Sinergia de Probabilidades, Optimización y Aprendizaje Automático para la Toma de Decisiones

Autor:
Lic. Enzo Rojas D'Toste

Tutor:
Dra. Aymée de los Ángeles Marrero Severo

Universidad de La Habana
Facultad de Matemática y Computación

Mayo 2025

Abstract

This work tackles decision-making problems in which an agent must choose, at each step, one option from a finite—or even countably infinite—set of candidates. Because the underlying optimisation tasks are NP-Complete, exact methods become infeasible as the problem size grows. We therefore propose a *probabilistic assignment framework*: a continuous function $f : [0, 1) \to [0, n)$ maps a uniform random value to a decision index, so that each option receives a tunable probability of being selected.

Three families of assignment functions are studied:

1. **Inverted logarithmic** curves, which strongly favour the best-ranked choices while preserving exploration.
2. **Inverse proportional** and **inverse exponential** curves, both featuring a vertical asymptote ($x \to 1^-$), allowing us to handle countably infinite sets of decisions. The first is more aggressive (assigns higher probability to extreme values); the second is more conservative.

Parameters of f are optimised with Particle Swarm Optimisation (PSO). When static parameters are insufficient, a Transformer—fed with a sequence reduced by Singular Value Decomposition (SVD)—produces *dynamic* parameters conditioned on the current state.

Case studies include an *imperfect-memory matching game*, a *risk-based betting scenario*, and the *minimum vertex-cover* problem on graphs. Across hundreds of random instances, the proposed approach never underperforms a greedy baseline and improves solution quality, all while maintaining comparable computational complexity. Hence, the probabilistic framework, coupled with meta-heuristic or neural optimisation, offers a robust and extensible alternative for difficult combinatorial decision problems.

Resumen

Este trabajo aborda problemas de toma de decisiones en los que, en cada paso, un agente debe escoger una opción dentro de un conjunto finito—o incluso infinito numerable—de candidatos. Dado que las variantes exactas son NP-Completas, los métodos exhaustivos se vuelven inviables a gran escala. Se propone, por tanto, un *marco de asignación probabilística*: una función continua $f : [0, 1) \to [0, n)$ transforma un número aleatorio uniforme en un índice, asignando a cada decisión una probabilidad ajustable.

Se analizan tres familias de funciones:

1. Curvas **logarítmicas inversas**, que favorecen con fuerza la mejor opción sin excluir las demás.
2. Curvas de **proporcionalidad inversa** y **inversa exponencial**, ambas con asíntota vertical en $x = 1$, adecuadas para conjuntos infinitos de decisiones; la primera es más agresiva, la segunda más conservadora.

Los parámetros de f se optimizan con *Particle Swarm Optimisation* (PSO). Cuando los parámetros estáticos no bastan, un Transformer—cuya entrada se reduce mediante *SVD*—genera parámetros dinámicos en función del estado actual.

Los casos de estudio incluyen un *juego de memoria con olvido*, un *entorno de apuestas con riesgo* y el problema del *cubrimiento mínimo de vértices* en grafos. Sobre cientos de instancias aleatorias, la estrategia jamás rinde peor que el algoritmo voraz (greedy) y mejora la solución, manteniendo complejidad temporal comparable.

En consecuencia, el marco probabilístico, combinado con optimización meta-heurística o neuronal, constituye una alternativa sólida y extensible para problemas combinatorios de decisión complejos.

Índice

1. Introducción **4**
1.1. Contexto, Motivación y Relevancia del Trabajo . . . 4
1.2. Planteamiento del Problema . . . 4
1.3. Propuesta y Enfoque . . . 4
1.3.1. Hipótesis de la Tesis . . . 5
1.4. Objetivos . . . 5
1.4.1. Objetivo General . . . 5
1.4.2. Objetivos Específicos . . . 5
1.5. Campo de Acción . . . 5

2. Estructura y Descripción General **6**
2.1. Componentes del algoritmo . . . 6
2.2. Pruebas y Resultados . . . 6

3. Estado del Arte **8**
3.1. Funciones de asignación probabilística para decisiones . . . 8
3.2. Metaheurísticas (PSO) para ajustar parámetros probabilísticos . . . 8
3.3. Transformers en decisiones secuenciales y RL . . . 8
3.4. SVD como técnica de reducción dimensional previa a modelos deep/Transformer . . . 9

4. Función de Asignación Probabilística de Decisiones **10**
4.1. Descripción General . . . 10
4.2. Condiciones de la Función . . . 10
4.3. Interpretación Probabilística . . . 10
4.4. Ventajas de la Aproximación . . . 11
4.5. Inversión de la Función Logarítmica . . . 11
4.5.1. Motivación del Uso de un Logaritmo . . . 11
4.5.2. Definición General . . . 11
4.5.3. Despeje de y . . . 11
4.5.4. Condiciones de Borde . . . 12
4.5.5. Solución para d_x y d_y . . . 12
4.5.6. Función Resultante $y(x)$ . . . 12
4.5.7. Interpretación . . . 13
4.6. Caso con $n \to +\infty$: Función de Proporcionalidad Inversa . . . 13
4.6.1. Definición General . . . 13
4.6.2. Condiciones de Borde . . . 13
4.6.3. Cálculo de d_x y d_y . . . 14
4.6.4. Función Final $y(x)$ . . . 14
4.6.5. Ajuste de la Estructura . . . 14
4.6.6. Forma Explícita . . . 15
4.6.7. Interpretación y Uso Práctico . . . 15
4.7. Combinación de Múltiples Funciones Probabilísticas . . . 15

5. Optimización de Parámetros Mediante Metaheurísticas **16**
5.1. Necesidad de la Optimización de Parámetros . . . 16
5.2. Metaheurísticas: Una Vía de Uso General . . . 16
5.3. Ejemplos de Metaheurísticas . . . 16
5.4. Enjambre de Partículas (PSO): Principios Básicos . . . 17
5.5. Aplicación al Caso de Asignación Probabilística . . . 17
5.6. Conclusiones . . . 18

6. Selección Dinámica de Parámetros Mediante Redes Neuronales **18**
6.1. Motivación de la Selección Dinámica . . . 18
6.2. Uso de Redes Neuronales para la Selección Dinámica . . . 18
6.2.1. Entrenamiento Supervisado . . . 18
6.3. Dimensión Variable en la Entrada . . . 19
6.4. Arquitectura de tipo Transformer . . . 19
6.4.1. Manejo de Estados Variados . . . 20
6.5. Proceso General de Entrenamiento . . . 20
6.6. Reflexiones y Proyecciones . . . 20
6.7. Manejo de la Escalabilidad: Reducción de Dimensionalidad con SVD . . . 21
6.7.1. Transformar la Secuencia de Estados en una Matriz 2D . . . 21
6.7.2. Aplicación de la Descomposición en Valores Singulares (SVD) . . . 21
6.7.3. Representación Fija y Compacta . . . 21
6.7.4. Beneficios para el Modelo . . . 22
6.7.5. Reflexiones Finales . . . 22

7. Pruebas y Resultados: Primer Caso de Estudio **23**
7.1. Descripción del Juego y Motivación . . . 23
7.1.1. Limitaciones de Memoria y Confusión . . . 23
7.2. Estrategia de Decisión Probabilística . . . 23
7.3. Configuración de las Pruebas . . . 24
7.4. Usando PSO . . . 24
7.4.1. Función Objetivo del PSO . . . 24
7.4.2. Configuración Experimental . . . 25
7.4.3. Visualización de la Función Logarítmica Invertida con Parámetros Obtenidos . . . 25
7.4.4. Comparación de Resultados con la Selección por Mayor Valor Esperado . . . 27
7.5. Usando Transformer . . . 28
7.5.1. Entrenamiento y Construcción del Conjunto de Datos . . . 28
7.5.2. Comparativa con PSO: Inferencia Dinámica de la Confusión . . . 29

8. Aplicación en un Entorno de Riesgo **30**
8.1. Modelo y Ajuste Mediante Proporcionalidad Inversa . . . 30
8.1.1. Pruebas y Resultados con PSO . . . 30
8.2. Pruebas con la Inversa de la Exponencial . . . 31
8.2.1. Configuración Experimental y Resultados . . . 31
8.2.2. Análisis de Resultados . . . 32
8.3. Combinando ambas Funciones . . . 32
8.3.1. Resultados y Análisis . . . 32

9. Mínimo Conjunto de Vértices de Cobertura **34**
9.1. Estrategia Aproximada Greedy . . . 34
9.2. Aplicando la Solución Propuesta . . . 34
9.3. Estrategia Conjunta: Mezclando el Greedy y el Algoritmo Probabilístico . . . 34
9.4. Pruebas con Grafos Aleatorios de Hasta 50 Nodos . . . 35
9.4.1. Generando y Comparando con el Óptimo . . . 35
9.4.2. Uso de la Función Logarítmica Inversa . . . 35
9.5. Entrenamiento de un Transformer . . . 35
9.6. Conclusiones . . . 36

10. Conclusiones Generales **37**
10.1. Repositorio de Código . . . 38

1 Introducción

1.1 Contexto, Motivación y Relevancia del Trabajo

Los problemas de toma de decisiones son fundamentales en diversas áreas de la ciencia, la ingeniería y la industria. Estos problemas surgen cuando es necesario elegir entre múltiples alternativas, optimizando un conjunto de objetivos bajo ciertas restricciones. Ejemplos notables incluyen la planificación de rutas en logística, la selección de inversiones en finanzas, el diagnóstico asistido por computadora en medicina, y la navegación autónoma en robótica.

En la actualidad, la resolución de problemas de toma de decisiones enfrenta diversos desafíos. La incertidumbre inherente a los datos y las condiciones dinámicas de los entornos reales complican el proceso de toma de decisiones, y la creciente complejidad de los problemas plantea altas demandas computacionales, lo que exige el desarrollo de algoritmos eficientes y escalables.

En este contexto, los enfoques basados en probabilidades, algoritmos de optimización y aprendizaje automático han demostrado ser herramientas poderosas. Las probabilidades permiten modelar la incertidumbre de manera formal, los algoritmos de optimización ofrecen métodos efectivos para encontrar soluciones óptimas, y el aprendizaje automático proporciona la capacidad de adaptar las estrategias a partir de datos. Sin embargo, integrar estas herramientas en un marco único y eficiente sigue siendo un desafío abierto.

Este trabajo busca abordar este desafío mediante el diseño de un algoritmo híbrido que combine probabilidades, optimización y aprendizaje automático para resolver problemas complejos de toma de decisiones. Esta combinación no solo tiene el potencial de superar las limitaciones de enfoques tradicionales, sino también de abrir nuevas posibilidades en aplicaciones reales, desde la industria hasta la investigación científica.

1.2 Planteamiento del Problema

El problema central que aborda esta tesis es el diseño y la estructuración de un algoritmo capaz de tomar decisiones óptimas dentro de un conjunto (finito o infinito contable) de opciones. Este algoritmo debe garantizar un resultado final que maximice el cumplimiento de los objetivos planteados en un problema específico.

La complejidad principal radica en la diversidad de los problemas y en la naturaleza dinámica del conjunto de decisiones. En muchos casos, este conjunto no es estático, sino que puede variar tanto en la cantidad de elementos como en sus características a medida que el problema evoluciona.

El objetivo es que el algoritmo sea lo suficientemente robusto y adaptable para encontrar estrategias efectivas independientemente de la naturaleza y las variaciones del problema. Esto requiere integrar de manera eficiente herramientas probabilísticas, de optimización y de aprendizaje automático, creando un enfoque capaz de abordar esta dinámica de manera generalizada y efectiva.

1.3 Propuesta y Enfoque

El algoritmo propuesto utiliza una función matemática adaptable que varía en función del problema a resolver. Esta función recibe como entrada un valor aleatorio generado de manera uniforme y devuelve un número entre 0 y n (sin incluir n), donde n representa el tamaño (finito o infinito contable) del conjunto de decisiones posibles. El valor devuelto se trunca a un entero, que se utiliza como índice para seleccionar una decisión de las opciones disponibles.

El orden de las decisiones está determinado por una función de *fitness*, la cual evalúa qué tan buena es cada decisión basada en un criterio heurístico específico. La función matemática, por lo tanto, define intervalos probabilísticos de confianza para cada decisión. Sin embargo, debido a la complejidad inherente de los problemas abordados, seleccionar la decisión con el mejor *fitness* no siempre garantiza un resultado óptimo, o incluso satisfactorio.

La optimización en este contexto consiste en ajustar los parámetros de la función matemática para definir los intervalos probabilísticos que maximicen los resultados. Este proceso otorga un poder significativo al criterio heurístico utilizado, el cual en muchos casos será lo suficientemente potente, como se demostrará en los resultados experimentales. Sin embargo, esta estrategia presenta limitaciones, ya que al elegir los parámetros de la función de forma discreta, se adopta un comportamiento probabilístico que no considera el estado actual o pasado del problema. Esto implica que dos instancias del mismo problema que podrían seguir trayectorias distintas usarán la misma estrategia, lo que disminuye la capacidad del algoritmo para adaptarse dinámicamente.

Para superar esta limitación, se introduce el uso del aprendizaje automático, específicamente redes neuronales, como parte del enfoque. Estas redes permiten ajustar los parámetros de la función de forma continua, basándose en el estado actual del problema. Esto proporciona al algoritmo una mayor flexibilidad y potencia, ya que los parámetros ya no se seleccionan de forma discreta, sino que se determinan mediante una "función continua"generada por la red neuronal. Este enfoque mejora la adaptabilidad del algoritmo y amplía significativamente su capacidad para abordar problemas complejos y dinámicos.

1.3.1 Hipótesis de la Tesis

La integración de un mecanismo de asignación probabilística, una estrategia de optimización y el uso de redes neuronales para ajustar dinámicamente los parámetros de decisión, permite encontrar soluciones más efectivas y adaptables que las obtenidas con métodos estáticos o puramente deterministas en problemas de toma de decisión, maximizando así el beneficio global.

1.4 Objetivos

1.4.1 Objetivo General

Brindar una herramienta computacional basada en el algoritmo que sea adaptable a diferentes problemas de decisión, dando flexibilidad para que los usuarios definan la estructura y criterios específicos de sus casos.

