ITM Web of Conferences **1**, 02003 (2013)
DOI: 10.1051/itmconf/ 20130102003

Randomizing activity in fire spreading simulations

Eric Innocenti[1,a] and Dominique Cancellieri[1]

[1]UMR CNRS 6134, University of Corsica Campus Grimaldi, 20250 Corte, France

Abstract. In this paper, we suggest the use of activity paradigm to model the uncertain and imprecise nature of the local dynamics in fire spreading simulations. Formerly based on a very simple set of rules, Cellular Automata (CA) models are remarkable in their ability to easily simulate fire spreading behaviour. A new algorithm to track activity concept in a (CA) model is presented. It is called random-activity and it combines activity and stochastic precepts. This algorithm enhances the realism of the spatial simulations. It resolves the problem of the spurious symmetries of the square lattice usually encountered which affect negatively outputs. Experiment shows that the approach enhances fire spreading simulations which are based on local description of the fire dynamics with general descriptive laws. A comparison between classical and random-activity algorithms is performed. The random-activity algorithm proposed outperforms the classical algorithm in terms of graphical restitutions' realism. It might be used to develop fire risk-management tools. A new way is presented to combine activity concept and stochastic modelling in a (CA) model.

1 Introduction

In many countries, forest fires are the cause of numerous and irreparable damages with deep ecological and socio-economic impacts. What happened in Corsica island in June 2009 was the worst damage of the last 30 years. More than 6 000 hectares of forest, corsican scrub and residential areas burnt in a few days. Thus, there is a need for designing and developing fire risk-management tools since those phenomena occur more and more. However, due to the chemical reactions intricacy and to the physical processes involved, the fire spreading modelling usually needs many equations. Such equations-based models are clearly not yet adapted to conceive risk-management tools. Consequently, we must simplify the phenomenon's description while remaining reasonably accurate in the representation of the reality. The models used must be able to quickly predict fire spreading in space and time, for both preventive and operational policies. That's why, the theory of Modelling and Simulation [1] is usually applied to describe the physical mechanisms of fire spreading systems. As discussed in literature, three main categories of (CA) modelling strategies are commonly used in this context [2, 3]: (1) physics-based or physical, (2) statistical, and (3) empirical or semi-empirical.
(1) The strategy based on physics consists in using more or less accurate descriptive laws that rule fluid mechanics, combustion and heat transfer, chemical and physical processes [4, 5, 6].
(2) The statistical strategy is only based on statistical correlations of a given experimental data set, and there are no physical laws [7, 8, 9].

[a] Corresponding author: ino@univ-corse.fr.

(3) The empirical strategy consists in using models mixing statistical correlations extracted from experiments, and simple general and theoretical expressions complete through experimentations [30].
In this paper, we are going to focus on the first strategy which is usually associated to (CA) in software applications [10, 11, 12].
The (CA) were first introduced by Von Neumann in 1966 to capture essential features of complex systems, where global behaviour arises from the collective effect of simple components locally interacting [13, 14]. Usually, the (CA) implements descriptive laws that rule local processes in transition functions. They use states to help expression of physical rules, i.e., the descriptive adjectives of the attributes used to model the system observed. Many works highlight that (CA) are appropriate to model fire spreading phenomenon [10, 11, 12, 15, and 16]. In this kind of models, a common improvement consists in restricting computations on the active elements, using the activity-tracking paradigm [17, 18]. Nevertheless, (CA) models also produce many spurious regular shapes, which greatly affect temporal fire patterns on output graphical restitutions, especially on flat areas [10, 19, and 20].

In this article, we present a new way to implement activity-tracking algorithm in (CA) models. We used pseudo-random numbers and activity paradigm in a new way called: "random-activity". We observe the effect of our approach in fire spreading simulations. The (CA) model implemented consists in a physics-based strategy which describes the local processes state's trajectory of fuel degradation (temperature evolution vs. mass loss). The results show that the random-activity proposed is adapted to this modelling strategy. It leads to realistic output restitutions in spite of simplifications. An object framework in Python is designed for investigating this new activity-based algorithm. The experiment shows that we can simulate in a quite adequate manner the (CA) model evolution in space and time. We potentially use it to develop a fire risk-management tool for real landscapes simulations.
In the next section, we briefly present the main paradigms used in our approach: (CA) models, (CA) stochastic modelling, and the activity concept. In a third part, we formulate a (CA) model with activity rules, and we describe the simulation algorithm dedicated to drive this activity. The object framework developed is also presented. The fourth part is dedicated to the presentation of the new random-activity concept. Pseudo-random numbers and activity precepts are then both combined in a new random-activity algorithm. It is used to randomize activity in (CA) models. Then, we describe in a fifth part, the new overall simulation algorithm, before discussing some numerical results in a sixth part.

2 Backgrounds

2.1 CA-based modelling precepts

The *(CA)* models have increasingly become important in spatial simulations as evolved forms of classical *(CA)* structures [21, 22]. In its basic description, a *(CA)* model of a system is an object structure divided into a set of square areas named cells [23]. Each cell interacts with a group of near others, belonging to predefined neighbourhood patterns through a simple set of rules.

Typically, a *(CA)* model consists in defining: Z^d, a grid of cells, S , a set of cell's state, N a neighbourhood pattern, and a transition rule δ. The transition rule describes how the cell's state considered changes from one S^t state to S^{t+1} next state, through the main simulation algorithm at each time step. A global clock is used to compute the next simulation time. Between two consecutive time steps, the transition rule expresses how each cell's state of the N neighbourhood influences the $S_{i,j}^{t+1}$ next cell's state considered at t time (cf. figure 1). Formally, a d-dimensional finite *(CA)* model is a structure:

$$CA = \langle Z^d, S, N, \delta \rangle$$

where:

- $Z^d = \{(i,j), 0 \leq i \leq h-1, 0 \leq j \leq w-1\}$, is the *d*-dimensional cellular space of w x h identical cells;
- *S* the finite states set of a (i,j) cell at *t* denoted by $S_{i,j}^t$;
- *N* the finite ordered subset of the (i,j) neighbour cells, at *t* denoted by $N_{i,j}^t = \{(i+\alpha_1, j+\beta_1), \ldots, (i+\alpha_k, j+\beta_k), k\ \varepsilon N^{*+}\}$
- δ*: S x N*, the local transition function in charge of updating the cell's states.

