

FACULTAT D'INFORMÀTICA DE BARCELONA

FIB UPC
OPTIMITZACIÓ

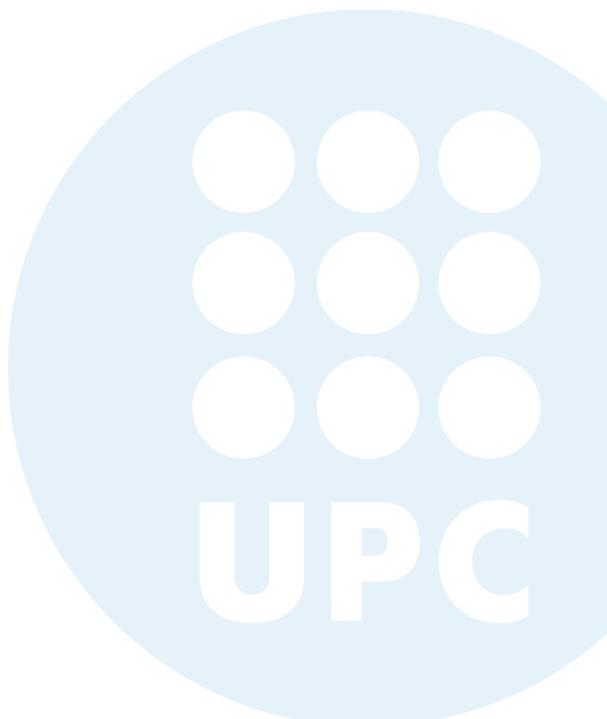
Pràctica 3: Treballant amb CA & Heurístiques

Alumnes :

Jerez Cubero, ALBERTO
Granja Bayot, JORDI

Tutors :

Fonseca, PAU



Contents

1	Introducció	2
1.1	Objectius	2
2	Autòmat Cel·lular amb Regles de Wolfram	3
2.1	Assumpcions del CA	4
2.2	Autòmat Cel·lular Multicapa	4
2.3	Resultats de la Simulació	5
2.4	Conclusions CA de Wolfram	6
3	Model de Propagació d'un Incendi Forestal	7
3.1	Model en SDL	7
3.2	Generació de les Dades	8
3.3	Autòmat Cel·lular del Model	8
3.3.1	Capa de foc	9
3.3.2	Capa de combustible	9
3.3.3	Capa d'humitat	9
3.4	Assumpcions del model	10
3.5	Anàlisi dels resultats	11
3.6	Conclusions de la Modelització	12
4	Ús de la IA	13

1 Introducció

El present informe constitueix la justificació de la solució proposada pel problema plantejat. Aquest és, implementar una sèrie d'autòmats cel·lulars: un CA amb regles de Wolfram, tant monocapa com multicapa; i una adaptació del CA a un model real, com és la propagació d'un incendi forestal. En el nostre cas, el llenguatge de programació escollit ha estat Python.

1.1 Objectius

En la primera part de la pràctica, ens concentrarem en la implementació d'un autòmat cel·lular basat en les regles de Wolfram. L'objectiu principal consistirà en crear aquest autòmat i comprendre els conceptes que el defineixen: la regla d'actualització, la regla de combinació, el veïnatge, entre altres. Un objectiu secundari serà la representació gràfica de les capes d'aquest autòmat per tal de visualitzar-ne l'evolució i el comportament, associant-lo amb les diverses classes de Wolfram. A més, aquesta primera part servirà com a fonament per a la realització de la simulació en la segona part del treball.

En una segona part de la pràctica, un cop interioritzats els conceptes bàsics dels autòmats cel·lulars, voldrem modelitzar la propagació d'un incendi forestal. Així, l'objectiu principal serà la simulació d'aquest sistema, ja que estudiar el seu comportament en un experiment real és inviable. En aquest sentit, podrem analitzar els resultats que s'esdevenen de les interaccions entre les diferents capes. En definitiva, aquesta segona part servirà per donar un enfocament més pràctic i real als nostres autòmats cel·lulars.

