Improved Cell-DEVS Models for Fire Spreading Analysis

Matthew MacLeod, Rachid Chreyh , Gabriel Wainer

Dept. of Systems and Computer Engineering
Carleton University. 1125 Colonel By Dr. Ottawa, ON. K1S 5B6. Canada
{mmacleod, rchreyh, gwainer}@sce.carleton.ca

Abstract. The spread of fire is a complex phenomenon that many have tried to study over the years. As one can imagine, the spread of fire depends on many different variables such as the material being burned, the geography of the area, and the weather. Here, we will show a Cell-DEVS model based on an existing model to speed up the simulation. We use Quantized DEVS and 'dead reckoning' to vary the length of the time steps taken by each cell. This paper explores how using one or both of the methods together can sometimes decrease the number of messages sent (and hence the execution time).

1 Introduction

The spread of fire is a complex phenomenon that many have tried to study over the years. As one can imagine, the spread of fire depends on many different variables such as the material being burned, the geography of the area, and the weather. It has been determined that finding an analytical solution for mathematical models of fire spread is almost impossible, and therefore many have looked to simulation as an attractive alternative. Simulations have been found that accurately represent the way in which fire spreads, and are now generally the preferred solution for predicting the behavior of fire. This goal of predicting fire behavior is important to firefighters (for example), because having a tool that is able to predict where the fire will be and how it will move will enable them to better plan strategies to control the fire quickly and safely. An aspect of great importance in such a tool is that it has to be able to predict the fire behavior at the very minimum faster than the fire itself moves, preferably much faster. In a real life situation, if we want to use a tool to help us predict how the fire will spread we have to be confident that it will give us a reasonable result on the order of minutes. Otherwise, valuable time will be lost as the fire spreads further.

The complexity of how fire spreads has made it the target of study in the modeling and simulation field. Mathematical models for this phenomenon are too complex to give an analytical solution; therefore, simulations have been used to study it and there has been some success in predicting fire behavior using cellular models. Although Cellular Automata have been used in defining the kind of models of our interest [1, 2, 3] CA poses precision constraints and extra computation time. Cell-DEVS [4] was proposed to solve these problems by defining cell spaces as DEVS (Discrete Events

systems Specifications) models [5]. Using Cell-DEVS, a cell space is described as a discrete event model in which explicit delays can be used to model accurately the cell timing properties. CD++ [6] allows implementing DEVS and Cell-DEVS models, while providing remote access to a high performance DEVS simulation server [4].

Here, we will show improvement to previously developed fire spreading model using Cell-DEVS [7]. In this model, the physical area of interest is divided into cells, with each cell exhibiting the same behavior. The model uses a simple set of equations to determine the temperature of each cell at regular time intervals. The temperature of a non-burning cell is an averaging function of its own temperature and that of its neighbors. Once ignited the temperature of burning cells, on the other hand, is also a function of time – the cell's temperature increases to a peak and then falls back down, modeling the exhaustion of fuel in the cell.

As all the cells are activated on every timestep, performance can be poor (especially for large models). Here, we explore modifications to the existing model to speed up the simulation. This can be done in several ways, including both using Quantized DEVS (Q-DEVS) to quantize the model output [8], and using 'dead reckoning' to vary the length of the time steps taken by each cell. By using Q-DEVS we will be able to reduce the number of messages exchanged between the cells so that messages are only sent when the output of a cell passes a quantization threshold. The use of dead reckoning contrasts with the traditional method of using an equation (fit from experimental data) that determines the temperature of a cell as a function of time. In the dead reckoning approach, we find an equation (or set of equations) that determines the time the next quanta will be passed as a function of the current temperature. In other words, we will only update the temperature of the cells at the quantum boundaries, by using an equation that determines at what time the next quantum will be reached. This paper explores how using one or both of the methods together can sometimes decrease the number of messages sent (and hence the execution time).