In spatial modelling, the neighbour cells are the spatial region from which the transition rule is allowed to search values to compute the next state. Usually, diverse neighbourhoods could be implemented. The most famous are: (1) the *von Neumann*, (2) the *Moore* and (3) the *extended Moore*, a Moore extension using more than one crown around the *(i,j)* cell executing the transition (cf. figure 1).

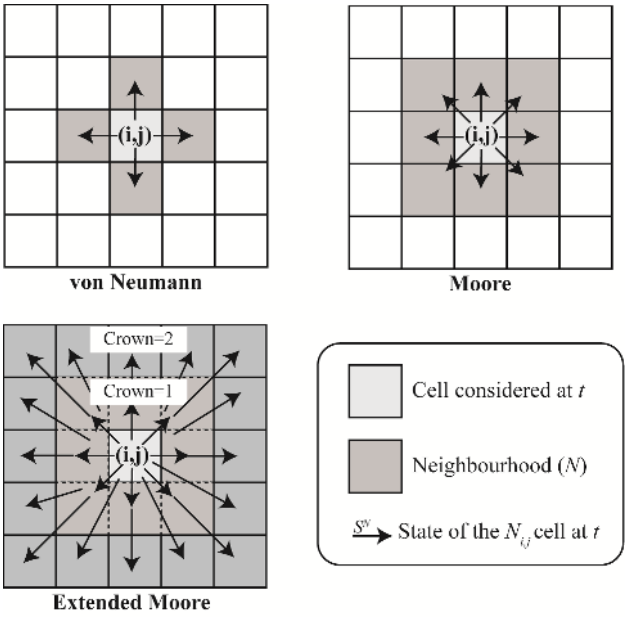

Figure 1. Generic representation of a cell neighbourhood in a (CA) model, and the main neighbourhoods used in spatial simulations.

During the simulation, the interactions take place in Δt discrete time steps. At any time period, each cell state is estimated considering the states of the neighbouring cells. This algorithm is continuously repeated in a recurrent self-reproductive mechanism in charge of calling the δ transition rule. Thus, the phenomenon expansion is observed through a bottom up approach (local level/global level). This last characteristic makes the *(CA)* a quite appropriate data structure to model spatial systems (spatial models). Each cell of the cellular space follows a deterministic *Discrete State Trajectory* (cf. figure 2).

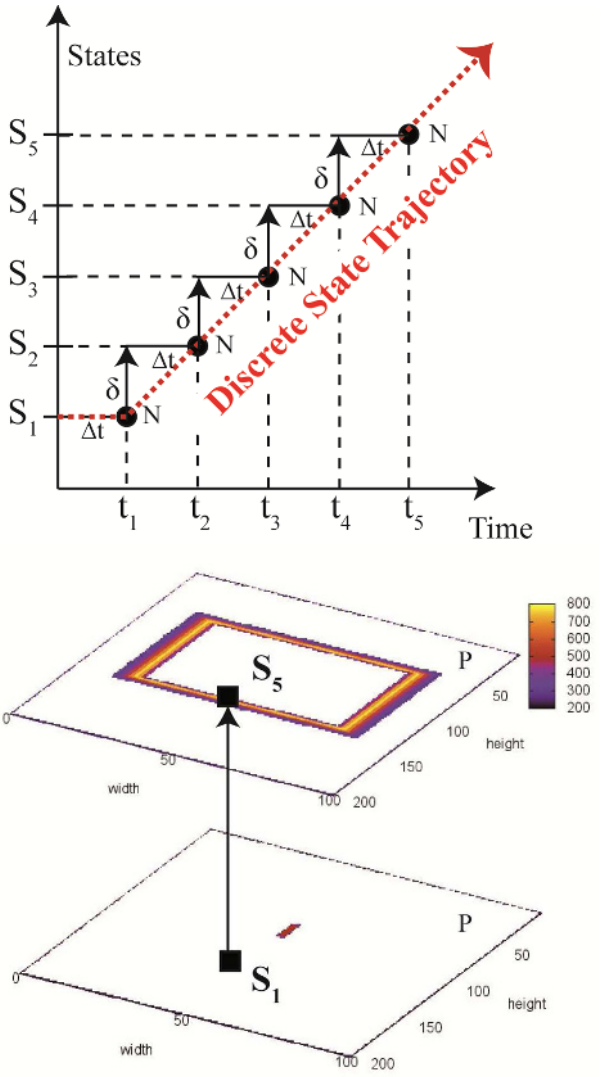

Figure 2. Example of a Discrete State Trajectory of a cell in a *(CA)* model.

The δ transition function explains how the $S_{i,j}^{t}$ cell's state located at the (i,j) place in the Z^{d} cellular space will evolve, according to the $N_{i,j}^{t}$ neighbourhood surrounding it.

While basic *(CA)* formulation is suitable for modelling spatial phenomena, in fire spreading simulations, this model usually needs some refinements to be more realistic and efficient. That's why recent developments use transition rules that are not restricted to the deterministic forms. When the system complexity implies uncertainty, the transition function's definition implements a non deterministic transition rule, i.e., a stochastic transition [24, 25, and 26].

2.2 Stochastic precepts

During these last years, in the areas of Modelling and Simulation, the literature on uncertainty has considerably grown. Many works have showed that to combine both pseudo-random numbers and *(CA)* models provides a proper foundation to efficiently simulate fire spreading [28].

In this sense, we consider the fire behaviour resulting from many mechanisms of local reactions (pyrolysis, local turbulences, heat transfer, etc.), integrating uncertainty. That's why it is difficult to

assign a discrete state trajectory skilled to reliably describe the local situation (i.e. an analytical expression). In this situation the combination of both *(CA)* models and pseudo-random numbers seem to be a solution [26]. Indeed, the presence of uncertainty at local level avoids us to describe all the local reactions phases, which are difficult to model, hard to implement, and time consuming in software applications. These new kinds of *(CA)* models which integrate uncertainty clearly seem to be suitable to conceive simulation tools for preventive and operational policies [11].

