

TP/2: Autómata Off-Lattice: **Bandadas de agentes** **auto-propulsados**



Grupo 5

- © Golmar, Agustín
- © Lobo, Daniel Alejandro



Fundamentos



“

*Investigar el agrupamiento,
transporte y transición de fases de
partículas en sistemas que no
están en equilibrio.*



Regla de Evolución

En cada momento, la dirección de una partícula está condicionada por las direcciones de sus partículas vecinas.

$$\theta_{t+1} = \langle \theta_t \rangle_{r_c} + \Delta\theta$$

$$\langle \theta_t \rangle_{r_c} = \text{atan2} \left(\frac{\langle \sin \theta \rangle_{r_c}}{\langle \cos \theta \rangle_{r_c}} \right)$$



Aplicaciones

Sistemas biológicos que incluyen
agrupamiento y migración: cardúmenes,
bandadas, colonias de bacterias, etc.



Modelos



Modelos Relevantes

- ◎ **Mobile Particle**
- ◎ **Particle Generator**
- ◎ **Cell Index Method**
- ◎ **Cellular Automaton**

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are double-lined, and the connections are thin grey lines. The diagram is partially cut off by the top and left edges of the slide.

Implementación

Parámetros de Operación

- ◎ Cantidad de ciclos/iteraciones
- ◎ Delta de tiempo por ciclo de evolución
- ◎ Cantidad de partículas
- ◎ Longitud del espacio bidimensional
- ◎ Radio de interacción
- ◎ Módulo de la velocidad de las partículas
- ◎ Amplitud de ruido

Cell Index Method

- ❖ Para calcular vecinos de forma más eficiente: $O(n)$.
- ❖ Pre-computa celdas que no estén vacías: mejoras a **baja densidad!!!**

Mobile Particle

- ❖ Es inmutable.
- ❖ Extiende de **Particle**.

```
public class MobileParticle extends Particle {  
  
    protected final double vx;  
    protected final double vy;  
  
    public MobileParticle()  
  
    public double getVx() {  
  
    public double getVy() {  
  
    public double getθ() {  
  
    public double getDegrees() {  
  
    public double getSpeed() {  
  
    public MobileParticle move(final double Δt) {  
  
    public MobileParticle rotateTo()  
  
    public MobileParticle rotateTo(final double angle) {  
  
    public String toString() {  
  
}
```

Cellular Automaton

Mobile Particle →

.move(...) →

.rotateTo(...)

→ **SINCRÓNICO!!!**

```
public class CellularAutomaton {  
  
    protected final double  $\eta$ ;  
    protected final double  $\Delta t$ ;  
    protected final double interactionRadius;  
    protected final MobileGenerator generator;  
    protected final DistanceProcessor processor;  
    protected final Space space;  
    protected final BiConsumer<Integer, List<MobileParticle>> spy;  
  
    protected CellularAutomaton(final Builder builder) {}  
  
    public static Builder from(final MobileGenerator generator) {}  
  
    public CellularAutomaton advance(final int step) {}  
  
    public double getOrder() {}  
  
    private double  $\theta$ (final List<Particle> neighbours) {}  
  
    private double  $\Delta\theta$ () {  
        return (Math.random() - 0.5) *  $\eta$ ;  
    }  
  
    public static final class Builder {}  
}
```

Main


```
// cycles dt N L RC V noise
private static void generate(String[] args, final long start) {
    Integer cycles = Integer.valueOf(args[0]);
    Double dt = Double.valueOf(args[1]);
    Integer N = Integer.valueOf(args[2]);
    Double L = Double.valueOf(args[3]);
    Double RC = Double.valueOf(args[4]);
    Double V = Double.valueOf(args[5]);
    Double noise = Double.valueOf(args[6]);

    // Generador móvil:
    final MobileGenerator generator = MobileGenerator.of(N)
        .maxRadius(0)
        .speed(V)
        .over(L)
        .spy(System.out::println)
        .build();

    // El autómata celular:
    Output output = Output.getInstance();
    final CellularAutomaton automaton = CellularAutomaton.from(generator)
        .spy((k, ps) -> {
            // Se ejecuta para cada frame:
            output.write(ps, k);
            // console logging:
            //System.out.println("Simulación " + k);
            //ps.stream().forEach(System.out::println);
        })
        .interactionRadius(RC)
        .spaceOf(L)
        .step(dt)
        .noise(noise)
        .build();

    // Para ver el orden antes de simular:
    System.out.println("Orden: " + automaton.getOrder());

    // Simulación:
    IntStream.range(0, cycles)
        .forEachOrdered(automaton::advance);
}
```