2 Background

A real system modeled using the DEVS formalism [5], can be described as a hierarchy of submodels. Each of them can be behavioral (atomic) or structural (coupled). A DEVS atomic model is described as $M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D >$. The interface is composed of input and output ports (X, Y) to communicate with other models. The input external events (those coming from other models) are received in input ports. The model specification defines the behavior of the external transition function under such inputs (δ_{ext}). Each state has an associated duration time (D). When this time is consumed, the output function (λ) is triggered, and then the internal transition function (δ_{int}) is activated to produce internal state changes. A DEVS coupled model is defined as: $CM = < I, X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\} >$. Here $\boldsymbol{X}$ is the set of input events, and $\boldsymbol{Y}$ is the set of output events. **D** is an index of components, and for each $i \in D$, $\mathbf{M_i}$ is a basic DEVS model (atomic or coupled). $\mathbf{I_i}$ is the set of influencees of model *i*. For each $j \in I_i$, $\mathbf{Z_{ij}}$ is the *i* to *j* translation function. Each coupled model consists of a set of basic models connected through the input/output ports. The influencees of a model will determine to which models one send the outputs. The translation function is in

charge of translating outputs of a model into inputs for the others. To do so, an index of influencees is created for each model (I_i). For every *j* in this index, outputs of the model M_i are connected to inputs in the model M_j.

Cell-DEVS allows defining complex cellular models that can be integrated with other DEVS. Each cell of a space is defined as an atomic DEVS. Transport and inertial delays allow defining timing behavior of each cell in an explicit and simple fashion. Cell-DEVS atomic models can be specified as TDC = < *X*, *Y*, S, N, delay, d, δ_{int}, δ_{ext}, τ, λ, D >. ***X*** represents the external input events, ***Y*** the external outputs. **S** is the cell state definition, and **N** is the set of input events. **Delay** defines the kind of delay for the cell, and **d** its duration. Each cell uses a set of N input values to compute the future state using the function t. These values come from the neighborhood or other DEVS models, and they are received through the model interface. A delay function can be associated with each cell, allowing deferring the outputs. A Cell-DEVS coupled model is defined by GCC = < X_{list}, Y_{list}, *X*, *Y*, n, {t_1,...,t_n}, N, C, B, Z >. Here, $\mathbf{Y_{list}}$ is an output coupling list, $\mathbf{X_{list}}$ is an input coupling list. ***X*** are the external input events and ***Y*** the external outputs. The **n** value defines the dimension of the cell space, {$\mathbf{t_1,...,t_n}$} is the number of cells in each dimension, and **N** is the neighborhood set. **C** is the cell space, **B** is the set of border cells and **Z** the translation function. The cell space defined by this specification is a coupled model composed of an array of atomic cells. Each of them is connected to the cells defined by the neighborhood. As the cell space is finite, the borders should have a different behavior than the remaining cells. Finally, the Z function allows one to define the internal and external coupling of cells in the model. This function translates the outputs of m-eth output port in cell *Cij* into values for the *m*-eth input port of cell *Ckl*. The input/output coupling lists can be used to transfer data with other models.

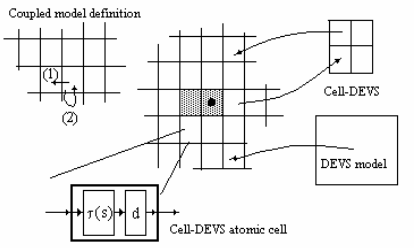

Fig. 1. Informal definition of a Cell-DEVS model

Recently, DEVS has been used recently for continuous systems simulation. In most cases, the techniques are based on Q-DEVS [8], whose main idea is to represent continuous signals by the crossing of an equal spaced set of boundaries. This approach requires a fundamental shift in thinking about the system as a whole. Instead of determining what value a dependant variable will have (its state) at a given time, we must determine at what time a dependant variable will enter a given state.

In [7], we showed how Cell-DEVS and CD++ could be used to model fire spread. Below is a simplified diagram of the temperature curve used in the original model. The temperature curve is divided into four stages, and, at any given instant, each cell

in the model will be in one of these stages. The first is the inactive stage, when a cell has no neighbors with a temperature higher than the ambient temperature (T_a). The second is the unburned stage in which the temperature of the cell is increasing due to heat from the neighboring cells; during this stage, the cell's temperature is between the ambient temperature and the ignition temperature (300ºC). The third is the burning stage in which the cell has reached the ignition temperature and fuel in the cell starts to burn. The cell's temperature increases until it reaches a peak temperature, and then begins to fall back down to 60ºC. The fourth and final stage is the burned stage in which a burning cell's temperature has fallen below 60ºC. Because it has exhausted its fuel, it can no longer reignite and it is considered inactive.

