Simulación de Sistemas

Trabajo Práctico 2: Autómata Celular Off-Lattice

Bandadas de Agentes Autopropulsados

Piñeiro, Eugenia Sol - 59449
Baiges, Matías Sebastián - 59076
Comerci Wolcanyik, Nicolás - 59520

3 de septiembre de 2021

Índice general

0.1. Introducción

El objetivo del siguiente trabajo práctico es implementar un Autómata Celular *Off-Latice*, a fin de analizar el comportamiento de partículas puntuales en base a distintos parámetros.

El autómata *Off-Latice* se basa en un modelo matemático inspirado en el comportamiento colectivo de algunos Sistemas Naturales. Estos sujetos biológicos tienden a moverse con sus vecinos. Por ejemplo, bandadas, cardúmenes, manadas de cuadrúpedos o una colonia de bacterias.

En cuanto al sistema, las partículas se desplazan con velocidad absoluta constante. Las mismas, en cada paso de tiempo, asumen el promedio de las direcciones de las partículas vecinas que se encuentran dentro de un radio r con un ruido añadido de forma aleatoria.

Este modelo exhibe un comportamiento cooperativo entre las partículas, las cuales a medida que va avanzando el tiempo comienzan a moverse con una simetría de forma espontánea.

La evolución temporal de las partículas viene dada por las siguientes ecuaciones:

$$x_i(t+1) = x_i(t) + v_i(t)\Delta t \tag{1}$$

donde x es la posición de la partícula, v la velocidad, y $\Delta\ t$ el paso de tiempo

$$\theta_i(t+1) = \langle\theta_i(t)\rangle_r + \Delta\theta \tag{2}$$

donde $\Delta\theta\epsilon[-\frac{\eta}{2};\frac{\eta}{2}]$ es el ruido añadido, siendo η la amplitud del ruido, y $\langle\theta_i(t)\rangle_r$ es el promedio de los ángulos de todas las partículas en un radio r

$$\langle\theta_i(t)\rangle_r = \frac{\langle sin(\theta_i(t))\rangle_r}{\langle cos(\theta_i(t))\rangle_r} \tag{3}$$

0.2. Implementación

La implementación del algoritmo *Off-Lattice* se desarrolló en el lenguaje de programación *Java*

El flujo principal del programa (ver Figura 1) consiste en diversos pasos. En primer lugar, se obtienen los parámetros de configuración para iniciar la

Simulación.

Luego, se generan partículas de forma aleatoria y se van actualizando tanto sus posiciones como sus ángulos con el algoritmo del autómata.

Por último, se guardan los resultados en distintos archivos que serán utilizados como *input* para generar la Animación en *Ovito* y realizar el postprocesamiento en *Python* para hacer un análisis de resultados.

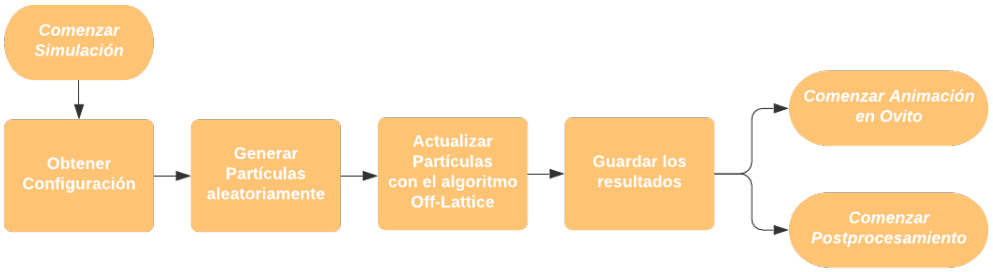

Figura 1: Diagrama de Flujo

En cuanto a la implementación de la Simulación, se cuenta con distintas clases en Java (ver Figura 2). La clase que implementa el flujo principal (main) es *OffLatticeSimulation*. Las partículas se modelaron con la clase *VelocityPaticle*, que extiende de una clase *Particle* y cuenta con su posición, ángulo y rapidez. Las mismas se generan a partir de la clase *VelocityParticleGenerator*.