The *(CA)* models integrating uncertainty contain a certain amount of randomness in their transitions. They have many possible states that the transition can enter at each time step. A classical stochastic *(CA)* model is non deterministic in the sense that the next S_{t+1} state cannot be unequivocally predicted between two consecutive time steps (cf. figure 3). Then, the discrete state trajectory is not deterministic, as illustrated on figure 3.

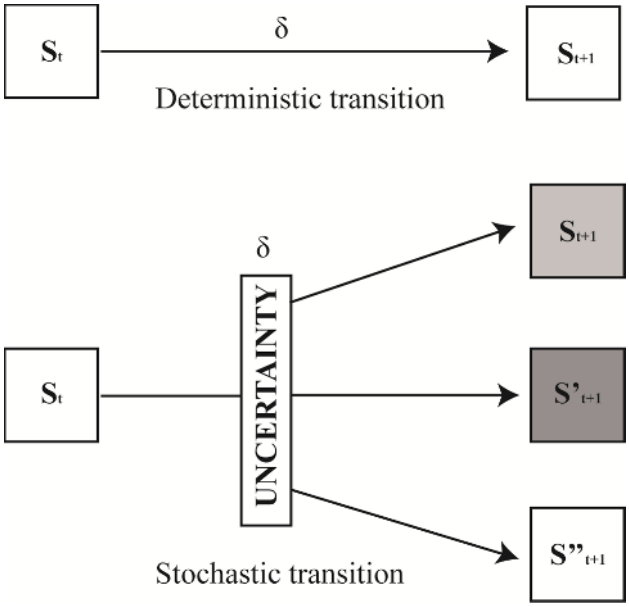

Figure 3. Deterministic vs. stochastic transitions in CA-models.

In physics-based modelling strategy, it is usually wise to use deterministic transitions. On the other hand, the use of stochastic precepts seems to allow a more accurate adequacy of the outputs with the real system observed. In this paper, we apply the stochastic precepts to describe the *activity rule* that relies on the activity precepts. The uncertainty is introduced during the propagation phase, which is described thanks to an *activity rule*. Using this technique, we allow more realistic simulations. The next section presents the activity precepts.

2.3 Activity precepts

In discrete event simulation, the process or event in charge of updating the model's state take place at the location of the activity. In a *(CA)* model, the activity is measured at global level, from a set of *active cells* and a common optimization consists in focusing the state transitions on it. The active cells are then defined as the components of the *(CA)* model that at time *t* will possibly change state, or will definitely be left unchanged between two consecutive transitions [17, 29]. The local level informs the global level on the model activity. The active cells which are referenced in an adapted data structure execute their local δ transition function. This technique comes from the activity tracking paradigm, and it is suitable to efficiency enhance *(CA)* model simulations [18, 30].

Technically, the implementation of the activity in *(CA)* models consists in indexing active cells in a data set structure (as linked list) to restrain the computations in the main simulation loop. At each time step, these active cells are scanned and their transition function δ is executed to calculate the next $S_{i,j}^{t+1}$ active states. A second copy of the data set is used to save the next states. It holds the next local states until all are computed. After all the next active cells' states are known, they constitute, with the inactive ones, the next global new state of the *(CA)* model. Then, the global simulation clock goes forward to the next simulation time (cf. figure 4).

The scan activity in a *(CA)* model relies on two main steps:

- Firstly, the active cells are scanned and their transition functions δ are executed to compute S_{t+1}. We call this phase: *"transition phase"*. The cells becoming inactive are removed from the set, according to the *transition rule* which implements the state trajectory of the local phenomenon;
- Secondly, at the same iteration, the inactive *N* neighbouring cells of the active cell are questioned to determine if they will become active at *t+1*. We call this phase *propagation phase*. The neighbouring cells becoming active are added to the set of active cells.

The figure 4 hereafter is an illustration of the activity cycle algorithm in the *(CA)* model implemented.

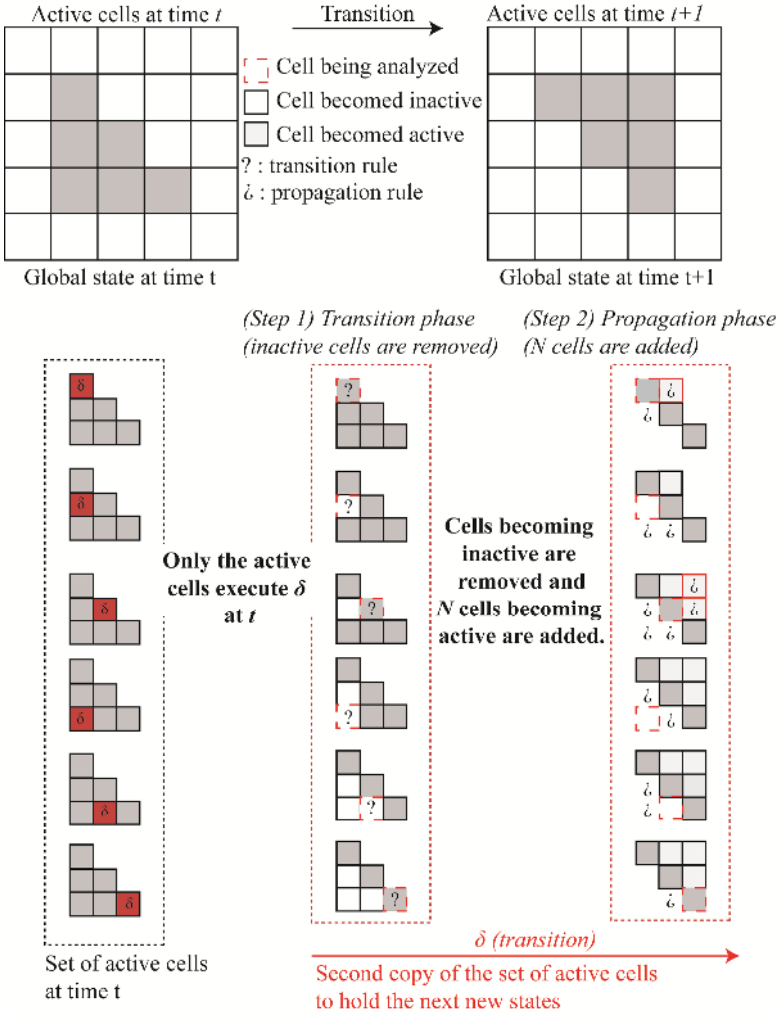

Figure 4. Algorithm's activity cycle in a *(CA)* model.