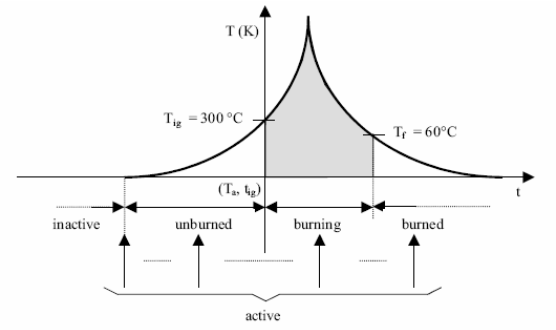

Fig. 2. Simplified Temperature curve [1]

The Cell-DEVS model contains two planes of cells. The first represents the fire spread itself, in which each cell calculates its temperature. The additional plane is used to store the ignition times for the cells (this can be considered merely a state variable of the true cells of interest). The cells in the ignition temperature plane have a simple rule – record the current simulation time when the corresponding cell in the fire spread plane reaches the ignition temperature. The cells in the fire spread plane compute fire spreading. When a cell is in the unburned phase, its temperature is calculated as the weighted average of the current cell's temperature with its neighbors' temperatures. When a cell is in the burning phase, its temperature is calculated as an exponential function of time that describes the fire's behavior, taking into account the same weighted average. In the other two phases (inactive and burned), the cell's temperature does not change, and thus these cells remain in the passive state (the inactive cells will of course respond to any temperature changes in their neighborhood, so they may eventually ignite). One of the advantages of using Cell-DEVS is that all cells in the inactive or burned phase will remain passive, and thus the calculations will be confined to the fire front. This saves on the simulation's execution time.

The reason for the slow execution time of the simulation is mainly due to the high number of messages being exchanged between cells. In the simplest version of the model, each cell in the unburned or burning phase will update its temperature once every millisecond and will as a result send messages to its neighbors. To remedy this problem a solution must be found that decreases the number of messages exchanged.

3 Quantizing the Fire Spread Cell-DEVS Model

In order to increase the speed of the simulation, we propose two simplifications. Firstly, if we are able to keep the unburned cells completely in the passive state until they reach the ignition temperature, we would reduce the number of cells that send out messages to their neighbors. The second is to use quantization to reduce the number of messages exchanged among the cells. This is explained in more detail below.

The model we propose will still have the same four phases of inactive, unburned, burning and burned. The main difference, however, is that in our model, in addition to the inactive and burned phase, the unburned cells also remain passive. We have found that a cell in the unburned phase will reach the ignition temperature when: a) one of its neighbors has reached a temperature above 650 °K , or b) two of its neighbors have reached a temperature above 474 °K. Using this result, we are able to keep all cells in the passive state until their neighbors meet these conditions, rather than constantly calculating weighted averages. When running the original model, we found that all cells more or less exhibit the same temperature curve when they are in the burning phase, implying that the temperatures of neighboring cells do not have a big impact on a burning cell's temperature. As a result, when a cell reaches the ignition temperature it can calculate its temperature by following only the temperature curve determined from experimentation, rather than taking into account its neighbors at every step. By doing this we have restricted the majority of calculations to the cells in the burning phase, and removed the need for messages from the neighbors in many cases.

The other method of reducing the messaging between cells is quantization [8, 9]. There are two quantization ideas that we have implemented in our model. The first is to use Q-DEVS to automatically quantize the model. In Q-DEVS, all cells in the model have a fixed quantum size and each cell has a quantizer. Each cell will only send output to its neighbors if its temperature has exceeded the next quantum threshold. The quantizer acts as the detector that decides when a threshold has been crossed, and it sends out the output only in that case. By implementing quantization as described here, the number of messages exchanged between cells will be reduced, thus increasing the speed of the simulation. However, the accuracy of the simulation will also be reduced. The key is to select a quantum size that gives a good performance increase for a small reduction in accuracy. The second method of quantization involves calculating time based on temperature, rather than temperature as a function of time. The first task was to find the inverse of the temperature curve for a typical cell [10]. Given such a function **f(T)**, we can calculate the amount of time it will take to reach the next quantum level as a simple difference $\mathbf{f(T_2)-f(T_1)}$. By doing this, cells can be kept quiescent until they reach the next quantum threshold. After this time has passed, they will wake up, calculate the next time at which they will cross a threshold, and return to the quiescent state. This saves unnecessary calculation, as cells will only become active when a significant change in temperature occurs. To obtain the required function **f**, we started with a typical temperature curve of a cell in the burning phase.