Para calcular la polarización de las mismas se cuenta con la clase *Polarization*. El algoritmo del autómata se implementó en la clase *OffLattice* que hace uso de las mencionadas anteriormente.

A su vez, se cuenta con las clases para obtener los parámetros de configuración y obtención de resultados *Config, XYZWriter, JsonWriter* y *Postprocessing*, sobre las cuales no se entrará en detalle ya que corresponden al postprocesamiento.

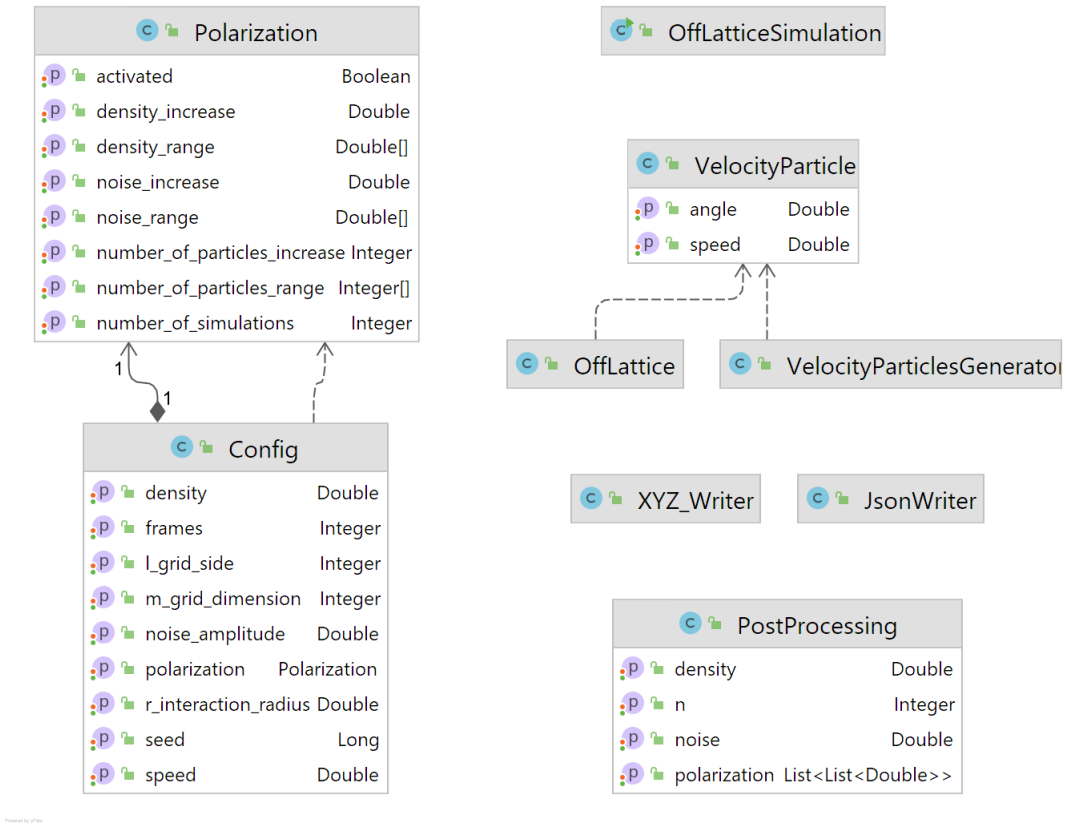

Figura 2: Diagrama UML

En el pseudocódigo correspondiente al flujo principal del programa (ver Algoritmo 1) se puede notar que se sigue el flujo mencionado previamente, en el cual se llama a la función *readValues()* para obtener la configuración del sistema, luego se generan partículas de forma aleatoria invocando la función *generateRandom()*, se llama al autómata Off-Lattice y por último se guardan los resultados en los archivos correspondientes.

En cuanto a la implementación del autómata (ver Algoritmo 2) cabe destacar que por cada pasa de tiempo (frame), en la línea 3 del pseudocódigo, se buscan los vecinos de las partículas a través del *Cell Index Method*. Este método define una grilla de dimensión MxM y se utilizan condiciones de contorno periódicas [1] Luego, se actualizan el ángulo y la posición de cada partícula utilizando las ecuaciones de Evolución Temporal (1) (2) y (3).

