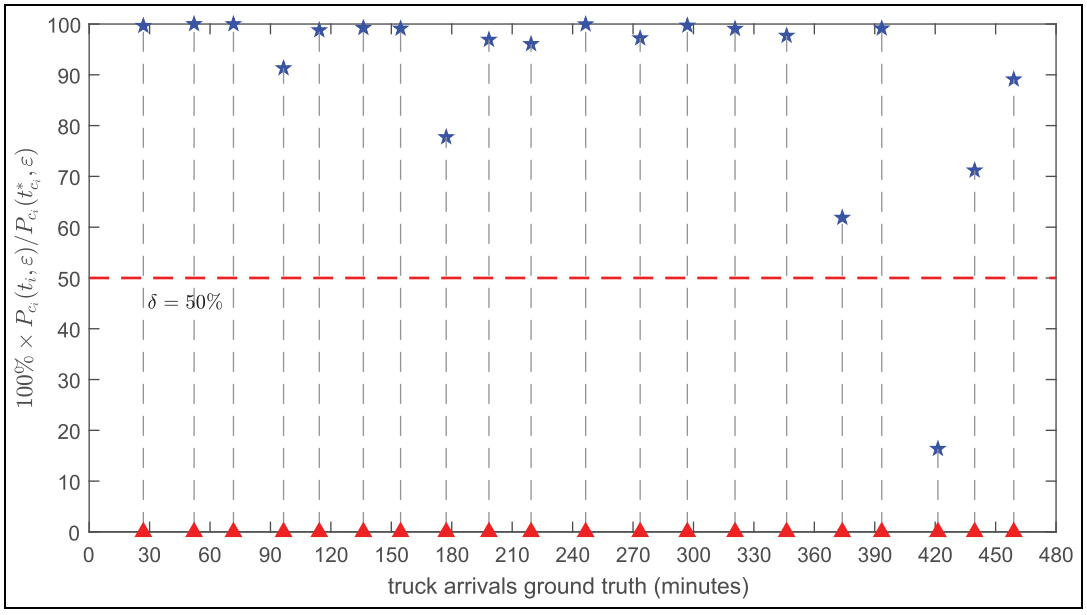


Figure 14. The match criterion $100\% \times P_{c_i}(t_i,\varepsilon)/P_{c_i}(t^*_{c_i},\varepsilon)$ (each red triangle represents a truck arrival in ground truth; color online only).

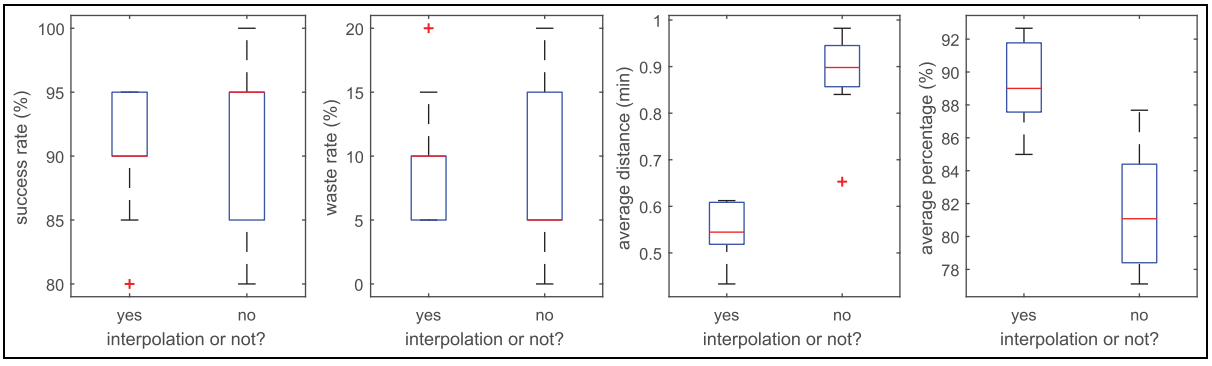


Figure 15. The influence of interpolation on the data assimilation results (noisy dataset ($\sigma_e = 3.0, \sigma_t = 3.0$); $N_p = 2000$; 10 independent runs; color online only).

5.3.2 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 four error measures (i.e., SR in Equation (30), WR in Equation (31), $\bar{d}$ in Equation (32), and $\bar{P}$ in Equation (33)) in Figure 15. The results show that although 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. Although 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, and 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.