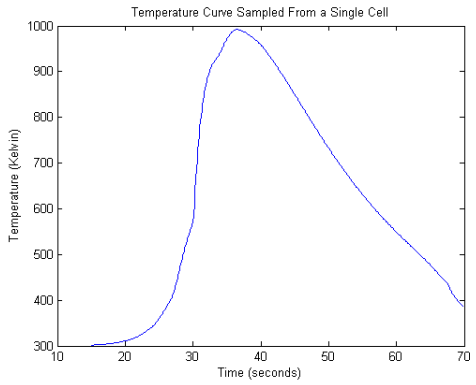

Fig. 3. Burning Cell's Temperature Curve

This function fails the horizontal line test, and therefore is not directly invertible. So, it must be divided into increasing and decreasing components, giving us two invertible functions. A state variable can then be used to choose between them during execution. We found two functions that approximate the two curves reasonably well. Note that for the scope of this paper we are not overly concerned about the accuracy of these functions, as the focus of this study is to analyze the performance of our proposed model, which if successful could be refined by fire experts to the desired level of accuracy. Collecting data for this version of the model would also be more efficient, as instead of sampling every cell of the real model every millisecond, samples would only have to be recorded at threshold crossings. This could potentially save much data storage, and make better use of network bandwidth in the test bed.

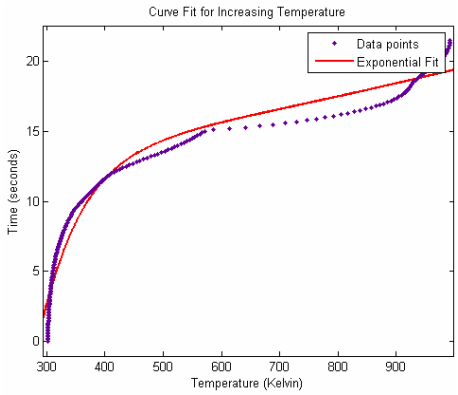

Fig. 4. Inverted increasing temperature function

The fit for the increasing temperature portion of the curve is shown above in figure 4. It uses a sum of two exponential functions:

$$f(T) = 11.56 * e^{0.0005187*T} - 784.7 * e^{-0.01423*T} \quad \textbf{(1)}$$

Where T is the temperature in degrees Kelvin. This will be used for cells in there "Burning Up" phase, i.e. the phase in which they are burning with increasing temperature. Similarly, the decreasing portion (or "Burning Down" phase) is fit with the linear function:

$$f(T) = 0.052 * T \quad \textbf{(2)}$$

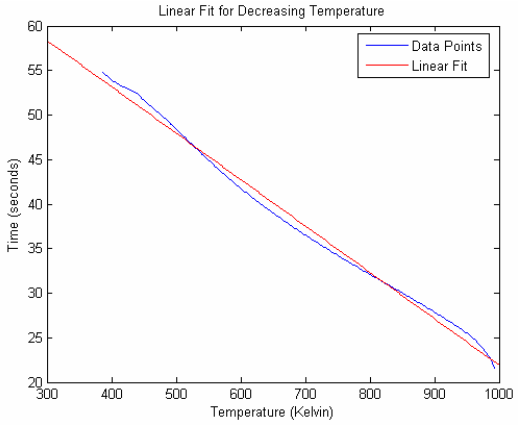

Fig. 5. Inverted decreasing temperature function

There are a few things to notice: although we found higher order functions that fit the functions with greater accuracy, these had issues related to local maxima and minima. As time is necessarily monotonically increasing, it is not acceptable for the functions to decrease at any point. Therefore, any function of time must also pass the horizontal line test, and therefore be invertible. As a consequence, we cannot model any up and down fluctuations in temperature within any of our piecewise curves, and we will need a new state for any change in direction. This restriction causes the obtained functions to be linear or near-linear in most regions (the exponential curve shown has two nearly linear regions joined by a knee). These functions will be used to develop the time advance portion of the model rules, which is easily implemented in Cell-DEVS delay functions.