1.4.2 Objetivos Específicos

- Explicar la estructura del algoritmo, incluyendo sus componentes de probabilidades, optimización y aprendizaje automático.
- Validar el algoritmo propuesto aplicándolo a casos de estudio representativos de distintos problemas de decisión.
- Analizar los resultados obtenidos en comparación con métodos existentes, evaluando su eficacia, eficiencia y adaptabilidad.
- Brindar la herramienta computacional que permita la configuración de problemas de decisión de forma parametrizable y flexible.

1.5 Campo de Acción

- **Áreas de aplicación:** logística, planificación de recursos, diseño de sistemas, toma de decisiones empresariales, sistemas de control y automatización, entre otros.
- **Usuarios potenciales:** investigadores, profesionales en ciencias de la computación, ingenieros de sistemas, analistas de datos, y cualquier persona interesada en resolver problemas de decisión de manera óptima.
- **Alcance del algoritmo:** la herramienta será capaz de adaptarse a diferentes problemas al permitir que los usuarios definan las estructuras y criterios heurísticos necesarios para modelar sus casos particulares. Además, será posible integrar mecanismos de aprendizaje automático para ajustar dinámicamente los parámetros según las condiciones del problema.

2 Estructura y Descripción General

2.1 Componentes del algoritmo

A continuación se presenta una visión global de los componentes principales que conforman el algoritmo propuesto. Cada uno de ellos se detallará en las siguientes secciones con ejemplos y fundamentos teóricos:

- **Función de Asignación Probabilística:** Se describen diversas formulaciones matemáticas (logarítmica inversa, etc.) que, partiendo de un valor aleatorio uniforme, permiten asignar probabilidades a un conjunto (finito o infinito) de decisiones. Se explican los requisitos de borde que garantizan una distribución coherente, y cómo estas funciones pueden favorecer, de manera controlada, unas decisiones sobre otras sin descartar la exploración de soluciones menos atractivas.

  Además, **se introduce la posibilidad de combinar varias de estas funciones** en una *función maestra* que seleccione probabilísticamente cuál función usar en cada momento. De este modo, se maximizan la *flexibilidad* y la capacidad de adecuación del modelo a distintas situaciones del problema, ampliando considerablemente el alcance de la aproximación probabilística.

- **Optimización de Parámetros:** Se introduce la necesidad de ajustar parámetros (por ejemplo, $\{a, b, c, d_x, d_y, \dots\}$) para distintos problemas de decisión. A fin de lograr este ajuste, se recurre a metaheurísticas (por ejemplo, optimización por enjambre de partículas, PSO) que permiten explorar eficazmente el espacio de soluciones sin requerir gradientes o modelos analíticos simples.

- **Aprendizaje Automático para la Selección Dinámica:** Se explica cómo, una vez determinados parámetros óptimos a nivel global o en subinstancias del problema, pueden usarse redes neuronales para *aprender* una función que ajuste dichos parámetros de forma dinámica, en tiempo real, conforme evoluciona el problema. Se abordan también las limitaciones de las redes neuronales clásicas para manejar entradas de tamaño variable y se presenta el uso de arquitecturas tipo *Transformer*, junto con técnicas de reducción de dimensionalidad (por ejemplo, SVD) para afrontar el reto de la escalabilidad.

2.2 Pruebas y Resultados

- **Juego de Confusión de Memoria:** En esta sección se presenta el uso de **PSO** para encontrar parámetros óptimos de la función logarítmica inversa en el juego de parejas de cartas con *memoria imperfecta*, así como la implementación de un **Transformer** entrenado con ejemplos generados por PSO. Primero, se mostraron los resultados de la búsqueda de parámetros en escenarios de distinta *confusión* (α) y cómo dicha búsqueda evitó que las partidas se volvieran "infinitas". Luego, se comparó esta estrategia con la selección determinista de la *mayor esperanza de acierto*, evidenciando la superioridad de PSO a medida que la confusión crece. Finalmente, se introdujo el *Transformer*, capaz de adaptarse dinámicamente a casos específicos sin requerir conocer α de antemano y aprovechando la reducción de dimensionalidad via **SVD** para garantizar la escalabilidad. Los resultados muestran que, si bien PSO conserva ventaja en algunas franjas de confusión, la arquitectura neuronal supera la solución puramente metaheurística en confusiones intermedias y se perfila como una vía más potente conforme se amplía el conjunto de entrenamiento y el tiempo de aprendizaje.

- **Ganancias en un Entorno de Riesgo:** En esta sección se describe múltiples experimentos realizados para demostrar cómo distintas *funciones probabilísticas* (proporcionalidad inversa, inversa exponencial) o su *combinación bajo una función maestra* pueden aplicarse exitosamente a un *problema de apuestas en un entorno de riesgo* desconocido, modelado mediante la probabilidad $\alpha \sin(i)$. Se mostró que:

  - **Proporcionalidad Inversa:**
    - Potencial de ganar enormes sumas de capital, especialmente si la confusión/riesgo es bajo o moderado.
    - El precio a pagar es la posibilidad de sufrir un número de pérdidas más alto en escenarios de mayor volatilidad (α).
  - **Inversa Exponencial:**

- Opción más conservadora, con notable reducción de pérdidas a costa de obtener ganancias promedio de menor magnitud.
- Proporciona estabilidad en confusiones elevadas, pero no alcanza los picos que logra la función de proporcionalidad inversa en ambientes de riesgo moderado.

- **Combinación de Estrategias con Función Maestra:**
  - Permite *conciliar* las ventajas de ambas funciones, decantándose por la más conveniente en cada situación.
  - En entornos de bajo riesgo, la función maestra favorece la proporcionalidad inversa para maximizar ganancias; en riesgos moderados a altos, puede preferir la inversa exponencial para minimizar pérdidas.
  - Ante confusiones extremas (α muy alta), el algoritmo ajusta las apuestas a valores próximos a cero, evitando así la bancarrota incluso si el margen de ganancia es bajo.

En conjunto, estas pruebas evidencian la **flexibilidad** de la aproximación: al poder elegir entre distintas funciones o combinarlas dinámicamente, el algoritmo responde de forma eficaz a variaciones en el nivel de riesgo, manteniendo un *balance* entre el incremento potencial del capital y el número de pérdidas que se está dispuesto a tolerar.

- **Mínimo Conjunto de Vértices de Cobertura:** Esta sección presenta, a grandes rasgos, cómo aplicar el marco probabilístico al **problema del Mínimo Conjunto de Vértices de Cobertura**. Tras recordar que el problema es NP-Completo, se describe la heurística *greedy* habitual (elegir el nodo de mayor grado en cada paso) y se señala su alta efectividad, aunque no siempre garantiza la solución óptima. Para reducir esos fallos se introduce una **función de asignación probabilística** que, en lugar de forzar siempre el grado máximo, asigna probabilidades a todos los nodos, fomentando cierta exploración; sus parámetros se ajustan con PSO para entrenar un Transformer, alimentado con estados comprimidos vía SVD, en los casos más difíciles. Finalmente se evalúa la estrategia conjunta (greedy + probabilístico), destacando que la combinación nunca empeora y puede mejorar las coberturas en los grafos donde el método voraz resulta subóptimo.

3 Estado del Arte

A continuación se relacionan los componentes del algoritmo con paralelos relevantes hallados en la literatura.

3.1 Funciones de asignación probabilística para decisiones

El uso de funciones que asignan probabilidades a cada decisión está bien fundamentado en la literatura de toma de decisiones bajo incertidumbre y aprendizaje por refuerzo. En estos enfoques, en lugar de seleccionar determinísticamente la mejor opción, se utiliza una función (por ejemplo, una política en un MDP) que devuelve una distribución de probabilidad sobre las acciones posibles en cada estado. Esto permite decisiones estocásticas informadas, equilibrio entre exploración-explotación, y modelar preferencias con incertidumbre. En la teoría de decisiones de Markov y en RL clásico se define la política $\pi(s, a)$ precisamente como la probabilidad de elegir la acción a estando en el estado s. Por ejemplo, Kaelbling et al. (1996) [1] discuten que una política puede considerarse un mapeo del estado a una distribución de probabilidad sobre acciones. De igual forma, Sutton y Barto (2018) [2] explican que en una política estocástica $\pi(a|s)$ asigna a cada acción a una probabilidad dada la situación s. Esta idea respalda este componente, ya que usar funciones probabilísticas para decidir introduce aleatoriedad controlada y está avalado por trabajos fundamentales en RL.

3.2 Metaheurísticas (PSO) para ajustar parámetros probabilísticos

El empleo de metaheurísticas de optimización como Particle Swarm Optimization (PSO) para aprender o ajustar los parámetros de las funciones probabilísticas de decisión ha sido explorado en múltiples trabajos técnicos. La idea es tratar los parámetros de la función (por ejemplo, pesos de una política o coeficientes que determinan las probabilidades) como una solución candidata a optimizar mediante búsqueda global estocástica. PSO, en particular, ha mostrado eficacia en sintonizar políticas sin gradientes, explorando el espacio de parámetros para maximizar rendimiento. Hein et al. (2016) [3] introdujeron una política basada en PSO (PSO-P) que optimiza secuencias de acción en entornos de control continuo, evidenciando que PSO puede reemplazar métodos gradientes para encontrar políticas cercanas al óptimo. Más recientemente, France & Sheppard (2023) [4] usan PSO para co-entrenar políticas en RL, ajustando los parámetros de la política en paralelo a la iteración de aprendizaje por refuerzo tradicional. Estos estudios respaldan este componente, mostrando que PSO y otras metaheurísticas pueden aprender parámetros de asignación probabilística (p. ej., pesos de una red de política o distribuciones de decisión) de forma efectiva, incluso en espacios de gran dimensión.

3.3 Transformers en decisiones secuenciales y RL

El uso de Transformers para abordar problemas de decisión secuencial, planificación por secuencias y aprendizaje por refuerzo ha ganado soporte en años recientes. Los Transformers, originalmente exitosos en secuencias de texto, han probado ser modelos eficaces para secuencias de estados/acciones debido a su capacidad de modelar dependencias de largo alcance. Por ejemplo, Chen et al. (2021) [5] propusieron el Decision Transformer, que reformula el problema de RL como uno de modelado de secuencia condicionada: el modelo (un Transformer causal) toma como entrada la secuencia de estados, acciones pasadas y una recompensa objetivo, y predice las acciones futuras óptimas. Este enfoque secuencial logró desempeños competitivos con algoritmos RL tradicionales, validando la efectividad de Transformers en RL offline. Asimismo, Janner et al. (2021) [6] introdujeron el Trajectory Transformer, entrenado con secuencias completas de transiciones (estado, acción, recompensa) para planificar en entornos MDP tratándolo "como un gran problema de modelado de secuencias". Esto mostró que un Transformer puede aprender dinámicas y políticas simultáneamente, facilitando la planificación mediante técnicas como beam search sobre el modelo entrenado. Además, en entornos de RL online, Parisotto et al. (2020) [7] demostraron que un Transformer modificado (la arquitectura GTrXL, Gated Transformer-XL) puede usarse en lugar de LSTMs para dotar de memoria a un agente, logrando mayor estabilidad y rendimiento en tareas secuenciales complejas. En conjunto, estos trabajos respaldan este componente, mostrando que los Transformers son modelos efectivos para decisiones secuenciales, ya sea formulando el control como un problema de predicción de secuencias o incorporándolos como parte del agente de RL para procesar historia y planificar acciones.

3.4 SVD como técnica de reducción dimensional previa a modelos deep/Transformer

La descomposición en valores singulares (SVD) es una técnica clásica para reducción de dimensionalidad que ha sido aplicada como paso de preprocesamiento antes de alimentar modelos de deep learning, incluyendo Transformers. El objetivo de aplicar SVD es obtener una representación compacta de los datos de entrada (extrayendo sus componentes principales) y así disminuir la dimensionalidad antes de la etapa de aprendizaje, reduciendo ruido y complejidad. Numerosos trabajos han explorado esta combinación. Por ejemplo, en señales temporales de mantenimiento predictivo, He et al. (2020) [8] utilizan SVD multi-resolución para extraer las características más importantes de series de vibración de un motor, antes de ingresarlas a una red LSTM que predice la degradación del sistema. Esto mostró mejoras en la precisión del pronóstico, atribuidas a que SVD filtró componentes irrelevantes y redujo la dimensión de entrada del LSTM. De forma similar, en visión por computador Kang & Kim (2013) [9] aplican SVD para extraer rasgos reducidos de señales eléctricas de motores y luego clasificarlas con modelos de diagnóstico, logrando alta exactitud con muchos menos atributos. En general, la literatura reporta que incorporar SVD (o PCA, su equivalente) antes de redes neuronales puede acelerar el entrenamiento y evitar sobreajuste al eliminar redundancias. Este uso frecuente respalda este componente, SVD es una herramienta válida para reducir dimensionalidad de entradas complejas (imágenes, secuencias, matrices de características) antes de alimentar modelos Transformers o deep learning, tal como han validado estudios en dominios de imágenes satelitales (integrando SVD antes de un 3D-CNN para clasificación hiperespectral), en detección de intrusos en redes (usando SVD para preprocesar vectores de características) y otros. En resumen, hay evidencia técnica de que una fase de SVD puede simplificar los datos de entrada conservando la información esencial, mejorando potencialmente el rendimiento del modelo subsecuente.

4 Función de Asignación Probabilística de Decisiones

En esta sección se describe una función matemática que, a partir de un valor aleatorio uniforme en el intervalo $[0, 1)$, permite asignar un índice válido dentro de un conjunto finito (o potencialmente infinito) de decisiones. El objetivo es que cada decisión reciba una probabilidad de ser elegida, cumpliendo con la condición de que la suma de todas estas probabilidades sea igual a 1.

4.1 Descripción General

Sea n el número de decisiones posibles, donde n puede ser un valor entero finito o tender a $+\infty$. Definimos una función

$$f : [0, 1) \to [0, n),$$

cuyo propósito es mapear un valor aleatorio $u \in [0, 1)$ (generado de manera uniforme) a un valor real en el rango $[0, n)$. A continuación, se trunca el valor obtenido, esto es,

$$\text{decision\_index} = \lfloor f(u) \rfloor,$$

para seleccionar un índice de decisión dentro del conjunto $\{0, 1, 2, \ldots, n-1\}$ (o hasta $+\infty - 1$ si n no está acotado superiormente).