Algorithm 1 Off-Lattice Simulation Main

```
1: procedure MAIN
2:     config ← mapper.readValues()
3:     particles ← VelocityParticlesGenerator.generateRandom(config)
4:     frames ← OffLattice.simulate(config)
5:     XYZ_Writer(FILENAME).addAllFrames(frames)
```

Algoritmo 1. Flujo Principal del Programa

Algorithm 2 Off-Lattice Algorithm

```
1: procedure SIMULATE
2: for frame in frames:
3:     neighbours ← CellIndexMethod.search()
4: for particle in neighbours + itself:
5:     newAngle ← atan2(sinAvg, cosAvg) + noise
6:     particle.setAngle(newAngle)
7:     particle.setCoords((x + speed * cos(newAngle)) %L, (y + speed *
   sin(newAngle)) %L)
```

Algoritmo 2. Algoritmo del Autómata Off-Lattice

0.3. Simulaciones

El esquema del sistema (ver Figura 3) consta de una grilla cuyo lado tiene longitud L, en la cual interactúan partículas con velocidad constante v e interactúan dentro un radio r.

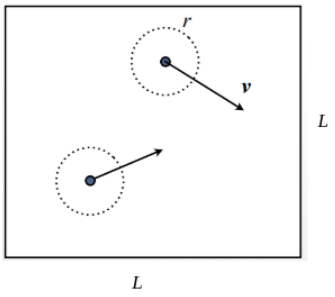

Figura 3: Esquema del Sistema

Para realizar las simulaciones se utilizaron los siguientes parámentros:

- radio de interacción entre partículas $r = 1$
- velocidad absoluta de las partículas $v = 0.03$
- amplitud del ruido $\eta \in [0; 2\pi]$
- lado de la grila L (Variable)
- cantidad de partículas N (Variable)
- densidad $\rho = \frac{N}{L^2}$
- cantidad de pasos *frames* = 2000

En cada simulación se observó la polarización del sistema, la cual se define con la siguiente ecuación:

$$v_a = \frac{1}{Nv} \left| \sum_{i=1}^{N} v_i \right| \tag{4}$$

Si la polarización tiende a cero, las partículas se encuentran en total desorden, mientras que si la polarización tiende a uno se dice que las partículas están polarizadas.

Para correr las simulaciones se utilizarán las siguientes configuraciones:

1. Para analizar el ruido se utilizó N=300, ρ=0.6 y η variable
2. Para analizar la densidad se utilizó N=300, η=0.8 y ρ varibale
3. Densidad y ruido pequeños: N = 300, L = 25, ($\rho = 0.48$), $\eta = 0.1$
4. Densidad y ruido grandes: N = 300, L = 7, ($\rho = 6.12$), $\eta = 2.0$
5. Densidad grande y ruido pequeño: N = 300, L = 5, ($\rho = 12$), $\eta = 0.1$

0.4. Resultados

A partir de las configuraciones mencionadas previamente, se obtuvieron los siguientes resultados tras analizar el efecto del ruido y la densidad.

0.4.1. Análisis del Ruido

Luego de realizar simulaciones bajo las configuraciones propuestas (Ver figuras 4, 5 y 6), se observó que a medida que el ruido aumentaba, las particulas tendían a dispersarse, dirigiendose hacia distintas direcciones. Es decir, manteniendo la cantidad de partículas y el ruido constante, /emphal aumentar el ruido, la polarización disminuye.

En caso de poco ruido, se podía observar como las partículas al

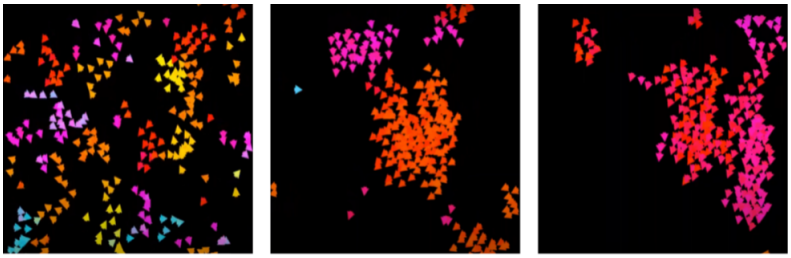

Figura 4: Capturas de los frames inicial, intermedio y final de una Animación con N=300; densidad=0.6 y ruido=0.5

