DEVS-FIRE: Design and Application of Formal Discrete Event Wildfire Spread and Suppression Models

Xiaolin Hu (corresponding author)  
Department of Computer Science  
Georgia State University  
Atlanta, GA, 30303, U.S.A.  
xhu@cs.gsu.edu

Yi Sun  
Bioinformatics  
Sigma-Aldrich Corporation  
St. Louis, MO 63103, U.S.A.  
syish@hotmail.com

Lewis Ntaimo  
Department of Industrial and Systems Engineering  
Texas A&M University  
College Station, TX  77843, U.S.A.  
ntaimo@tamu.edu

Abstract  
DEVS-FIRE is a discrete event system specification (DEVS) model for simulating wildfire spread and suppression. It employs a cellular space model to simulate fire spread and agent models that interact with the cellular space to simulate fire suppression with realistic tactics. The complex interplay among forest cells and agents calls for formal treatment of the fire spread and fire suppression models to verify the correctness of DEVS-FIRE. This paper gives formal design specifications of fire spread and suppression agent models used in DEVS-FIRE and applies DEVS-FIRE to both artificially generated and real topography, fuels and weather data for a study area located in the US state of Texas. The paper also develops a new method, called pre_Schedule, for scheduling ignition events of forest cells more efficiently than the original onTime_Schedule event scheduling method used in DEVS-FIRE. Simulation results show the performance improvement of the new method, and demonstrate the utility of DEVS-FIRE as a viable discrete event model for wildfire simulations.

Keywords:  Discrete event simulation, DEVS, DEVS-FIRE, fire spread, fire suppression

1. Introduction  
Every year wildfires cause tremendous loss of natural resources, endangered species, human lives and property. The wildland urban interface (WUI), the area where structures and other human development overlap with undeveloped wildland, is usually in danger of wildfires. In the US several communities are within the WUI and are at risk from wildfires. Since 2000 US federal wildfire suppression expenditures alone have averaged more than $1 billion annually [1]. In 2008 a total of 78,949 wildfires were reported and they burned a total of 5,292,468 acres [2].
The ability to understand and predict wildfire location and behavior is important for effective wildfire initial response. Predictions regarding when a given wildfire will reach a certain location, its intensity, and the effect of firefighting tactics on the fire, are crucial for wildfire response planning. Faster and better prediction of fire behavior and firefighting would help in both fire containment planning and evacuation, and would limit the destruction wrought by wildfires.

However, modeling and simulating wildfire occurrence and behavior is difficult and time consuming. The complexity of fire behavior is due to the wide range of landscape, fuel, and environmental conditions in both time and space, which determine the dynamics of fire behavior. Consequently, modeling and simulating realistic wildfires adequately requires mathematically sound models of fire spread and a significant amount of computer memory and computational speed to accommodate the large-scale spatial and temporal data. Precise and reliable mathematical models for fire spread and fire suppression are needed. Motivated by this need, our previous work developed a discrete event model called DEVS-FIRE [3] for simulating wildfire behavior and wildfire suppression based on the DEVS formalism [4][5]. DEVS-FIRE models the forest as a cellular space where fire spreading is simulated as a contagion process between neighbor forest cells, whereby burning cells ignite their unburned neighbor cells. DEVS-FIRE uses agent models that interact with the cellular space to simulate wildfire suppression. Different types of agents are used to model different firefighting tactics based on realistic firefighting tactics.

The dynamic ignition process among forest cells during fire spread simulation and the complex interplay between agents and forest cells during fire suppression simulation call for formal treatment of the fire spread and fire suppression models to verify the correctness of DEVS-FIRE. A specification of an early implementation of the fire spread model was provided in [6]. Since then the model went through extensive changes, including a major extension on fire suppression simulation using agents, to become a fully operational model for both wildfire spread and wildfire suppression simulations. In this work we give mathematical formal specifications of both the fire spread models and fire suppression agent models used in DEVS-FIRE. The formal specification in this paper is necessary for adequately implementing sound models of fire spread and fire suppression in DEVS, and to provide a formal foundation for others to derive further work from this model. Besides formal specification, another focus of this work is on validation of the DEVS-FIRE model. Towards that goal, we apply DEVS-FIRE to both artificially generated and real topography, fuels and weather data for a study area located in the US state of Texas, and compare the results from DEVS-FIRE with those from FARSITE [21], a wildly used and validated model. Simulation results on wildfire suppression with different suppression tactics are also demonstrated. Finally, in order to reduce the number of simulation cycles and to speed up simulation execution time, we also extend the ignition event scheduling algorithm that we developed in earlier work for DEVS-FIRE. A new method for scheduling ignition events of forest cells, named pre_Schedule, is developed in this paper and integrated into DEVS-FIRE. This new scheduling method allows an ignited cell to immediately send out eight scheduling messages all at once to its neighbor cells, each of which schedules to ignite itself based on the received ignition delays. This compares to the original method, named onTime_Schedule, where an ignited cell send out ignition messages to its neighbor cells individually when they are about to be ignited. Formal specifications of both methods are provided and their performance results are reported. The simulation results show that DEVS-FIRE based on the new scheduling method has better performance resulting in a four-fold reduction in simulation cycles, a two-fold reduction in activity; and a two-fold reduction in
simulation execution time. The simulation results demonstrate the utility of DEVS-FIRE as a viable discrete event model for wildfire simulations.

We believe the contributions of this work are mathematical formal specifications of both wildfire spread and suppression agent models used in DEVS-FIRE; a new method for scheduling ignition events in cellular wildfire spread models; and application and validation of the extended DEVS-FIRE to simulate wildfire spread and suppression based on both artificially generated and real topography, fuels and weather data for a study area located in the US state of Texas. The rest of the paper is organized as follows: In Section 2 related work is reviewed and in Section 3 the architecture of DEVS-FIRE is presented. Section 4 first reviews the original fire spread model based on the onTime_Schedule method, and then presents the new one based on the pre_schedule method. Formal specifications for the two models are given in this section. Section 5 describes and gives a formal specification of the fire suppression agent models in DEVS-FIRE. Simulation results are reported in Section 6 and concluding remarks are given in Section 7.

2. Related work
This paper uses the cellular approach based on DEVS to model fire spread and is based on several previous work [3][6][7][8][9][10][11][12]. In particular, it builds on the DEVS-FIRE model in [3] and the integrated framework in [12]. DEVS-FIRE incorporates control response measures and represents an advance toward developing a real-time decision support system for fire spread prediction and the effects of suppression attempts. It uses the Rothermel model to calculate one-dimension fire spread and decomposes it into two-dimensions to produce elliptically shaped fires. The rate of spread within each cell assumes uniform topography and fuels, and constant weather conditions for the elapsed time. The work in this paper is closely related to [13], who develop mathematical specifications for fire spread models using DEVS and Cell-DEVS. Comparative simulation results from controlled laboratory experiments in [13] validate both simulation models of fire spread.

Several fire behavior models based on the cellular discrete event M&S have also been proposed and include [13][14], who considers a physical model of fire spread in [15] based on Timed Cell-DEVS [16]. In the model in [15] fire spread in each cell is calculated based on heat transfer partial differential equations. Cell-DEVS has also been used to model environmental systems in general [17]. A general object-oriented framework for M&S of propagation processes such as fire spread has been proposed in [18]. An integrated GIS and cellular automata (CA) [19] fire behavior model is presented in [20]. The GIS-CA model incorporates topography, forest fuel and weather data and its performance is evaluated by comparison with fire spread simulations derived from Prometheus using spatial data from a historical fire in Alberta, Canada. The study shows that the GIS-CA model can simulate realistic forest fire scenarios.

The widely used fire behavior models include FARSITE [21], BehavePlus [22][23], Prometheus [24], SiroFire [25] and HFire [26]. These models are designed to be used by fire behavior analysts who are familiar with the topography (slope, aspect and elevation), fuels, weather, and fire management issues. FARSITE models fire growth based on Huygens’ principle of wave propagation [27], which models fire evolution as a two-dimensional elliptical wave [28] using spatial topography data, fuels data and weather data. The perimeter of the fire is modeled as a polygon. First, a one-dimension fire spread speed and direction at discrete points along the fire front is computed, and then projections of the fire front are made over a finite time-step. The one-dimension fire spread is calculated using the Rothermel [29] surface fire spread model, which takes in as input local raster fuels, topography and weather data. Finally, a two-
dimensional fire spread is produced by considering all the points around the fire perimeter. Unlike FARSITE, BehavePlus computes fire spread based on an elliptically shaped fire [30][31]. Like FARSITE, BehavePlus uses the Rothermel surface fire spread model to compute the rate of spread of the head fire which is used to determine the size of the ellipse. HFire also uses the Rothermel model but is a raster-based surface fire spread model designed for fire behavior in Southern California chaparral.

