PRÁCTICA COMPUTACIONAL

Evolución de la Cooperación: Dilema del Prisionero Iterado

Replicación Axelrod & Hamilton (1981) - Actualización 2026

Curso:	Técnicas Computacionales Avanzadas
Profesor:	Dr. Leonardo González Tejeda
Créditos:	15% de la calificación final
Modalidad:	Equipos de 3 estudiantes
Duración:	2 semanas

OBJETIVO

Replicar el experimento clásico de Axelrod & Hamilton (1981) utilizando técnicas computacionales modernas de 2026 para analizar la evolución de estrategias cooperativas en el Dilema del Prisionero Iterado. El entregable principal será un dashboard interactivo que permita explorar resultados y ejecutar simulaciones en tiempo real.

MARCO TEÓRICO

Dilema del Prisionero: Matriz de pagos estándar con T=5, R=3, P=1, S=0 donde T>R>P>S. La estrategia TIT FOR TAT (cooperar en primera ronda, luego copiar acción previa del oponente) demostró ser evolutivamente estable cuando $w \geq \max((T-R)/(T-P), (T-R)/(R-S)) \approx 0.5$, donde w es la probabilidad de interacción futura.

REQUERIMIENTOS TÉCNICOS

Hardware: CPU multi-core (4+ cores), 8GB RAM mínimo
Software: Python 3.11+, NumPy 1.26+, Pandas 2.1+, Matplotlib 3.8+, SciPy 1.11+, Plotly 5.18+ (para dashboard), Dash 2.14+ o Streamlit 1.29+
RNG Obligatorio: numpy.random.Generator con PCG64 (seed reproducible)

OPCIONES DE EJERCICIO

Seleccionar UNA opción y completarla exhaustivamente con dashboard interactivo.

OPCIÓN 1: TORNEO ROUND-ROBIN CLÁSICO

Descripción: Implementar torneo con 10 estrategias clásicas + 5 modernas. Cada par juega 5 veces con 200 rondas por juego.
Estrategias requeridas: TIT FOR TAT, GRIM, PAVLOV, ALL-D, ALL-C, TIT FOR TWO TATS, RANDOM, JOSS, GRADUAL, ADAPTIVE, EVOLVED-NN (red neuronal 2 capas), PSO-PLAYER, MEMORY-3, FRIEDMAN, TESTER.
Tareas: (1) Implementar todas las estrategias con clase base común, (2) Ejecutar torneo completo con w=0.995, (3) Generar matriz de scores, (4) Identificar ganador y top-3, (5) Análisis estadístico con ANOVA.
Dashboard debe incluir: Selector de estrategias, matriz de payoffs interactiva (heatmap con hover), ranking dinámico, gráfica head-to-head configurable, histograma de distribución de scores, controles para ajustar w y matriz de pagos en tiempo real.
Hints: • TIT FOR TAT: if round==0: return 'C'; else: return opp_history[-1] • Usar numpy.random.Generator(PCG64(seed=42)) • Verificar T>R>P>S en cada juego • Para ADAPTIVE: actualizar probabilidades con Bayesian updating • Plotly heatmap con go.Heatmap() o px.imshow()

OPCIÓN 2: ALGORITMO GENÉTICO PARA DESCUBRIR ESTRATEGIAS

Descripción: Usar AG para evolucionar estrategias representadas como Autómatas Finitos Determinísticos (DFA) de 6 estados. Objetivo: superar a TIT FOR TAT.

Representación: DFA con estados S■...S■, entradas {CC,CD,DC,DD}, salidas {C,D}. Cromosoma: vector binario codificando transiciones y salidas.
Tareas: (1) Codificar DFA como cromosoma, (2) Implementar AG con DEAP (población=100, generaciones=50, Cx=0.7, Mut=0.2), (3) Fitness=score promedio contra baseline (ALL-D, TFT, PAVLOV, GRIM), (4) Analizar top-3 estrategias evolucionadas, (5) Visualizar DFAs como grafos.
Dashboard debe incluir: Animación de evolución generación por generación, gráfica de fitness (mejor, promedio, peor), visualización interactiva de DFAs con Cytoscape, comparador de estrategias evolucionadas vs clásicas, control para ejecutar nuevas evoluciones con diferentes parámetros (tamaño población, Cx, Mut).
Hints: • Longitud cromosoma: n_states*(1 + 4*log2(n_states)) bits • Usar tools.selTournament(tournsize=3) • Elitismo: preservar top 5% • Diversidad: medir Hamming distance promedio • Dash Cytoscape para visualizar grafos de DFA • Guardar historial completo de evolución para animación

