Informe de la Práctica 3

Pablo Barrenechea Perea[*], Pau Prat Moreno[**]

Abril de 2024

Universitat Politècnica de Catalunya

Grado en Inteligencia Artificial

Optimización

[*]16106624T
[**]47599937B

Resumen

Este documento corresponde al informe de la tercera práctica de la asignatura de Optimización, perteneciente al grado en Inteligencia Artificial de la Universitat Politècnica de Catalunya (UPC).
El informe está dividido en 2 partes principales. La primera de ellas trata la ejecución de automátas celulares siguiendo las normas de Wolfrang. La segunda parte, trata la ejecución de autómatas multidimensionales combinados, con el objetivo de representar una simulación sencilla de un incendio bidimensional.

Índice

1. Primera Sessió - Regles de Wolfram

En aquesta primera sessió, se'ns ha demanat implementar un autòmat cel·lular que segueixi les regles elementals de Wolfram, tal com es defineixen a *Wolfram MathWorld*. Aquestes regles són un conjunt de normes per a determinar l'estat d'una cèl·lula en una nova generació, basant-se en l'estat de la mateixa cèl·lula i les seves dues veïnes immediates en la generació anterior.

1.1. Objectius i Assumpcions

L'objectiu principal d'aquesta sessió és desenvolupar un codi (hem escollit fer-ho en Python) que pugui:

- Implementar les regles elementals de Wolfram per a un autòmat cel·lular.
- Generalitzar l'implementació per permetre l'ús de múltiples regles diferents en un mateix autòmat, creant una estructura multicapa.
- Visualitzar l'evolució temporal de l'autòmat en un gràfic per a una regla simple o una combinació d'elles.

Assumim que l'usuari introduirà regles vàlides en format decimal i que l'entorn de desenvolupament té les llibreries necessàries instal·lades (per exemple, *matplotlib* per a la visualització).

1.2. Descripció del Codi

El codi desenvolupat es divideix en diverses funcions que permeten una fàcil expansió i manteniment:

Funció apply_rule

Aquesta funció és el nucli de l'autòmat cel·lular. Rep com a paràmetres el número de regla en format decimal i els estats actuals de tres cèl·lules consecutives: esquerra, centre i dreta. La funció converteix la regla a format binari i utilitza una operació matemàtica per calcular l'índex corresponent a la combinació actual de cèl·lules. Retorna l'estat nou de la cèl·lula central segons la regla.

Funció generate_initial_state

Aquesta funció crea l'estat inicial de l'autòmat. Per defecte, configura una longitud de 100 cèl·lules i activa només la cèl·lula central, amb la resta de cèl·lules inicialment inactives. Aquesta configuració inicial simplifica la visualització dels efectes de les regles i facilita l'anàlisi dels patrons que es desenvolupen.

Funció evolve_multilayer

Desenvolupa l'autòmat a través de múltiples generacions, aplicant un conjunt de regles que l'usuari pot variar. Permet la creació de diverses çapes.º "nivells"d'autòmats, cadascun amb la seva pròpia regla. Les capes són processades de manera independent en cada generació, i els resultats de totes les capes són combinats per determinar els estats de la capa superior en la següent generació.

Funció plot_evolution

Utilitza la llibreria *matplotlib* per visualitzar gràficament l'evolució de l'autòmat cel·lular. Genera una imatge on cada fila correspon a una generació de l'autòmat, mostrant els canvis al llarg del temps.

1.3. Conclusions

El codi que hem desenvolupat compleix amb els requisits de la pràctica, doncs és capaç de simular autòmats cel·lulars simples i també de configurar-los en estructures multicapa. Això ens permet explorar i comprendre millor com es formen els patrons complexos a partir de regles simples. A més, la funció de visualització gràfica que hem implementat ens ajuda a veure clarament com evolucionen les cèl·lules amb el temps, facilitant-nos l'enteniment dels efectes de les diferents regles quan s'apliquen conjuntament.