Prometheus [24] is a Canadian fire simulation model designed for the boreal forest of Alberta, Canada. It is a deterministic fire growth simulation model and uses spatial fire behavior input data on topography and the Canadian forest fire behavior prediction (FBP) [32] fuel types and weather stream. Like FARSITE, Prometheus simulates fire growth based on Huygens' principle of wave propagation and uses the simple ellipse as the underlying model for the shape of fire growth. Fire behavior outputs are calculated using the Canadian Fire Weather Index (FWI) [33] and the FBP Systems which constitute the two primary subsystems of the Canadian Forest Fire Danger Rating System (CFFDRS).

SiroFire [25] is an Australian fire simulation model designed for the two major fuel types found in Australia's grass and forest. SiroFire is based on the elliptical wave model and uses a similar method to FARSITE to calculate the spread of the fire. The perimeter of the fire is represented by a polygon and a similar algorithm to FARSITE is used to calculate fire spread based on the input parameters. Another class of wildfire spread models are the coupled atmosphere-fire models [34][35]. These fire models include atmospheric effects on fire spread and account for the role of local convective wind patterns and dynamic fingering at the fire front, especially for large scale or very intense fires [36]. We refer the interested reader to [37][38] for surveys on fire spread models.

3. The Architecture of DEVS-FIRE
The DEVS-FIRE model is based on the DEVS formalism and supports discrete event simulation of both wildfire behavior and fire suppression tactics. In DEVS-FIRE the forest is modeled as a two-dimension cell space of rectangular cells whose dimensions depend on the resolution of the GIS fuel and topography data. Each cell performs its local computation of the rate of fire spread and direction based on its fuel, topography, and prevailing weather conditions. Different states are used to represent the stages of fire spread and communication with neighbor cells is done through message passing. Fire spread is modeled as a propagation process involving burning cells that ignite their unburned neighbor cells. Firefighting resources (e.g., dozers, fire crews, etc.) are modeled as firefighting agents and fire suppression is modeled as a process of constructing firelines to suppress or contain a burning fire. Firefighting agents execute fire suppression plans by interacting with the forest cells through coupling and message passing. While the cell space model simulates the dynamics of wildfire spread, firefighting agents simulate the fire suppression effort. This modeling approach separates the design concerns of fire behavior and fire suppression and makes it relatively easy to evolve each one independently.

Figure 1 shows the architecture of the DEVS-FIRE model, where the components in gray color are implemented as DEVS atomic or coupled models. At the center of DEVS-FIRE is the Forest Cell Space coupled model. This model comprises a grid of forest cells with the fuel, topography, and weather conditions assumed to be uniform within the cell. Each cell is a DEVS atomic model and transitions through different states (e.g., unburned, burning, burned) during the simulation. When ignited, each cell uses the Rothermel model [29] to compute a one-dimension fire spread (speed and direction), which is then decomposed into two-dimensions
based on an elliptical fire spread. Each cell has input and output ports through which couplings are made for exchanging messages, and it is coupled to the eight (if it is not on the boundary) adjacent neighbor cells as illustrated by cell\textsubscript{ij} in Figure 1. Through these couplings, fire spread across the cell space is enabled via message exchange between neighbor cells. Topography data (including slope data and aspect data) and fuel data are read through a topography and fuel data interface. This allows each forest cell to be initialized with the correct fuel and topography data. A cell receives weather data through a wind flow atomic model, which is coupled to all the cells for updating weather information. This temporal weather information is assumed to be available from a meteorological station closest to the forest fire. The forest cell igniter atomic model is responsible for igniting an initial set of cells to start the simulation. The visualization component displays the simulation results. It changes the display color of a cell whenever the cell’s state changes.

Figure 1. The DEVS-FIRE model architecture

To support fire suppression simulation, two more types of models, firefighting agent models and a dispatch model, are needed. A firefighting agent model is a DEVS atomic model of a firefighting resource such as a dozer. Therefore, its characteristics are based on the actual resource it represents. Firefighting resource characteristics include fireline production rate, arrival time to the fire, fixed rental cost and operational cost. Multiple firefighting agents can be added in a fire suppression simulation to model multiple groups of firefighting resources. To simulate fire suppression with realistic tactics, three types of firefighting agents are developed: agent\textsubscript{directAttack}, agent\textsubscript{parallelAttack}, and agent\textsubscript{indirectAttack}. These different types of agents carry out different fire suppression tactics (see [39] for a discussion of different firefighting tactics). Further details regarding the operations of firefighting agents are provided later in Section 5. The dispatch model is responsible for creating firefighting agents and adding them to the simulation. It reads from a firefighting resource dispatch plan and creates the firefighting agents to execute that plan. We have developed stochastic optimization models to generate such dispatch plans based on the available firefighting resources and the predicted fire spread.
spread behavior [12][40]. The fire suppression simulations provide a tool to assess such plans, and to experiment with different firefighting tactics [12].

Figure 2 shows the class diagram for the key classes in DEVS-FIRE. Based on their functions, the classes are grouped into two packages: one for fire spread simulation and one for fire suppression simulation. The figure shows how the DEVS-FIRE models are integrated into the DEVSJAVAs class hierarchy by inheriting two key classes from DEVSJAVAs: ViewableAtomic and ViewableDigraph, which correspond to an atomic model and a coupled model in DEVS, respectively. For the classes in the fire spread simulation package, the ForestCellSpace is a sub-class of ViewableDigraph (through the TwoDimCellspace and OneDimCellspace classes). It consists of several ForestCell atomic models and a ForestCellIgniter and a WindFlowModel. The ForestCell class is a subclass of ViewableAtomic (through the TwoDimCell and OneDimCell classes) and uses Rothermel’s model for calculating fire spread behavior. The ForestCellIgniter and WindFlowModel are DEVS atomic models and are subclasses of ViewableAtomic as well. The inheritance relationships between those two classes and the ViewableAtomic are not shown in Figure 2. For the classes in the fire suppression simulation package, the ForestCellSpace_Suppression class inherits the ForestCellSpace class from the fire spread simulation package. It includes multiple ForestCell_Suppression, which inherits the ForestCell class. The main difference between ForestCell_Suppression and ForestCell is that the ForestCell_suppression model has more states representing the suppressed situations of the cell (see Section 5 for further details). The ForestCellSpace_Suppression class also includes multiple Agent models, which are subclasses of ViewableAtomic. There are three types of agents implemented by three different classes: Agent_DirectAttack, Agent_ParallelAttack, and Agent_IndirectAttack. Finally, the ForestCellSpace_Suppression includes a Dispatch model, which is a subclass of ViewableAtomic, and is responsible for creating and adding Agents to the simulation. Next we discuss how the fire spread model and fire suppression model work, and provide formal specifications of both models.

Figure 2. DEVS-FIRE major classes
4. The Fire Spread Model

The fire spread model includes a forest cell space coupled model, which is composed of a grid of forest cell atomic models, an igniter model and a wind flow model. The igniter model and the wind flow model are straightforward to implement and understand. The former is coupled to one (or more) forest cell and sends out an ignition message to start a fire. The later is coupled to all the cells. It reads a weather data file and sends out weather information (including wind speed and direction) based on their time indexes, e.g., in every 30 minutes. In this section we focus on the forest cell model and omit the specifications of the igniter and wind flow models.

In DEVS-FIRE the two-dimensional forest cell space is composed of many forest cells. Each cell has eight fixed spread directions (propagation lines) N, NE, E, SE, S, SW, W, and NW according to the center to center directions between the cell and its neighbor cells. These directions correspond to an azimuth (degrees measured clockwise from the north) of 0, 45, 90, 135, 180, 225, 270, and 315 degrees, respectively. Accordingly, each cell has eight input ports, denoted \{"inN", "inNE", "inE", "inSE", "inS", "inSW", "inW", "inNW"\}, through which it can receive messages from its neighbor cells; each cell has eight output ports, denoted \{"outN", "outNE", "outE", "outSE", "outS", "outSW", "outW", "outNW"\} through which it can send output to its neighbor cells. A cell is coupled to its neighbor cells (the Moore neighborhood) using these input and output ports. With the couplings, fire spread across the cell space is enabled via message exchange between neighbor cells.