3.1 Cell-DEVS model definition

In our Cell-DEVS model, temperatures remain on the first plane, and ignition times are in the second plane, and a third plane stores information about each cell that will help us determine which rule to apply. The second plane has a value of 0 by default, and the following values as indicated below:

- -100: If the temperature of the cell is between 301 and 474, meaning it is burning but not hot enough to cause a neighbor to ignite.

- -200: If the temperature of the cell is between 474 and 650, meaning it is burning and hot enough to cause a neighbor to ignite if another neighbor is in this state.
- -300: If the temperature of the cell is above 650, meaning it is burning and is by itself hot enough to cause a neighbor to ignite.
- -400: If the temperature has reached the peak temperature (992) from our data curve, and is now starting to burn with a decreasing temperature.
- -500: When the cell has burned out.

The neighborhood is as follows: Looking at a cell in the first plane (the fire spread plane) each cell has its corresponding cell in the second plane (the supporting info plane) and the Von Neumann neighborhood of that cell as its neighbors. The local computing function described below:

- **A cell whose neighbor in above plane has values of -100, -200 or -300:** Cells in the Burning Up phase; they have not yet reached their peak temperature. These cells will calculate (according to the burning up function) the time delay after which they should increment their temperature by the quantum amount and then sleep for this time
- **A cell whose neighbor in other plane has values of - 400:** Cells in the Burning Down phase; they are still burning but have reached their peak temperature and their temperature is falling from here on in. These cells will calculate (according to the burning down function) the time delay after which they should decrement their temperature and then sleep for this time
- **A cell whose value is 0 and its neighbor in other plane has a value between 301 and 474:** cells in the "Supporting Info" Plane. After a short time delay, they are to get a value of -100 indicating that their corresponding cell has ignited but is still below 474 °K.
- **A cell whose value is 0 or -100, and its neighbor in other plane has a value > 474:** cells are in the "Supporting Info" Plane. After a short time delay, they are to get a value of -200 indicating that their corresponding cell has ignited and has reached 474 °K. Two of these cells can cause a neighbor to ignite.
- **A cell whose value is 0 or -200, and its neighbor in other plane has a value > 650:** cells are in the "Supporting Info" Plane. After a short time delay, they are to get a value of -300 indicating that their corresponding cell has ignited and has reached 650 °K. This cell alone can cause a neighbor to ignite.
- **A cell whose value is -300 and its neighbor in other plane has a value > 992:** cells in the "Supporting Info" Plane. After a short time delay, they are to get a value of -400 indicating that their corresponding cell has just reached the peak temperature and should use the burning down equation.
- **A cell whose value is -400 and its neighbor in other plane has a value < 332:** cells in the "Supporting Info" Plane. After a short time delay, they are to get a value of -500 indicating that their corresponding cell has Burned out.
- **Ignition Rules:** A cell in the "fire spread plane" that has not ignited yet (i.e. has a value of 300 °K) will ignite if at least two of its neighbors have a value of -200 or at least one neighbor with a value of -300. The cell will ignite by being assigned a temperate of 301 °K.

4 Simulation Results

We ran the fire spread simulation in CD++ using our proposed model in comparison with the original model. The initial values used represented a line ignition scenario (i.e. the initial burning cells are in a straight line).

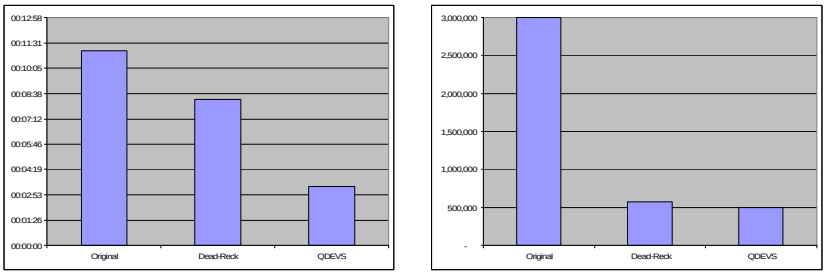

Fig. 6. (a) Execution times (b) Number of messages passed.