4.2 Condiciones de la Función

Para garantizar una distribución probabilística coherente, la función $f(u)$ debe ser continua y cumplir con los siguientes requisitos:

1. **Dominio y Rango:** El dominio debe ser el intervalo $[0, 1)$, mientras que el rango se extiende hasta $[0, n)$. Esto asegura que un valor de entrada u (aleatorio uniforme) se convierta en un índice válido de decisión.

2. **Condiciones de Borde:**
$$u \in [0, 1), \quad \begin{cases} \exists\, u : & f(u) = 0, \\ \exists\, u : & \lim_{x \to u} f(x) = n \end{cases}$$
Esta característica garantiza que la función cubra todas las decisiones posibles cuando u varía desde 0 hasta valores cercanos a 1.

3. **Asignación de Probabilidades:** El uso de un valor aleatorio uniforme u y la partición del intervalo $[0, 1)$ en subintervalos que correspondan a cada decisión asegura que la suma de las probabilidades asignadas a todas las decisiones sea 1.

4.3 Interpretación Probabilística

Si se analiza la función $f(u)$ como un mecanismo de asignación de probabilidades, cada decisión $d \in \{0, 1, \ldots, n-1\}$ queda representada por el subconjunto de valores $u \in [0, 1)$ tales que

$$\lfloor f(u) \rfloor = d.$$

La medición de Lebesgue [10] (básicamente, la longitud) de ese subconjunto en $[0, 1)$ representa la probabilidad de que la decisión d sea elegida. Por definición de la variable u como uniforme en $[0, 1)$, la suma de dichas mediciones para todas las decisiones será igual a la longitud total del intervalo, es decir, 1.

En consecuencia, el algoritmo puede controlar la probabilidad asignada a cada decisión ajustando la forma de la función f. Por ejemplo, si para ciertos valores de u se hace crecer más rápido (o más lento) la función, la distribución de probabilidades variará, otorgando mayor (o menor) peso a decisiones específicas.

4.4 Ventajas de la Aproximación

- **Flexibilidad:** Cualquier función $f(u)$ que cumpla con las condiciones anteriores es válida, lo cual permite diseñar distintos perfiles de asignación de probabilidad según el problema de decisión.
- **Escalabilidad:** Si n tiende a infinito (por ejemplo, en escenarios con un número muy grande o dinámico de decisiones), la forma de f puede adaptarse para seguir cumpliendo con las restricciones de borde.
- **Simplicidad:** El mapeo directo de un número real en $[0, 1)$ a un índice entero es intuitivo y fácil de implementar.

4.5 Inversión de la Función Logarítmica

Se presenta un ejemplo concreto de cómo construir la función $f(x)$ descrita en la sección empleando un logaritmo. El objetivo principal es ilustrar cómo asignar probabilidades a un conjunto de n decisiones (ordenadas según un criterio *fitness*) de tal manera que las primeras decisiones tengan mayor probabilidad de ser escogidas, sin descartar por completo las que se consideran menos prometedoras.

4.5.1 Motivación del Uso de un Logaritmo

La elección de una función logarítmica surge de la idea de favorecer de forma significativa las decisiones con mejor *fitness*, pero permitiendo que, con menor probabilidad, también se seleccionen opciones menos sobresalientes. De esta manera, se logra un equilibrio entre la *explotación* de las mejores decisiones y la *exploración* de soluciones potencialmente útiles en el largo plazo.

4.5.2 Definición General

Partimos de la forma:

$$x \;=\; b\,\log_a\Big(c\,(y - d_x)\Big) \;+\; d_y,$$

donde:

- $a > 0 \wedge a \neq 1$ es la base del logaritmo.
- $b \neq 0$ es un factor de escala que ajusta la pendiente de la curva.
- $c > 0$ es un factor de escala dentro del logaritmo.
- d_x y d_y son términos de traslación en los ejes de x y y, respectivamente.

El objetivo es *despejar* y en función de x. Además, se desea que esta función satisfaga las dos condiciones de borde anteriormente mencionadas:

$$\begin{cases} y(0) = 0, \\ \lim\limits_{x \to 1^-} y(x) = n, \end{cases}$$

4.5.3 Despeje de y

Iniciamos con la ecuación:

$$x \;=\; b\,\log_a\Big(c\,(y - d_x)\Big) \;+\; d_y.$$

Reordenamos para aislar el logaritmo:

$$x - d_y \;=\; b\,\log_a\Big(c\,(y - d_x)\Big).$$

Aplicando la definición de logaritmo, pasamos a la forma exponencial:

$$c\,(y - d_x) \;=\; a^{\frac{x - d_y}{b}}, \quad \Longrightarrow \quad y \;=\; d_x \;+\; \frac{1}{c}\,a^{\frac{x - d_y}{b}}.$$

De este modo, obtenemos y en función de x, con la salvedad de que $\big(d_x, d_y\big)$ deben elegirse (o ajustarse) para cumplir las condiciones de borde.

4.5.4 Condiciones de Borde

Deseamos que sea monótona creciente, además de cumplir con las condiciones generales:

$$y(0) = 0 \quad \text{y} \quad y(1) = n.$$

Por tanto,

$$\begin{cases} y(0) \;=\; d_x \;+\; \dfrac{1}{c}\, a^{\frac{0-d_y}{b}} \;=\; 0, \\ y(1) \;=\; d_x \;+\; \dfrac{1}{c}\, a^{\frac{1-d_y}{b}} \;=\; n. \end{cases}$$

- Primera Ecuación: $y(0) = 0$

$$d_x \;+\; \frac{1}{c}\, a^{\frac{-d_y}{b}} \;=\; 0 \quad \Longrightarrow \quad d_x \;=\; -\,\frac{1}{c}\, a^{\frac{-d_y}{b}}. \tag{1}$$

- Segunda Ecuación: $y(1) = n$

$$n \;=\; d_x \;+\; \frac{1}{c}\, a^{\frac{1-d_y}{b}} \;=\; \Big(-\,\frac{1}{c}\, a^{\frac{-d_y}{b}}\Big) \;+\; \frac{1}{c}\, a^{\frac{1-d_y}{b}} \;=\; \frac{1}{c}\Big[a^{\frac{1-d_y}{b}} \;-\; a^{\frac{-d_y}{b}}\Big].$$

Agrupando términos:

$$\frac{1}{c}\, a^{\frac{-d_y}{b}}\Big[a^{\frac{1}{b}} - 1\Big] \;=\; n, \quad \Longrightarrow \quad a^{\frac{-d_y}{b}} \;=\; \frac{c\,n}{a^{\frac{1}{b}} - 1}. \tag{2}$$

4.5.5 Solución para d_x y d_y

A partir de (2), resolvemos d_y:

$$\frac{-d_y}{b} = \log_a\Big(\frac{c\,n}{a^{\frac{1}{b}} - 1}\Big), \quad \Longrightarrow \quad d_y = -b\,\log_a\Big(\frac{c\,n}{a^{\frac{1}{b}} - 1}\Big).$$

Con este valor de d_y, usamos (1) para obtener d_x:

$$d_x \;=\; -\,\frac{1}{c}\, a^{\frac{-d_y}{b}} \;=\; -\,\frac{1}{c}\,\frac{c\,n}{a^{\frac{1}{b}} - 1} \;=\; -\,\frac{n}{a^{\frac{1}{b}} - 1}.$$

Por tanto, las constantes quedan fijadas como:

$$\boxed{d_y = -b\,\log_a\Big(\frac{c\,n}{a^{\frac{1}{b}} - 1}\Big), \quad d_x = -\,\frac{n}{a^{\frac{1}{b}} - 1}.}$$

4.5.6 Función Resultante $y(x)$

Sustituyendo d_x y d_y en la expresión

$$y \;=\; d_x \;+\; \frac{1}{c}\, a^{\frac{x-d_y}{b}},$$

obtenemos:

$$y(x) = -\,\frac{n}{a^{\frac{1}{b}} - 1} \;+\; \frac{1}{c}\, a^{\frac{x \,+\, b\,\log_a\left(\frac{c\,n}{a^{\frac{1}{b}} - 1}\right)}{b}}.$$

Podemos simplificar la potencia:

$$a^{\frac{b\,\log_a\left(\frac{c\,n}{a^{\frac{1}{b}} - 1}\right)}{b}} = a^{\log_a\left(\frac{c\,n}{a^{\frac{1}{b}} - 1}\right)} = \frac{c\,n}{a^{\frac{1}{b}} - 1}.$$

Así, la forma final resulta:

$$y(x) = -\frac{n}{a^{\frac{1}{b}} - 1} + \frac{1}{c}\left(\frac{c\,n}{a^{\frac{1}{b}} - 1}\right) a^{\frac{x}{b}} = \boxed{n\frac{a^{\frac{x}{b}} - 1}{a^{\frac{1}{b}} - 1}},$$

4.5.7 Interpretación

- **Dominio:** Para $x \in [0, 1)$, $y(x)$ se desplaza desde 0 hasta valores cercanos a n. Esto modela una curva logarítmica (o su inversa exponencial) que puede adaptarse vía a y b.
- **Escalabilidad:** El parámetro n define el "tope" al que llega $y(x)$ cuando $x \to 1^-$, y los parámetros (a, b) controlan la forma de la curva.
- **Eficiencia de Cálculo:** La ventaja de esta inversión es que se sustituyen operaciones logarítmicas por potencias, generalmente más eficientes en implementaciones de gran escala o en sistemas embebidos.

4.6 Caso con $n \to +\infty$: Función de Proporcionalidad Inversa

Cuando se desea modelar un conjunto de decisiones de tamaño indefinido (o incluso infinito), las expresiones estudiadas en apartados anteriores pueden resultar poco prácticas, pues al hacer $\lim_{n\to+\infty}$ a menudo el valor de la función también tiende a infinito, imposibilitando la selección de un *índice finito* de decisión.

Para afrontar este escenario, se propone un enfoque alternativo: usar una **función de proporcionalidad inversa**. La idea central es que, a medida que el parámetro de entrada ($x \in [0, 1)$ generado aleatoriamente) se acerca a 1, la función crezca de manera no acotada, simulando la disponibilidad de decisiones más allá de cualquier número finito. A su vez, cerca de $x = 0$, la función valdrá 0 (o un valor muy pequeño), representando la primera decisión.

4.6.1 Definición General

Se adopta la siguiente forma general para la función:

$$y(x) = \frac{a}{b\,(x - d_x)} + d_y,$$

donde:

- $x \in [0, 1)$ es el parámetro de entrada.
- $a > 0$ y $b > 0$ son factores de escala que determinan la curvatura y la velocidad de crecimiento de la función.
- d_x y d_y son términos de traslación que ajustan el dominio y el rango de la función.

El objetivo es imponer condiciones de borde que permitan que:

$$y(0) = 0 \quad \text{y} \quad \lim_{x\to 1^-} y(x) = +\infty.$$

4.6.2 Condiciones de Borde

- **Condición 1:** $y(0) = 0$

$$0 = \frac{a}{b\,(0 - d_x)} + d_y.$$

$$\Longrightarrow \quad d_y = -\frac{a}{b\,(-d_x)} = \frac{a}{b\,d_x}.$$

- **Condición 2:** $\lim_{x\to 1^-} y(x) = +\infty$

Para que la fracción $\frac{a}{b\,(x-d_x)}$ crezca sin acotación conforme $x \to 1^-$, se requiere que el denominador tienda a 0 desde el *lado positivo*:

$$b\,(1-d_x) \quad \overset{x\to 1^-}{\longrightarrow} \quad 0^+,$$

lo que implica

$$1-d_x = 0 \quad \Longrightarrow \quad d_x = 1.$$

4.6.3 Cálculo de d_x y d_y

Combinar ambas condiciones permite determinar $(d_x,\, d_y)$.

De la segunda condición:

$$d_x \;=\; 1.$$

De la primera condición:

$$0 \;=\; \frac{a}{b\,\big(0-1\big)} \;+\; d_y \;=\; -\,\frac{a}{b} \;+\; d_y \quad \Longrightarrow \quad d_y = \frac{a}{b}.$$

4.6.4 Función Final $y(x)$

Sustituyendo $d_x = 1$ y $d_y = \frac{a}{b}$ en la expresión original, obtenemos:

$$y(x) \;=\; \frac{a}{b\,\big(x-1\big)} \;+\; \frac{a}{b}.$$

Sin embargo, observemos el signo: para $0 \le x < 1$, $x - 1 < 0$, lo que hace que el término $\frac{a}{b(x-1)}$ sea negativo y, al acercarse $x \to 1^-$, tienda a $-\infty$. Esto *no* cumple el requisito de crecer a $+\infty$, sino que diverge en el sentido opuesto.

4.6.5 Ajuste de la Estructura

Para que el límite sea $+\infty$ cuando $x \to 1^-$, conviene invertir la posición de $(1-x)$ en el denominador:

$$y(x) \;=\; \frac{a}{b\,\big(1-x\big)} \;+\; d_y.$$

De este modo, al acercarse $x \to 1^-$, $(1-x) \to 0^+$ y la fracción $\frac{a}{b\,(1-x)}$ crece hacia $+\infty$.

Ahora se imponen las mismas condiciones de borde:

$$\begin{cases} \text{(i)}\ y(0) = 0, \\ \text{(ii)}\ \lim\limits_{x\to 1^-} y(x) = +\infty. \end{cases}$$

Condición (i): $y(0) = 0$

$$0 \;=\; \frac{a}{b\,\big(1-0\big)} \;+\; d_y \;=\; \frac{a}{b} \;+\; d_y \quad \Longrightarrow \quad d_y = -\,\frac{a}{b}.$$

Condición (ii): $\lim_{x\to 1^-} y(x) = +\infty$ Con $\big(1-x\big) \to 0^+$ en el denominador,

$$y(x) \;=\; \frac{a}{b\,\big(1-x\big)} \;-\; \frac{a}{b} \quad \overset{x\to 1^-}{\longrightarrow} \quad +\infty,$$

lo cual cumple el requisito.