When a cell is ignited (either by the igniter model or by a neighbor cell), Rothermel’s model [29] is used to compute the cell’s fire behavior. Based on the cell’s fuel model type and environmental parameters (slope, wind speed, and wind direction), Rothermel’s model computes the maximum fire spread and the corresponding direction of spread, as well as flame length and fire intensity. Once the maximum fire spread and direction are known, the fire spread is decomposed into the eight spread directions using a model that defines fire shape as an ellipse as proposed by [30] and [31]. This is similar to what is used in FARSITE, BEHAVE and HFIRE, which also assume elliptical fire shapes in decomposing the one-dimension maximum rate of spread and direction from Rothermel’s mathematical model to achieve two-dimension spread. The decomposition gives the rate of spread in each of the eight directions. Then the time delays it takes for the fire to spread from the center of this cell to the centers of the neighbor cells are calculated by dividing the center distances by the corresponding rates of spread. These time delays are referred to as ignition delays in this paper. Based on these ignition delays, the forest cell schedules to ignite its neighbor cells, and thus simulates the fire spreading from its center to its neighbors. Below we explain how a forest cells schedules to ignite its neighbors.

We developed two different scheduling methods to enable a burning cell to ignite its neighbors: onTime-Schedule and pre-Schedule. The pre_Schedule method is a new method developed in this paper in order to improve simulation performance. These two scheduling methods result in two different implementations of the fire spread model, which are named onTime_Schedule model and pre_Schedule model. Figure 3 illustrates the two models. In the figure, the center cell (in gray color) is just ignited (burning). Based on the computation described above, the ignition delays for this cell to ignite its eight neighbor cells are \{t0, t1, t2, t3, t4, t5, t6, t7\} as shown in the figure. Figure 3(a) shows the onTime_Schedule model. In this model, the center cell is responsible for keeping track of which cell should be ignited at what time. When that time arrives, it sends out an ignition message to the corresponding neighbor cell, which ignites itself right away after receiving the message (unless it is unburnable). To
implement this, the center cell sorts the ignition delays and sends out messages according to the sorted order. For example, in Figure 3(a) we assume \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\} is a sorted order from small to large. Based on this order, the center cell will send an ignition message to the outN cell at \(t_0\). Then after elapsed time \(e = t_1 - t_0\), it will send an ignition message to the outNE cell (at time \(t_1\)). This continues until all ignition messages are sent out. As can be seen in the onTime_Schedule model, to ignite all its eight neighbor cells, a cell needs to send out eight messages at eight different times, one message each time.

![Diagram of onTime_Schedule](image)

**Figure 3(a) onTime_Schedule**

![Diagram of pre_Schedule](image)

**Figure 3(b) pre_Schedule**

Figure 3(b) shows the pre_Schedule model. In this model, the center cell does not keep track of which cell should be ignited at what time. Instead, it sends the schedule information to the neighbor cells and asks the neighbor cells to keep track of their own ignition time. To implement this, whenever a cell is ignited it sends out eight scheduling messages to its eight neighbors all at once. Each scheduling message contains an ignition delay, indicating the cell receiving the message should be ignited after that ignition delay. A cell that receives such a message will store the ignition delay and schedule to ignite itself when that time arrives. For example, in Figure 3(b) the center cell sends out eight messages all at once: the message sent to the outN cell contains the ignition delay \(t_0\), the message sent to the outNE cell contains the ignition delay \(t_1\), etc. As a result, the outN cell schedules to ignite itself after \(t_0\). Other cells also schedule to ignite themselves based on their own ignition delays. Compared to the onTime_Schedule model that sends out the ignition messages at eight different times, the pre_Schedule model sends out the scheduling messages (contains the ignition delays) all at once at the time when the cell is ignited.

The two scheduling methods require implementing the forest cell using different states and state transition rules. The onTime_Schedule model is easier to understand and implement than the pre_Schedule model. This is because in the onTime_Schedule model, the ignition delays are managed in a centralized place: the center cell in Figure 3. In the pre_Schedule model, however, this information is distributed across the eight neighbor cells. Thus whenever there is weather update and the ignition delays are re-calculated, the pre_Schedule model needs to re-send the updated ignition delays to its neighbor cells. To make this work, an unburned cell needs to keep all the received ignition delays from its neighbor cells because these ignition delays could be changed later. More details about the implementations and formal specifications of these two models are given in Sections 4.1 and 4.2. Despite their different implementation logic, it is
important to note that both models rely on the same Rothermel’s model and decomposition
schema to compute the fire spread behavior and then decompose into eight spreading directions.
They have the same ignition events but schedule them in different ways. Because of this, both
models should give exactly the same simulation results.

The motivation for developing the pre_Schedule model is because of its superior
performance results. This is due to the different number of simulation iterations (simulation
cycles) resulting from simulating the two models. In the onTime_Schedule model each ignited
cell sends out eight ignition messages at eight different times. Each message is an event and
needs a simulation cycle for processing the event. Thus each ignited cell will result in eight
simulation cycles (due to the eight ignition messages it sends out). However, in the pre_Schedule
model the eight scheduling messages are sent out all at once. This happens in one simulation
cycle in the discrete event simulation. Although the pre_Schedule model introduces two more
states and state transitions (see Section 4.2), typically it will result in less simulation cycles, and
thus gives better performance result than the onTime_Schedule model. A special case is when
there are frequent weather updates. In the onTime_Schedule model, a weather update simply
makes a burning cell recalculate its remaining ignition delays. It does not cause any extra
messages to be sent out. But in the pre_Schedule model, a weather update will make a burning
cell send a new set of scheduling messages to its neighbors to update the previously sent ignition
delays. Some performance results that measure the simulation performance of the two models are
provided in Section 5.

4.1 The onTime_Schedule Fire Spread Model

In the onTime_Schedule model, an ignited cell sends out ignition messages to its neighbor cells
when they should be ignited. In DEVS-FIRE, an onTime_Schedule forest cell is a DEVS atomic
model and has three states: unburned, burning, and burned. Figure 4 shows the state transition
diagram of the onTime_Schedule forest cell. The rounded rectangles represent the cell’s states
and their time advances (sigma). A solid filled arrow represents an external transition and a
dashed filled arrow represents an internal transition of the DEVS model. An arrow attached to an
internal transition represents the output of the model that happens right before the internal
transition.

![Figure 4. State transitions of the onTime_Schedule model](image)

Each cell is initialized in the unburned state (with sigma equals infinity) and remains in that
state until it is ignited. The cell transitions to the burning state if it receives an ignition message
from the igniter model or from a neighbor cell and its fireline intensity is above the ignition threshold. In Figure 4, the fireline intensity is denoted as $F_I$ and the ignition threshold is denoted as threshold. The inIgniter represents the input port coupled to the igniter model, and inNeighbor$_i$ represents one of the eight input ports for receiving messages from the neighbor cells, $i \in \{\text{inN}, \text{inNE}, \text{inE}, \text{inSE}, \text{inS}, \text{inSW}, \text{inW}, \text{inNW}\}$. As described above, once a cell is ignited it calculates the ignition delays for its eight neighbors, and sorts these delays from small to large. The set of sorted delays is denoted as $\{d_i\}_{\text{ordered}}$, $i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ in Figure 4. Based on these ignition delays, the cell needs to send out ignition messages one by one when the delays expire. To implement this, when the cell transitions from the unburned state to the burning state it sets its time advance $\bullet$ to $d_0$, which is the smallest in the delay set. When that time expires, the cell sends out an ignition message to the corresponding neighbor cell. It then removes $d_0$ from $\{d_i\}_{\text{ordered}}$ and carries out an internal transition back to the burning state with a new time advance $\bullet = d_1 - d_0$. This continuous until $\{d_i\}_{\text{ordered}} = \emptyset$, meaning all the ignition delays have elapsed (and all the ignition messages have been sent out). Then the cell transitions to the burned state as shown in Figure 4. When a cell is in the burning state and receives a weather update from the inWeather input port, it calculates the remaining ignition delays using the new weather information. These remaining ignition delays are sorted and then the cell transitions to the burning state with the time advance set to the smallest one of them.

Based on the above description, we present a formal specification of the onTime_Schedule forest cell model in the DEVS formalism. Note that our specification focuses on the changes of states and their associated times. It leaves out the implementation details and does not include state variables such as fireline intensity.