2 Autòmat Cel·lular amb Regles de Wolfram

En aquesta primera part, ens centrem en la implementació d'un autòmat cel·lular elemental, que basarà la generació dels seus estats en les regles de Wolfram. En aquest sentit, l'autòmat és cel·lular perquè opera sobre una quadrícula de cel·les (espai discret), i elemental, perquè aquestes prenen dos valors possibles (0 o 1), els quals es determinen amb regles definides sobre el veïnatge directe de cada cel·la. D'aquesta manera, podrem simular i estudiar el comportament de sistemes complexos, concretament quan les interaccions locals depenguin només de dos estats. Per exemple, la propagació d'un incendi, en què una cel·la podria estar cremant-se o, al contrari, no estar afectada pel foc.

En aquest sentit, amb 256 regles de Wolfram podrem elegir les més apropiades per una aplicació específica. D'aquesta manera, la implementació del autòmat incorporà aquesta flexibilitat. De cara a simular diversos escenaris, amb variacions en el nombre de cel·les de cada estat, la regla d'actualització o bé el nombre de generacions (temps discret), la implementació s'ha articulat amb una classe `Wolfram_CA`. Així, després d'una inicialització de l'autòmat, és a dir, de l'estat inicial i la funció d'actualització, podem dur a terme simulacions de temps variats.

```
1 def simulate(self, num_steps):
2
3     states = np.zeros((num_steps, self._state_len), dtype=int)
4
5     # Configurar l'estat inicial
6     states[0, :] = self._initial_state
7
8     for step in range(1, num_steps):
9         for i in range(self._state_len):
10
11             left = states[step - 1, (i - 1) % self._state_len]
12             center = states[step - 1, i]
13             right = states[step - 1, (i + 1) % self._state_len]
14
15             # Regla de Wolfram
16             states[step, i] = self._update_function(left,
17                 center, right)
18
19     return states
```

2.1 Assumpcions del CA

Pel que fa a les assumpcions, a banda de la homogeneïtat de les regles i les interaccions reduïdes als veïns directes (elementalitat), en un primer moment assumirem les condicions de frontera periòdica o cíclica. Això significa que l'estat del CA està fitat, i que les cel·les dels marges interactuen amb les cel·les del costat oposat de la quadrícula. Aquesta política és molt freqüent en simulacions de sistemes sense límits externs o que presenten periodicitat, com per exemple en dinàmica molecular o de fluids.

No obstant això, existeixen altres condicions de frontera pels autòmats cel·lulars, que poden ser fixes, reflexives, etc. La seva elecció, depèn del problema, concretament de les característiques del sistema a simular. En el nostre cas, per simplicitat, assumirem un estat de longitud fixa, amb condicions de frontera periòdica.

2.2 Autòmat Cel·lular Multicapa

En aquest apartat, desenvolupem un autòmat cel·lular multicapa, a partir de la combinació de CAs simples ($1 : 1 - AC^1$) implementant diferents regles. En aquest sentit, tornem a plantear la implementació en torn a una classe, `Multilayer_Wolfram_CA`. Aquesta rep una llista de nombres de regles de Wolfram, així com el nombre de cel·les de cada estat i la funció de combinació. Anàlogament, incorpora un mètode per simular el CA multicapa, que en el nostre cas, només tindrà una capa principal ($m : 1 - AC^1$).

```

1 class Multilayer_Wolfram_CA:
2
3     def __init__(self, layers_rules, state_len = 101, comb_f
4                 = np.bitwise_and.reduce):
5
6         self._m = len(layers_rules)
7         self._CAs = [Wolfram_CA(rule_number, state_len =
8                         state_len) for rule_number in layers_rules]
9         self._comb_f = comb_f
10
11    def simulate(self, num_steps):
12
13        states = [wolfram_CA.simulate(num_steps) for
14                  wolfram_CA in self._CAs]
15        comb = self._comb_f(states)
16
17        return comb

```

Novament, tornem a tenir una elecció que dependrà del sistema a simular: la funció de evolució dels estats de la capa principal. Aquesta és, la funció de combinació de les diferents capes, ja que les funcions de proximitat i nucli no s'han considerat, per simplicitat. En el cas concret de regles de Wolfram, el més lògic és implementar funcions de combinació a nivell de bit (e.g. AND).