3 Activity formulation

3.1 Activity rules

In our fire spreading *(CA)* model, the local behaviour is computed from a basic description of the fire dynamics (cf. figure 5). The *(i,j)* cell activity is formulated thanks to a mathematical expression of the combustible pyrolysis degradation. The transition rules are expressed in function of the five "*physical states*": '*unburnt*', '*heating*', '*burning*', '*cooling*' and '*burnt*'.

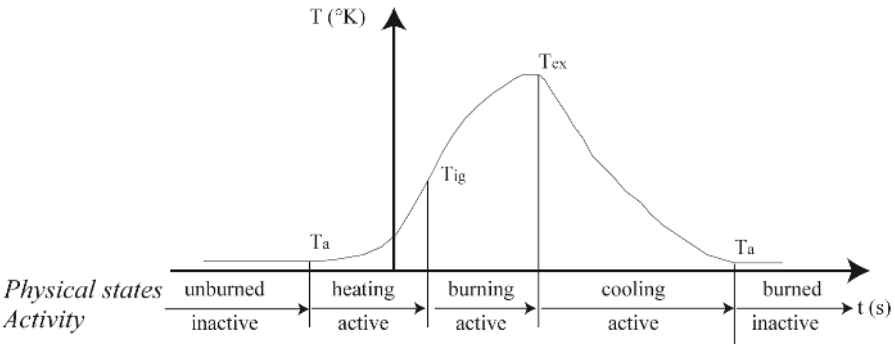

Figure 5. The physical states of the cell's state trajectory considered for tracking the activity (local level).

In this model, the combustion phenomenon is described as resulting from the kinetic fuel degradation in square areas, i.e., as a cell which contains a quantity of fuel degrading over the time. When a *(i,j)* cell starts its burning process, it is "*activated*", until it is completely burnt. The *(i,j)* cell becomes "*inactive*" when its physical state becomes '*burnt*'. Through the simulation process, the activity rules supplies the new cells of the fire front, i.e., the cells reach in the neighbourhood of the active cells at time *t+1*. Each characteristic time interval corresponds to a couple of both physical state and typical temperature ($T_{ambient}$, $T_{ignition}$, $T_{extinction}$). The discrete state trajectory describes the evolutions over the time of the temperature and mass loss using an analytical expression. The deterministic behaviour illustrated on the figure 5 corresponds to the cell's temperature evolution modelled in this work. The "*physical states*" describe the cell's behaviour over the time.

The active cells of the *(CA)* model represent the combustion zone, i.e. the *active-area* where the thermal fuel components degradation occurs. The degradation approximately occurs over a wide range of temperature around 300-900°C. We presume the fuel as a mixture of cellulose, hemicelluloses and lignin [31]. The simulation is focused on the active cells according to the discrete time event simulation (DTS) algorithm, in its activity formulation [12, 17] (cf. figure 6).

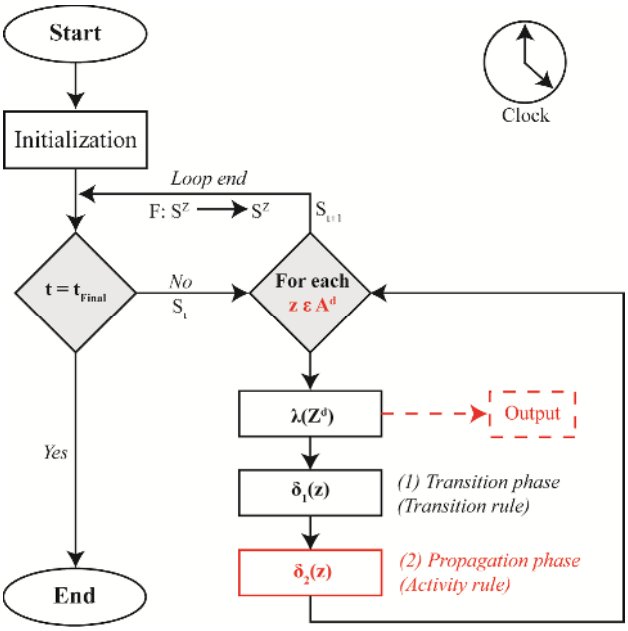

Figure 6. The physical states of the cell's state trajectory considered for tracking the activity (local level).

As previously mentioned, the active-area perimeter is updated at each time step: the transition function δ of the active cells is executed. We proceed in two steps.

(1) The first step is dedicated to the transition rule execution (*"transition phase"*). The algorithm determines if the cell considered will remain active at the next time step.

(2) In the second step, the attribute defined in each cell *"activityState"*, is updated according to the *"activity rule"*. The *"activity rule"* is in charge of determining if the inactive *N* neighbouring cells of the active cells will become active (propagation phase). In this step, the cells which can potentially change state at the next time step are activated. The activity rule also flips the cell activity using the "*active*" or "*inactive*" values of the activity (cf. algorithm 1).

```
def updateState(self):
if(self.__state=="unburnt"):
     self.__activityState="active"
     self.__nextActive=True
     self.__nextState="heating"
               if(self.__nextTem.>self.__ignition):
               print("Next temperature Error")
               self.infoCell()
               exit()
//Heating zone
if(self.__state=="heating"):
     if(self.__nextTem.>=self.__ignition):
               self.__nextState="burning"
               self.__activityState="active"
               self.__nextActive=True
          else:
               self.__nextState="heating"
               self.__activityState="active"
               self.__nextActive=True
//Burning zone
if(self.__state=="burning"):
     if(self.__nextTem.>=self.__extinction):
               self.__nextState="cooling"
               self.__activityState="active"
               self.__nextActive=True
               self.__nextTem.=self.__extinction
```

```
     else:
               self.__nextState="burning"
               self.__activityState="active"
               self.__nextActive=True
//Cooling zone
     if(self.__state=="cooling"):
          if(self.__nextTem.>self.__TA):
               self.__nextState="cooling"
               self.__activityState="active"
               self.__nextActive=True
          else:
               self.__nextState="burnt"
               self.__activityState="inactive"
               self.__nextActive=False
//Burnt zone
     if(self.__state=="burnt"):
          self.__activityState="inactive"
          self.__nextActive=False
          self.__nextState="burnt"
          if(self.__nextTem.<self.__TA):
               self.__nextTem.=self.__TA
```

Algorithm 1. Algorithm to track the activity in the CA-model.