An onTime_Schedule forest cell atomic model, denoted $\text{Forestcell}_{\text{onTime_Schedule}}$, can be formally expressed in DEVS as follows:

$$\text{Forestcell}_{\text{onTime_Schedule}} = <X_M, Y_M, S, \cdot_{\text{ext}}, \cdot_{\text{int}}, \cdot_{\text{con}}, \bullet, \tau>,$$

where

$X_M = \{(p, v) | p \in \text{InPorts}; v \in X_p\}$ is the set of input ports and values;

$Y_M = \{(p, v) | p \in \text{OutPorts}; v \in Y_p\}$ is the set of output ports and values;

$S = \{(\text{phase}, \bullet) | \text{phase} \in \text{States}; \bullet \in R_{0,\infty}\}$ is the set of states and associated time durations.

Table 1 shows the definitions for $X_M$, $Y_M$, and $S$ for the onTime_Schedule model. In the table, ignition denotes an ignition message; $(\text{wsp}, \text{wdir})$ denotes a wind speed and wind direction pair.
 Specification of the initial state, internal transition function, external transition function, confluent transition function, output function, and time advance function of the onTime_Schedule model is given below. In the specification, \{d_i\}_{ordered} represents the sorted ignition delays; \{d_i\}'_{ordered} is the recomputed ignition delays set after receiving new weather data; inNeighbor \in \{“inN”, “inNE”, “inE”, “inSE”, “inS”, “inSW”, “inW”, “inNW”\} is one of the input ports; outNeighbor \in \{“outN”, “outNE”, “outE”, “outSE”, “outS”, “outSW”, “outW”, “outNW”\} is one of the output ports. The symbols \lor and \land denote logic OR and AND, respectively.

**Initial state:**

("unburned", \infty).

**Internal transition function:**

\[\delta_{int}("burning", \sigma) = ("burning", \tau), \tau = d_i - d_{i-1} \quad \text{if} \quad \{d_i\}_{ordered} \neq \emptyset;\]
\[\delta_{int}("burning", \sigma) = ("burned", \infty) \quad \text{if} \quad \{d_i\}_{ordered} = \emptyset.\]

**External transition function:**

\[\delta_{ext}("unburned", \sigma, e, x) = ("burning", \tau), \tau = \min\{d_i\}_{ordered} \quad \text{if} \quad x = ("inIgnited", ignition) \lor (\text{inNeighbor}_i, ignition) \land FI \geq \text{threshold};\]
\[\delta_{ext}("burning", \sigma, e, x) = ("burning", \tau'), \tau' = \min\{d_i\}'_{ordered} \quad \text{if} \quad x = ("inWeather", (wsp, wdir));\]
\[\delta_{ext}(phase, \sigma, e, x) = (phase, \sigma - e), \quad \text{otherwise.}\]

**Confluent transition function:**

\[\delta_{con}(phase, \sigma, x) = \delta_{ext}(\delta_{int}(phase, \sigma), 0, x).\]

**Output function:**

\[\lambda("burning", \sigma) = (\text{outNeighbor}_i, ignition).\]

**Time advance function:**

\[X_M \rightarrow Y_M \equiv \{("inN", ignition), ("inNE", ignition), ("inE", ignition), ("inSE", ignition), ("inS", ignition), ("inSW", ignition), ("inW", ignition), ("inNW", ignition), ("inWeather", (wsp, wdir)), ("inIgniter", ignition)\} \rightarrow \{("outN", ignition), ("outNE", ignition), ("outE", ignition), ("outSE", ignition), ("outS", ignition), ("outSW", ignition), ("outW", ignition), ("outNW", ignition)\} \rightarrow ("unburned", \infty);

("burning", \tau), \tau \in [0, \infty);

("burned", \infty)\]
\[ ta(\text{phase}, \sigma) = \sigma. \]

### 4.2 The pre-Schedule Fire Spread Model

In the pre_Schedule model, an ignited cell sends out eight scheduling messages all at once to its neighbors, each of which schedules to ignite itself based on the received ignition delays. Besides the unburned, burning, and burned states as in an onTime_Schedule forest cell, a pre_Schedule forest cell has two new states: schedule_to_burn and resend_schedule. The schedule_to_burn state is used to schedule the cell to ignite itself after receiving scheduling messages from neighbor cells. Note that in this state the cell is not ignited yet. The resend_schedule state is used to send updated ignition delays to the neighbor cells after receiving new weather information from the weather model. Besides the two new states, another difference between the two models is the type of messages passed between the neighbor cells. In the onTime_Schedule model, the message sent from an ignited cell to a neighbor cell is the ignition message. In the pre_Schedule model, this message is the scheduling message that contains the ignition delay for the neighbor cell. Figure 5 shows the state transition diagram of the pre_Schedule forest cell. The notation and descriptions used in this figure have the same meaning as that used in Figure 4 in Section 4.1.

![State transition diagram](image)

Each cell is initialized in the unburned state (with \( \sigma = \infty \)). It transitions to the schedule_to_burn state when it receives a message from the igniter model or from a neighbor cell. If the message comes from the igniter model, it means the cell needs to ignite itself right away. In this case the time advance for the schedule_to_burn state is set to zero (\( \tau = 0 \)). Otherwise, the time advance is set to the ignition delay contained in the message from the neighbor cell. When a cell is in the schedule_to_burn state, it may receive other scheduling messages from its neighbor cells. These messages may come from a neighbor cell that has already sent a scheduling message before, or from a new neighbor cell. The former case corresponds to the situation when there is weather change and thus the previous ignition delay needs to be updated. In this case, the previous ignition delay is replaced with the new received one. The latter case means another neighbor cell schedules to ignite this cell. In this case, a new entry of ignition delay is added into the existing set of ignition delays stored by the cell. In the pre_Schedule model, these ignition delays are referred to as the schedule-to-burn delays. The set of sorted schedule-to-burn delays (from small to larger) is denoted as \{d_i\}_{ordered} in Figure 5.
When a cell receives a scheduling message in the schedule_to_burn state, it stays in the same state and sets its time advance to the smallest of the updated schedule-to-burn delays ($\cdot = \min \{d_i\}_{\text{ordered}}$). We note that as an implementation detail, whenever a new scheduling message is received, all the existing schedule-to-burn delays in $\{d_i\}_{\text{ordered}}$ will also be updated appropriately to ensure they start from the same time base.

When the time advance associated with the schedule_to_burn state expires, the cell removes the current $d_i$ from $\{d_i\}_{\text{ordered}}$, and invokes Rothermel’s model to calculate its fireline intensity $FI$. Three situations may occur: 1) $FI \cdot \text{threshold}$. In this case the cell is successfully ignited, and thus transitions to the burning state; Figure 5 shows that in this case the cell also sends out eight scheduling messages (containing the ignition delays) to its neighbors before transitioning to the burning state. 2) $FI < \text{threshold}$ and $\{d_i\}_{\text{ordered}} \neq \emptyset$. This means the cell cannot be ignited at this time, but other neighbor cells have scheduled to ignite this cell at a later time. In this case, the cell transitions back to the schedule_to_burn state and sets its time advance to the difference between the previous schedule-to-burn delay and the next earliest schedule-to-burn delay, denoted as $\cdot = d_i - d_{i-1}$ in the specification below; 3) $FI < \text{threshold}$ and $\{d_i\}_{\text{ordered}} = \emptyset$. This means the cell cannot be ignited at this time and there is no other pending ignition event for this cell. Thus the model transitions to the unburned state. When the cell transitions to the burning state, its time advance (denoted as duration in Figure 5 and the specification below) is set to the maximum ignition delay sent out. After that delay expires, the cell transitions to the burned state with time advance $\sigma = \infty$. When the cell is in burning state and receives a weather update from the inWeather port, it calculates the remaining ignition delays using the new weather information and sends out new scheduling messages to its neighbor cells. To support this, the cell transitions to a transient state (whose time advance $\sigma = 0$) named resend_schedule and then transitions back to the burning state as shown in Figure 5.

The formal specification of the pre-Schedule model is given below. In the specification, the $\{d_i\}_{\text{updated}}$ represents the updated and sorted set of schedule-to-burn delays after receiving new scheduling messages; duration is the remaining duration of the burning state after receiving a weather update. Similarly, this specification focuses on the changes of states and their associated times, and leaves out the implementation details.

A pre_Schedule forest cell atomic model, denoted as Forestcell_pre_Schedule, can be formally expressed in DEVS as follows:

$$\text{Forestcell}_\text{pre\_Schedule} = <X_M, Y_M, S, \cdot \text{ext}, \cdot \text{int}, \cdot \text{con}, \cdot, \cdot \text{ta}>$$

where

- $X_M = \{(p, v) \mid p \in \text{InPorts}; \ v \in X_p\}$ is the set of input ports and values;
- $Y_M = \{(p, v) \mid p \in \text{OutPorts}; \ v \in Y_p\}$ is the set of output ports and values;
- $S = \{(\text{phase}, \cdot) \mid \text{phase} \in \text{States}; \ \cdot \in R_{0,\infty}\}$ is the set of states and associated time durations.