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by circles of varying sizes, some with concentric rings, and the lines are thin and grey. The diagram is partially cut off by the left edge of the slide.

Pruebas y Resultados

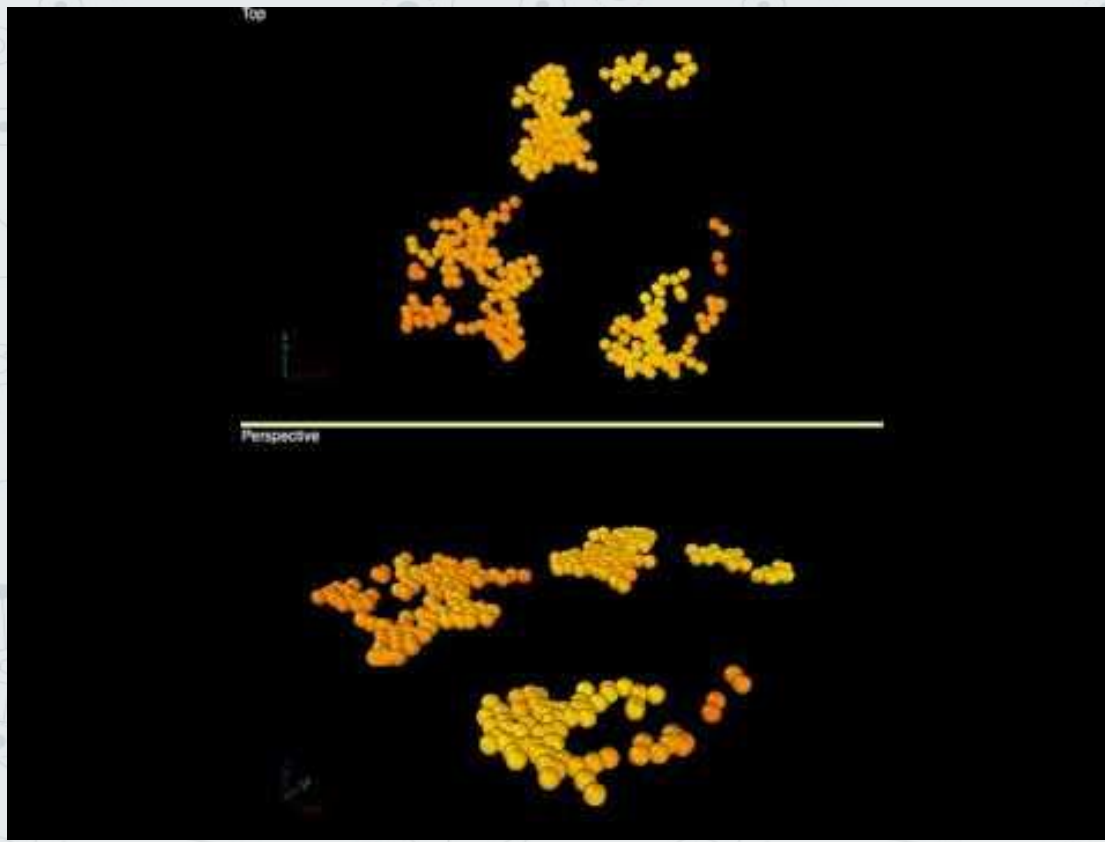
Formato del Archivo de Output

El algoritmo arroja como resultado un archivo de texto con el siguiente formato:

```
number_of_particles
number_of_cycle
position_x_particle_1 position_y_particle_1 angle
position_x_particle_2 position_y_particle_2 angle
position_x_particle_3 position_y_particle_3 angle
...
position_x_particle_n position_y_particle_n angle
```