4.6.6 Forma Explícita

La función queda:

$$y(x) \;=\; \frac{a}{b\,\big(1-x\big)} \;-\; \frac{a}{b} \;=\; \frac{a-a\,(1-x)}{b\,(1-x)} \;=\; \frac{a\,x}{b\,(1-x)}.$$

$$\boxed{y(x) \;=\; \frac{a\,x}{b\,\big(1-x\big)}} \quad \text{para } x \in [0,1), \quad \lim_{x\to 1^-} y(x) = +\infty, \quad y(0) = 0.$$

4.6.7 Interpretación y Uso Práctico

- **Indice de Decisión Infinito:** Al no haber un tope en el valor de $y(x)$, conceptualmente puede considerarse que existen *ilimitadas* decisiones ordenadas, y la variable x determina cuál de ellas se escoge.
- **Distribución Sesgada:**
  - Para valores de x cercanos a 0, $y(x)$ será pequeña, favoreciendo la selección de las "primeras" decisiones.
  - A medida que x crece, la función aumenta más rápidamente, si bien la probabilidad de llegar a valores muy altos de y (i.e., de decisiones muy alejadas) existe, pero es pequeña (el usuario controla esta curva vía a y b).
- **Aplicaciones:** Este tipo de asignación es útil cuando se desea explorar un espacio teóricamente infinito de opciones, pero con alta preferencia por soluciones con *fitness* superior (asociadas a valores más bajos de y) y una reducida, aunque existente, probabilidad de explorar opciones lejanas en el orden.

En resumen, la **función de proporcionalidad inversa** ofrece una solución elegante para el caso de $n \to +\infty$, conservando la propiedad de asignar mayor prioridad a las primeras decisiones, pero sin truncar la posibilidad de explorar otras con un *índice* arbitrariamente grande.

4.7 Combinación de Múltiples Funciones Probabilísticas

En ciertas situaciones, distintas funciones pueden resultar óptimas según la condición o etapa específica del problema. Para aprovechar las ventajas de cada una, es posible crear una *combinación* que seleccione, en cada momento, cuál de ellas se empleará para determinar la probabilidad de escoger cada decisión. El proceso se basa en:

1. Definir un **conjunto de funciones** probabilísticas, por ejemplo, la *función logarítmica inversa* y otras (polinómicas, exponenciales, etc.).
2. Diseñar una **función maestra** (por ejemplo, otra función logarítmica inversa) que devuelva la *probabilidad* de seleccionar cada una de las funciones del conjunto.
3. Al momento de tomar una decisión, *elegir probabilísticamente* cuál función del conjunto usar, de acuerdo con la distribución que genera la función maestra.
4. Aplicar la función elegida para calcular la probabilidad final de cada decisión en el problema, asegurando que se cumpla la condición de que la suma de probabilidades de todas las decisiones sea 1.

De esta manera, la **combinación** de funciones se comporta *como si fuese una sola*, pero con un *potencial de adaptación superior*, puesto que en cada situación puede optar por la función que mejor se ajuste a la dinámica o la información disponible del entorno. Además, se mantienen las **reglas de consistencia probabilística**, porque tanto la función maestra como las funciones individuales se encargan de normalizar sus respectivos valores, cumpliendo así con los requerimientos de una distribución válida.

En síntesis, la combinación de múltiples funciones probabilísticas **maximiza la flexibilidad** y la capacidad de adecuación a problemas donde un solo tipo de función puede no ser suficiente para capturar todas las facetas de la decisión. Esta extensión amplía el alcance de la aproximación probabilística y permite un mejor desempeño en escenarios de mayor complejidad y variabilidad.

5 Optimización de Parámetros Mediante Metaheurísticas

En la sección anterior se describieron diversas funciones (logarítmicas, inversas, etc.) para asignar probabilidades a decisiones en un problema de optimización, cada una con un conjunto de parámetros (a, b, c, d_x, d_y, etc.) cuyo valor determina la forma específica de la curva de asignación probabilística. Sin embargo, cada problema de decisión presenta características particulares, por lo que la configuración de parámetros que maximice los resultados de un caso podría no ser óptima para otro. Con esto surge la necesidad de un **proceso de optimización** que encuentre, de manera automática, aquellos valores de parámetros que produzcan los mejores resultados para el problema bajo estudio.

5.1 Necesidad de la Optimización de Parámetros

La búsqueda manual de parámetros adecuados puede resultar inviable cuando:

- El espacio de parámetros es muy grande (por ejemplo, varias dimensiones).
- Existe no linealidad en cómo los parámetros afectan el desempeño final.
- No se dispone de un modelo analítico simple para relacionar los parámetros con el valor objetivo a maximizar.

Por ende, se recurre a métodos sistemáticos de optimización que, dada una *función objetivo* y unas *restricciones*, exploran el espacio de soluciones en búsqueda de los parámetros que optimicen (maximicen o minimicen) el desempeño esperado.

5.2 Metaheurísticas: Una Vía de Uso General

Existen numerosos algoritmos de optimización, desde métodos deterministas como la búsqueda en gradiente, hasta métodos estocásticos que no requieren información explícita de derivadas. *Las metaheurísticas* conforman un conjunto de procedimientos de búsqueda de propósito general que pueden adaptarse a una amplia variedad de problemas y restricciones:

- **Idea fundamental:** una metaheurística no está diseñada para un problema específico, sino que, a través de reglas de exploración y explotación, es capaz de iterar sobre un espacio de soluciones, refinando paulatinamente la búsqueda.
- **Ventaja principal:** se pueden aplicar *incluso cuando la función objetivo sea una* caja negra (black-box), siempre que sea posible evaluar la calidad de una solución (por ejemplo, simulando o ejecutando el problema con los parámetros propuestos).
- **Convergencia:** muchas metaheurísticas cuentan con mecanismos que les permiten escapar de máximos locales y seguir explorando. Sin embargo, no siempre garantizan un óptimo global estricto, sino una *buena* solución en un tiempo razonable.

5.3 Ejemplos de Metaheurísticas

Dentro de las metaheurísticas más populares se encuentran:

- **Algoritmos Genéticos (GA):** inspiran su funcionamiento en la selección natural. Parten de una población de soluciones (denominadas individuos o cromosomas), que se reproduce y muta bajo una presión selectiva hacia soluciones de mayor *fitness*.
- **Recocido Simulado (SA):** inspirado en el proceso de temple de metales. Comienza con soluciones aleatorias y permite movimientos que empeoren la calidad de la solución con cierta probabilidad, reduciendo esta probabilidad progresivamente para refinar la búsqueda.
- **Enjambre de Partículas (PSO):** un conjunto de partículas se desplaza por el espacio de soluciones, ajustando su velocidad en función de su mejor posición personal y la mejor posición encontrada por el enjambre. Esta técnica resulta especialmente poderosa en espacios continuos.
- **Búsqueda Tabú, Optimización por Colonia de Hormigas (ACO), etc.:** cada una con inspiración en sistemas naturales o enfoques combinatorios que las hacen flexibles ante distintos problemas.

5.4 Enjambre de Partículas (PSO): Principios Básicos

La optimización por enjambre de partículas (*Particle Swarm Optimization*, PSO) [11] es particularmente útil cuando:

1. El espacio de parámetros es de dimensión moderada o alta.
2. No se cuenta con información sobre el gradiente de la función objetivo.
3. Se puede evaluar rápidamente la calidad (o *fitness*) de una configuración de parámetros.

Estructura de PSO.

- Una **partícula** representa un conjunto de parámetros $\vec{p}$ que definen la función de asignación de probabilidades.
- Cada partícula tiene ademas una **velocidad** $\vec{v}$ en el espacio de búsqueda.
- **Cálculo de fitness:** para cada $\vec{p}$, se evalúa el resultado final del problema de decisión usando los parámetros propuestos. Ese valor sirve de guía para mejorar la búsqueda.
- **Movimiento:** en cada iteración, las partículas ajustan su velocidad y posición basadas en:
  1. Su mejor solución personal (memoria individual).
  2. La mejor solución global encontrada por cualquier partícula del enjambre (memoria colectiva).

Ventajas de PSO.

- **Facilidad de implementación:** PSO se resume en pocas líneas de código y no requiere gradiente.
- **Convergencia rápida:** particularmente en espacios de dimensión moderada.
- **Escapatoria de máximos locales:** no está completamente garantizada, pero el componente estocástico y la influencia colectiva suelen permitir escapar de algunos valles locales.

5.5 Aplicación al Caso de Asignación Probabilística

Para optimizar la función de asignación probabilística (sea logarítmica, inversa, etc.) mediante metaheurísticas como PSO, se definen los siguientes elementos:

1. **Parámetros a Optimizar:** $\{a, b, c, d_x, d_y, \dots\}$ (dependiendo de la función escogida).
2. **Función Objetivo:** generalmente, la medida de *efectividad* de la asignación de probabilidades en la resolución del problema de decisión. Por ejemplo, la ganancia promedio, la proporción de veces que se encuentra una solución factible de cierto nivel, etc.
3. **Restricciones:** valores válidos de los parámetros, relaciones entre ellos (por ejemplo, $a > 1$, $b > 0$), y aspectos prácticos (tiempo de cómputo).
4. **Proceso Iterativo:**
   a) Generar o actualizar un conjunto de candidatos (N partículas en PSO, población en GA, etc.).
   b) Evaluar cada candidato en la función objetivo.
   c) Modificar las soluciones según las reglas de la metaheurística (velocidades en PSO, cruces y mutaciones en GA, vecinos en SA, etc.).
   d) Repetir hasta cumplir un criterio de parada (número máximo de iteraciones, convergencia, etc.).

5.6 Conclusiones

Las metaheurísticas representan un bloque de construcción versátil en la optimización de parámetros para problemas complejos, incluidas las funciones probabilísticas que se han definido en este trabajo. Ideas extras que podrían considerarse:

- La adaptación dinámica de parámetros de la metaheurística (por ejemplo, disminuir la inercia en PSO a lo largo del tiempo).
- El híbrido con métodos deterministas (por ejemplo, iniciar con PSO y finalizar con un algoritmo de búsqueda local).
- El análisis teórico de la convergencia y un estudio de sensibilidad de cómo la variación en los parámetros afecta la probabilidad de elegir decisiones rentables.

En suma, la elección de una metaheurística depende de la *naturaleza del problema*, de las restricciones computacionales y de la facilidad de implementación. PSO, dada su sencillez y eficacia en espacios continuos, es una de las alternativas más llamativas para hallar los parámetros óptimos de las funciones de asignación probabilística propuestas.

6 Selección Dinámica de Parámetros Mediante Redes Neuronales

En la sección anterior se discutió cómo determinar los parámetros de la función de asignación probabilística utilizando metaheurísticas, lo cual conduce a un *conjunto discreto* de valores "óptimos" para un problema dado. Sin embargo, una limitación de dicho enfoque es que los parámetros encontrados suelen fijarse de forma estática. A medida que el problema evoluciona o se presentan diversas *subinstancias* dentro de un mismo escenario, podría resultar beneficioso *ajustar los parámetros en tiempo real* conforme cambia el estado del problema. En este apartado se explora la posibilidad de dotar al algoritmo de una **capacidad de selección dinámica de parámetros** a través de una *función entrenable*, que se aproximará mediante redes neuronales.

6.1 Motivación de la Selección Dinámica

La búsqueda de parámetros que se ajusten *globalmente* a todo un problema de decisión puede ser insuficiente cuando:

- El **entorno** o las **restricciones** varían a lo largo de la resolución del problema.
- Se deseen explotar *estados intermedios*: en cada etapa, la configuración "óptima" de parámetros para la función de decisión puede diferir de la que se tendría en etapas iniciales o finales.
- Exista un **alto costo computacional** al recomputar parámetros con metaheurísticas en cada paso (dada la complejidad de los problemas reales), de forma que se requiera un modelo entrenado que prediga rápidamente los parámetros oportunos.

6.2 Uso de Redes Neuronales para la Selección Dinámica

Para aproximar una función $\mathcal{F}$ que, dado el *estado actual* del problema, devuelva un vector de parámetros $\vec{\theta}$ $(a, b, c, \dots$ o equivalentes), se recurre a *redes neuronales*. Estas redes disponen de la capacidad de capturar relaciones complejas entre la entrada (estado) y la salida (parámetros) tras un proceso de entrenamiento supervisado.

6.2.1 Entrenamiento Supervisado

El entrenamiento de la red neuronal requiere un **conjunto de datos** con pares $(\text{entrada}, \text{salida deseada})$. En este caso:

1. **Entrada**: Una representación del estado (o el histórico de estados) del problema en un instante dado.

2. **Salida**: El vector de parámetros $\vec{\theta}$ que optimiza la decisión en ese estado.

Sin embargo, **la pregunta clave es cómo se genera dicho conjunto de entrenamiento**. Una estrategia viable es:

- Para diversas *instancias* del problema, aplicar el enfoque discreto basado en metaheurísticas para encontrar los parámetros óptimos en cada *subetapa* del problema.
- Almacenar cada par (estado, parámetros óptimos) como un ejemplo de entrenamiento.
- Repetir en un número suficientemente grande de instancias (y subinstancias) para cubrir la variedad de situaciones posibles.

De este modo, la red *aprende* a reproducir la solución que la metaheurística habría producido, pero *en tiempo de ejecución* y de manera *continua*.

6.3 Dimensión Variable en la Entrada

Una de las dificultades más notables es que los **problemas pueden tener estados con distinto tamaño de información**:

- **Secuencias de longitud variable**: algunos problemas tienen un número de pasos que cambia dinámicamente.
- **Diferente representación de datos**: según la instancia, podrían variar la cantidad de entidades, restricciones activas, etc.

La mayoría de *redes neuronales clásicas* (ejemplo, perceptrones multicapa) tienen un tamaño fijo de entrada. Para abordar entradas de longitud variable se contemplan arquitecturas que puedan procesar secuencias de manera flexible:

- **Redes Recurrentes (RNN, LSTM, GRU):** Adecuadas para procesamiento secuencial, donde la entrada se "alimenta" paso a paso. Son una opción para tratar datos con distintas longitudes, aunque pueden presentar problemas de gradiente en secuencias muy largas.
- **Redes Convolucionales con mecanismos de *pooling* adaptativo**: Permiten analizar subsecuencias y extraer características, aunque se adecúan mejor a datos 2D (imágenes) o secuencias con cierto alineamiento.
- **Transformers**: Arquitectura que maneja secuencias usando *atención*, capaz de procesar representaciones de diverso tamaño y capturar dependencias a larga distancia de manera más efectiva.