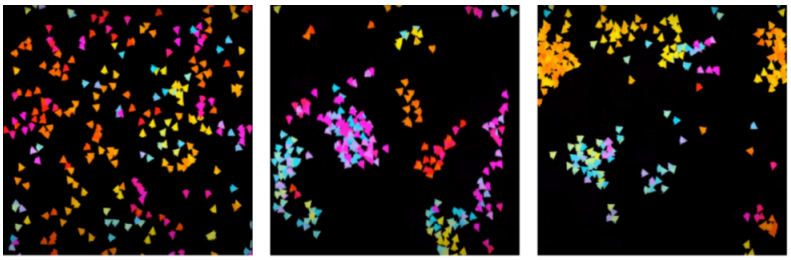

Figura 5: Capturas de los frames inicial, intermedio y final de una Animación con N=300; densidad=0.6 y ruido=1.5

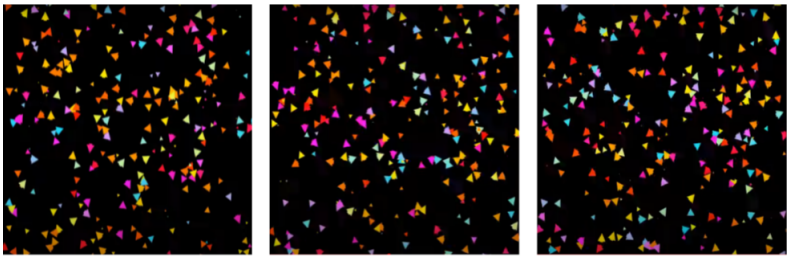

Figura 6: Capturas de los frames inicial, intermedio y final de una Animación con N=300; densidad=0.6 y ruido=5

0.4.2. Análisis de la Densidad

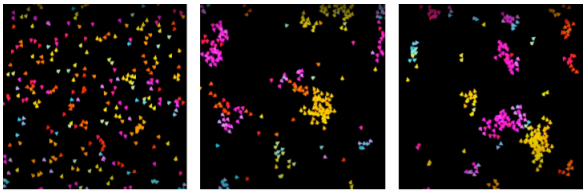

Figura 7: Capturas de los frames inicial, intermedio y final de una Animación con N=300; densidad=0.25 y ruido=0.8

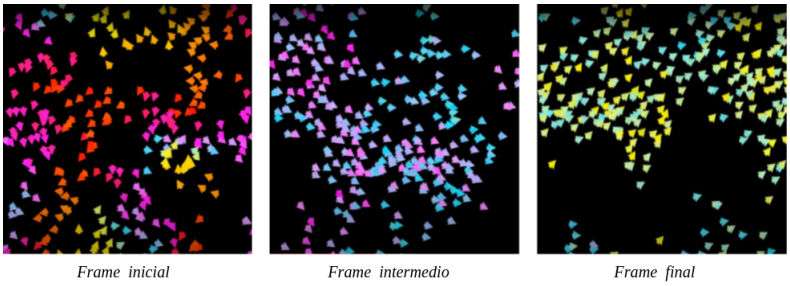

Figura 8: Capturas de los frames inicial, intermedio y final de una Animación con N=300; densidad=3 y ruido=0.8

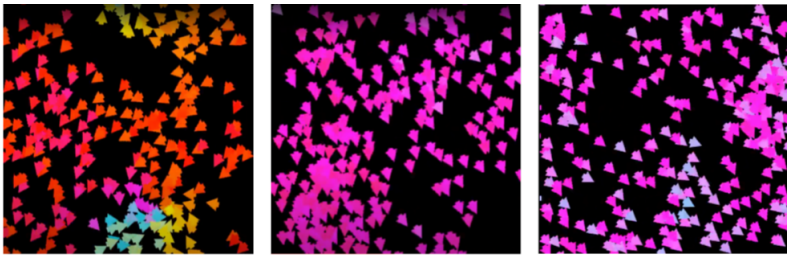

Figura 9: Capturas de los frames inicial, intermedio y final de una Animación con N=300; densidad=6 y ruido=0.8

0.4.3. Comportamientos Destacados

Se utilizaron distintas configuraciones propuestas en el paper [1] con el objetivo de evaluar distintos comportamientos destacados del sistema.