Table 2 defines the $X_M$, $Y_M$, and $S$ for the pre_Schedule model.
The initial state, internal transition function, external transition function, confluent transition function, output function, and the time advance function of the pre_Schedule model can now be defined as follows:

**Initial state:**
("unburned", ∞).

**Internal transition function:**
\[ \delta_{\text{int}}("\text{schedule\_to\_burn}\), \sigma) = ("\text{burning}\), \text{duration}) \]
if \( F I \geq \text{threshold} \);
\[ \delta_{\text{int}}("\text{schedule\_to\_burn}\), \sigma) = ("\text{schedule\_to\_burn}\), \tau) = d_i - d_{i-1} \]
if \( F I < \text{threshold} \) \& \{d_i\}_\text{ordered} \neq \emptyset ;
\[ \delta_{\text{int}}("\text{schedule\_to\_burn}\), \sigma) = ("unburned", ∞) \]
if \( F I < \text{threshold} \) \& \{d_i\}_\text{ordered} = \emptyset ;
\[ \delta_{\text{int}}("\text{resend\_schedule}\), \sigma) = ("\text{burning}\), \text{duration'}) ;\]
\[ \delta_{\text{int}}("\text{burning}\), \sigma) = ("\text{burned}\), ∞).

**External transition function:**
\[ \delta_{\text{ext}}("\text{unburned}\), \sigma, e, x) = ("\text{schedule\_to\_burn}\), 0) \]
if \( x = (\text{inIgniter}, \text{ignition}) ;\)
\[ \delta_{\text{ext}}("\text{unburned}\), \sigma, e, x) = ("\text{schedule\_to\_burn}\), d_i) \]
if \( x = (\text{inNeighbor}, d_i) ;\)
\[ \delta_{\text{ext}}("\text{schedule\_to\_burn}\), \sigma, e, x) = ("\text{schedule\_to\_burn}\), \tau) = \text{min}\{d_i^{\text{updated}}\}_\text{ordered} \]
if \( x = (\text{inNeighbor}, d_i) ;\)
\[ \delta_{\text{ext}}("\text{burning}\), \sigma, e, x) = ("\text{resend\_schedule}\), 0) \]
if \( x = (\text{inWeather}, (\text{wsp, wdir})) ;\)
\[ \delta_{\text{ext}}(\text{phase}, \sigma, e, x) = (\text{phase}, \sigma - e) \), \text{otherwise} .\]

**Confluent transition function:**

<table>
<thead>
<tr>
<th>( X_M )</th>
<th>( Y_M )</th>
<th>( S )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&quot;inN&quot;, ( d_N ))</td>
<td>(&quot;outN&quot;, ( d_N ))</td>
<td>(&quot;unburned&quot;, ∞)</td>
</tr>
<tr>
<td>(&quot;inNE&quot;, ( d_{NE} ))</td>
<td>(&quot;outNE&quot;, ( d_{NE} ))</td>
<td>(&quot;resend_schedule&quot;, 0)</td>
</tr>
<tr>
<td>(&quot;inE&quot;, ( d_E ))</td>
<td>(&quot;outE&quot;, ( d_E ))</td>
<td>(&quot;schedule_to_burn), ( \tau ), ( \tau \in [0, \infty) )</td>
</tr>
<tr>
<td>(&quot;inSE&quot;, ( d_{SE} ))</td>
<td>(&quot;outSE&quot;, ( d_{SE} ))</td>
<td>(&quot;burning), \text{duration}) ), duration ( \in [0, \infty) )</td>
</tr>
<tr>
<td>(&quot;inS&quot;, ( d_S ))</td>
<td>(&quot;outS&quot;, ( d_S ))</td>
<td>(&quot;burned), ∞)</td>
</tr>
<tr>
<td>(&quot;inSW&quot;, ( d_{SW} ))</td>
<td>(&quot;outSW&quot;, ( d_{SW} ))</td>
<td></td>
</tr>
<tr>
<td>(&quot;inW&quot;, ( d_W ))</td>
<td>(&quot;outW&quot;, ( d_W ))</td>
<td></td>
</tr>
<tr>
<td>(&quot;inNW&quot;, ( d_{NW} ))</td>
<td>(&quot;outNW&quot;, ( d_{NW} ))</td>
<td></td>
</tr>
<tr>
<td>(&quot;in\text{Weather}), (\text{wsp,wdir}))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&quot;in\text{Igniter}), \text{ignition})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\( \delta_{\text{con}}(\text{phase}, \sigma, x) = \delta_{\text{ext}}(\delta_{\text{int}}(\text{phase}, \sigma), 0, x). \)

**Output function:**

\[
\lambda("\text{schedule}_\text{to}_\text{burn}") = (\text{out}Nei\text{ghbor}_i, d_i), \text{for all } i \in \{0, 1, 2, 3, 4, 5, 6, 7\} \\
\text{if } \text{FI} \geq \text{threshold}; \\
\lambda("\text{resend}_\text{schedule}" ) = (\text{out}Nei\text{ghbor}_i, d_i), \text{for all } i \in \{0, 1, 2, 3, 4, 5, 6, 7\}
\]

**Time advance function:**

\( ta(\text{phase}, \sigma) = \sigma. \)

5. The Fire Suppression Model

The fire suppression model is built based on the fire spread model and extends it for fire suppression simulation. The extensions include adding new states to the forest cell model to support suppression, and using firefighting agents to simulate fire suppression efforts. This section describes the forest cell model and the agent model used in fire suppression simulation.

5.1. The Forest Cell Model in Fire Suppression Simulation

A wildfire suppression simulation not only needs to simulate the fireline construction from firefighting resources, but also needs to simulate the spread of wildfire while the firelines are constructed. To simulate fire spread, a forest cell in the fire suppression simulation needs to fulfill the same functions as in fire spread simulation. It is thus implemented by inheriting the forest cell model in fire spread simulation, including using one of the two scheduling methods: onTime_Schedule or pre_Schedule. We use the onTime_Schedule forest cell as an example to illustrate how the forest cell is extended to allow for fire suppression.

Whenever a forest cell receives a suppression message from an agent, it transitions to a “suppressed” state and remains there forever to represent fire containment. A suppressed state is a passive state and cannot be ignited. A cell may be suppressed when it is in the unburned, burning, or burned state. To differentiate them, the cell transitions to three different suppressed states: unburned_suppressed, burning_suppressed, or burned_suppressed, respectively. Figure 6 shows the state transition diagram of the forest cell model in fire suppression simulation. In the figure, inSuppress denotes the cell’s input port that receives the suppression message from firefighting agents. As can be seen, this model closely follows the model in Figure 4. The state transitions for simulating the fire spread are the same as those described in Section 4.1, and thus their descriptions are omitted here. The only difference is the addition of the three new suppressed states and the associated external transitions to these states.
The formal specification of this model also follows that in Section 4.1. To save space we only provide the specifications for the three additional external transitions for processing the suppression message, denoted as suppression, from firefighting agent. These specifications and the specifications in Section 4.1 together form the full specification of the forest cell model in fire suppression simulation.

\[
\delta_{ext}("unburned", \sigma, e, x) = ("unburned\_suppressed", \infty)
\]

if \(x = ("in\_Suppress", \text{suppression})\);

\[
\delta_{ext}("burning\_\tau", \sigma, e, x) = ("burning\_suppressed", \infty)
\]

if \(x = ("in\_Suppress", \text{suppression})\);

\[
\delta_{ext}("burned", \sigma, e, x) = ("burned\_suppressed", \infty)
\]

if \(x = ("in\_Suppress", \text{suppression})\).

Extending the \text{pre\_Schedule} forest cell to support fire suppression is also straightforward. For a \text{pre\_Schedule} forest cell, both the unburned state and the \text{schedule\_to\_burn} states mean the cell is not ignited yet. Thus in both these states, the cell will transition to the \text{unburned\_suppressed} state when receiving a suppression message from the \text{in\_Suppress} port. This is the only difference from extending the \text{onTime\_Schedule} model described above.