2.3 Resultats de la Simulació

En aquesta darrera part, ens interessem per donar una representació gràfica a l'evolució dels autòmats cel·lulars en cada simulació. En aquest sentit, implementem una funció `plot_automaton`, que ens permetrà visualitzar l'estat dels CAs en cada pas de temps o generació. En la Fig. 1 podem observar el comportament generat per la regla 45 de Wolfram. Pel que fa a l'autòmat cel·lular multicapa, trobem un exemple de simulació a la Fig. 2, implementat amb funció de combinació AND i regles de Wolfram 30, 90 i 110.

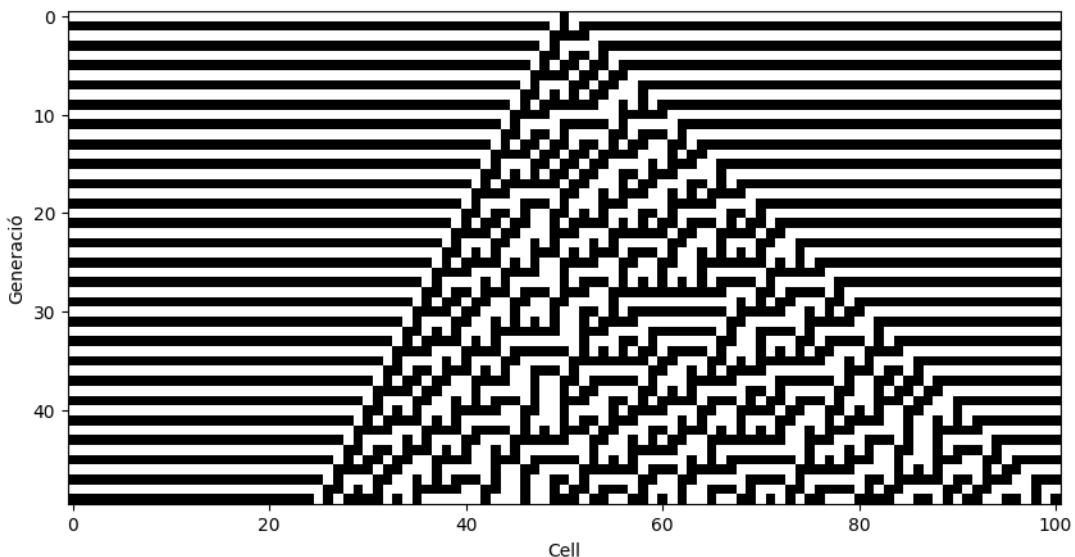


Figure 1: autòmat Cel·lular amb Regla 45 de Wolfram

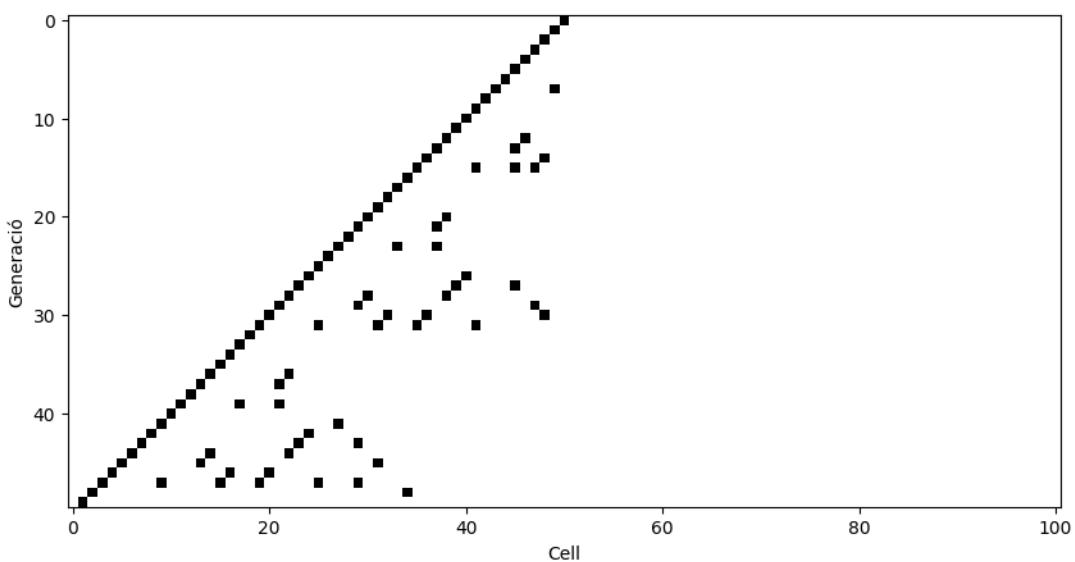


Figure 2: autòmat Cel·lular Multicapa amb Regles 30, 90 i 110 de Wolfram

2.4 Conclusions CA de Wolfram

En resum, en aquesta primera part del treball hem interioritzat la noció d'autòmat cel·lular, mitjançant un enfocament simple, ja que s'han considerat les regles de Wolfram com a funcions d'actualització. En aquest sentit, hem hagut d'investigar sobre les assumpcions que hi ha darrere els CAs, les quals depenen de l'aplicació en qüestió, com també les funcions d'actualització o evolució.

D'altra banda, en quant als dos tipus d'autòmats cel·lulars implementats, podem concloure que l'extensió a un CA multicapa aporta una altra dimensió a les capacitats de simulació. Sempre que la funció d'evolució sigui adequada, podrem modelar dinàmiques més complexes, que emergeixen de la interacció entre diferents capes de CAs.

En definitiva, en aquesta secció hem implementat models de simulació arbitraris i simples, per tal d'entendre el funcionament dels autòmats cel·lulars. Així, de cara a un futur model real amb CA, tant les assumpcions com les regles d'actualització o d'evolució, les haurem d'elegir nosaltres com a modeladors, en funció del sistema d'estudi.

3 Model de Propagació d'un Incendi Forestal

3.1 Model en SDL

En una primera aproximació a la modelització d'un incendi forestal mitjançant un autòmat cel·lular, hem optat per definir un model SDL simple per descriure la funció d'evolució de la capa principal que representa el foc. El diagrama resultant es mostra a continuació:

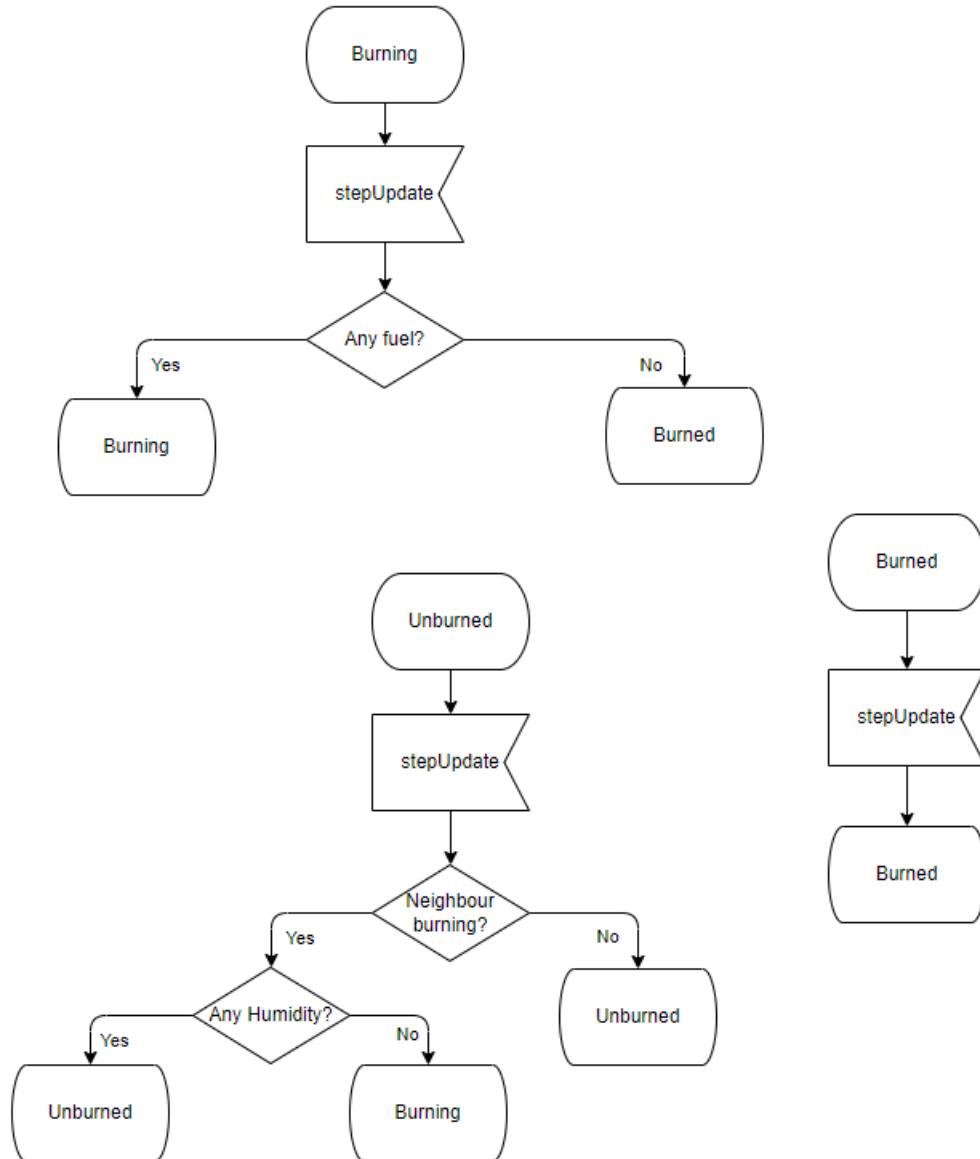


Figure 3: Model SDL capa de foc.

En el diagrama anterior, les etiquetes **Burning**, **Unburned** i **Burned** representen els tres possibles estats de cada cel·la de la capa de foc. L'event **stepUpdate** simbolitza l'única transició present en el nostre sistema, és a dir, l'actualització de les capes. Amb la finalitat de simplificar, hem optat per ometre les funcions que actualitzen els valors de cada capa en cada iteració.

3.2 Generació de les Dades

En el context particular d'un model amb finalitat acadèmica, ens trobem en la tessitura d'haver de generar artificialment les nostres dades d'entrada. Concretament, haurem d'indicar a l'autòmat cel·lular la inicialització de cada capa, és a dir, la imatge inicial de cada atribut: foc, combustible i humitat. En aquest sentit, les dades es representaran en el format de fitxer IDRISI32. No obstant, la generació es durà a terme amb Python, mitjançant una matriu, estructura més adient per representar informació relativa a un espai discretitzat en cel·les.

En primer lloc, la capa en què indicarem la propagació del foc s'inicialitzarà fàcilment. D'entre totes les cel·les de l'autòmat, n'elegirem una aleatoriament, que representarà el focus de l'incendi. La resta de cel·les, romandran en l'estat base *Unburned*.

Seguidament, assignem a cada cel·la de l'autòmat la seva informació relativa al combustible (i.e. vegetació). En el nostre cas, la decisió és un nombre en el rang [5, 60], que representarà el potencial nombre d'hores de foc actiu d'una cel·la. D'altra banda, el canvi d'estat de *Unburned* a *Burning* podrà retardar-se quan hi hagi humitat a les zones afectades. Per tant, inicialitzem també la capa d'humitat, amb valors en el rang [2, 24], representant el retard d'emissió de foc a una cel·la. En aquest sentit, la lògica de generació de les dades és la mateixa per les dues capes. Donada una discretització del terreny en cel·les, s'agrupen les cel·les en *clusters*, pels quals s'assigna una densitat. Després, per cada cel·la, es determina un nombre d'hores aleatori de combustible/humitat, proporcional a la densitat del seu *cluster*. D'aquesta manera, obtindrem comportaments més realistes, esperant resultats més interessants de cara a la simulació.