En primer lugar, se utilizaron valores de densidad y ruido pequeños y se pudo notar que las partículas tienden a formar grupos moviendose de forma coherente en direcciones aleatorias (ver Figura xxxx)

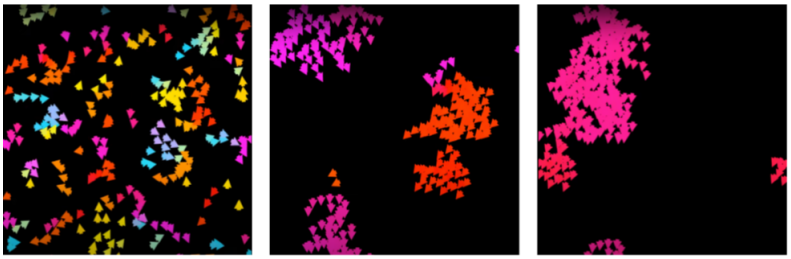

Figura 10: Capturas de los frames inicial, intermedio y final de una Animación con N = 300; L = 25; ($\rho = 0.48$); $\eta = 0.1$

Luego, para valores de densidad y ruido grandes se notó que las partículas se desplazan de forma aleatoria pero con cierta correlación.

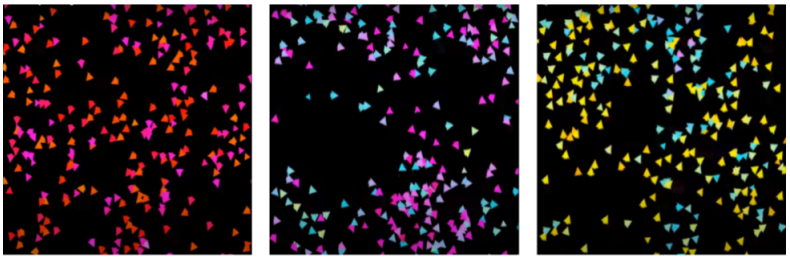

Figura 11: Capturas de los frames inicial, intermedio y final de una Animación con N = 300; L = 7; ($\rho = 6.12$); $\eta = 2.0$

Por último, con una densidad grande y un ruido pequeño se notó que el movimiento se vuelve ordenado para todas las partículas.

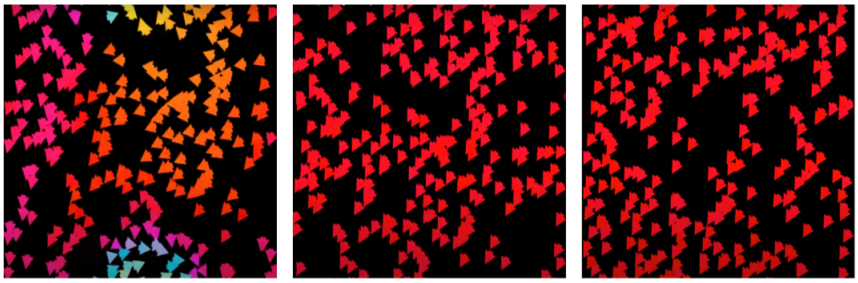

Figura 12: Capturas de los frames inicial, intermedio y final de una Animación con N = 300; L = 5; ($\rho = 12$); $\eta = 0.1$

0.5. Conclusiones

Tomando como base los resultados obtenidos a partir de las simulaciones, se llegó a la conclusión que, al aumentar el ruido disminuye la polarización y al aumentar la densidad de partículas aumenta la polarización.

A su vez, se puede observar que el sistema presenta distintos comportamientos según los parámetros elegidos.

Bibliografía

[1] Tamás Vicsek, András Czirók, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet. Novel type of phase transition in a system of self-driven particles. *Physical review letters*, 75(6):1226, 1995.