In the algorithm 1, the activity of each cell at time *t* depends on the physical state embedded in qualities indexed by the strings *"unburnt"*, *"heating"*, *"burning"*, *"cooling"* and *"burnt"*. They cut the continuous range of the combustion curve in discrete time intervals used by the "*transition rule*" algorithm to define the cell's state (cf. figure 5). When the simulation starts, i.e. the propagation ignites, all the physical states of the ignited cells directly change from *"unburnt"* to *"burning"*, and the *"nextActivity"* attribute changes from *"False"* (inactive) to *"True"* (active). Through simulation, if the physical state of a cell reaches the *"burnt"* state, its *"activity attribute*" definitively flips to *"False"* (inactive), and the cell is dereferenced from the active set.

3.2 Object-Oriented framework

In Modelling and Simulation the Object-Oriented Programming (O.O.P.) concepts and techniques are used to ease implementation, and to provide modular, evolutive and efficient software frameworks. In this work, the object framework has been already described in a previous paper [17], that's why only a succinct description of the class hierarchy is done. Improvements are realized to dynamically instantiate the cell objects needed. In the framework, all the landscape cells belong to a land characteristics class derived from the *Cell* superclass. The *CellsBag* class is the container of the cell components. The *Model* class describes the *(CA)* model and implements the global algorithms in charge of the execution of the transition phase (transition rule) and the propagation phase (activity rule). It uses a container to store the active-cells (dynamic array). The *cellsBag* usually inherit from the classical containers implemented in the Object Oriented Programming languages. The hierarchy and the relationships of these objects are depicted on the figure 7 hereafter.

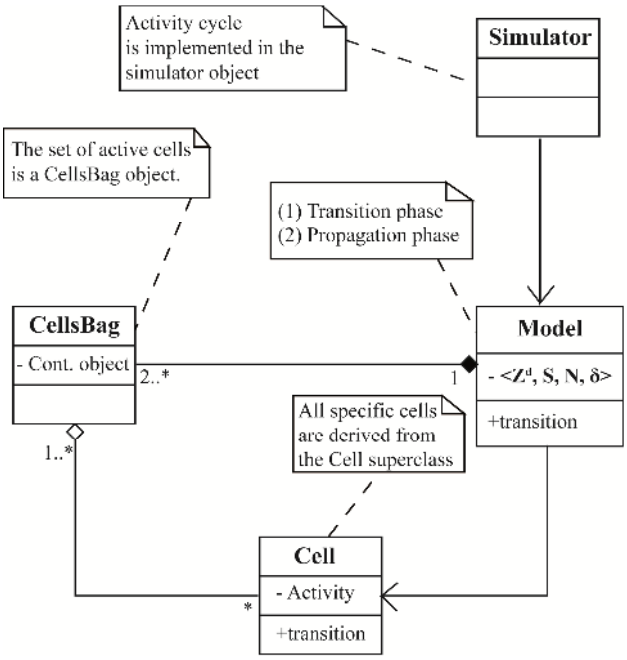

Figure 7. Object Oriented Pattern to track activity in a *(CA)* model.

4 Random-Activity

4.1 Randomization process

As in classical *(CA)* models, in stochastic (CA) model all the *(i,j)* cells of the *N* neighbourhood also execute the δ transition function to update the S^t cells' states. The cell's transition allows a gradual and nuanced transition rules implementing the introduction of the uncertainty. In this work, we called *random-activity*, the uncertainty is introduced in a quite different manner. The fire spreading behaviour is not considered as the result of a precise multi-scale description. The pseudo-random numbers are used to randomize the "*active area*", i.e. the burning area. The uncertainty is only located in the active area, thus, the cell transition rule remains deterministic. The uncertainty results at global level as an inhomogeneous outlines of fire spreading (cf. figure 12). The activity rule is non-deterministic and depends on a probability '*P*' for an active cell to execute its transition function at the next time step. In this context, only a small part of the active cells in the neighbourhood are activated during the propagation phase. As consequence, many active cells do not execute the δ transition function as they should, and they remain active. These particular active cells go to the next time remaining in the same state ($S_{t+1} = S_t$). Consequently, when an active cell execute its transition rule (transition phase), the S_{t+1} new state is different that it should do. In fact, in its neighbourhood some cells are remained constant several time steps, depending on *(P)*. The state trajectory of a cell is rather different as in its deterministic form, due to the probability *(P)*. The overall principle of the random activity is illustrated on the figure 8 hereafter.

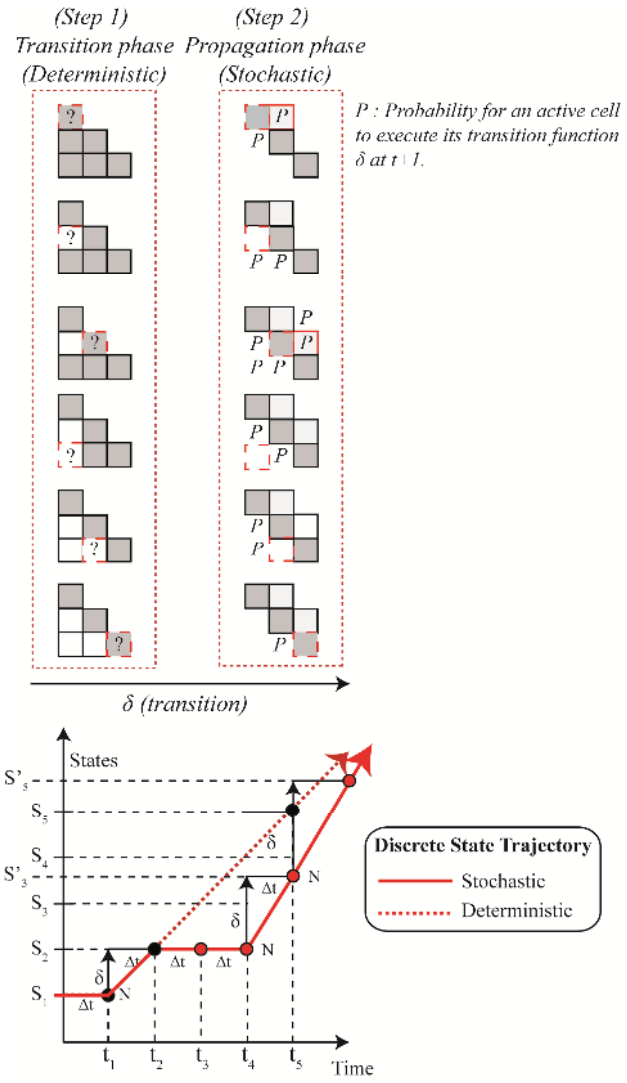

Figure. 8. Illustration of the random activity process**.** The discrete state trajectory brings forward over the time.