6.4 Arquitectura de tipo Transformer

Los **Transformers** [12] han ganado popularidad por sus impresionantes resultados en PLN (Procesamiento de Lenguaje Natural), pero su uso se ha generalizado a otros dominios.

Aspectos clave:

- **Entrada como secuencia:** Se codifica el input en un conjunto de *tokens*. Cada token recibe una representación embedding que se enriquece con información posicional o de segmentación.
- **Atención multi-cabeza (Multi-head Attention):** Permite que cada posición (token) preste atención a cualquier otra posición en la secuencia, aprendiendo las relaciones importantes entre ellos.
- **Aplicación más allá del Lenguaje:** Si se aplanan los datos del estado del problema a una secuencia (token tras token), el Transformer puede procesarlo y mapearlo a una salida de tamaño fijo (los parámetros a predecir).

6.4.1 Manejo de Estados Variados

Para lidiar con **información heterogénea** en un mismo problema, una estrategia:

- **Aplanar cada estado** (o cada componente relevante) en un vector que se convierte en un token.
- Insertar un *token separador* (ejemplo, un símbolo especial $\langle SEP \rangle$) entre estados sucesivos.
- Construir una *secuencia n-única* compuesta por todos los tokens de los estados relevantes, de modo que el Transformer "vea" el conjunto total de información como un bloque continuo.

La atención del Transformer permite que el modelo descubra las relaciones importantes entre los elementos de esta secuencia, sin requerir una arquitectura específica para la variable longitud.

6.5 Proceso General de Entrenamiento

1. **Definir la estructura de entrada**: representar cada estado como un vector de características. Encadenar todos los estados transcurridos con un token separador.
2. **Diseñar la red Transformer**:
   - Bloques de atención multi-cabeza.
   - Capas feed-forward intermedias.
   - Mecanismos de *positional encoding* para indicar el orden o la secuencia temporal.
3. **Salida de la red**: un bloque feed-forward final (o cabeza de salida) produce un vector de parámetros $\vec{\theta}$.
4. **Objetivo de aprendizaje**: Mínimo error (por ejemplo, en norma cuadrática) entre la salida de la red y los parámetros $\vec{\theta}^*$ que determinan la solución óptima en cada subinstancia, obtenida por la metaheurística.
5. **Entrenamiento**:
   - Se divide el conjunto de datos (instancias + subinstancias) en entrenamiento, validación y test.
   - Se aplica un método de optimización estándar (por ejemplo, Adam) para ajustar los pesos de la red.
   - Al final, se evalúa la calidad prediciendo los parámetros en instancias nunca antes vistas y midiendo el desempeño del problema de decisión.

6.6 Reflexiones y Proyecciones

- **Flexibilidad:** El uso de un Transformer no se limita a problemas con una secuencia temporal; cualquier información que pueda representarse como tokens puede beneficiarse de la atención multi-cabeza.
- **Escalabilidad:** Aunque los Transformers han probado su eficacia a gran escala, pueden volverse costosos en términos de memoria y cómputo si la secuencia es muy larga. Se han propuesto variantes (Longformer, Performer, etc.) para mitigar estas limitaciones.
- **Calidad del conjunto de entrenamiento:** Cuanta más instancias y subinstancias se analicen con metaheurísticas de alta calidad, mejor podrá la red neuronal aproximar la selección de parámetros en tiempo real.

En conclusión, la selección dinámica de parámetros vía redes neuronales es un paso más *ambicioso* que busca convertir la asignación discreta (obtenida por metaheurísticas) en una selección continua, adaptable al estado presente. El uso de **Transformers** se presenta como una estrategia particularmente prometedora para manejar entradas de tamaño variable y capturar relaciones complejas entre estados, permitiendo que el modelo "aprenda" cómo deberían evolucionar los parámetros en el tiempo o conforme varíen las condiciones del problema.

6.7 Manejo de la Escalabilidad: Reducción de Dimensionalidad con SVD

Pese a su *gran potencial*, los Transformers presentan una limitación crucial en términos de **escabilidad**: la capa de atención requiere construir y procesar una matriz de atención de tamaño $N \times N$ para una secuencia de longitud N. En problemas donde la secuencia de estados crece a lo largo del tiempo (o abarca gran cantidad de elementos), la memoria necesaria para representar y procesar dicha matriz puede volverse prohibitivamente grande.

6.7.1 Transformar la Secuencia de Estados en una Matriz 2D

Una forma de abordar este inconveniente es *transformar la secuencia de estados* en una matriz 2D que capture de forma compacta la información esencial. Por ejemplo, si en cada paso del problema se produce un vector de características (información sobre el entorno o las decisiones pasadas), podemos concatenar o apilar dichos vectores en la dirección de las filas o las columnas, obteniendo una representación del estado global:

$$\mathbf{M} \in \mathbb{R}^{N \times d},$$

donde N es la longitud de la secuencia (número de estados considerados) y d es la dimensión de cada vector de características.

6.7.2 Aplicación de la Descomposición en Valores Singulares (SVD)

Para reducir la dimensionalidad y *controlar* el tamaño de la representación, se emplea la **Descomposición en Valores Singulares (SVD, por sus siglas en inglés) [13]**:

$$\mathbf{M} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^{\top},$$

donde:

- $\mathbf{U} \in \mathbb{R}^{N \times N}$ es una matriz unitaria (columnas ortonormales).
- $\mathbf{\Sigma} \in \mathbb{R}^{N \times d}$ es una matriz diagonal (o cuasi-diagonal) con los valores singulares de $\mathbf{M}$.
- $\mathbf{V} \in \mathbb{R}^{d \times d}$ es también una matriz unitaria.

Los valores singulares se ordenan de mayor a menor. De esta forma, **los primeros valores singulares** capturan la mayor parte de la energía (información) de la matriz $\mathbf{M}$.

Selección de los vectores más relevantes Para reducir la dimensionalidad de $\mathbf{M}$, se toma un número $r < \min(N, d)$ de los valores singulares más altos. Esto equivale a truncar $\mathbf{\Sigma}$ y las columnas/filas correspondientes de $\mathbf{U}$ y $\mathbf{V}$, quedando:

$$\mathbf{M} \approx \mathbf{U}_r\,\mathbf{\Sigma}_r\,\mathbf{V}_r^{\top},$$

donde $\mathbf{U}_r \in \mathbb{R}^{N \times r}$, $\mathbf{\Sigma}_r \in \mathbb{R}^{r \times r}$ y $\mathbf{V}_r \in \mathbb{R}^{d \times r}$. Esta aproximación minimiza en sentido de *norma de Frobenius* [14] ($\|\mathbf{M} - \mathbf{M}_r\|_F$) el error al representar $\mathbf{M}$ con rango r.

6.7.3 Representación Fija y Compacta

Para propósitos de *entrada al Transformer* (o cualquier otra red neuronal de tamaño fijo), puede diseñarse un esquema que:

1. Elija un r constante para todas las instancias del problema.
2. Construya una representación vectorial tomando los vectores columna (o fila) en $\mathbf{U}_r$, $\mathbf{\Sigma}_r$, $\mathbf{V}_r$, o alguna combinación de ellos. Por ejemplo, concatenar las columnas principales de $\mathbf{U}_r$ y $\mathbf{V}_r$ para obtener una descripción de dimensiones fijas.
3. Ignore los valores singulares de menor magnitud (colocados al final de $\mathbf{\Sigma}$) que no aportan significativamente a la estructura del estado.

Con este procedimiento, **no importa cuán larga sea la secuencia original** (N), porque finalmente se *proyecta* a una representación $\mathbf{M}_r \in \mathbb{R}^{r \times r}$, o bien a un vector de dimensión fija (por ejemplo, $2\,r\,d$ si se concatenan parte de $\mathbf{U}_r$ y $\mathbf{V}_r$). Adicionalmente, los vectores ortonormales en SVD poseen propiedades algebraicas que pueden resultar más fáciles de explotar o aprender para la red neuronal que los datos originales "sin procesar".

6.7.4 Beneficios para el Modelo

- **Ahorro de Memoria:** Al representar la información relevante en un espacio de dimensión r, se evita la construcción de matrices $N \times N$ cuando N es grande.
- **Generalización Más Rápida:** La proyección SVD tiende a eliminar el ruido o la redundancia en los datos, facilitando a la red neuronal detectar relaciones más claras y robustas.
- **Tamaño de Entrada Fijo:** Aun si el problema evoluciona indefinidamente y la secuencia de estados se vuelve muy larga, la representación de SVD puede mantener un tamaño constante, evitando la explosión de recursos.

6.7.5 Reflexiones Finales

La **aplicación de SVD** u otras técnicas de reducción de dimensionalidad (por ejemplo, PCA, proyecciones aleatorias, autoencoders) resulta clave para hacer *escalable* el uso de Transformers en problemas donde la secuencia de estados es muy grande. Aunque cada problema demandará su propio criterio de cuántos vectores (o qué umbral de energía) conservar en la descomposición, este enfoque proporciona un *control preciso* sobre la representación de la información, combinando:

1. La *capacidad explicativa* de la reducción SVD (que concentra la mayor parte de la variabilidad en menos dimensiones).
2. La *potencia* del mecanismo de atención de los Transformers, sin quedar limitado por los requerimientos explosivos de memoria en secuencias largas.

En conclusión, integrar la reducción de dimensionalidad mediante SVD antes de alimentar un Transformer constituye una solución elegante al problema de la *escalabilidad* en el contexto de selección dinámica de parámetros. Esta aproximación equilibra la retención de información clave con la necesidad práctica de mantener un tamaño de entrada manejable en la red neuronal.

7 Pruebas y Resultados: Primer Caso de Estudio

En esta sección se presentan las pruebas realizadas para validar el desempeño del algoritmo en un juego de parejas de cartas con memoria imperfecta. El objetivo de este caso de estudio es ilustrar cómo el mecanismo de asignación probabilística, la optimización de parámetros y la selección dinámica mediante aprendizaje automático pueden adaptarse a un escenario donde la información no se conserva de manera perfecta.

7.1 Descripción del Juego y Motivación

El juego consiste en un conjunto de cartas boca abajo, dispuestas en posiciones fijas sobre la mesa. Cada carta tiene una pareja idéntica, también presente en algún lugar del tablero. La dinámica básica es:

1. En cada ronda, el jugador selecciona dos cartas para descubrirlas.
2. Si las cartas son iguales, se retiran del tablero (pareja encontrada).
3. Si son diferentes, se voltean nuevamente boca abajo.

El objetivo final es *quedarse sin cartas boca abajo*, es decir, descubrir todas las parejas.

7.1.1 Limitaciones de Memoria y Confusión

Para una máquina con memoria perfecta, el juego es trivial, ya que cualquier carta vista una vez puede recordarse con exactitud. Sin embargo, en el caso de los humanos, la *memoria no es perfecta*: a menudo se confunden las posiciones de las cartas descubiertas e, incluso, se puede olvidar por completo haber visto cierto naipe.

Con el fin de modelar esta limitación en la máquina, se introduce una *matriz de probabilidades* que contiene la información sobre dónde se cree que está cada carta, con cierto grado de incertidumbre. Específicamente:

- La matriz asocia a cada posición del tablero un vector de probabilidades, donde cada componente p_i indica la probabilidad de que *la carta* i se encuentre en esa posición.
- Cuando se descubren dos cartas en la ronda, si se obtiene información certera (dos cartas distintas u iguales), se actualiza la matriz para reflejar que, en las posiciones reveladas, la probabilidad de tener tal carta es 1, y la probabilidad de tener otras cartas es 0.
- Tras cada ronda, se aplica un mecanismo de **pérdida de memoria**, donde cierto α (porcentaje de olvido) redistribuye parte de la probabilidad asociada a cada carta entre las posiciones vecinas. De esta manera, se simula la *confusión* propia de la memoria imperfecta: mientras mayor sea α, más se propaga la incertidumbre al resto del tablero.

7.2 Estrategia de Decisión Probabilística

La estrategia básica para escoger qué dos posiciones revelar en cada ronda se basa en **valores esperados**. Para cada par de posiciones del tablero (x, y), se multiplican sus correspondientes vectores de probabilidad y se estima el valor esperado de la carta que se podría descubrir en ese par. A grandes rasgos:

- Si existe alta certeza de que dos posiciones contienen la misma carta, el valor esperado de elegirlas es mayor.
- Si la confusión (o pérdida de memoria) es muy elevada, estos valores esperados pueden volverse menos fiables.

El *algoritmo* que proponemos debe *aprender* cuándo es conveniente seguir los valores esperados más altos (que tienden a favor de las opciones aparentemente más seguras) o explorar otras posiciones cuya información se desconoce con mayor grado de incertidumbre.

7.3 Configuración de las Pruebas

Para llevar a cabo las pruebas, se define:

- **Tamaño del Tablero:** Se trabaja con un número de cartas equilibrado para que el problema sea desafiante pero manejable (10 pares).
- **Rango de Pérdida de Memoria (α):** Se evalúan escenarios con distintos niveles de confusión para medir la robustez de la estrategia.
- **Iteraciones Metaheurísticas:** La optimización se ejecuta bajo un límite de iteraciones con el fin de explorar soluciones factibles en un tiempo de cómputo razonable.
- **Entrenamiento de la Red Neuronal:** Se generan múltiples instancias simuladas con diferentes valores de α y composiciones de cartas.

7.4 Usando PSO

En esta sección se muestran los resultados obtenidos al aplicar el algoritmo de Enjambre de Partículas (*Particle Swarm Optimization*, PSO) para ajustar los parámetros de la función logarítmica invertida de asignación probabilística descrita anteriormente. El objetivo es minimizar el número de pasos necesarios en promedio para resolver el juego de parejas con un cierto nivel de confusión (α), teniendo en cuenta que **existe un límite de rondas** máximo para no permitir partidas excesivamente largas.

7.4.1 Función Objetivo del PSO

Para cuantificar la calidad (o aptitud) de una configuración de parámetros, se define una **función objetivo** basada en:

- **Cantidad de pasos** promedio necesarios para resolver el juego cuando el número de rondas totales *no* supera un cierto límite (limit).
- **Partidas fallidas o sobre el límite**, es decir, aquellas que exceden la cantidad de pasos permitidos, lo cual podría considerarse un fracaso en la práctica.