As we had noted earlier the aim of our model is to reduce the execution time of the simulation without reducing the accuracy. Our model successfully reduced the simulation execution time. The following figure shows the execution time and the number of messages involved in the simulation of the original model and the two new versions here presented. As we can see, the reduction in the number of messages involved is exponential, thus providing an excellent technique for execution in parallel/distributed environments, in which message passing between the components are the cause of most of the execution time of the model. Gains were greater when only a few cells were initially activated.

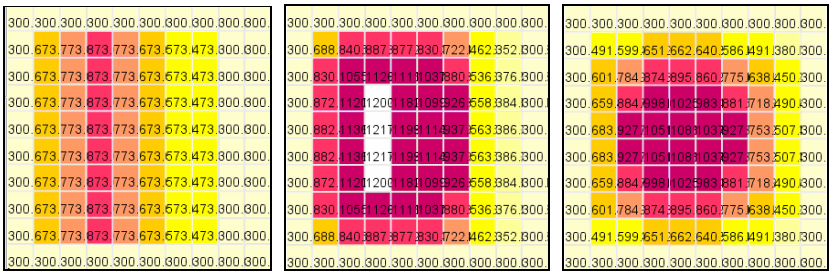

Fig. 7. Execution results at 0, 300 and 1000 time units

The previous diagrams depict the results we obtained using our model, which were similar to the original model. The cumulative average weighted error for the simulations was below 2%, following the trend presented in multiple studies on DEVS quantization [8, 9].

5 Conclusion

Our study concurred that the original model was too slow in execution time, due to the large number of messages exchanged among the cells in the model. Our techniques were able to improve performance. Our first proposed model used quantization to reduce the number of messages between the cells and thus increase the speed of the simulation. Quantization was implemented by calculating the time steps between temperatures, instead of the temperatures at time steps. This achieved the goal of keeping all cells "asleep" until a significant event takes place. The effect of neighboring cells was ignored, as in previous test runs all cells were seen to develop similarly. Another modification was to keep cells in the unburned state "passive" until they are seen to reach the ignition temperature. This increased performance, but had problems with accuracy, and required some prior knowledge of how the fire would develop to obtain good equations. We found that the general direction and speed of fire spread was maintained by our model, although some finer details such as peak temperatures and temperatures of cells at the fire front lost accuracy.

References

1. Rothermel, R.C.: A Mathematical Model for Predicting Fire Spread in Wasteland Fuels. USDA Forestry Service Research Paper, INT-115 (1972)
2. Barros, F., Ball, G.L.: Fire Modelling Using Dynamic Structure Cellular Automata. In: III International Conference On Forest Fire Research. 14th Conference on Fire and Forest Meteorology. Luso, Portugal (1998)
3. Balbi, J.; Santoni, P.; Dupuy, J.: Dynamic Modelling of Fire Spread Across a Fuel Bed. In: International Journal of Wasteland Fire. Vol. 9 (1999) 275-284
4. Wainer, G.; Giambiasi, N.: N-dimensional Cell-DEVS. In: Discrete Events Systems: Theory and Applications. Kluwer, Vol. 12, No. 1 (2002) 135-157
5. Zeigler, B.; Kim, T.; Praehofer, H.: Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press (2000)
6. Wainer, G: CD++: a Toolkit to Define Discrete-Event Models. In: Software, Practice and Experience. Wiley, Vol. 32, No.3 (2002) 1261-1306
7. Muzy, A., Innocenti, E., Aiello, A., Santucci, J., Wainer, G.: Specification of Discrete Event Models for Fire Spreading. In: Transactions of the Society for Modeling and Simulation International, Vol.81, Issue 2 (2005)
8. Wainer, G., Zeigler, B.: Experimental Results of Timed Cell-DEVS Quantization. In: Proceedings of AIS'2000. Tucson, Arizona. USA (2000)
9. Zeigler, B.: DEVS Theory of Quantization. DARPA Contract N6133997K-007: ECE Dept., University of Arizona, Tucson, AZ (1998)
10. Lin, K.-C.: Dead Reckoning and Distributed Interactive Simulation. In: Distributed Interactive Simulation Systems for Simulation and Training in the Aerospace Environment; Proceedings of the Conference. Orlando, FL, USA (1995) 16-36.