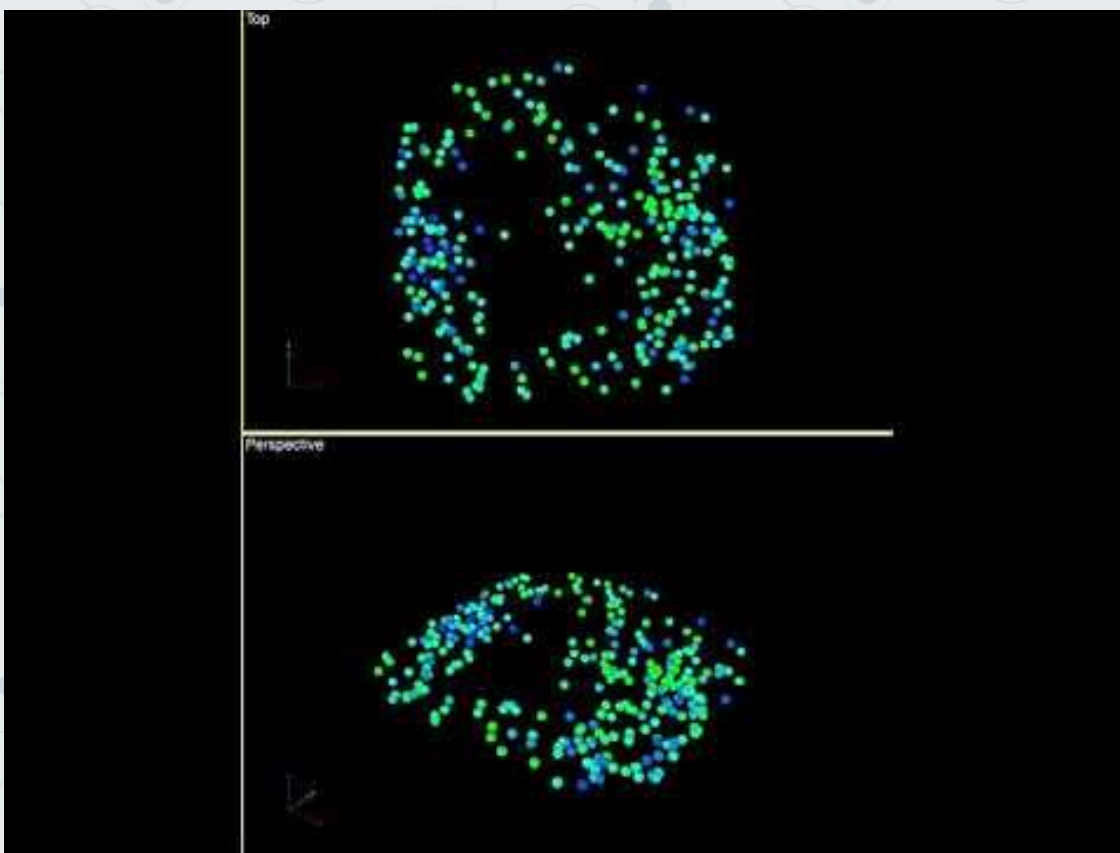

Parámetros Fijos

⦿	Módulo de la velocidad de las partículas	0.03 m/s
⦿	Tiempo de simulación	5000 segundos
⦿	Delta de tiempo de evolución	1.0 segundo
⦿	FPS en animaciones	4 FPS
⦿	Radio de interacción	1 metro