\section{5.2. The Agent Model}

Fire suppression is modeled as a process for firefighting resources to construct suppression firelines to contain a burning fire. In DEVS-FIRE, the firefighting resources are modeled by firefighting agents. Each firefighting agent is a DEVS atomic model. The interaction between an agent and the forest cells is supported by the couplings between them. An agent is always coupled to the cell where it is located. This coupling allows the agent to send a suppression message to the cell when the cell is suppressed. In a cellular space model where the space is divided into discrete cells, it is assumed that an agent can only proceed, i.e., construct the fireline, from center to center between two neighbor cells. Each agent has a production rate that defines
how fast this agent can construct a fireline. Based on the production rate and the cell size, an agent calculates the time needed to finish a fireline segment. Once that time expires, the agent sends a suppression message to the cell, which then transitions to a suppressed state as described in Section 5.1. After the agent finishes suppressing a cell, it chooses a neighbor cell to move to and to work on. When the agent moves, the coupling between the agent and the old cell is removed, and a coupling between the agent and the new cell is dynamically added. This process continues until the fire is completely contained or the simulation ends. The dynamical coupling between agents and cells is modeled using the dynamic structure modeling capability of DEVSJAVA. In DEVS-FIRE each agent is responsible for changing its own couplings as it moves from one cell to another.

Figure 7 shows the state transition diagram of the agent model. The agent is initialized in the passive state and is coupled to the cell where the agent locates. After it receives a message from the inStart port (sent from the dispatch model), it transitions to the suppressing state and begins to suppress the forest cell. The time advance of suppressing is set to the duration for suppressing the cell. It is calculated from the cell size and the agent’s production rate. When the time advance expires, the agent sends out a suppression message. It then selects a neighbor cell (uses a “choose-a-neighbor” algorithm described later) to move to and transitions back to the suppressing state to suppress the new selected cell. When the fire is completely contained, the agent transitions to the passive state. Once the agent begins fire suppression, it moves through and suppresses a sequence of cells for constructing a fireline. As can be seen, the state transitions of the agent model are fairly straightforward and we therefore, omit a formal specification of the agent model.

While the agent model has simple state transitions, it however requires a relatively involved procedure for choosing a right sequence of cells to construct a fireline in order to simulate different firefighting tactics. In DEVS-FIRE we consider three types of firefighting tactics: direct attack, parallel (indirect) attack, and indirect attack. Direct attack refers to the tactic in which fireline is constructed on the flaming fire front. Two specific direct attack tactics are direct head attack and direct tail attack where the attacks start from the head and tail of the fire, respectively. Parallel (indirect) attack refers to the tactic in which fireline is constructed parallel to, but at a safe distance (offset) away from, the fire front perimeter. This is usually applied when the fire is intense and spreading fast, and thus having the potential for causing serious injuries or fatalities to the firefighters. Indirect attack refers to the tactic in which fireline is constructed according to a predetermined route. More information about these different tactics can be found in [39].

The different firefighting tactics result in different dynamical processes of fireline construction that need to be simulated by firefighting agents. Each agent, after completing a fireline segment, needs to decide where to proceed for constructing the next fireline segment.
This decision of where to proceed is either based on a pre-defined plan (such as in indirect attack) or on the dynamical behavior of fire spread (such as in direction attack and parallel attack). In DEVS-FIRE, since an agent can only proceed from a cell to its neighbor cell in a discrete fashion, the modeling of firefighting agent thus concerns how an agent chooses an appropriate neighbor cell to construct the next fireline segment. This is governed by an algorithm called the “choose-a-neighbor” algorithm that we developed. In DEVS-FIRE, the different firefighting tactics are simulated by different types of firefighting agents, which have different “choose-a-neighbor” algorithms to govern them to choose a neighbor cell. Below we take the firefighting agent in direct attack as an example and present its “choose-a-neighbor” algorithm. The algorithms for firefighting agents in parallel attack and in indirect attack can be found in [41].

In direct attack, agents build a fireline along the fire front where cells are burning. Since it takes time for a fireline segment to be constructed, a burning cell may ignite its neighbor cells while it is being suppressed by an agent. Thus in choosing which neighbor cell to construct the next fireline segment, an agent needs to ensure that the to-be-constructed fireline segment will be completed before any neighbor cells “outside” this fireline segment are ignited. In our design, this is achieved by a “predict-and-scan” schema (see [41] for more details). The basic idea of the “predict-and-scan” schema is that an agent needs to predict how far the current fire can spread based on how fast the agent can finish constructing the next fireline segment, and then uses the predicted fire front as a guidance to decide the direction for constructing the next fireline segment. Based on the “predict-and-scan” schema, the “choose-a-neighbor” algorithm for a firefighting agent in direct attack is given below. This algorithm includes two stages of prediction because the lengths for constructing a diagonal fireline segment and that for constructing a non-diagonal fireline segment are different, i.e., the former is $\sqrt{2}$ times the latter. In the algorithm, cellSize is the size of a cell, and production_rate is the agent’s production speed (meters per second) for constructing a fireline.

**The “choose-a-neighbor” algorithm for firefighting agent in direct attack**

Step 1: Calculate the time for building a fireline to a non-diagonal neighbor cell: $T_{\text{lookahead}} = \text{cellSize}/\text{production\_rate}$.

Step 2: Use $T_{\text{lookahead}}$ as the look-ahead time window to predict the fire spreading situation of the neighbor cells. Mark them as unburned, burning, burned, or suppressed.

Step 3: Apply the “scan” process described in [41] to scan its neighbor cells until meet the first burning cell. Two situations may happen:

i. If this cell is a non-diagonal cell, the cell is chosen as the destination cell for constructing the next fireline segment. The algorithm stops here.

ii. If this cell is a diagonal cell, aborts the scan and goes to Step 4 to start the second “prediction-and-scan” stage.

Step 4: Calculate the time for building a fireline to a diagonal neighbor cell: $T_{\text{lookahead}} = \sqrt{2} * \text{cellSize}/\text{production\_rate}$.

Step 5: Use $T_{\text{lookahead}}$ as the look-ahead time window to predict the fire spreading situation of the neighbor cells. Mark them as unburned, burning, burned, or suppressed.

Step 6: Apply the “scan” process again to scan its neighbor cells until meet the first burning cell. Choose this cell (diagonal or non-diagonal) as the destination cell.
In the algorithm, the procedure that is used to predict fire spread is the fire spread simulation itself. Specifically, an agent creates a new cell space model that duplicates the local area of the “original” cell space. The states of the cells in this new cell space are initialized to the current states of the corresponding cells in the original cell space. This new cell space model is then simulated until the given look-ahead time window is reached. Using the developed “choose-a-neighbor” algorithms, we were able to simulate different firefighting tactics in wildfire suppression [12][41]. Some simulation results are provided in Section 6.

The last component of the fire suppression model is the dispatch model. The behavior of the dispatch model is relatively simple. It creates firefighting agents according to a dispatch plan and adds the agents to the fire suppression simulation. Dynamic structure modeling is used when agents are dispatched dynamically during the simulation. A dispatch plan includes important information such as agent ID, production rate, dispatch time, dispatch location, and firefighting tactic type (direct attack, parallel attack, or indirect attack). It is an essential part of fire suppression simulation and is specified by a fire manager before the simulation starts [12]. When firefighting resources are divided into multiple groups working on different segments of the fireline, they are modeled by multiple agents, each of which corresponds to one group.

6. Experimental Results
We present three experiments and their results. The first experiment shows some simulation results to illustrate the fire spread and fire suppression simulations using DEVS-FIRE (Section 6.1). The second experiment compares the simulation performance of the onTime_Schedule model and the pre_Schedule model for fire spread simulation (Section 6.2). Finally, the third experiment uses real GIS data to demonstrate the applicability of DEVS-FIRE for simulating realistic fire scenarios (Section 6.3).