Hem utilitzat també eines d'intel·ligència artificial com ChatGPT per ajudar-nos a escriure i millorar parts del codi. Aquesta tecnologia ha estat molt útil per accelerar el desenvolupament, però també hem après que és molt important saber comunicar-se clarament amb aquesta eina. Si no li diem exactament el que necessitem, pot ser que no obtinguem els resultats esperats. Això ens ha ensenyat la importància de tenir un bon coneixement del problema que estem tractant per poder guiar eficaçment l'ajuda que ens ofereix la IA.

També cal remarcar que hem fet ús de Copilot (que el tenim integrat al Visual Studio Code) per tal de fer més eficient certes parts del codi, així com arreglar algunes línies que no funcionaven del tot bé. Per tant, la IA ha estat una eina de molta ajuda en aquesta primera sessió, ja que ens ha permès millorar el nostre procés de desenvolupament i aconseguir resultats més ràpids i precisos.

2. Segona Sessió - Propagació d'un incendi forestal

La segona sessió tracta d'implementar una sèrie d'automates cellulars, amb l'objectiu d'aplicar-los per tal que puguin representar un mapa bidimensional que simuli la propagació d'un incendi forestal. Les diferents capes que utilitzaran els autòmats com a dades són les següents:

- Vegetació
- Humitat
- Aigua

Per a cada capa, existeixen diferents normes que les correlacionen, combinant-se i formant la simulació simplificada del incendi forestal.

2.1. Definició de les capes i el seu comportament

Amb l'objectiu de la representació, el model inicial constava de 3 capes, una representant la humitat de les cel·les, una segona en representació de la quantitat de vegetació i la tercera representant el foc per a cada cel·la. Finalment, també hem implementat una 4a capa que s'aprofiti d'un generador de terreny basat en soroll de perlin, per a afegir llacs i rius en totes aquelles cel·les que tinguin una alçada menor que un llindar escollit.

Cadascuna d'aquestes capes tenen un comportament definit, de forma que l'estat d'una de les capes per a una cel·la tingui una certa influència sobre altres cel·les, sigui en la mateixa capa o en una altra. Per determinar el comportament, hem creat les següents normes.

Fire_status

La primera capa a explicar és la més complicada d'elles, l'anomenada "fire_status". Aquesta capa representa l'estat del foc d'una cella, seguint la següent codificació:

- 0: No hi ha foc en aquesta cel·la.
- 1: La casella està cremant-se i pot tenir foc.
- 2: La casella s'ha cremat.

El funcionament del foc és bastant senzill. Si una casella no té foc, la resta de variables no canviaran fins que alguna casella adjacent a aquesta propagui el seu foc. Hem considerat com a adjacents aquelles cel·les que es trobin sobre, sota, a la dreta o a l'esquerra de la cella que vagi a propagar el seu foc. Un cop el foc s'hagi propagat fins a la casella que estiguem estudiant, aquesta començarà a cremar-se, perdent humitat. En perdre la seva humitat, direm que aquesta casella té foc i s'està cremant, de forma que comenci a propagar el foc a les cel·les adjacents. També s'ha de nomenar el fet que en cas que la casella estigui al final de la quadrícula, aquesta no podrà propagar el foc més enllà de la quadrícula visible. Es tracta, bàsicament, de la capa més important en la simulació, donat que és la part dinàmica de la simulació, l'encarregada que varií amb el temps.

Humitat

Aquesta capa, tal com explica el seu nom, representa el nivell d'humitat de cada casella. Bàsicament, està representada per un nombre enter per cada casella, i es regeix per les següents normes:

- Si una cella té foc, per cada unitat de temps que passi, el nivell d'humitat d'aquesta cella es reduirà en una unitat.
- Si una cella arriba a tenir un valor d'humitat zero, no podrà baixar d'aquest nivell.
- Si una cella té foc i el seu valor d'humitat és zero, considerarem que aquesta cella s'està cremant.