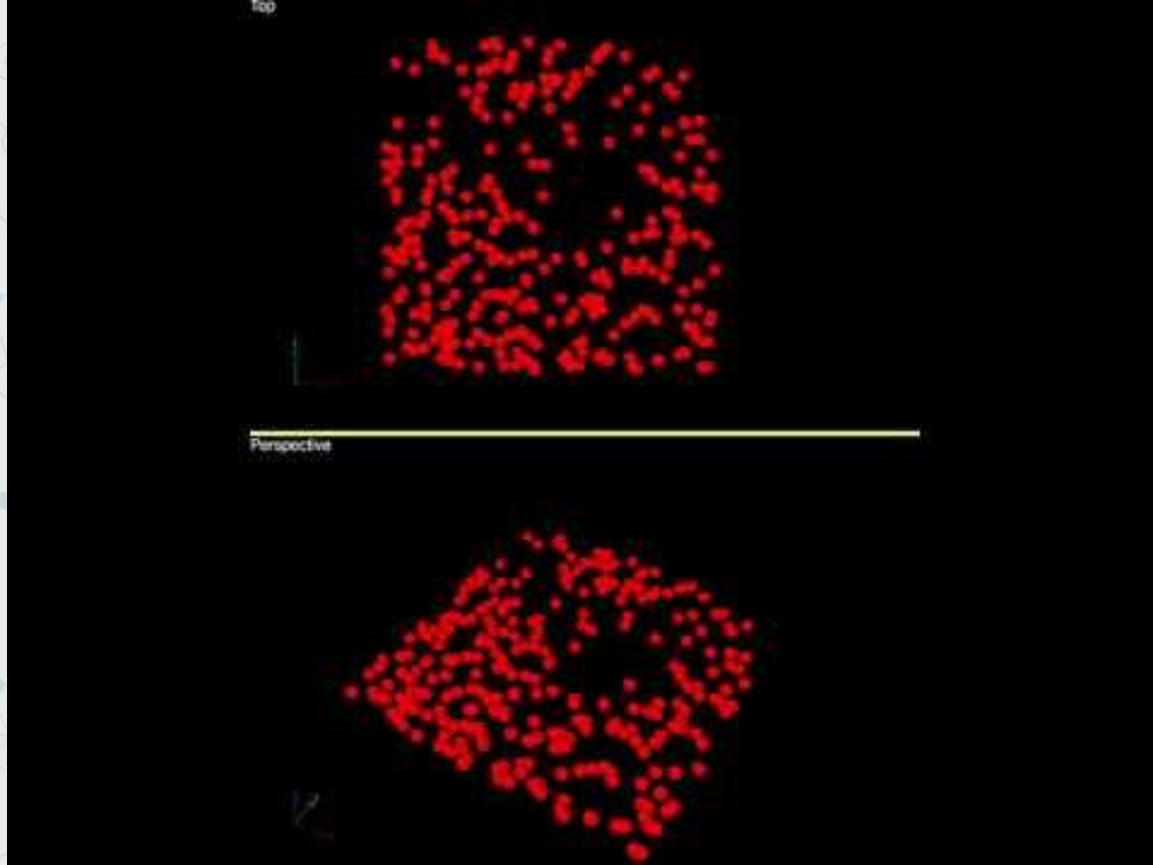
Animación 1: Grupos Coherentes

5000 cycles | **1.0** Δt | **300** partículas | **25** L | **1.0** Rc | **0.03** v | **0.1** noise



Animación 2: Aleatorio con Correlación

5000 cycles | **1.0** Δt | **300** partículas | **7** L | **1.0** Rc | **0.03** v | **2** noise