Sea:

- count: la cantidad total de partidas ejecutadas durante la evaluación.
- ol (*over limit*): la cantidad de partidas que superaron el límite de pasos permitido.
- total: la suma de los pasos utilizados en las partidas que finalizaron por debajo del límite.

Entonces, la función objetivo (o **fitness**) se define como:

$$\text{fitness} = \begin{cases} \text{ol} + 2, & \text{si count} = \text{ol}, \\ \text{ol} + 2 \times \left(\frac{\text{total}}{\text{count} - \text{ol}}\right)/\text{limit}, & \text{en caso contrario.} \end{cases}$$

La expresión anterior se interpreta de la siguiente manera:

- Si todas las partidas (count = ol) se fueron por encima del límite, el algoritmo retorna ol + 2 como penalización máxima, evidenciando un desempeño muy deficiente.
- De lo contrario, se penaliza el número de partidas fallidas (ol) y, además, se considera el promedio de pasos (total/(count − ol)), escalado por limit.

7.4.2 Configuración Experimental

Para llevar a cabo los experimentos se fija:

- **Número de partículas** en el enjambre: parámetro usual de PSO (por ejemplo, 20 o 30 partículas).
- **Iteraciones máximas** de PSO: determinado para equilibrar la exploración y el tiempo computacional.
- **Niveles de confusión** (α): se prueban valores representativos (por ejemplo, 0.1, 0.3, 0.5, etc.) para cubrir escenarios de memoria más confiable y otros sumamente confundidos.

7.4.3 Visualización de la Función Logarítmica Invertida con Parámetros Obtenidos

Con el fin de ilustrar cómo varía la *función logarítmica invertida* (empleada en la asignación probabilística) al ajustar los parámetros encontrados por PSO, se muestra el caso particular con $n = 5$. Esto corresponde a un escenario donde existen 5 decisiones potenciales (o parejas) por elegir, lo que brinda una representación gráfica clara de cómo la probabilidad se distribuye entre ellas. En la práctica del juego, n varía a medida que se retiran parejas del tablero, por lo que estos porcentajes se adaptan dinámicamente, aunque la forma cualitativa de las curvas se conserva.

A continuación, se presentan distintas gráficas que muestran la forma de la función logarítmica invertida para diferentes valores de α (porcentaje de confusión):

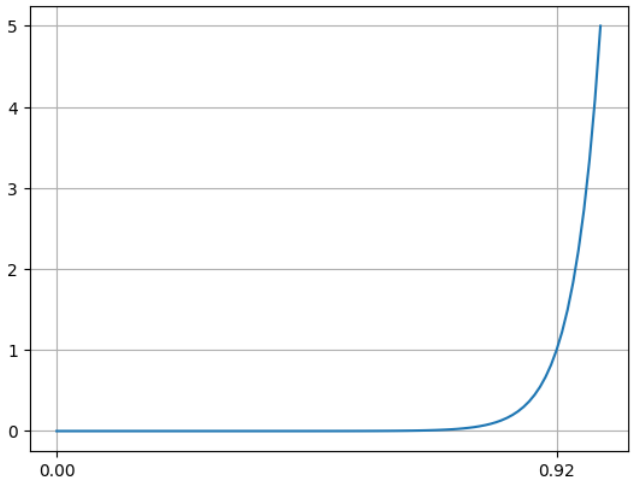

Figura 1: Curva logarítmica invertida para α entre 0 % y 2 %. Se observa que la primera opción recibe en promedio un 92 % de confianza.

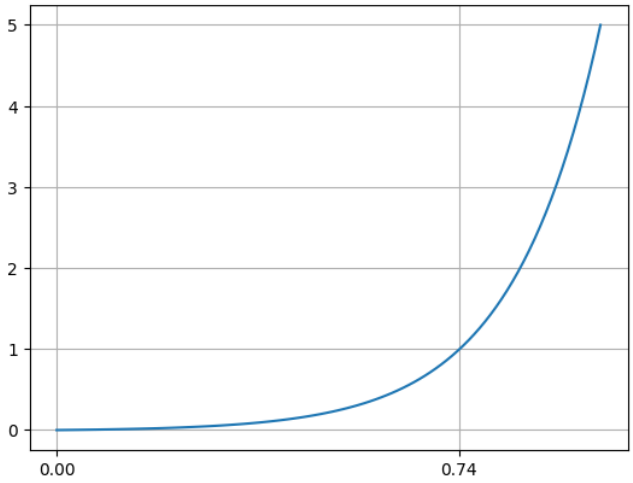

Figura 2: Curva logarítmica invertida para $\alpha = 5\,\%$. La primera opción obtiene un $74\,\%$ de confianza.

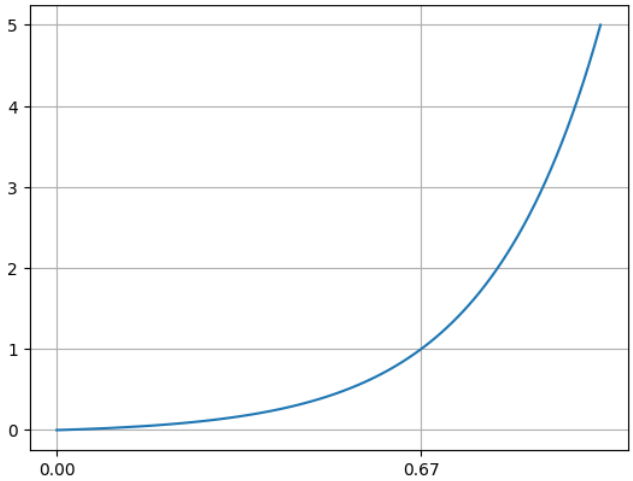

Figura 3: Curva logarítmica invertida para $\alpha = 10\,\%$. La primera opción baja a un $67\,\%$ de confianza, indicando más distribución hacia las demás.

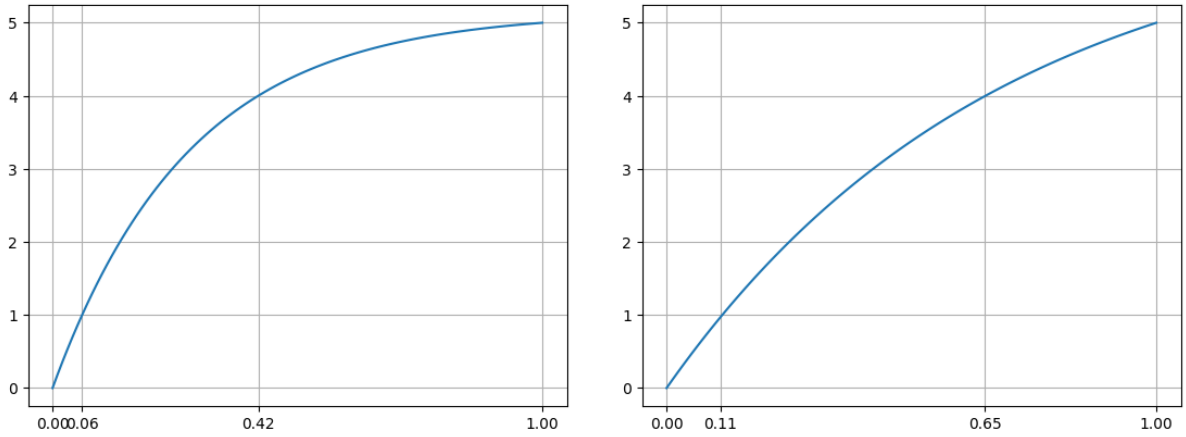

Figura 4: Curvas logarítmicas invertidas para α entre 50 % y 80 %. Se observa que la primera opción cae entre 6 % y 11 %, mientras que la última asciende entre 35 % y 58 %.

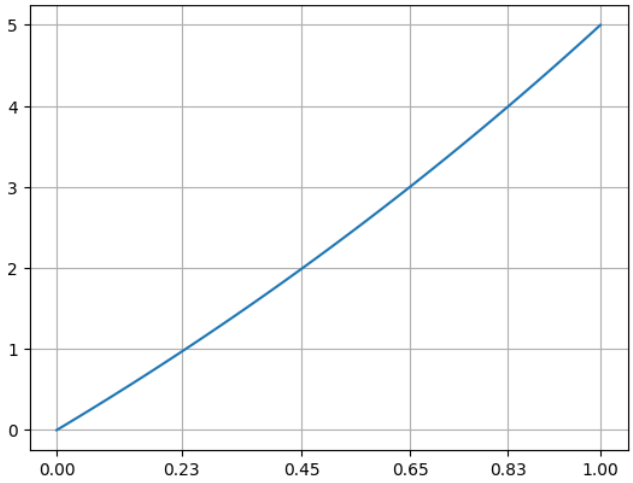

Figura 5: Curva logarítmica invertida para $\alpha = 100\,\%$. Todas las opciones reciben aproximadamente la misma probabilidad (alrededor de un 20 % para $n = 5$).

Se aprecia cómo, a medida que incrementa α, la función resultante va reduciendo la confianza en la primera decisión y distribuyéndola cada vez más entre el resto de las opciones. De hecho, para niveles de confusión muy altos (50 %–80 %), se observa un comportamiento inverso, favoreciendo las opciones finales en mayor proporción. Finalmente, con $\alpha = 100\,\%$, las probabilidades quedan *uniformemente* repartidas, dado que la información se considera completamente confusa.

Estos resultados muestran cómo los parámetros determinados por PSO ajustan la función logarítmica invertida de forma adaptativa, favoreciendo la explotación cuando la memorización del juego es confiable (baja confusión) y diversificando la búsqueda de parejas cuando la memoria es sumamente inestable (alta confusión).

7.4.4 Comparación de Resultados con la Selección por Mayor Valor Esperado

Para evaluar la eficacia de los parámetros encontrados mediante PSO (en conjunto con la función logarítmica) se compara su desempeño con el de una estrategia que siempre elige la decisión con mayor valor esperado. Se simularon 100 partidas para cada nivel de confusión (α), con un límite de hasta 1000 rondas. En caso de superar ese número de rondas, consideramos que la partida entra en un ciclo indefinido y, por tanto, la categorizamos como un fallo (*over limit*).

En la tabla siguiente se muestran, para cada valor de α:

- **Promedio debajo del límite** (sólo considerando las partidas que no superaron las 1000 rondas).
- **Cantidad de partidas por encima del límite** (las que potencialmente se vuelven infinitas).

Cuadro 1: Comparación de la media de rondas y fallos entre el algoritmo propuesto (PSO + función logarítmica) y la selección por mayor valor esperado. Cada celda muestra (Promedio, # over limit).

Estrategia	$\alpha = 0$	$\alpha = 0{,}01$	$\alpha = 0{,}02$	$\alpha = 0{,}05$	$\alpha = 0{,}1$	$\alpha = 0{,}3$	$\alpha = 0{,}5$	$\alpha = 0{,}8$	$\alpha = 1{,}0$
Algoritmo (PSO)	(21.52, 0)	(24.95, 0)	(25.14, 0)	(30.74, 0)	(60.43, 0)	(120.67, 0)	(108.49, 0)	(100.22, 0)	(98.65, 0)
Mayor Valor Esperado	(21.53, 0)	(25.38, 0)	(26.65, 0)	(36.08, 0)	(170.67, 2)	(443.11, 17)	(474.96, 20)	(429.92, 29)	(414.17, 40)

Análisis de Resultados

- **Escenario sin confusión** ($\alpha = 0$): Ambas tienen un rendimiento muy parecido y sin partidas fallidas.
- **Confusión baja a moderada** ($0{,}01 \leq \alpha \leq 0{,}3$):
  - El algoritmo propuesto mantiene su capacidad de resolver todas las partidas por debajo del límite, con promedios crecientes a medida que α sube, pero sin llegar a fallo.
  - La estrategia de mayor valor esperado, en cambio, comienza a acumular partidas fallidas; a partir de $\alpha = 0{,}1$ aparecen 2 fallos, esta tendencia se agrava a 17 fallos con $\alpha = 0{,}3$.
- **Confusión alta** ($0{,}5 \leq \alpha < 1{,}0$):
  - El método con PSO sigue sin generar ninguna partida fallida, con promedios entre 100 y 108 rondas.
  - La selección por mayor valor esperado se vuelve inviable, con un número considerable de partidas (entre 20 y 29 de cada 100) rebasando el límite. Incluso el promedio entre las partidas que no fallan es muy alto (por ejemplo, 474.96 rondas para $\alpha = 0{,}5$).
- **Confusión total** ($\alpha = 1{,}0$):
  - El algoritmo propuesto resuelve la totalidad de las partidas sin fallar, con un promedio de 98.65 rondas.
  - La estrategia de mayor valor esperado falla en una gran cantidad de las partidas, y con un alto promedio de rondas en el resto.

Estos datos reflejan que, para $\alpha = 0$ el algoritmo decide que la selección por mayor valor esperado es lo más eficiente, pero **en cuanto existe un mínimo grado de confusión la estrategia de mayor valor esperado deja de ser confiable** y provoca ciclos infinitos en una fracción importante de las partidas. Por el contrario, el algoritmo que combina PSO con la función logarítmica invertida y un método probabilístico de selección consigue adaptarse a la pérdida de información y culminar exitosamente el juego incluso para altos valores de α.

Conclusión preliminar: La capacidad de explorar nuevas decisiones en lugar de confiar exclusivamente en la *opción aparentemente más prometedora* resulta crítica en escenarios donde la información puede ser parcial o confusa.

7.5 Usando Transformer

En la sección anterior (Subsección 7.4), se mostró cómo, mediante metaheurísticas como PSO, es posible encontrar conjuntos de parámetros que optimizan el desempeño del juego de parejas para un *nivel de confusión* (α) en particular. Sin embargo, en la práctica, **no siempre conocemos el valor de α con exactitud**. Por ello, el siguiente paso consiste en incorporar el componente de **aprendizaje automático** (redes neuronales) para que el algoritmo se *adapte* dinámicamente a la situación real del juego, infiriendo la confusión a partir de los estados.

7.5.1 Entrenamiento y Construcción del Conjunto de Datos

Para implementar el componente de **Transformers** se siguen las adaptaciones explicadas en secciones anteriores, incluyendo:

- El uso de **SVD** para asegurar la *escalabilidad* cuando el número de estados o la longitud de la secuencia aumenta significativamente.

- La *codificación* de los estados del juego de parejas, representando la matriz de probabilidades, las cartas eliminadas y otras características relevantes.