The random sampling required by the model to randomize the active cells is computed by a random number generator whose output is assumed to be a sequence of continuous random values uniformly distributed over the interval (0,1). These random numbers are then transformed to simulate the random probability *(P)* of a cell to be activated.

5 Simulation algorithm for random-activity

5.1 The activity-cycle

The simulation algorithm is implemented in the *Simulator* object (cf. figure 7). Before starting the simulation, the simulator object proceeds to the initialization phase, i.e., loads the map of the land characteristics and execute the ignition conditions. The set of active cells is then created and initialized to seed the activity cycle algorithm. The simulation starts and the simulator object continuously repeats in a self-reproductive mechanism the "*activity-cycle"*. The simulation flow chart of the "*random-activity algorithm*" is presented on the figure 9 hereafter.

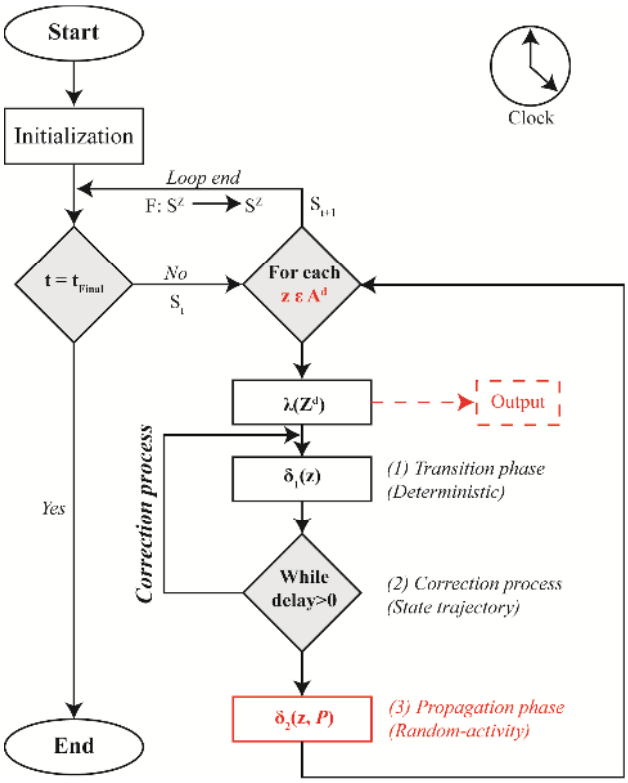

Figure 9. Simulation flowchart of the random-activity algorithm.

The active cells set is scanned at each time step, and the δ transition function of the *(i, j)* active cells is executed according to the *(P)* probability. The cell "*activity state*" is updated according to the combustion rules of the transition rule. The cells which change the activity state from "*inactive*" to "*active*" are checked, and they are added to the active cells set. The cells burnt are definitively removed from the active cells set which is continuously updated in order to reference the cells that are active at each time step.

5.2 Correction process

Due to the randomization process of the activity cycle described in 4.1, the active cells simultaneously draw delays over time. Consequently, the discrete state trajectory of the active cells brings forward over the time and the outputs cumulate error on states (cf. figure 8). The random-activity process needs the implementation of a correction process. It restrains the error of the state trajectory to obtain tolerable values. The correction process keeps the output values reasonably accurate to represent the system simulated in spite of cell's transition delays. In order to keep this error in a coherent state trajectory, a "*delay*" attribute is added in each cell to count the number of transitions skipped over the time. The transitions skipped before are recomputed, when a cell execute its transition function. This adjustment process minimizes the error and recovers the lost transitions, thanks to the "*delay*" attribute. The cell activated executes the correction process after its transition phase to adjust its state trajectory (cf. figure 9).

6 Experiments

6.1 Random-activity simulation

The effectiveness of the method is evaluated and tested on hypothetical landscapes. The simulations presented in this section are performed on a 350 x 250 lattice consisting of square cells with no slope and no wind. Each cell contains parameters describing fuel characteristics. The model is coded in the Oriented Object Programming language Python. Gnuplot® is used for graph visualizations [32, 33].

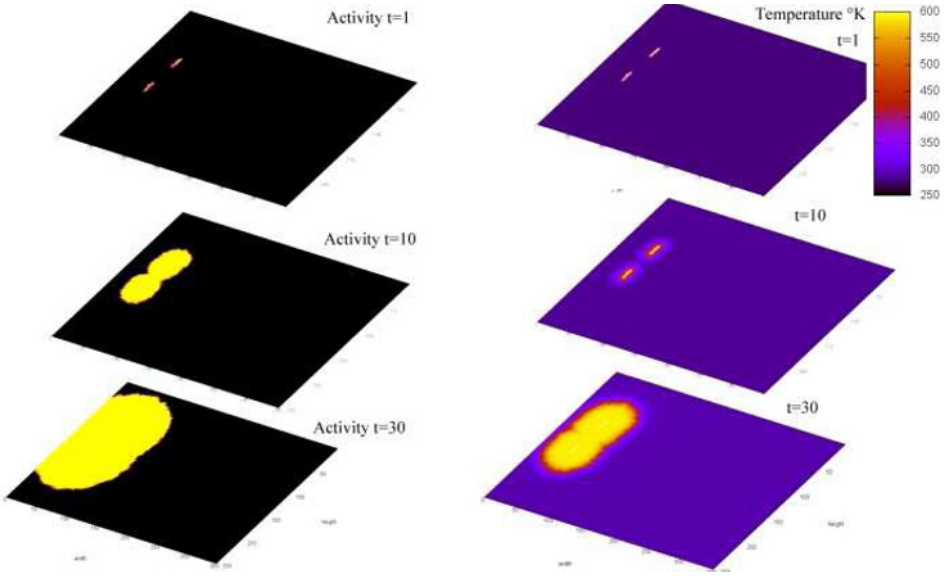

Figure 10. Sequences of three simulation steps for the test case: figures (a), (b), (c) correspond respectively to steps 1, 10, 30 with random-activity.