3.3 Autòmat Cel·lular del Model

La implementació de l'autòmat cel·lular per a la simulació de l'incendi forestal es fonamenta en l'evolució d'un estat inicial, representat com un tensor de 3 dimensions (capes), el qual està definit en els fitxers *.img* generats conforme a l'apartat 3.2. Aquest procés evolutiu té lloc de manera separada en les tres capes, seguint les seves respectives funcions d'actualització, creant així nous estats.

Quant a les decisions preses en el modelatge d'aquesta simulació, s'ha optat per modelar el veïnatge seguint el principi de Moore. Aquesta elecció es fonamenta en la seva capacitat per capturar una àmplia gamma d'interaccions entre les cel·les veïnes. Al considerar les vuit cel·les adjacents, en lloc de només les quatre direccions cardinals com ho fa el Veïnatge de von Neumann, obtenim una simulació més complexa i realista dels fenòmens de propagació i difusió del foc que són rellevants per a la nostra simulació.

Per comprendre de manera clara el flux d'execució de la simulació, analitzarem cada capa de forma individual. Per a una millor comprensió, farem servir com a guia el model SDL ubicat a la part superior.

3.3.1 Capa de foc

En aquesta capa principal, necessitem informació de les altres dues capes. Específicament, requerim el valor de la mateixa cel·la de les capes d'humitat i combustible, així com el veïnatge de la capa de foc. Un cop disposem d'aquests valors, cridem la funció d'actualització.

La funció d'actualització d'aquesta capa, com hem vist en el diagrama SDL anteriorment, segueix aquests criteris: si la cel·la actual ja està cremada, la mantindrem en aquest estat. D'altra banda, si està cremant, la mantindrem cremant, sempre i quant tingui de combustible. Quan s'esgoti el combustible, la cel·la passarà a estar cremada. Finalment, si la cel·la està a punt de cremar, comprovarem que tingui de combustible i que la seva humitat sigui 0. En cas afirmatiu, passarà a estar cremant; sinó, seguirà en l'estat de per cremar.

3.3.2 Capa de combustible

En aquesta capa, necessitem informació de la mateixa capa i de la capa de foc. Específicament, només requerim els valors de la mateixa cel·la. Un cop disposem d'aquests valors, invoquem la funció d'actualització.

La funció d'actualització d'aquesta capa simplement resta 1 al combustible si aquesta cel·la està cremant-se; en cas contrari, no canvia. Noti's que no permetem que prengui valors negatius.

3.3.3 Capa d'humitat

En aquesta última capa, necessitem el veïnatge de la capa de foc i el valor actual de la humitat. La funció d'actualització d'aquesta capa està dissenyada amb l'objectiu de considerar que un major nombre de cel·les adjacents cremant provoquen una major reducció de la humitat. Tot i això, optem per no modelar aquesta reducció amb una relació lineal perquè la considerem poc realista. La solució adoptada és la següent funció:

$$\begin{cases} 0 & x \leq 0 \\ (1.5 \ln(x)) + 1 & x > 0 \end{cases}$$

on x representa el nombre de veïns en flames segons el veïnatge de Moore. Aquesta funció passa pel punt (1,1) al seu domini i creix de forma logarítmica, un comportament que s'ajusta a la nostra definició del model.

Un cop calculat el valor de la reducció, el restem al valor actual de la humitat per obtenir el nou valor. En cas que sigui negatiu, el fixem a zero.

3.4 Assumpcions del model

Pel que fa a les assumpcions, a part de la homogeneïtat de les regles i les interaccions reduïdes als veïns directes, com ja hem definit en la primera part del treball, en aquesta ocasió assumim la condició de **frontera estricta**. Això implica que l'estat del CA està limitat, i que les cel·les dels marges interactuen només amb les cel·les presents a la quadrícula. Tot i que aquesta política pot afectar significativament la velocitat de propagació de les cel·les a les cantonades (a causa de com hem modelat la reducció de la humitat), fronteres com la periòdica no tindrien cap sentit en el context del modelatge d'un incendi forestal.