Bàsicament, la humitat funciona com un factor que indicarà el temps que una casella amb foc trigarà a començar a cremar-se i estendre el foc.

Vegetació

La vegetació és la capa que indicarà la quantitat de vegetació per cella existeix. Aquesta, estarà representada per un valor enter, com també la humitat, i segueix les següents normes:

- En cas que una cella tingui foc, i la seva humitat sigui zero, aquesta cella perdrà una unitat d'humitat per cada unitat de temps.
- Quan el valor de vegetació sigui zero, no podrà reduir-se més.
- Si el valor de vegetació és zero, i la cella s'està cremant, aquesta deixarà de cremar-se i passarà a l'estat de foc 2 (cremat).

Corpus d'aigua

Aquesta capa representa tots els corpus d'aigua amb mida considerable que podem trobar en un bosc, siguin llacs o rius. Aquesta capa serà generada a partir d'un generador de terreny, de forma que el terreny es generi assignant una alçada específica a cada cella d'una quadrícula de les mateixes dimensions que aquella del bosc. Un cop existeixi aquesta grilla addicional, la capa de corpus d'aigua s'omplirà en funció del terreny, de forma que si l'alçada és més baixa o alta que un llindar, se li assigni els valors 1 (és part d'un cos d'aigua) o 0 (no es tracta d'aigua).

En el que al seu funcionament respecta, aquestes cel·les no podran tenir foc en cas de pertànyer a un cos d'aigua; i, per tant, tampoc podran estendre'l. Destacar també que el generador de terreny es basa en un generador de soroll de perlin per escollir les diferents alçades de la quadrícula addicional.

2.2. Diagrama de Fluxos

Per tal de realitzar el diagrama de fluxos de la propagació de l'incendi, ens hem volgut centrar en el comportament d'una única cel·la, ja que totes es comporten de la mateixa manera. Així doncs, el dibuix del diagrama de fluxos sobre la nostra implementació d'aquest projecte ha estat el següent:

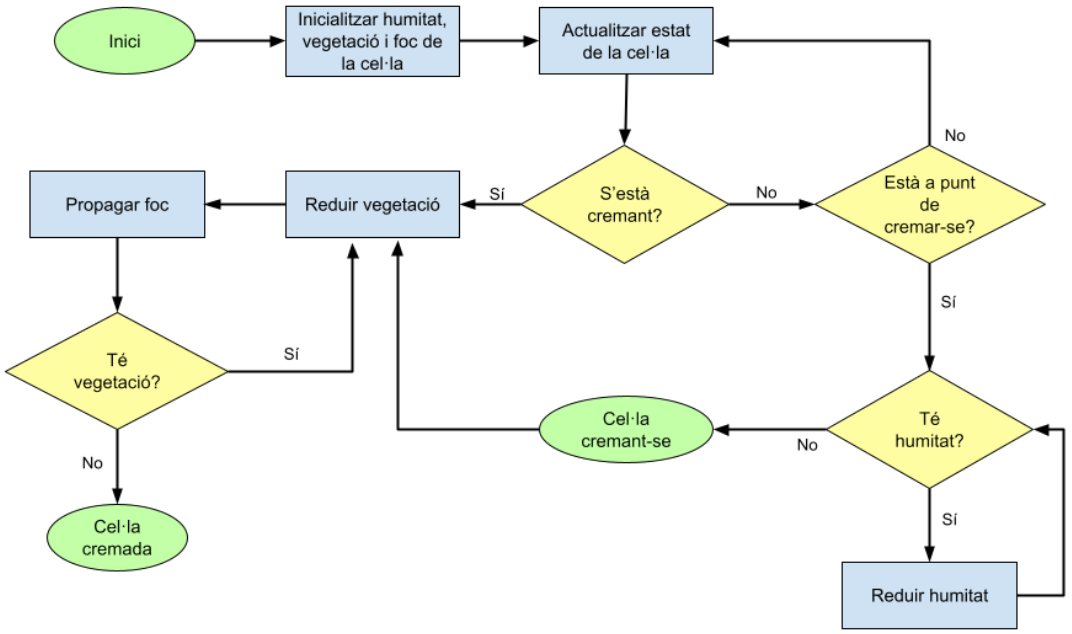

Figura 1: Diagrama de fluxos d'una cel·la

Com podem veure, en un inici s'estableixen els diferents paràmetres per a totes les cel·les, és a dir, el seu nivell d'humitat, de vegetació, i de foc. Cal remarcar que en aquest diagrama de fluxos no representem el fet que una cel·la pugui ser riu, ja que en aquest cas, simplement la cel·la anirà actualitzant-se fins que s'aturi el nombre de generacions, i una cel·la propagarà foc a una cel·la veïna només si aquesta no pertany a *rivers*, la qual cosa s'assumeix dins del procés **Propagar foc**.

Així doncs, un cop s'han inicialitzat els paràmetres per a totes les cel·les, s'anirà actualitzant l'estat de la cel·la fins que aquesta estigui a punt de cremar-se, que vol dir que s'haurà d'esgotar la humitat de la cel·la en qüestió per tal que aquesta comenci a cremar-se. Així doncs, si la cel·la s'està cremant, es reduirà la vegetació de la cel·la i a continuació es propagarà el foc a cel·les contínues, que tindran exactament el mateix comportament. Si la cel·la encara té vegetació, aquesta s'anirà reduint fins que ja no hi quedi, en què finalment la cel·la haurà estat cremada per complet.

2.3. Implementació en Python

Un cop definits els diferents comportaments que presenta la nostra quadrícula de simulació, podem passar a explicar el codi que aconsegueix ajuntar totes aquestes regles en una animació bidimensional. El codi està dividit en dos programes: **segona_sessio.py**, l'encarregat de o bé inicialitzar o bé crear les dades, i **Grid.py**, arxiu que conté la classe Grid, amb la qual es creen les diferents simulacions.

segona_sessio

Aquest programa consta dels següents apartats:

- **Paràmetres**: La definició d'una llista de paràmetres, com la mida de la quadrícula en la qual se simularà, el nombre d'iteracions de l'animació (o número de frames), una llista amb tuples on un vulgui iniciar el o els diferents focs i una variable que indiqui si es volen generar noves capes o carregar dels csv's ja creats. També inclouen els paràmetres pel soroll de perlin i el llindar a partir del qual s'omplirà d'aigua.
- **Funcions de generació de capes**: Generen les diferents capes de la quadrícula.
- **Generació de csv's o càrrega de dades**: En cas d'escollir generar les dades, aquestes es generaran i guardaran en els seus arxius .img i csv's corresponents. En cas contrari, es carregaran dels csv's creats anteriorment.
- **Crida a Grid i animació**: Finalment, el programa cridarà a Grid amb l'objectiu de crear l'animació corresponent als paràmetres escollits per l'usuari.

Grid

Grid consta tan sols d'una gran classe anomenada Grid, seguint aquesta l'encarregada de simular i crear l'animació en funció de les dades que hagi rebut en inicialitzar-se. Conté les següents variables de self utilitzades per gestionar el foc:

- **self.fire_status**: Estat del foc en cada cel·la, representant de la capa fire_status.
- **self.fire_start_cells**: Diccionari que emmagatzema les cel·les on s'ha iniciat el foc.
- **self.fire_burning_cells**: Diccionari que emmagatzema les cel·les on el foc està cremant.
- **self.fire_burnt_cells**: Diccionari que emmagatzema les cel·les on el foc ha cremat, aquests últims 3 diccionaris s'utilitzen per eficiència a l'hora d'avaluar cel·les, i així avaluar només aquelles cel·les que hagin patit algun canvi en aquests àmbits.

Aquestes variables són les utilitzades per la resta de capes:

- **self.vegetation**: Quantitat de vegetació en cada cel·la.
- **self.humidity**: Nivell d'humitat en cada cel·la.
- **self.rivers**: Indica si hi ha un riu en cada cel·la.