5.4 Sensitivity analysis

In this section, we explore the influence of several key factors on the data assimilation results based on the simple gold mine case. These factors include data quality, modeling errors, and the number of particles used. For each set

Table 4. The influence of data quality, that is, 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	Success rate *SR*(%)	Waste rate *WR*(%)	Average distance $\bar{d}$(min)	Average percentage $\bar{P}$(%)
$\sigma_e = 3.0, \sigma_t = 3.0$	90.00 (90.00, 95.00)	10.00 (5.00, 10.00)	0.54 (0.52, 0.61)	89.00 (87.56, 91.77)
$\sigma_e = 5.0, \sigma_t = 5.0$	90.00 (90.00, 90.00)	10.00 (10.00, 10.00)	0.62 (0.59, 0.75)	88.53 (86.91, 90.48)
$\sigma_e = 10.0, \sigma_t = 10.0$	90.00 (90.00, 95.00)	10.00 (5.00, 10.00)	0.73 (0.70, 0.85)	86.40 (85.31, 88.60)
$\sigma_e = 15.0, \sigma_t = 15.0$	85.00 (85.00, 90.00)	15.00 (10.00, 15.00)	0.79 (0.77, 0.93)	86.50 (83.77, 87.98)
$\sigma_e = 20.0, \sigma_t = 20.0$	85.00 (85.00, 90.00)	15.00 (10.00, 15.00)	0.83 (0.76, 0.93)	85.82 (83.61, 87.04)

Table 5. 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 25th and 75th percentiles.

Model	Success rate *SR*(%)	Waste rate *WR*(%)	Average distance $\bar{d}$(min)	Average percentage $\bar{P}$(%)
Perfect model (drilling time: Triangular distribution with varying mode)	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)	90.00 (85.00, 90.00)	12.14 (10.00, 15.00)	0.63 (0.57, 0.69)	88.83 (86.65, 90.95)

of parameters, we run the experiment 10 times with different random seeds.

We should note that many factors can influence the quality of the data assimilation results. Besides the factors mentioned above, other factors may include the sensor deployment information, such as the number of sensors and data collection frequency. In this simple gold mine case, we do not consider these extra factors. For a more comprehensive analysis of the particle filter-based data assimilation methods, please refer to Gu and Hu.[30]

5.4.1 Effect of the data quality. In the gold mine case, only position data of entities (Elevator and Truck) are noisy, and the quality of the noisy position data is characterized by the standard deviation of the zero mean Gaussian noise, that is, 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 4. 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-m standard deviation on entity positions, the performance does not degenerate too much. Specifically, the performance indicators of estimating the truck arrivals are 85.00% (success rate), 15.00% (waste rate), 0.83 min (average distance), and 85.82% (average percentage).

5.4.2 Effect of the model errors. In the experiment in Section 5, the model we used to carry out data assimilation is the same as 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 the 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 modes (i.e., perfect model) to a standard Triangular distribution with lower bound 15 min, upper bound 30 min, and mode 20 min (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.

The results in Table 5 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

Table 6. The influence of the 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 25th and 75th percentiles.

N_p	Success rate *SR*(%)	Waste rate *WR*(%)	Average distance $\bar{d}$(min)	Average percentage $\bar{P}$(%)
100	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	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	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	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	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	90.00 (90.00, 95.00)	10.00 (5.00, 10.00)	0.54 (0.52, 0.61)	89.00 (87.56, 91.77)

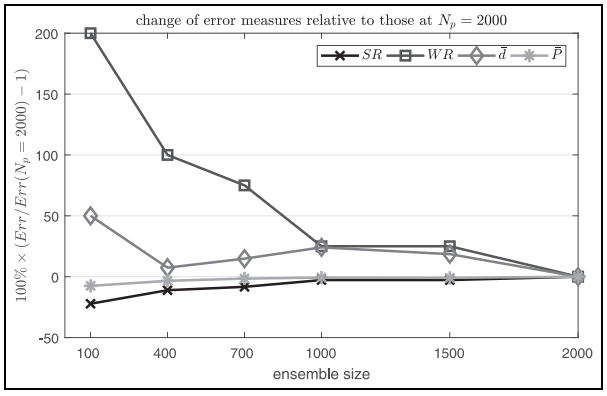


Figure 16. The influence of N_p on the data assimilation results (states are retrieved through interpolation; $\sigma_e = 3.0, \sigma_t = 3.0$); the performance indicators are relative to those at $N_p = 2000$.

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.

5.4.3 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 6. 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 this results in different event sequences and entity positions/phases, which will lead to a better coverage of the system state space.

Figure 16 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.