OPCIÓN 3: DEEP REINFORCEMENT LEARNING (DQN/PPO)

Descripción: Entrenar agente RL usando DQN o PPO (Stable-Baselines3) para aprender estrategia óptima contra TIT FOR TAT, luego generalizar a múltiples oponentes.
Ambiente: Crear Gymnasium environment custom. Estado: últimas 5 acciones de ambos jugadores. Acción: {0=D, 1=C}. Recompensa: payoff inmediato. Episodio termina probabilísticamente con w=0.995.
Tareas: (1) Implementar IPD como gym.Env, (2) Entrenar DQN (500k steps) contra TFT, (3) Entrenar PPO (500k steps) contra TFT, (4) Curriculum learning: ALL-C → TFT → PAVLOV → Mixed, (5) Evaluar en torneo contra 10 estrategias clásicas, (6) Comparar DQN vs PPO.
Dashboard debe incluir: Curvas de aprendizaje en tiempo real (rewards, episode lengths), visualizador de política aprendida (mapa de estados → acciones), simulador interactivo (usuario juega contra agente entrenado), comparación DQN vs PPO en múltiples métricas, control para cargar diferentes checkpoints de entrenamiento, análisis de comportamiento emergente con ejemplos de juegos.
Hints: • Gamma debe ser ≈ w para valorar futuro correctamente • Si reward no sube tras 50k steps, ajustar learning_rate • PPO más estable pero más lento que DQN • Guardar checkpoints cada 50k steps • Usar callbacks de SB3 para logging • Streamlit permite modo 'juega contra el agente' fácilmente

OPCIÓN 4: SIMULACIÓN BASADA EN AGENTES CON REDES COMPLEJAS

Descripción: Modelar evolución de cooperación en redes con Mesa framework. Comparar topologías: scale-free (Barabási-Albert), small-world (Watts-Strogatz), lattice, random (Erd■s-Rényi).
Parámetros: N=1000 agentes, 500 generaciones, 50% inicial cooperadores (TFT), 50% defectores (ALL-D). Regla actualización: imitación (imitar vecino con mayor score).
Tareas: (1) Implementar modelo Mesa con 4 topologías, (2) Experimento 1: comparar topologías, (3) Experimento 2: invasión de defectores (99% TFT, 1% ALL-D), (4) Experimento 3: variar mutation_rate [0, 0.001, 0.01, 0.05, 0.1], (5) Análisis estadístico (ANOVA) comparando % cooperadores final entre topologías.
Dashboard debe incluir: Visualización de red animada (nodos cambian de color según estrategia), gráfica de evolución temporal (% cooperadores vs generación), controles para pausar/continuar/reiniciar simulación, selector de topología con regeneración dinámica, sliders para N, mutation_rate, proporción inicial, estadísticas en tiempo real (grado promedio, clustering, assortativity), modo comparación lado-a-lado de 2 topologías.
Hints: • Usar networkx para crear redes • Para scale-free: nx.barabasi_albert_graph(N, m=4) • Mantener grado promedio ≈6 en todas las topologías • Plotly Network Graph con go.Scatter para visualización • Para animación fluida, actualizar cada 10 generaciones • Mesa tiene servidor integrado pero Dash da más control • Precomputar simulaciones y cargar en dashboard para fluidez