6.1. Fire spread and fire suppression simulations
This experiment demonstrates both the fire spread simulation and fire suppression simulation using uniform topography and fuel data. In this experiment, all simulation runs are based on a 200 × 200 cell space (the center cell has coordinates (100, 100)). Each cell has size 30 meters × 30 meters, 0 slope and 0 aspect, and standard fuel model 7 (southern rough), which represents chaparral and shrub fields. The wind speed and direction are randomly generated every hour for a six-hour period. The wind speed is generated from \( \text{uniform}(8.05, 24.14) \) kilometers/hour while the wind direction is sampled from \( \text{uniform}(0, 180) \) degrees azimuth. We use the conventional sense for the wind direction, that is, the direction that the wind is coming from (0 degrees means the wind is coming from the north). The hourly wind speed (kilometers/hour) and wind direction (degrees azimuth) for the six-hour period starting from the first hour are: (20.92, 101), (16.09, 2), (11.27, 168), (24.14, 157), (19.31, 13), and (9.66, 130). The fire is arbitrarily chosen to be ignited at the cell (120, 100) of the study area at time \( t = 0 \).
We ran fire spread simulations using both the onTime_Schedule model and the pre_Schedule model in DEVS-FIRE. For validation purposes, we also ran the fire spread simulations in FARSITE using the same input data. We then compared the simulation results from DEVS-FIRE to those from FARSITE, a widely used and validated model. For each simulation run we recorded the time-indexed fire shapes, fire perimeters, and burned areas at the end of every hour for the six-hour prediction horizon. The fire growth maps obtained using the DEVS-FIRE models are shown in Figure 8(a) and those for FARSITE are shown in Figure 8(b). The two DEVS-FIRE models, onTime_Schedule and pre_Schedule, gave exactly the same results displayed in Figure 8(a). In Figure 8(a) and 8(b), the different circles represent the evolving fire shapes (starting from the inner to the outer) at the end of each hour. As can be seen, the results from DEVS-FIRE follow the same trend as those from FARSITE. We should point out that since the study area has uniform fuels and topography (0 slope and 0 aspect), the fire growth is mainly driven by the wind speed and direction. The higher the wind speed, the faster the fire spreads. As wind direction changes, the wildfire changes its spreading direction too. For example, in the 4th hour, the fire spread towards the northwest direction because the wind direction is 157 degree; in the 5th hour, the fire spreads towards the south direction because the wind direction is 13 degree. In Figure 8, fire growth is relatively slow for the 3rd and the 6th hour because the wind speed at these two time periods is relatively low (11.27 kilometers/hour and 9.66 kilometers/hour, respectively). The reason for that the fire perimeters in DEVS-FIRE are not as smooth as those in FARSITE is because DEVS-FIRE uses discretized cells to represent the space and it decomposes the fire spreading only along eight spreading directions.

The hourly perimeter and burned area for each simulation model are reported in Table 3. As can be seen in the table, onTime_Schedule and pre_Schedule give the same results. Also, the results from both these two models are on average within 10% of the results from FARSITE. Extensive comparative simulation results between DEVS-FIRE and FARSITE for various fuel models, topography data, and weather data are reported in [42]. The results from these experiments show that DEVS-FIRE produces simulation results that are consistent with FARSITE.
Next, we used the same input data to run fire suppression simulations. We simulated direct attack with two firefighting agents (using the Agent_DirectAttack model), each with a production rate of 0.3 meters/second. The two agents are dispatched to the fire front at time $t = 3$ hours, i.e., after the fire has spread for 3 hours. Both agents started fire suppression from the same location with one constructing the fireline clockwise and the other counter-clockwise. We ran simulations with two different starting locations and compared the results. Figure 9(a) shows the fire shape at $t = 3$ hours, and the two starting locations: cell (107,115) on the north side of the fire, and cell (110,74) on the south side of the fire. Figure 9(b) and 9(c) show the fire suppression results starting from the two locations. The figure is better seen in color. In the figure, the green color means a forest cell is unburned, red means burning, black means burned, and yellow means the cell is suppressed by firefighting agents. Besides showing the contained fire, Figure 9(b) and 9(c) also show the “free burn” fire perimeter (i.e., if no fire suppression is applied) at the time when the fire is contained. The suppressed fire shape is displayed by the inner perimeter in yellow color, while the “free burn” fire shape is displayed by the outer perimeter in red color. The “free burn” fires shapes clearly reveal how much forest area is saved from burning by the firefighting agents.

**Figure 9. Fire suppression simulation results**

<table>
<thead>
<tr>
<th>Time (hour)</th>
<th>onTime_Schedule</th>
<th>pre_Schedule</th>
<th>FARSITE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Perimeter (kilometers)</td>
<td>Burned area (hectares)</td>
<td>Perimeter (kilometers)</td>
</tr>
<tr>
<td>1.0</td>
<td>2.30</td>
<td>31.05</td>
<td>2.30</td>
</tr>
<tr>
<td>2.0</td>
<td>4.15</td>
<td>118.62</td>
<td>4.15</td>
</tr>
<tr>
<td>3.0</td>
<td>5.47</td>
<td>209.70</td>
<td>5.47</td>
</tr>
<tr>
<td>4.0</td>
<td>7.99</td>
<td>442.08</td>
<td>7.99</td>
</tr>
<tr>
<td>5.0</td>
<td>10.16</td>
<td>710.91</td>
<td>10.16</td>
</tr>
<tr>
<td>6.0</td>
<td>11.32</td>
<td>892.44</td>
<td>11.32</td>
</tr>
</tbody>
</table>

Figure 9 also reveals that starting firefighting efforts from different locations will lead to different results including having different fire shapes, perimeters, burned areas, and fire
containment times. These differences are as a result of the dynamic nature of fire behavior (due to the dynamic wind speed and direction) and fire suppression efforts. When fire suppression is initiated from the north location shown in Figure 9(b), the agents fully contain the fire at time $t = 5.19$ hours and the perimeter of the suppressed fire is 10.30 kilometers and the burned area is 731.97 hectares. When the fire suppression begins from the south location as shown in Figure 9(c), the agents fully contain the fire at time $t = 5.68$ hours. The perimeter of the suppressed fire is 10.93 kilometers and the burned area is 830.70 hectares. It takes less time, and thus smaller fire perimeter and burned area, to start fire suppression from the north location. This can be explained by the fact that, in the fourth hour the fire spreads towards the north (see Figure 8). Thus starting fire suppression from the north prevents the fire from spreading rapidly. More simulation results on using different firefighting tactics are reported in [12].

6.2. Performance Measurement of the onTime_Schedule and pre_Schedule Models

This experiment focused on measuring the simulation performance of onTime-Schedule and pre-Schedule fire spread models. The aim of the experiment was twofold: 1) to compare the performance of the two models and to demonstrate that the pre-Schedule model has performance advantage over the onTime-Schedule model; 2) to show the key factors that affect the performance of the two models. We analyzed the performance of the models based on the relationships among the simulation cycles, the number of active cells (the cells that perform state transitions in the simulation cycles), and the execution time. The number of active cells is also referred to as activity in this paper. A description of these measurement metrics can be found in [43] for the interested reader.

The experiments were conducted on a Lenovo laptop with Intel Core(TM)2 CPU T7300 @ 2.00GHz, 0.97G memory, and Windows XP OS running DEVSJAVA version 3.0 with a heap-based simulation engine [43]. Simulation runs for both models using the same simulation configurations and input data. We ran simulations using a 300 × 300 cell space with constant wind speed and direction arbitrarily set at (9.66 kilometers/hour, 180 degrees). The forest cell size was set to 15 meters × 15 meters with both slope and aspect set to zero and fuel model 7 was used. The simulations were run for 40000 seconds (about 11 hours) simulation time.

The results of the simulations are plotted in Figure 10. The figures show the execution time, the activity and the simulation cycles for the two fire spread models. In the figure the total simulation time of 40000 seconds on the horizontal axis is divided into 20 intervals. Therefore, each interval represents a simulation time of 2000 seconds. Figure 10(a) shows the execution time, activity and the number of simulation cycles for pre-Schedule while Figure 10(b) shows the same measurements for onTime-Schedule. Figure 10(c) shows the ratios (onTime_Schedule to pre_Schedule) of the execution time, simulation cycles and activity between the two models.
Figures 10(a) and 10(b) show that the execution time has a positive relationship with the number of simulation cycles and active cells, i.e., the execution time increases with the increase of simulation cycles and active cells. Compared with the simulation cycle, the execution time follows more closely with the trend of the activity. This is because the execution time is mainly determined by the total number of active cells participating in the simulation. Compared to the onTime_Schedule model, the pre_Schedule model has less simulation cycles, less activity, but faster execution time. Figure 10(c) shows the ratios of these measurements between the two models. As can be seen, the pre-Schedule model reduces the simulation cycles by a factor of up to more than four times. The pre_Schedule model reduces the activity up to two times, which is not as large as the reduction of simulation cycles. This is because in the pre-Schedule model, when a cell sends out the eight messages in one simulation cycle, there is a total of nine active cells (the center cell plus the eight neighbor cells). But in the onTime_Schedule model, each cycle has only two active cells (the center cell plus a neighbor cell). Figure 10(c) shows that the ratio of the execution time follows closely with the ratio of the activity. It shows that the speedup of the pre_Schedule model over the onTime_Schedule model is up to two times.