També ens hem aprofitat en**self.updated_cells**, un vector que indica quines cel·les han estat actualitzades des de l'última iteració, per millorar l'eficiència a l'hora d'escollir els colors de la quadrícula. I, finalment, les següents variables s'utilitzen per emmagatzemar colors i poder representar-los en l'animació. Els colors representen els diferents valors d'humitat, vegetació i foc. La humitat està representada amb el blau i la vegetació amb el verd, tal que les caselles més "grogues" siguin aquelles més seques i amb menys vegetació. Després, un cop hi hagi foc en la cella, es torna tot de color ataronjat, que s'anirà fent fosc a mesura que la vegetació es crema, quedant en negre en haver-se cremat del tot. Finalment, un blau intens representarà les zones amb aigua. Tot això s'emmagatzema en les següents variables:

- **self.humidity_colors**: Diccionari que emmagatzema els colors de la humitat.
- **self.vegetation_colors**: Diccionari que emmagatzema els colors de la vegetació.
- **self.fire_colors**: Diccionari que emmagatzema els colors del foc.

Un cop definides les variables de la classe, les funcions s'encarreguen d'actualitzar la quadrícula en funció de **self.t** (la variable que guarda l'instant o el valor de temps pel qual s'estan fent els càlculs). Aquestes serien unes breus explicacions per les funcions de Grid:

- **__init__**: És l'encarregada d'inicialitzar les diferents capes, diccionaris i rangs de colors.
- **init**: Afegeix totes les cel·les a updated_cells, perquè es pintin els seus colors, i afegeix foc en les cel·les indicades a la llista que serà la variable fire_init.
- **update_fire**: Actualitza les cel·les que tinguin foc o se'ls hagi propagat foc des d'altres cel·les, i a la vegada propaga el foc de cel·les cremant-se a les seves veïnes.
- **update_humity**: Actualitza la humitat d'aquelles cel·les on s'estigui començant un foc.
- **update_vegetation**: Actualitza la vegetació de les caselles que estiguin cremant-se.
- **update_colors**: Canvia els colors de totes les cel·les on s'hagi fet algun canvi
- **execute**: Executa totes les anteriors funcions dins d'una sub-funció **animate**, tal que pugui retornar una animació que s'hagi construït seguint els passos indicats en les funcions i les dades amb les quals s'hagi inicialitzat la classe.

Ús del ChatGPT i copilot dins del codi

A l'hora de desenvolupar el codi que simula la propagació del foc al bosc, ens hem aprofitat d'aquestes dues eines a l'hora d'arrancar el projecte. Les hem notat de gran ajuda a l'hora de construir l'animació de forma molt senzilla. Així i tot, trobem que encara que siguin eines molt espectaculars i funcionen prou bé, a vegades cometia algun que altre fallo, i que molts altres cops no feien exactament el que els hi estaven demanant. Per un altra banda, cal destacar també que a l'hora d'ordenar el que en un inici va ser un codi tot seguit i de funcions, en una classe i el programa d'inicialització ha sigut fet a mà, degut als problemes que donava el ChatGPT en intentar implementar algunes de les idees que teníem, i que vam poder implementar de manera més ràpida i organitzada que tant copilot com el ChatGPT.

2.4. Conclusions

En conclusió, hem pogut definir una sèrie de normes que ens defineixen autòmats cel·lulars de manera que interactuïn entre ells, cosa que per molt que sembli senzilla, ens dona molta flexibilitat i potencia com per fer una simulació senzilla d'esdeveniments importants, com és en aquest cas la propagació d'un incendi en un bosc. Cal remarcar també, que l'ús d'eines com ChatGPT i copilot ha ajudat a l'hora de desenvolupar codi, tot i que les idees principals i el comportament dels autòmats han sigut definides per nosaltres, basant-nos en les explicacions rebudes a classe.