El **dataset de entrenamiento** se compone de *instancias específicas* del juego (misma disposición de cartas y confusión), para las cuales se obtienen los *parámetros óptimos* usando PSO. A diferencia de la estrategia pura de PSO, donde se busca un conjunto de parámetros que funcionen *en promedio* para mútiples casos de una misma confusión, aquí se recolectan soluciones *individualizadas* (una por cada instancia), enriqueciendo los ejemplos con escenarios más específicos y precisos.

Para fines demostrativos y por *limitaciones de tiempo*, se generó un **conjunto de datos relativamente pequeño** y se entrenó el Transformer en *pocos épocas* (epochs). Aún así, los resultados obtenidos fueron prometedores, sugiriendo que un entrenamiento más extenso y con más casos podría mejorar significativamente el desempeño del modelo.

7.5.2 Comparativa con PSO: Inferencia Dinámica de la Confusión

A continuación, se realizó una prueba cubriendo valores de α en el rango $[0, 1]$. La estrategia basada **exclusivamente** en PSO emplea un *conjunto discreto* de soluciones (las que optimizan cada α discreto considerado), y en tiempo de decisión elige el conjunto de parámetros cuyo α entrenado sea más cercano al valor real. El **Transformer**, en cambio, produce un *conjunto continuo* de parámetros y *no* requiere conocer α, pues *infiere* el nivel de confusión implícitamente a partir de la información de estado.

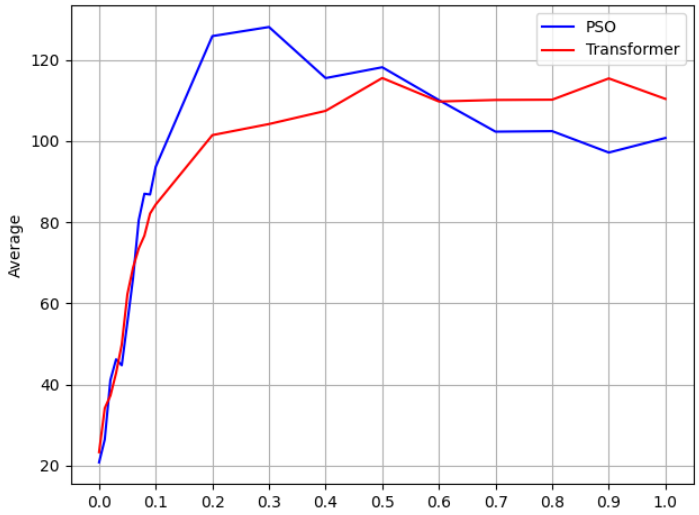

Figura 6: Comparación de resultados entre el Transformer y la estrategia puramente basada en PSO, para valores de α entre 0 y 1.

En la Figura 6, se aprecia la evolución de los resultados de ambos métodos al variar α. Ninguno de los dos generó partidas por encima del límite, lo cual es una buena señal de la robustez de ambas propuestas. Al desglosar el análisis:

- **Confusión hasta 10 %**: Los resultados de PSO y Transformer son muy similares, estando casi empatados. PSO tiende a mostrar un ligero dominio en α más bajos, posiblemente por haber encontrado parámetros muy precisos para ese entorno estable.
- **Confusión entre 10 % y 60 %**: El *Transformer* se convierte en un claro ganador, lo que evidencia el **poder de tener parámetros específicos para cada instancia** y la capacidad de adaptación dinámica. El PSO pierde eficiencia al intentar cubrir un espectro genérico de confusiones, mientras que el Transformer aprovecha su entrenamiento instancia a instancia.
- **Confusión entre 60 % y 100 %**: El PSO recupera cierta ventaja, superando ligeramente al Transformer. Aun así, los desempeños son de nuevo cercanos, pues ambos muestran una monotonía similar a medida que se acerca la confusión total.

Esto indica que **un Transformer con un conjunto de entrenamiento más amplio, abarcando múltiples instancias y confusiones variadas, podría superar en la mayoría de los casos la estrategia basada exclusivamente en PSO**, además de su ventaja inherente de *no requerir el conocimiento previo* de α.

8 Aplicación en un Entorno de Riesgo

En esta sección se presenta otra prueba del algoritmo, enfocada en un **problema de ganancia bajo riesgo**. Se tienen n rondas, y en cada ronda el agente decide cuánto apostar ($\in [0, +\infty)$). Existe una *función de riesgo*, simulada en este caso por una probabilidad desconocida, que determina si la cantidad apostada se *suma* o *resta* al capital acumulado. El objetivo final es **maximizar el capital** tras completarse las n rondas.

8.1 Modelo y Ajuste Mediante Proporcionalidad Inversa

Dado que el valor a apostar puede ser arbitrariamente grande, se opta por la **función de proporcionalidad inversa** como mecanismo de asignación probabilística, asegurando que para valores de entrada $x \in [0, 1)$ se mapee a $[0, +\infty)$. Posteriormente, se emplea **PSO** para encontrar los parámetros de dicha función que maximizan la ganancia promedio en un *entorno de riesgo* dado por:

$$\text{Probabilidad de Ganar/Apostar} \;=\; \alpha\,|\sin(i)|,$$

donde i es el número de ronda, y α varía en el rango [0, 1]. La idea es que, cuanto mayor sea α, *más volátil* es el ambiente de apuestas, incrementando el riesgo de pérdida.

8.1.1 Pruebas y Resultados con PSO

Para evaluar la efectividad de los parámetros de la función de proporcionalidad inversa, se realizaron 100 ejecuciones del juego, ajustando la duración a un número fijo de rondas y registrando:

- **Promedio de ganancia**: capital final promedio tras las 100 ejecuciones.
- **Cantidad de veces que se pierde dinero**: número de ejecuciones (de 100) en las que el capital final resulta menor que 0.

La *función objetivo* usada por PSO se definió para equilibrar la maximización de la ganancia y la reducción del número de pérdidas. A continuación, se presentan los resultados obtenidos para distintas configuraciones de α.

Cuadro 2: Resultados al variar α en el rango $[0, 1]$. Cada valor se basa en 100 ejecuciones.

α	Perdidas	Ganancia Promedio
0.0	0	1.05×10^{15}
0.1	0	2.13×10^{15}
0.2	3	3.22×10^{14}
0.3	3	5.74×10^{14}
0.4	5	9.59×10^{14}
0.5	18	6.81×10^{14}
0.6	18	4.06×10^{14}
0.7	33	5.05×10^{14}
0.8	52	2.76×10^{14}
0.9	65	3.83×10^{14}
1.0	83	7.57×10^{14}

Análisis de los Resultados

- $\alpha = 0{,}0$ y $\alpha = 0{,}1$: Se obtienen ganancias extremadamente altas, con **cero pérdidas** en 100 ejecuciones. El entorno es, en práctica, muy estable, lo que facilita el incremento sostenido del capital.

- $\alpha = 0{,}2$ a $\alpha = 0{,}4$: Comienza a existir cierto *riesgo*, reflejado en unas pocas ejecuciones con pérdidas, mientras los valores promedio de ganancia siguen siendo del orden de 10^{14} o 10^{15}.
- $\alpha \geq 0{,}5$: A medida que sube la volatilidad, aumenta notablemente la probabilidad de pérdida, llegando a más de la mitad de las ejecuciones para $\alpha = 0{,}8$ y superiores. Sin embargo, el capital promedio sigue resultando muy alto, indicando que en las partidas favorables se logran ganancias enormes, compensando en parte las caídas.

Estos resultados evidencian que, si bien se pueden alcanzar ganancias elevadas incluso en entornos muy volátiles, *el precio a pagar es un incremento sustancial en la posibilidad de terminar en números rojos*. Esto es consistente con la idea de que, al usar la función de proporcionalidad inversa, se pueden tomar apuestas más elevadas en condiciones de alto riesgo, lo que dispara tanto la ganancia como la pérdida potencial.

8.2 Pruebas con la Inversa de la Exponencial

Además de la función de proporcionalidad inversa, se consideró una **función exponencial inversa** para la selección del valor a apostar. La expresión, partiendo de un valor $x \in [0, 1)$, se define como:

$$f(x) \;=\; -\lambda \,\ln\big(1 - x\big),$$

donde $\lambda > 0$ es el parámetro a optimizar. Nótese que:

- Para $x \approx 0$, $f(x) \approx 0$, de modo que se opta por apuestas conservadoras.
- A medida que x se acerca a 1, $-\ln(1 - x)$ tiende a $+\infty$, haciendo crecer $f(x)$ sin cota superior, lo que sigue permitiendo apuestas elevadas.
- El valor λ controla la *escala* de la apuesta, similar a la media deseada en la distribución exponencial.

8.2.1 Configuración Experimental y Resultados

Se mantiene el mismo escenario de n rondas con una función de riesgo desconocida, modelada por $\alpha \sin(i)$ para cada ronda i. Se realizaron 100 ejecuciones para cada valor de $\alpha \in \{0; 0{,}1; 0{,}2; 0{,}3; 0{,}4; 0{,}5; 0{,}6; 0{,}7; 0{,}8; 0{,}9; 1{,}0\}$. El **PSO** se encarga de encontrar el λ que maximice la ganancia promedio y minimice el número de pérdidas.

A continuación se presentan los resultados, mostrando (en 100 ejecuciones):

- **Veces que se pierde dinero** (capital final menor que 0).
- **Promedio de ganancia** entre todas las partidas jugadas.

Cuadro 3: Resultados con la función inversa de la exponencial para distintos valores de α. Cada valor corresponde a (Veces que se pierde dinero, Promedio de ganancia).

α	Perdidas	Ganancia Promedio
0.0	0	6633.87
0.1	0	3748.56
0.2	0	9675.27
0.3	0	4645.20
0.4	0	3673.89
0.5	0	3582.46
0.6	0	1953.96
0.7	1	727.12
0.8	61	31.89
0.9	28	0.32
1.0	8	0.05

8.2.2 Análisis de Resultados

- **Rangos bajos de α (0 a 0,6):** Se observa un número de pérdidas cero, mientras que las ganancias promedio varían. El comportamiento es más *conservador* que con la función de proporcionalidad inversa, pues no se realizan apuestas desmesuradas: esto se refleja en ganancias más moderadas, pero también en menores riesgos.
- **Riesgo mayor ($\alpha = 0{,}7$ a 1,0):** Empiezan a surgir pérdidas más frecuentes. El promedio de ganancia baja a niveles cercanos a cero. Esto indica que la inversa exponencial se vuelve menos lucrativa a medida que la volatilidad del riesgo aumenta, aun siendo más "estable" que la otra función.
- **Comparación con la Proporcionalidad Inversa:** Mientras la función logarítmica inversa puede generar ganancias enormes a costa de un *alto riesgo*, la inversa exponencial resulta más conservadora en sus apuestas, ofreciendo valores más consistentes y reduciendo considerablemente las ocasiones en que se pierde dinero.

En conclusión, **la función exponencial inversa** constituye una opción más moderada para entornos muy volátiles, reduciendo el número de pérdidas incluso cuando α se torna elevado, a expensas de no alcanzar *picos de ganancias* tan impactantes como la solución más agresiva. Este compromiso entre potencial de ganancia y estabilidad es clave al diseñar estrategias de decisión en entornos de riesgo.

8.3 Combinando ambas Funciones

Dado que cada función propuesta, la *proporcionalidad inversa* y la *inversa exponencial*, presenta fortalezas y debilidades para distintos niveles de riesgo, surge la idea de combinar múltiples opciones bajo una *función maestra*. En esta prueba, se utilizó la *función logarítmica inversa* como la encargada de asignar probabilidades a cada una de las dos funciones candidatas:

- **Proporcionalidad inversa**, cuya capacidad de generar ganancias masivas se ve compensada por un riesgo más alto.
- **Inversa exponencial**, que mantiene un comportamiento más conservador, reduciendo las pérdidas a costa de no alcanzar picos de ganancia tan elevados.

La **función maestra** determina, en cada ronda, la probabilidad de elegir una u otra estrategia para fijar la apuesta. Posteriormente, la función seleccionada asigna el valor final de la decisión según sus propios parámetros. De este modo, el algoritmo puede adaptarse dinámicamente conforme varía el entorno de riesgo.

8.3.1 Resultados y Análisis

Para comprobar la eficacia de esta combinación, se llevaron a cabo experimentos con distintos valores de α, manteniendo la misma función de riesgo $\alpha \sin(i)$. A continuación, se muestran (en 100 ejecuciones) la cantidad de partidas con pérdidas y la ganancia promedio:

Cuadro 4: Resultados al combinar la proporcionalidad inversa y la inversa exponencial, usando una función logarítmica inversa como maestra. Los datos corresponden a (Veces que se pierde dinero, Promedio de ganancia) en 100 ejecuciones.

α	**Pérdidas**	**Ganancia Promedio**
0.0	0	3.60×10^{14}
0.1	0	4.01×10^{14}
0.2	0	4.02×10^{15}
0.3	6	1.37×10^{14}
0.4	5	3.07×10^{14}
0.5	10	8.29×10^{13}
0.6	0	2490.56
0.7	0	1978.15
0.8	51	1.47×10^{13}
0.9	1	0.02
1.0	2	0.25

Observaciones Principales:

- $\alpha \leq 0{,}5$**:** Para riesgos bajos o moderados, la estrategia *tiende a seleccionar* con mayor frecuencia la *proporcionalidad inversa*, pues el número de pérdidas es relativamente bajo y los incrementos de capital pueden ser muy elevados. En estas condiciones, el algoritmo explota las grandes ganancias potenciales.
- $\alpha = 0{,}6$ **y** $\alpha = 0{,}7$**:** Cuando la volatilidad incrementa, la *proporcionalidad inversa* empieza a mostrar más pérdidas. La función maestra, entonces, **decide favorecer la inversa exponencial**, que garantiza un número de pérdidas cercano a cero a cambio de ganancias más moderadas.
- $\alpha = 0{,}8$**:** En este valor intermedio-alto, la *inversa exponencial* también exhibe problemas (mayores pérdidas y menor ganancia), de modo que la estrategia vuelve a inclinarse hacia la *proporcionalidad inversa*, a pesar del riesgo, esperando aprovechar rachas favorables para compensar las pérdidas.
- α **muy alta (**0,9 **y** 1,0**):** Ambas funciones pierden efectividad; el algoritmo opta, en la práctica, por *apostar valores muy próximos a 0* con alta probabilidad, reduciendo así el riesgo de pérdida. Esto se refleja en las cifras: el promedio de ganancia es casi despreciable, pero también se controla el número de partidas negativas.