6 Conclusions

In this paper, we presented a particle filter-based data assimilation framework for discrete event simulations (of closed systems), in which we assume that the measurements fed at time step $k \in \{1, 2, \ldots\}$ are distributed over the last measurement interval $[(k-1)\Delta T, k\Delta T]$, implying that the measurements are dependent on the state transitions during that interval. The data assimilation framework was formally defined based on the DEVS formalism. In this framework, two key problems (i.e., the state retrieval problem and the variable dimension problem) that hinder the application of particle filtering in discrete event simulations were addressed. The state retrieval problem was solved by introducing an interpolation operation, 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 in order to obtain updated state values. The variable dimension problem was addressed based on the results of Godsill et al.,[19] which imply that in practice we can safely apply the standard sequential importance sampling algorithm to update the

posterior distribution $p(s_{0:k}|m_{1:k})$, where $s_{0:k}$ (see the definition in Equation (17)) has a variable dimension.

To illustrate the working of the proposed data assimilation framework, a case was studied in a gold mine system, in which noisy data (partial event sequences, entity positions with Gaussian errors) was assimilated into a discrete event gold mine simulation model to estimate truck arrival times at the bottom of the vertical shaft. The experiment results show that the proposed data assimilation framework is able to provide accurate estimation results in discrete event simulations. Assimilating (with interpolation) the noisy dataset with Gaussian error $\mathcal{N}(0, 3^2)$ added on entity positions, the performance indicators of estimating the truck arrivals are 95.00% (success rate), 5.00% (waste rate), 0.53 min (average distance), and 92.66% (average percentage) (see Section 5.3.1). In contrast, the simulation without data assimilation totally lost its prediction ability from $t = 150$ min onwards. The experiment results also prove that a proper interpolation operation can significantly improve the estimation results compared to those obtained without interpolation. With a linear interpolation to obtain entity positions in the gold mine case, all performance indicators improved in the statistic sense compared with those obtained without interpolation (see section 5.3.2), since the interpolation operation can capture the real-time state evolution, which is not described in the discrete event model but does happen in reality.

Sensitivity analysis reveals that the proposed data assimilation framework is robust to error assumptions. In the gold mine case, even with a 20-m standard deviation on entity positions, the performance does not degenerate too much. Specifically, the performance indicators of estimating the truck arrivals are 85.00% (success rate), 15.00% (waste rate), 0.83 min (average distance), and 85.82% (average percentage) (see Section 5.4.1). Similarly, the framework is robust to model errors (i.e., differences between the model generating the ground-truth data and the model used in the case study), although we cannot claim to have tested this exhaustively. The result shows that using a model with errors does not significantly affect the estimation results (see Section 5.4.2). 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.

The results of sensitivity analysis also imply several possible future research directions in order to improve the quality of the estimation results, such as developing simulation models that can make more valid predictions of the real system behavior, 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 in order to deal with more complex scenarios.

Funding

This research was supported by the China Scholarship Council (Grant no. 201306110027) and the National Natural Science Foundations of China (Grant no. 61374185 and 61403402).

References

1. Bouttier F and Courtier P. *Data assimilation concepts and methods*. Meteorological Training Course Lecture Series. Reading: ECMWF (European Centre for Medium-Range Weather Forecasts), 1999.
2. Atzori L, Iera A and Morabito G. The Internet of Things: a survey. *Comput Network* 2010; 54: 2787–2805.
3. Lee J, Lapira E, Bagheri B, et al. Recent advances and trends in predictive manufacturing systems in big data environment. *Manuf Lett* 2013; 1: 38–41.
4. Wu X and Liu HX. Using high-resolution event-based data for traffic modeling and control: an overview. *Transp Res C Emerg Technol* 2014; 42: 28–43.
5. Ho Y-C. Introduction to special issue on dynamics of discrete event systems. *Proc IEEE* 1989; 77: 3–6.
6. Nance RE. The time and state relationships in simulation modeling. *Commun ACM* 1981; 24: 173–179.
7. Schriber TJ, Brunner DT and Smith JS. How discrete-event simulation software works and why it matters. In: Laroque C, Himmelspach J, Pasupathy R, Rose O and Uhrmacher AM (Eds.) *Proceedings of the 2012 winter simulation conference*, Berlin, Germany, 9–12 December 2012, pp.1–15. Piscataway, NJ: IEEE.
8. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed. New York: Academic Press, 2000.
9. Nichols NL. *Data assimilation: aims and basic concepts*. Dordrecht: Springer Netherlands, 2003, pp.9–20.
10. Arulampalam MS, Maskell S, Gordon N, et al. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Trans Signal Proc* 2002; 50: 174–188.
11. Gillijns S, Mendoza O, Chandrasekar J, et al. What is the ensemble Kalman filter and how well does it work? In: *Proceedings of the 2006 American control conference*, Minneapolis, MN, USA, 14–16 June 2006, pp.4448–4453. Piscataway, NJ: IEEE.
12. Evensen G. The ensemble Kalman filter: theoretical formulation and practical implementation. *Ocean Dynam* 2003; 53: 343–367.
13. Yuan Y. *Lagrangian multi-class traffic state estimation*. PhD Thesis, Delft University of Technology, 2013.
14. Djurić PM, Kotecha JH, Zhang J, et al. Particle filtering. *IEEE Signal Proc Mag* 2003; 20: 19–38.
15. Gu F and Hu X. Towards applications of particle filters in wildfire spread simulation. In: Mason SJ, Hill RR, Mönch L,