Animación 3: Movimiento Ordenado

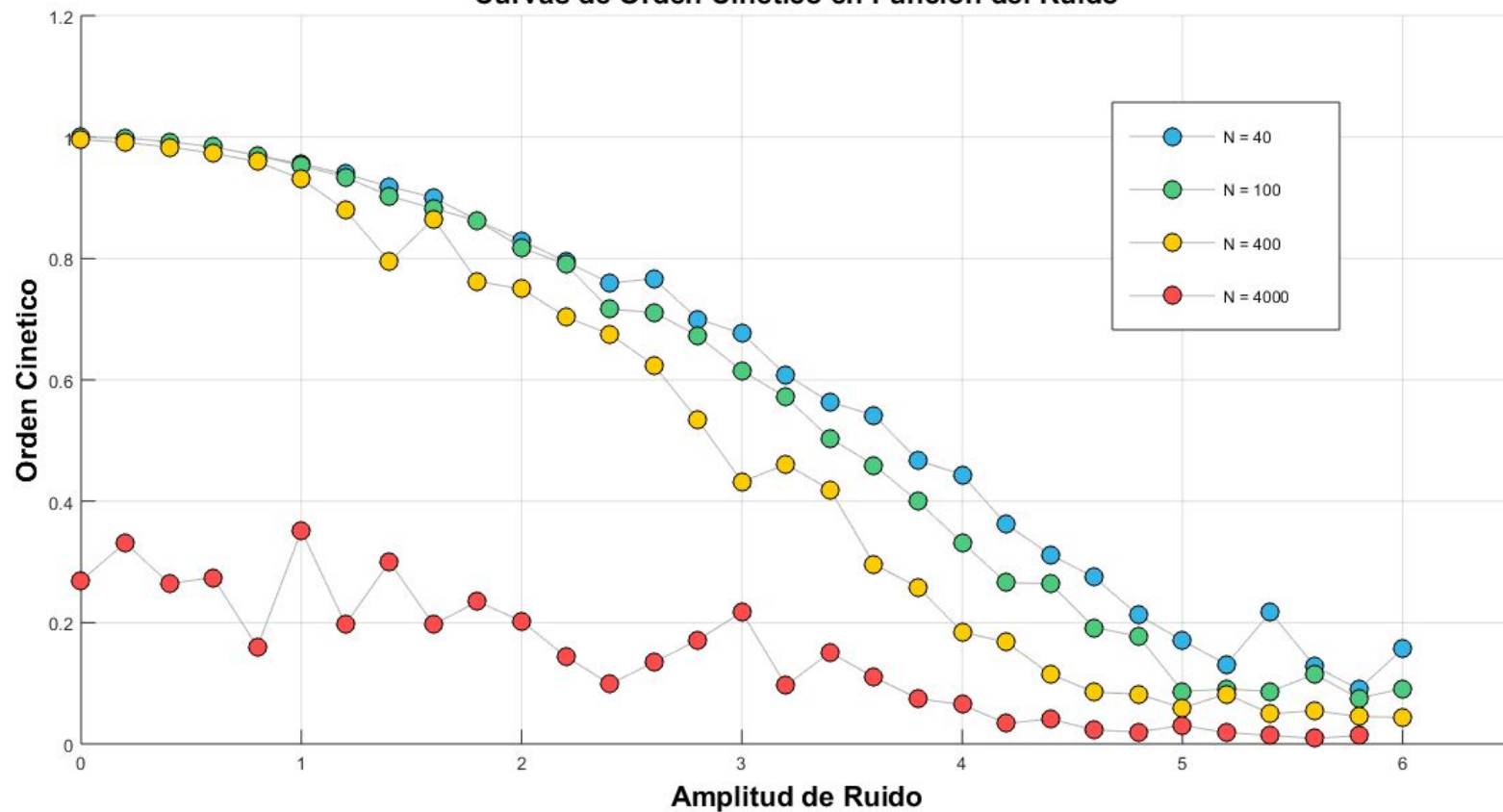
5000 cycles | **1.0** Δt | **300** partículas | **5** L | **1.0** Rc | **0.03** v | **0.1** noise