To measure how frequent weather updates may affect the performance of the two fire spread models, we ran a new set of simulations using the same configurations as above but with weather updates for every 1 minute, 2 minutes, 5 minutes, 10 minutes, 30 minutes, and 60 minutes, respectively. To ensure that different simulations will give the same results and thus can be fairly compared, we fixed the wind direction and wind speed for each weather update in all the simulation runs. The ratios of the execution time, simulation cycles and activity at the end of the simulations between the two models are shown in Figure 11. From the figure we see that the ratio of simulation cycles is almost the same for different frequencies of weather updates. This is because even in the case of frequent weather updates, the added number of simulation cycles is relatively small compared to the total number of simulation cycles. However, both the activity ratio and execution time ratio decreases when the frequency of weather update increases. This is because each weather update causes extra message passing between a cell and its neighbors for the pre_Schedule model as explained in Section 4. Thus when there is frequent weather update, the speedup of the pre_Schedule model decreases slightly (from 1.96 under 60-minute weather updates, to 1.69 under 1-minute weather updates). Nevertheless, the decrease is not drastic. Even in case of weather updates every minute, the pre_Schedule model still has a speedup of 1.69. The results of this experiment show that the performance of the pre_Schedule model is better than that of the onTime_Schedule model in general.
6.3. Simulations using GIS data

In this experiment we ran both fire spread and fire suppression simulations using GIS data from a study area in East Texas District 12 of the Forest Texas Service. The simulations are based on a 200 × 200 cell space with a cell size of 30 meters × 30 meters. The wind speed and wind direction were randomly generated every 30 minutes: the wind speed was generated from uniform(16.09, 32.19) kilometers/hour and the wind direction generated from uniform(210, 300) degrees azimuth. The fire was arbitrarily ignited at cell (80, 100). Similar as before, we ran fire spread simulations using both the onTime_Schedule model and the pre_Schedule model in DEVS-FIRE. For validation purposes, we also ran the fire spread simulation in FIRSITE using the same input data.

Figure 12 shows the results of a six-hour fire spread simulation from the DEVS-FIRE models (Figure 12(a)) and from FARSITE (Figure 12(b)). To show the fire growth, the fire perimeters at the end of three hours are also displayed in the figure. The figure is best seen in color. Note that the onTime_Schedule DEVS-FIRE model and the pre_Schedule DEVS-FIRE model gave exactly the same results, thus are shown in the same figure in Figure 12(a). In Figure 12(a), the black color means burned out, red means burning, and all other different colors represent different fuel models of the area. The inner perimeter in red color represents the fire front at the end of three hours, and the outer perimeter in red color is the fire front at the end of six hours. Figure 12(b) shows the results from FARSITE (which has a different coloring schema for displaying the fuel models). The two perimeters in white color show the fire fronts at the end of three hours and six hours respectively. By comparing the two figures, one can see that the results obtained from DEVS-FIRE are consistent with those from FARSITE.
To quantitatively compare the results, Table 4 lists the recorded hourly fire perimeter and burned area for the six-hour fire spread simulation period. The table shows that the onTime_Schedule model and the pre_Schedule model give the same results. By comparing the results from DEVS-FIRE to those of FARSITE, one can see that at the end of six hours the burned area computed from DEVS-FIRE is about 12% higher than that from FARSITE. This is likely due to the fact that in DEVS-FIRE the burned area is computed cell by cell – even if a cell is just ignited, the whole cell is computed as part of the burned area. Table 4 shows that the perimeters computed from DEVS-FIRE and FARSITE are consistent in the first four hours. Starting from the fifth hour the perimeter from FARSITE becomes about 30% larger than that from DEVS-FIRE. This is mainly due to the two models' different methods of calculating perimeters. Based on the technical documentation of FARSITE [21], the fire perimeters calculated in FARSITE includes not only the outer perimeter but also the perimeters of the enclaves inside the outer perimeter. This is different from DEVS-FIRE that only computes the outer perimeter. In this experiment, several enclaves were formed because of the non-uniform fuel model and complex landscapes. The perimeters of these enclaves were calculated in FARSITE but not in DEVS-FIRE. This makes FARSITE give larger perimeter values in the last two hours of the simulation.

Figure 13 shows the same fire scenario but now with fire suppression using direct attack. In this simulation we used two firefighting agents: one builds the fireline clockwise and the other builds the fireline counter-clockwise. Each agent has production rate of 0.4 meters/second. The two agents were dispatched to the fire front at time \( t = 2.5 \) hours and at location cell (78, 99). The two agents were able to contain the fire at time \( t = 5.62 \) hours. The perimeter of the suppressed fire is 8.98 kilometers. This compares to 12.84 kilometers for the uncontained fire (Figure 12(a)). Similarly, the burned area is 245.25 hectares after containing the fire. This compares to 489.87 hectares if the fire is not contained. This experiment based on real GIS data demonstrates that DEVS-FIRE is a viable discrete event model for simulating realistic fire scenarios.
7. Conclusion
Wildfires continue to cause tremendous losses of natural resources, human lives and property every year and are expensive to contain. The ability to make predictions regarding when a given wildfire will reach a certain location, its intensity, and the effect of firefighting tactics on the fire, are crucial for wildfire response planning. Faster and accurate prediction of fire behavior and firefighting effects would help in effective fire containment planning and evacuation and potentially limit the destruction from wildfires. In this paper we give formal design specifications of fire spread and suppression agent models used in DEVS-FIRE and demonstrate the application of DEVS-FIRE to a real study area located in the US state of Texas and to artificially generated landscape, fuels and weather data. We develop a new method, called pre_Schedule, for scheduling ignition events of forest cells more efficiently than the original onTime_schedule event scheduling method used in DEVS-FIRE. Simulation results show the performance

Table 4. Perimeters and burned areas from fire spread simulations using GIS Data

<table>
<thead>
<tr>
<th>Time (hour)</th>
<th>onTime_Schedule</th>
<th>pre_Schedule</th>
<th>FARSITE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Perimeter (kilometers)</td>
<td>Burned area (hectares)</td>
<td>Perimeter (kilometers)</td>
</tr>
<tr>
<td>1.0</td>
<td>1.49</td>
<td>10.44</td>
<td>1.49</td>
</tr>
<tr>
<td>2.0</td>
<td>2.77</td>
<td>35.64</td>
<td>2.77</td>
</tr>
<tr>
<td>3.0</td>
<td>5.39</td>
<td>102.06</td>
<td>5.39</td>
</tr>
<tr>
<td>4.0</td>
<td>7.84</td>
<td>184.23</td>
<td>7.84</td>
</tr>
<tr>
<td>5.0</td>
<td>10.09</td>
<td>331.92</td>
<td>10.09</td>
</tr>
<tr>
<td>6.0</td>
<td>12.84</td>
<td>489.87</td>
<td>12.84</td>
</tr>
</tbody>
</table>

Figure 13. DEVS-FIRE Fire Suppression Simulations Using Real GIS data
improvement of the new method, and demonstrate the utility of DEVS-FIRE as a viable discrete event model for wildfire simulations. Future work includes the application of DEVS-FIRE to historical fire events to allow for full validation of the fire spread and fire suppression models.

8. Acknowledgements.
This research was supported in part by grants CNS-0841170, CNS-0720675 and CNS-0720470 from the National Science Foundation. The authors are grateful to Thomas Spencer and Curt Stripling of the US Texas Forest Service for providing the GIS data used in the experiments.

9. References


**Xiaolin Hu** is an Associate Professor in the Computer Science Department at Georgia State University, Atlanta, Georgia. He received his Ph.D. degree from the University of Arizona, M.S. degree from Chinese Academy of Sciences, and B.S. degree from Beijing Institute of Technology in 2004, 1999, and 1996 respectively. His research interests include modeling and simulation theory and application, agent and multi-agent systems, and complex systems science. Dr. Hu is a National Science Foundation (NSF) CAREER Award recipient.
Yi Sun is a Senior Scientist in Sigma-Aldrich Bioinformatics Group. She got her Ph.D. degree in Computer Science Department at Georgia State University. Her research interests include performance improvement of discrete event system, modeling and simulation, and text mining.

Lewis Ntaimo is an associate professor of Industrial and Systems Engineering at Texas A&M University, College Station, Texas. He received his Ph.D., M.S. and B.S. degrees from the University of Arizona in 2004, 2000 and 1998, respectively. His research interests include stochastic programming, systems modeling and simulation and their application to wildfire management, wind farm operations and maintenance, and healthcare problems.