És així com en el nostre cas, assumim un estat de longitud fixa, amb condicions de frontera estricta o fixa.

3.5 Anàlisi dels resultats

Un cop detallades les parts essencials de la implementació, entrem a fons en l'anàlisi dels resultats. En el nostre cas, voldrem visualitzar amb gràfics la propagació de l'incendi, cosa que es tradueix a mostrar l'evolució de les capes del CA. En el context de *layers* representades amb matrius, les nostres funcions reben el conjunt de matrius de cada generació per una simulació de temps t . En funció del valor de cada cel·la, s'assigna al seu píxel del gràfic un color més o menys fosc, en el cas de la humitat i la vegetació, o bé un color diferent, en el cas de la capa de foc.

Concretament, les funcions que s'encarreguen de la representació gràfica són *visualize_gif* i *visualize_realtime*, que proporcionen imatges estàtiques i dinàmiques, respectivament. Noti's que la visualització *gif* és més costosa computacionalment, ja que requereix emmagatzemar les imatges a cada generació. No obstant, ens permet visualitzar la simulació d'exemple amb $t = 100$ per les diferents capes.

Figure 4: Evolució Incendi Figure 5: Evolució Humitat Figure 6: Evol. Vegetació

En aquest exemple, podem observar com el comportament del sistema és l'esperat. A mesura que l'incendi es va propagant a les diferents cel·les, la humitat de les mateixes i la vegetació es van consumint. A més, noti's com la propagació no és uniforme, sinó que el comportament del foc està governat per les densitats de *cluster* que hem definit a la generació de dades (3.2). En aquest sentit, jugant amb diferents conjunts de dades, les simulacions són diferents, però el patró de propagació és molt similar. Si afegim més humitat, el foc es retarda, mentre que si augmentem el combustible, prolonguem el temps que roman actiu.

3.6 Conclusions de la Modelització

La utilització d'autòmats cel·lulars com a eina principal per simular l'evolució d'un incendi forestal ofereix una aproximació simple i dinàmica dels processos involucrats en aquest fenomen. Cada capa de l'autòmat cel·lular està caracteritzada per funcions d'actualització específiques, les quals implementen les regles que determinen l'evolució de l'incendi. Aquestes funcions tenen en compte factors com el combustible disponible, la humitat present i l'estat de les cel·les veïnes, permetent una simulació detallada del comportament de l'incendi.

En conclusió, hem pogut complir l'objectiu d'implementar un autòmat que ens permet realitzar la simulació de l'evolució d'un incendi forestal. Si bé un incendi real és més complex, aquesta ha estat una bona introducció a la simulació de sistemes amb CA's.

4 Ús de la IA

Per a la realització del treball exposat, l'ús de la Intel·ligència Artificial ha estat present. En el context d'introducció als autòmats cel·lulars, eines com el *ChatGPT* han ajudat a clarificar conceptes. Per exemple, en la primera part del treball, l'extractor de la funció d'evolució corresponent a una certa regla de Wolfram.

```
1 def _get_rule_function(self, rule_number):
2
3     rule_binary = format(rule_number, '08b')
4
5     def update_function(left, center, right):
6         return int(rule_binary[7 - (left * 4 + center * 2 +
7             right)])
8
9     return update_function
```

En la nostra opinió, les eines de IA destinades a l'àmbit acadèmic resulten beneficioses, tant que nosaltres ja les hem incorporat en el nostre flux de treball. En el context d'aquesta pràctica, en tasques com la generació de visualitzacions en Python, en les quals no estem especialitzats i les possibilitats són enormes (varietat de mòduls), ens sentim més còmodes recorrent a la IA, alhora que podem centrar-nos a implementar nosaltres mateixos les parts més importants (e.g. les classes dels autòmats de la solució).

En aquest sentit, la considerem com una eina complementària, que ens ajuda en les tasques més pesades, si bé s'ha de supervisar en tot moment, per evitar errors i donar més qualitat als treballs. En resum, la IA proporciona una bona línia de partida que, si es complementa amb coneixement personal, pot potenciar el rendiment de les nostres produccions.