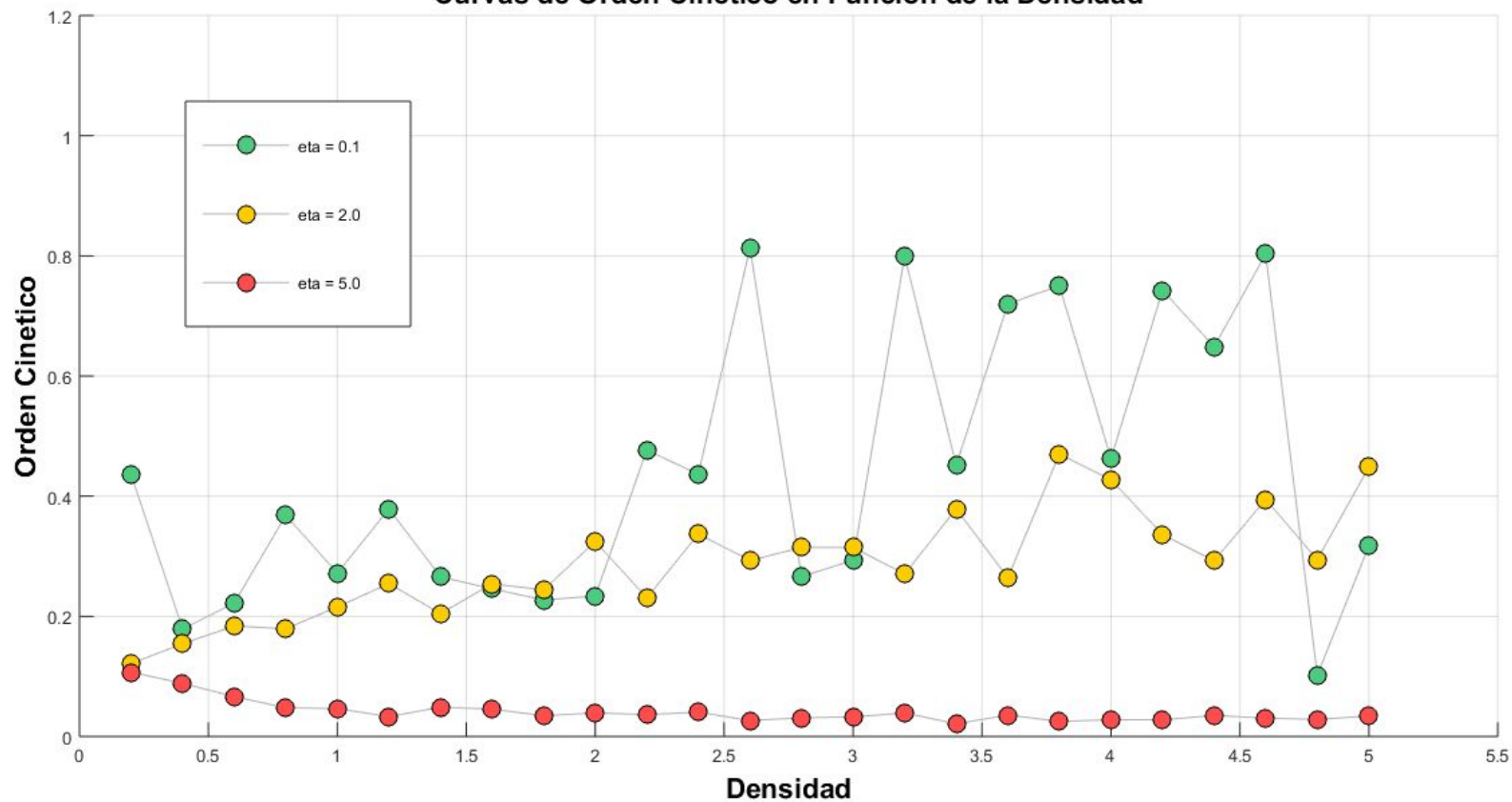
Gráficos



Curvas de Orden Cinetico en Funcion del Ruido



Curvas de Orden Cinetico en Funcion de la Densidad





Conclusiones



Conclusiones

- ◎ El comportamiento se controla solo mediante la **densidad** (ρ), y la **interferencia** (η).
- ◎ **Baja densidad** implica mayor resistencia a la interferencia.
- ◎ Ningún sistema soporta una interferencia **destructiva**.
- ◎ **Alta densidad** se beneficia de bajas interferencias. Demasiado caos es contraproducente.
- ◎ Interacción a **corta distancia** implica estructuras emergentes y *building-blocks* (lo que explica los modelos de baja densidad).



Gracias!