Conclusiones:

- La **función maestra** resulta una excelente extensión, pues permite combinar *lo mejor* de múltiples estrategias en respuesta al *nivel de riesgo real* que se va detectando.
- En *riesgos bajos*, apostar más agresivamente (proporcionalidad inversa) maximiza ganancias; conforme la volatilidad aumenta, es preferible la inversa exponencial; y en casos extremos de altísimo riesgo, lo mejor es escoger *apuestas cercanas a cero.*
- Usando la estrategia del Transformer este enfoque podría ajustarse aún más finamente, *infiriendo dinámicamente* la situación de riesgo y adaptando la decisión en cada ronda.

De esta forma, la **combinación de funciones probabilísticas, gestionada por una función maestra**, pone de manifiesto cómo el algoritmo puede *evolucionar* según el nivel de riesgo presente en el entorno, sin limitarse a una sola estrategia que podría fallar rotundamente en determinadas circunstancias.

9 Mínimo Conjunto de Vértices de Cobertura

El **problema de encontrar la mínima cantidad de vértices de un grafo cuyas aristas alcancen a todos los nodos** (Mínimo Conjunto de Vértices de Cobertura, o *Minimum Vertex Cover*), es **NP-Completo**, implicando que, al crecer el número de nodos, encontrar la solución óptima en tiempo polinómico se considera inalcanzable a menos que P = NP. Por ello, se usan *estrategias aproximadas* que potencialmente devuelven valores cercanos al óptimo, aunque pueden excederlo.

9.1 Estrategia Aproximada Greedy

Una estrategia común de aproximación es:

1. Ordenar los nodos por su número de aristas (grado).
2. Escoger el nodo con **mayor grado**.
3. Remover ese nodo y todos los nodos que alcanza a *distancia de una arista* (junto a sus aristas correspondientes).
4. Repetir en el *grafo resultante* hasta cubrir todos los nodos.

En grafos donde es posible calcular el óptimo (digamos, de hasta 50 nodos), se observa que este método *casi siempre* retorna el valor óptimo o (óptimo + 1). Sin embargo, que sea muy efectivo *no* significa que no haya margen de mejora. Si se traslada a un problema logístico (por ejemplo, localización de centros de datos para cubrir todas las regiones de un país), un centro adicional podría representar *millones de dólares*.

9.2 Aplicando la Solución Propuesta

La idea es que en cada ronda, en lugar de *forzar* el nodo de mayor grado, se introduzca **exploración** gracias a la **función de asignación probabilística**, pudiendo descubrir combinaciones que el greedy omitiría.

9.3 Estrategia Conjunta: Mezclando el Greedy y el Algoritmo Probabilístico

El **método aproximado** descrito (ordenar por número de aristas y seleccionar el nodo con mayor grado) resulta muy eficiente, aunque no garantiza siempre la solución óptima. De hecho, puede haber *empates* entre varios nodos de máximo grado, y aun así el método no exploraría opciones con grado algo menor que podrían, combinadas con otros nodos, *cubrir* al grafo con menos nodos.

Incorporando la Función de Asignación Probabilística

- **Decisiones en cada ronda**: se dispone de un conjunto de nodos candidatos (los del grafo aún no cubiertos).
- **Criterio original**: escoger *siempre* el nodo de mayor grado.
- **Función de Asignación (FA)**: en vez de forzar el nodo de máximo grado, se asigna a cada nodo una probabilidad proporcional a su grado; a su vez, existe cierta *probabilidad no nula* de escoger un nodo que no sea el máximo. Esto introduce **exploración**, pudiendo así descubrir coberturas no alcanzadas por el greedy puro.

Estrategia Conjunta

- **Ajustar Parámetros de FA en general**: si se entrenan los parámetros para *todas* las instancias posibles, es probable que la estrategia resultante se parezca al greedy (dado lo efectivo que es en la mayoría de casos).
- **Enfoque Específico en los "casos difíciles"**:
  - Se generan n grafos y se compara *greedy* vs. óptimo.

- Sólo en *los casos* donde el greedy no alcanza el óptimo se enfoca la optimización de la estrategia probabilística.

- **Combinación Final**:

  1. Para cada grafo, se corre el *greedy* y el *algoritmo probabilístico*.
  2. Se elige la *cobertura mínima* entre ambas salidas.

  Esta combinación garantiza no ser peor que el greedy, y *puede* superarlo en ciertos escenarios.

Complejidad y Práctica

- **Complejidad similar al Greedy**: El algoritmo base mantiene la misma complejidad temporal de la heurística (ordenar nodos, etc.).
- **Constante Adicional**: Surge al introducir la FA, sobre todo si se aplica un *Transformer* con SVD. Aunque esto añada un coste computacional, es todavía *polinómico* y, en la práctica, sólo un factor multiplicativo.
- **Beneficio**: Nunca empeora el resultado del greedy y, en casos donde el greedy falla, *logra mejoras* gracias a la exploración probabilística.

9.4 Pruebas con Grafos Aleatorios de Hasta 50 Nodos

9.4.1 Generando y Comparando con el Óptimo

- Se generaron 400 grafos con ≤ 50 nodos, probabilidad 50 % de arista entre cada par de nodos.
- Para cada grafo, se calculó el óptimo (vía algoritmo exhaustivo).
- El greedy resultó no óptimo en 44 grafos, confirmando que *puede fallar*.

9.4.2 Uso de la Función Logarítmica Inversa

Dado que en cada ronda tenemos un conjunto finito de decisiones (nodos), y se quiere favorecer con *mayor probabilidad* al de grado alto, se escoge la **función logarítmica inversa**. Para ajustar sus parámetros, se usa **PSO** con la siguiente función de *fitness* (pseudocódigo):

```
def fitness(graph):
    worst_case = None
    for _ in range(10):
        v = problem.run(graph)
        if worst_case is None or worst_case < v:
            worst_case = v
    return worst_case
```

Resultados:

- Para cada uno de los 44 grafos donde el greedy fallaba se optimizaron los parámetros de la función, y se obtuvo que la solución *empató* al greedy en 34 y *ganó* en 10, *sin perder en ninguno*.

9.5 Entrenamiento de un Transformer

- Se entrenó el Transformer (con la técnica de SVD para entradas de estado) en **los 10 casos** donde se venció al greedy.
- Posteriormente, se generaron 100 nuevos grafos (hasta 50 nodos, probabilidad 50 % de arista).
- *Greedy* acierta el óptimo en 89 casos, mientras la *estrategia conjunta* llega a 93.
- Si bien la mejora es discreta, se acerca un poco más al valor perfecto de 100 % de casos óptimos, además de que en problemas donde *un solo nodo extra* implica millones de dólares, cualquier avance es valioso.

9.6 Conclusiones

- El **greedy** es muy eficaz en grafos aleatorios de tamaño moderado, pero no siempre logra el óptimo.
- La estrategia probabilística, combinada con una metaheurística (PSO) o un modelo neuronal (Transformer), mejora *ligeramente* en los casos donde el greedy falla.
- La *combinación* final: tomar la *solución mínima* entre {greedy, algoritmo probabilístico} garantiza que nunca empeora los resultados, y potencialmente gana en ciertos grafos difíciles.
- Entrenar con más instancias y permitir arquitecturas de *mayor complejidad* (más capas neuronales) podría incrementar el número de casos donde se supera el greedy.

En definitiva, aunque el **ganar 4 casos más** en 100 grafos *parezca* poco, en un problema logístico a gran escala, *un solo nodo menos* puede equivaler a un ahorro masivo en recursos, justificando plenamente el uso de estrategias probabilísticas adaptativas.

10 Conclusiones Generales

A lo largo de este trabajo se propuso y validó un **algoritmo híbrido** que integra la asignación probabilística de decisiones, la optimización mediante metaheurísticas (PSO) y el uso de redes neuronales (Transformer) para abordar diversos problemas de toma de decisión bajo incertidumbre. Se presentan a continuación las conclusiones más destacadas:

1. **Asignación Probabilística:**
   - El uso de funciones tales como la *logarítmica inversa*, la *proporcionalidad inversa* o la *inversa exponencial* permite convertir un valor aleatorio uniforme en un índice (o cantidad) que asigna probabilidades a las decisiones disponibles.
   - Se demostró que, mediante condiciones de borde y normalización, es posible garantizar que la asignación cumpla las propiedades deseadas, favoreciendo la exploración y explotando las mejores opciones.
2. **Optimización por Metaheurísticas (PSO):**
   - La selección de parámetros óptimos para dichas funciones probabilísticas mediante PSO mostró resultados sólidos en diversos problemas, desde la *memorización imperfecta* hasta entornos de *riesgo* con funciones desconocidas.
   - La capacidad de PSO para explorar el espacio de parámetros de manera estocástica y eficiente garantiza que, con un número moderado de iteraciones, se encuentren configuraciones competitivas.
3. **Redes Neuronales para Selección Dinámica (Transformer):**
   - El componente de **aprendizaje automático** introdujo la capacidad de *ajustar los parámetros de forma continua y en tiempo real*, infiriendo condiciones como el nivel de confusión o riesgo directamente del estado del problema.
   - El uso de técnicas como la *Descomposición en Valores Singulares (SVD)* se demostró esencial para la escalabilidad, permitiendo manejar secuencias largas y datos de entrada variables sin agotar recursos computacionales.
   - Las comparaciones mostraron que, si bien PSO puede bastar para ciertos rangos de incertidumbre, el *Transformer* con un conjunto de entrenamiento amplio y diverso puede superar la estrategia puramente metaheurística, al no requerir el conocimiento exacto de parámetros externos (por ejemplo, el valor de confusión).
4. **Combinación y Flexibilidad:**
   - Se evidenció la ventaja de **combinar varias funciones probabilísticas** bajo una *función maestra*, que decide cuál resulta más adecuada en cada situación. Esto incrementó la adaptabilidad frente a diferentes escenarios, desde riesgo bajo hasta situaciones de alta volatilidad.
   - En problemas de riesgo, tal flexibilidad permitió un *balance* entre maximizar la ganancia (funciones más agresivas) y minimizar la probabilidad de pérdidas (funciones conservadoras), evitando caer en un desempeño deficiente en entornos particularmente adversos.
5. **Aplicaciones y Futuras Líneas de Investigación:**
   - Los casos estudiados confirman la versatilidad de la propuesta, abriendo la puerta a nuevas aplicaciones, por ejemplo, en *planificación financiera*, *logística adaptativa* o *control de procesos*.
   - Entre las oportunidades de mejora destacan:
     - Profundizar en el entrenamiento del *Transformer* con **datasets más amplios** y *modelos de memoria* más complejos.
     - Hibridar PSO con otros métodos de optimización o aprendizaje (por ejemplo, algoritmos genéticos o recocido simulado) para cubrir un espectro todavía más amplio de condiciones.

En conclusión, este trabajo confirma que la *integración de modelos probabilísticos, metaheurísticas y técnicas de aprendizaje automático* es una vía poderosa para resolver problemas complejos de decisión, ya sea bajo incertidumbre, riesgo o restricciones cambiantes. La posibilidad de ajustar dinámicamente los parámetros y la capacidad de explorar soluciones en un espacio continuo permite al algoritmo adaptarse eficazmente a un amplio abanico de entornos, demostrando así su **robustez** y **versatilidad**.

10.1 Repositorio de Código

Para quienes deseen explorar la implementación práctica de los conceptos y algoritmos expuestos en este trabajo, se ha creado un repositorio en *GitHub* que contiene:

- El código fuente de las funciones y modelos desarrollados.
- Notebooks con las pruebas realizadas y ejemplos de uso.

El repositorio se encuentra disponible en la siguiente dirección:

https://github.com/EnzoDtoste/Decision-Making-Optimization

Referencias

[1] Kaelbling, L. P., Littman, M. L., Moore, A. W. (1996). Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, 4, 237–285.

[2] Sutton, R. S., Barto, A. G. (2018). Reinforcement Learning: An Introduction (2ª ed.). MIT Press.

[3] Hein, D., Hentschel, A., Runkler, T. A., Udluft, S. (2016). Reinforcement Learning with Particle Swarm Optimization Policy (PSO-P) in Continuous State and Action Spaces. International Journal of Swarm Intelligence Research, 7(3), 23–47.

[4] France, K. K., Sheppard, J. W. (2023). Factored Particle Swarm Optimization for Policy Co-training in Reinforcement Learning. En Proc. Genetic and Evolutionary Computation Conference (GECCO 2023), pp. 30–38.

[5] Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., et al. (2021). Decision Transformer: Reinforcement Learning via Sequence Modeling. En NeurIPS 34 (Advances in Neural Information Processing Systems).

[6] Janner, M., Li, Q., Levine, S. (2021). Offline Reinforcement Learning as One Big Sequence Modeling Problem. En NeurIPS 34.

[7] Parisotto, E., Song, H. F., Rae, J. W., et al. (2020). Stabilizing Transformers for Reinforcement Learning. En Proc. 37th ICML (International Conference on Machine Learning), PMLR 119:7487–7498.

[8] He, M., Zhou, Y., Li, Y., Wu, G., Tang, G. (2020). Long short-term memory network with multi-resolution singular value decomposition for prediction of bearing performance degradation. Measurement, 156, 107582.

[9] Kang, M., Kim, J.-M. (2013). Singular value decomposition-based feature extraction approaches for classifying faults of induction motors. Mechanical Systems and Signal Processing, 41(1–2), 348–356.

[10] Wikipedia contributors, "Lebesgue measure," *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Lebesgue_measure

[11] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, Perth, Australia, 1995, pp. 1942–1948, https://www.cs.tufts.edu/comp/150GA/homeworks/hw3/_reading6%201995%20particle%20swarming.pdf

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All You Need," *arXiv preprint*, arXiv:1706.03762, 2017, https://arxiv.org/pdf/1706.03762

[13] Wikipedia contributors, "Singular value decomposition," *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Singular_value_decomposition

[14] Wikipedia contributors, "Matrix norm," *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Matrix_norm#Frobenius_norm