OPCIÓN 5: ANÁLISIS DE SENSIBILIDAD Y CONDICIONES LÍMITE

Descripción: Análisis exhaustivo de sensibilidad del modelo de Axelrod a parámetros clave. Identificar condiciones límite donde cooperación colapsa o emerge inesperadamente.
Experimentos: (1) Variar w: [0.1, 0.3, 0.5, 0.7, 0.85, 0.95, 0.99, 0.999], (2) Variar matriz de pagos: T/R ∈ [1.2, 1.5, 2.0, 2.5, 3.0], (3) Introducir ruido: 'trembling hand' con error_rate ∈ [0%, 1%, 5%, 10%], (4) Variar N: [10, 50, 100, 500, 1000, 5000] para estudiar drift estocástico.
Tareas: Para cada experimento ejecutar 50 réplicas, calcular media y desviación estándar de % cooperadores final, identificar phase transitions (cambios cualitativos abruptos), ajustar modelos de regresión (logística o splines) para predecir cooperación en función de parámetros.
Dashboard debe incluir: Tabs para cada experimento, sliders para ajustar parámetros y ver actualización en tiempo real, superficie 3D interactiva (2 parámetros vs cooperación), gráficas de phase transitions con intervalos de confianza, comparador de condiciones (seleccionar 2+ puntos y ver diferencias), tabla resumen con estadísticas, exportador de datos configurables en CSV/JSON.
Hints: • Paralelizar con joblib.Parallel(n_jobs=-1) para precomputar datos • Phase transition aparece cuando derivada es máxima • Plotly Surface con go.Surface() para 3D • Guardar grid completo de resultados precalculados • Dashboard carga datos y permite exploración interactiva • Para ruido: con probabilidad error_rate ejecutar acción opuesta • Verificar predicción teórica: TFT óptimo cuando w>0.5

IMPLEMENTACIÓN OBLIGATORIA: RANDOM NUMBER GENERATOR

Código mínimo requerido en todos los ejercicios:

```
import numpy as np from numpy.random import PCG64, Generator class ReproducibleRNG: def __init__(self, seed: int = 42): self.seed = seed self.rng = Generator(PCG64(seed)) def uniform(self, low=0.0, high=1.0, size=None): return self.rng.uniform(low, high, size) def choice(self, a, size=None, replace=True, p=None): return self.rng.choice(a, size, replace, p) def integers(self, low, high=None, size=None): return self.rng.integers(low, high, size) # Uso rng = ReproducibleRNG(seed=20260211) # YYYYMMDD random_value = rng.uniform(0, 1) random_action = rng.choice(['C', 'D'])
```

Tests de Calidad RNG (incluir en dashboard como anexo técnico):
• Test Kolmogorov-Smirnov para uniformidad (p-value > 0.05) • Gráfica de autocorrelación (lag plot) • Histograma de 10■ muestras

ENTREGABLES FINALES

1. DASHBOARD INTERACTIVO (60% de la evaluación)

Tecnología: Dash (Plotly) o Streamlit (elegir uno según preferencia del equipo)
Funcionalidad mínima: • Navegación multi-página o tabs para diferentes secciones
• Visualizaciones interactivas (mínimo 6) con controles dinámicos (sliders, dropdowns, checkboxes)
• Simulaciones ejecutables desde el dashboard (con barra de progreso)
• Exportación de resultados (CSV, JSON, imágenes PNG/SVG)
• Diseño responsive y profesional (considerar colores colorblind-friendly)
• Documentación inline con tooltips explicativos
• Sección de comparación de estrategias/resultados
• Panel de tests de calidad RNG como anexo técnico
Estructura recomendada: Home (intro + paper original), Simulación (ejecutar experimentos), Resultados (visualizaciones principales), Análisis (estadísticas y comparaciones), Anexos (RNG tests, metodología, código fuente destacado)
Despliegue: Local (con instrucciones claras en README) o en cloud (Streamlit Cloud, Render, Heroku)

2. ANEXOS TÉCNICOS (20% de la evaluación)

A. Código fuente completo: • Estructura modular: main.py, strategies.py (o módulo relevante), dashboard.py, utils.py, rng_tests.py
• requirements.txt con versiones específicas
• README.md con instrucciones de instalación y uso (incluir screenshots del dashboard)
• Comentarios y docstrings en todo el código
B. Documentación técnica (PDF, 5-8 páginas): • Metodología detallada con pseudocódigo o diagramas de flujo
• Decisiones de diseño y arquitectura del sistema
• Análisis de complejidad computacional
• Limitaciones y posibles extensiones
• Referencias en formato APA (mínimo 5)
C. Datos: • Resultados experimentales en CSVs (mínimo 3 archivos con datos crudos)
• Metadata JSON describiendo estructura de datos
• Scripts de procesamiento de datos si aplica