In this experiment the classical Moore neighbourhood is used considering all the cells on the external crown becoming active at each time step. Spurious symmetries of the square lattice appear on outputs (cf. figure 11).

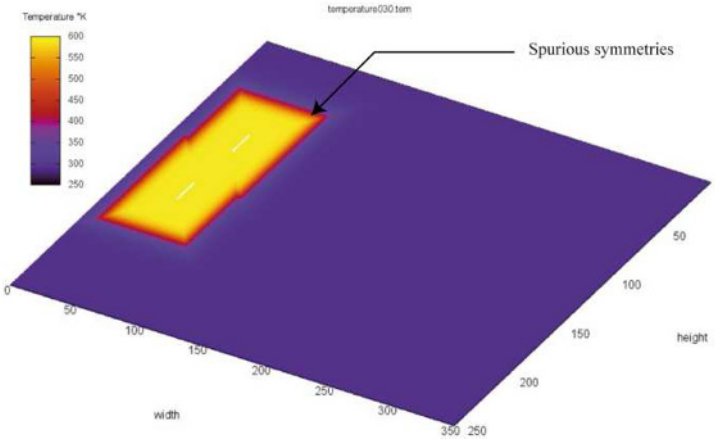

Figure 11. Fire spread simulation without random-activity, (classical activity) t = 30.

When the random-activity precepts are implemented, flames seem to appear with turbulence on the edges. The problem of the spurious symmetries which affect negatively the simulation outputs in the classical approach is resolved (cf. figure 11).

The random-activity algorithm outperforms the classical activity algorithm in terms of realism of graphical restitution. Thus, the outputs are reasonably accurate in the representation of the reality.

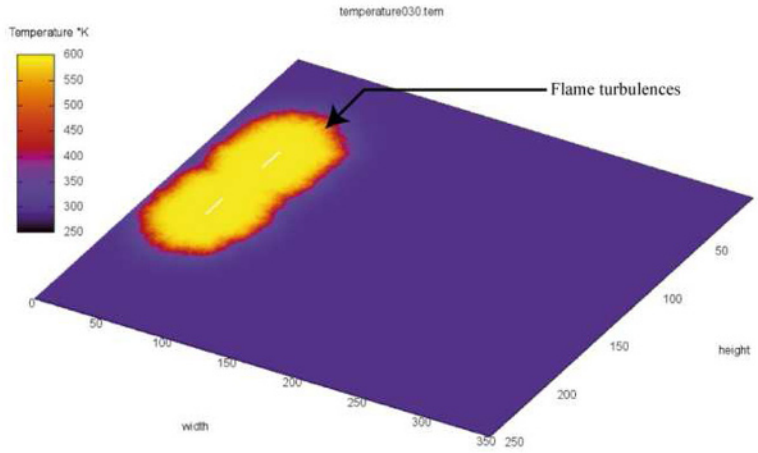

Figure 12. Fire spread simulation with random-activity, t = 30.

The experiment shows that our object framework is capable of simulating fire spreading reliably, according to the reality of the phenomenon. These preliminary results show plainly the potential of the random-activity in *(CA)* modelling.

In spite of the error correction process, a percentage near 2% of error still remains in the state trajectory. In this experiment it is sufficiently low to be tolerable. The results show that the random-activity concept can enhance fire spreading simulation based on *(CA)* models (cf. figure 13).

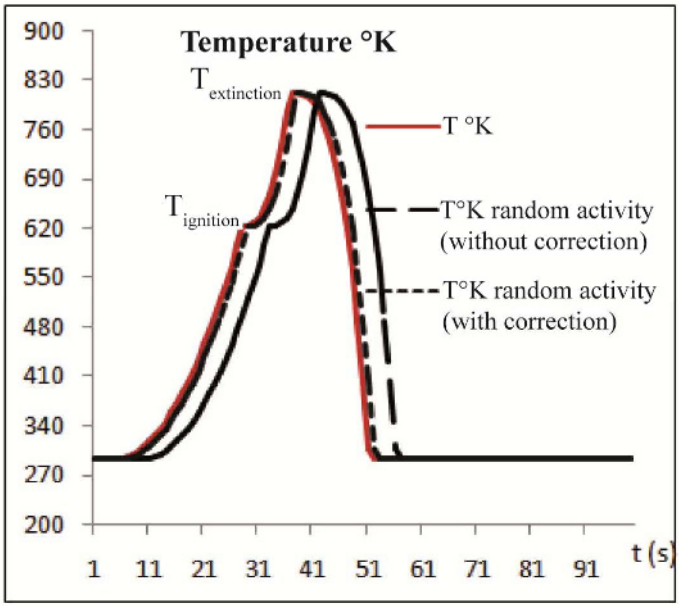

Figure 13. Comparisons of cell behaviour with and without correction implementing the random-activity process.

7 Conclusion

We have presented and illustrated a new way to implement activity-tracking algorithm in a *(CA)* model. Pseudo-random numbers and paradigm activity are coupled to simulate fire spreading in a (CA) model describing locally the fuel combustion. This new approach emphasizes the advantage to associate pseudo-random numbers and activity paradigm to acknowledge and to represent the uncertainty inherent in real systems. The approach consists in a *(CA)* model, using a simplified combustion model in each cell. The physical states used by the activity rule allow to compute faster the evolution temperature over the time. The random-activity concept shows a great potential in reproducing qualitatively the fire spreading features. Classical and random-activity simulations are both performed for comparison. This work shows that random-activity predicts in a quite adequate manner the real phenomenon characteristics in space and time in spite of state's error delays. The activity algorithm could be potentially used to develop a fire risk-management tool at the real landscapes scale. This study confirms the potential of the random-activity approach for fire spreading simulations. We have demonstrated a new way to couple activity and pseudo-random numbers in a *(CA)* modelling context. A calibration within experimental data and a sensitivity analysis will be performed in future works.