Rose O, Jefferson T and Fowler JW (Eds.) *Proceedings of the 2008 winter simulation conference*, Miami, FL, USA, 7–10 December 2008, pp.2852–2860. Piscataway, NJ: IEEE.
16. Xue H, Gu F and Hu X. Data assimilation using sequential Monte Carlo methods in wildfire spread simulation. *ACM Trans Model Comput Simulat* 2012; 22: 23:1–23:25.
17. Wang M and Hu X. Data assimilation in agent based simulation of smart environments using particle filters. *Simulat Model Pract Theor* 2015; 56: 36–54.
18. Godsill S and Vermaak J. Variable rate particle filters for tracking applications. In: *IEEE/SP 13th workshop on statistical signal processing*, Bordeaux, France, 17–20 July 2005, pp.1280–1285. Piscataway, NJ: IEEE.
19. Godsill S, Vermaak J, Ng W, et al. Models and algorithms for tracking of maneuvering objects using variable rate particle filters. *Proc IEEE* 2007; 95: 925–952.
20. Long Y. *Data assimilation for spatial temporal simulations using localized particle filtering*. PhD Thesis, Georgia State University, 2016.
21. Wu P. *Sequential Monte Carlo based data assimilation framework and toolkit for dynamic system simulations*. PhD Thesis, Georgia State University, 2017.
22. Ntaimo L, Hu X and Sun Y. DEVS-FIRE: Towards an integrated simulation environment for surface wildfire spread and containment. *Simulation* 2008; 84: 137–155.
23. Hu X, Sun Y and Ntaimo L. DEVS-FIRE: design and application of formal discrete event wildfire spread and suppression models. *Simulation* 2012; 88: 259–279.
24. Vangheluwe HL. The Discrete EVent System specification (DEVS) formalism. Technical report, McGill University, School of Computer Science, Montreal, Quebec, Canada, 2001.
25. Douc R, Cappé O and Moulines E. Comparison of resampling schemes for particle filtering. In: *Proceedings of the 4th international symposium on image and signal processing and analysis*, Zagreb, Croatia, 15–17 September 2005, pp.64–69. Piscataway, NJ: IEEE.
26. Honig HJ and Seck MD. ϕ DEVS: phase based discrete event modeling. In: *Proceedings of the 2012 symposium on theory of modeling and simulation*, Orlando, FL, USA, 26–30 March 2012, pp.39:1–39:8. Orlando, FL: ACM.
27. Giambiasi N and Carmona JC. Generalized discrete event abstraction of continuous systems: GDEVS formalism. *Simulat Model Pract Theor* 2006; 14: 47–70.
28. Mannila H and Ronkainen P. Similarity of event sequences. In: *fourth international workshop on temporal representation and reasoning*, Daytona Beach, FL, USA, 10–11 May 1997, pp.136–139. Piscataway, NJ: IEEE.
29. Kanungo T, Mount DM, Netanyahu NS, et al. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Trans Pattern Anal Mach Intell* 2002; 24: 881–892.
30. Gu F and Hu X. Analysis and quantification of data assimilation based on sequential Monte Carlo methods for wildfire spread simulation. *Int J Model Simulat Sci Comput* 2010; 1: 445–468.

Author biographies

Xu Xie was a PhD student at the Department of Multi Actor Systems, Faculty of Technology, Policy and Management, Delft University of Technology, the Netherlands. He is now working as an assistant professor at the Department of Modeling and Simulation, College of System Engineering, National University of Defense Technology, Changsha, China.

Alexander Verbraeck is a professor at the Department of Multi Actor Systems, Faculty of Technology, Policy and Management, Delft University of Technology, the Netherlands.