3. PRESENTACIÓN Y EXPOSICIÓN GRUPAL (20% de la evaluación)

Formato: 15 minutos de exposición + 5 minutos de preguntas
Estructura obligatoria: 1. Introducción (2 min): Contexto del Dilema del Prisionero y relevancia actual
2. Metodología (3 min): Descripción técnica del enfoque elegido, arquitectura del sistema
3. Demostración en vivo del dashboard (5 min): Ejecutar simulaciones, mostrar funcionalidades clave
4. Resultados principales (3 min): Hallazgos más importantes, comparación con paper original
5. Conclusiones (2 min): Aprendizajes, limitaciones, trabajo futuro
Requisitos: • Todos los integrantes deben participar activamente
• Slides profesionales (máximo 12 slides, mínimo 10pt font size)
• Demostración en vivo del dashboard funcionando
• Preparar respuestas a preguntas técnicas sobre implementación
• Incluir al menos 2 visualizaciones impactantes del dashboard
Evaluación: Claridad técnica, calidad de visualizaciones, dominio del tema, coordinación del equipo, manejo de preguntas

ESPECIFICACIONES TÉCNICAS DEL DASHBOARD

Componente	Dash (Plotly)	Streamlit
Estructura	Multi-page con dash.page_registry	Multi-page con st.pages
Gráficas	plotly.graph_objects, plotly.express	plotly charts + st.plotly_chart()
Controles	dcc.Slider, dcc.Dropdown, dcc.Checklist	st.slider(), st.selectbox(), st.multiselect()
Layout	html.Div + dbc.Row/Col (dash-bootstrap)	st.columns(), st.sidebar
Callbacks	@app.callback con Input/Output	Automático con st.button, reruns
Deploy	Render, Railway, pythonanywhere	Streamlit Cloud (gratis, fácil)

REFERENCIAS ESENCIALES

Paper original:
• Axelrod, R. & Hamilton, W.D. (1981). "The Evolution of Cooperation". *Science*, 211(4489), 1390-1396.
Libros recomendados:
• Axelrod, R. (2006). *The Evolution of Cooperation* (Revised Ed.). Basic Books.
• Nowak, M.A. (2006). *Evolutionary Dynamics: Exploring the Equations of Life*. Harvard Univ. Press.
Papers complementarios:
• Nowak, M.A. & Sigmund, K. (1993). "A strategy of win-stay, lose-shift". *Nature*, 364, 56-58.
• Santos, F.C. & Pacheco, J.M. (2005). "Scale-free networks provide a unifying framework". *PRL*, 95, 098104.
Documentación técnica:
• Dash: https://dash.plotly.com/tutorial • Streamlit: https://docs.streamlit.io
• DEAP: https://deap.readthedocs.io • Stable-Baselines3: https://stable-baselines3.readthedocs.io
• Mesa: https://mesa.readthedocs.io • NetworkX: https://networkx.org

CONTACTO Y SOPORTE

Profesor: Dr. Leonardo González Tejeda | **Email:** leonardo.gonzalez@tec.mx | **Horario consultas:** Ma-Ju 16:00-18:00 | **Sesión de dudas dashboard:** Jueves de la semana 1, 17:00-19:00 (obligatoria asistencia de al menos 1 integrante por equipo)

CRITERIOS DE ÉXITO DEL PROYECTO

✓ Dashboard completamente funcional con todas las características especificadas
✓ Reproducibilidad total: cualquier persona puede clonar el repo y ejecutar todo
✓ Código limpio, modular y bien documentado
✓ Visualizaciones profesionales y autoexplicativas
✓ Presentación clara que demuestre dominio técnico del equipo
✓ Comparación explícita con resultados del paper original de Axelrod
✓ Tests de RNG implementados y presentados en anexo técnico del dashboard