References

1. B.P. Zeigler, H. Praehofer, T.G. Kim. Theory of Modeling and simulation. Second Edition. (Academic Press 2000).

2. E. Pastor L. Zàrate E. Planas J. Arnaldos Mathematical models and calculation systems for the study of wildland fire behavior. Prog. in Ener. and Comb. Scie. J. **29**, 139-153 (2003).

3. R.O. Weber. Modelling fire spread through fuel beds, Progr. Energy Combust. Sci. **17** 67-82 (1991).

4. N. Gobeau X.X. Zhou Evaluation of CFD to predict smoke movement in complex enclosed spaces: Application to three real scenarios: an underground station, an offshore accommodation module and a building under construction. HSL Report **255** (2004).

5. D. Morvan, J.L. Dupuy, B. Porterie and M. Larini. Multiphase formulation applied to the modelling of fire spread through a forest fuel bed. Combustion Institute. **28**, 2803-2809 (2000).

6. D. Morvan, S. Méradji, G. Accary. Physical modelling of firespread in Grasslands. Fire Safety Journal **44**, 50-61 (2009).

7. A.G. McArthur, Weather and grassland fire behavior, Australian Forest and Timber Bureau Leaflet N°100, Canberra, (1966).

8. M.V. Moreno, B.D. Malamud, E.A. Chuvieco. Wildfire Frequency-Area Statistics in Spain. Procedia Environmental Sciences, **7**, 182-187 (2011).

9. C. Ordóñez, A. Saavedra, J.R. Rodríguez-Pérez, F. Castedo-Dorado, E. Covián. Using model-based geostatistics to predict lightning-caused wildfires. Env. Mod. & Soft. (**29**) 1, 44-50 (2012).

10. S.G. Berjak, J.W. Hearne. An improved cellular automaton model for simulating fire in a spatially heterogeneous Savanna system. Eco. Mod. J. **148**, 133-151 (2002).

11. A. Ohgai Y. Gohnai S. Ikaruga M. Murakami K. Watanabe Cellular Automata Modeling For Fire Spreading As a Tool to Aid Community-Based Planning for Disaster Mitigation. Rec. Adv. in Des. and Dec. Sup. Sys. in Arch. and Urb. Plan. Springer.193-209 (2004).

12. A. H. Mathey E. Krcmar S. Dragicevic I. Vertinsky. An object-oriented cellular automata model for forest planning problems. Eco. Mod. J. **212**, 359-371 (2008).

13. I.G. Georgegoudas G. C. Sirakoulis I. Andreadis Modelling earthquakes activity features using cellular automata. Math. and Comp. Mod. J. **46**, 124-137 (2007).

14. E. Yacoubi A. El Jai Cellular Automata Modelling and Spreadability. Math. and Comp. Mod. **36**, 1059-1074 (2002).

15. Y.E. Zhang Y. Han T. Zou S. Wang J. Zou A CA-based Information System for Surface Fire Spreading Simulation Proc. of Int. Geo. and Rem. Sens. Symp., USA **5,** 3484-3487 (2005).

16. Niloy Ganguly Biplab K Sikdar Andreas Deutsch Georey Canright P Pal Chaudhuri. A survey on cellular automata. Technical report, Centre for High Performance Computing, Dresden University of Technology (2003).

17. E. Innocenti A. Muzy A. Aiello J.F. Santucci, D.R.C. Hill. Design of a multithreaded parallel model for fire spread. Sim. in Ind. 15 th Eur. Sim. Symp. 104-109 (2003).

18. A. Muzy J.J. Nutaro B.P. Zeigler P. Coquillard Modeling and simulation of fire spreading through the activity tracking paradigm. Eco. Mod. **219**, 212-225 (2008).

19. G.A. Trunfio Predicting Wildfire Spreading Through a Hexagonal Cellular Automata Model. ACRI'04-Proc. of the Sixth Inter. Conf. on C. A. **3305**, 385-394 (2004).

20. A. H. Encinas L.H. Encinas S.H.White A. Martin del Rey G. Sanchez Rodrigez Simulation of forest fire fronts using cellular automata. Adv. in eng. Soft. J. **38**, 372-378 (2007).

21. A.L. Sullivan Wildland surface fire spread modelling, 1990-2007. 3: Simulation and mathematical analogue models. Int. Wild. Fire. J. E **18**, 387-403 (2009).

22. S. Bandini, G. Mauri, R. Serra Cellular automata: From a theoretical parallel computational model to its application to complex systems. Para. Comp. J. **27**, 539-553 (2001).

23. Robert J. Krawczyk. Architectural Interpretation of Cellular Automata, Poster presented at NKS, Boston, (2003).

24. J. Podrouzek. Stochastic Cellular Automata in Dynamic Environmental Modeling: Practical Applications. Elec. Notes in Theor. Comp. Scie. **252**, 1, 143-156 (2009).

25. V. Guinot. Modelling using stochastic, finite state cellular automata: rule inference from continuum models. App. Math. Mod. **26**, 6, 701-714 (2002).

26. Z.L. Krougly, I.F. Creed, D.A. Stanford. A stochastic model for generating disturbance patterns within landscapes. Comp. & Geo. **35**, 7, 1451-1459 (2009).

27. A. Alexandridis, D. Vakalis, C.I. Siettos, G.V. Bafas. A cellular automata model for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990. App. Math. and Comp. **204**, 1, 191-201 (2008).

28. W. U. Pooch, A. James Wall Discrete Event Simulation: A Practical Approach. (CRC Press, 1993)

29. I. Karafyllidis A. Thanailakis A model for predicting forest fire spreading using cellular automata. Eco. Mod. J. **99**, 87-97 (1997).

30. A. Muzy D.R.C. Hill Stochastic Modeling Strategies for the Simulation of Large (Spatial) Distributed Systems: Application to Fire Spread. Discrete event modeling and simulation. Theory and applications (CRC PRESS, 2011).

31. D. Baroudi A discrete dynamical model for flame spread over combustible flat solid surfaces subject to pyrolysis with charring – an application example to upward flame spread. Fire Saf. J. **38**, 53-84 (2003).

32. Python Programming Language – Official Website. http://www.python.org.

33. Gnuplot – portable command-line driven graphing utility – Official Website. http://www.gnuplot.info.