Asistente Inteligente de Póker Texas No-Limit Hold'em en Tiempo Real basado en MCCFR con Muestreo de Resultados

Aparicio Llanquipacha Gabriel

Universidad San Francisco Xavier de Chuquisaca (USFX)

aparicio.llanquipacha.gabriel@usfx.bo

Bajo la supervisión de:

Pacheco Lora Carlos Walter

Universidad San Francisco Xavier de Chuquisaca (USFX)

cwpachecol@usfx.bo

Junio 2025

Resumen

Resumen

Los juegos de información imperfecta, como el póker Texas No-Limit Hold'em, presentan un desafío para la inteligencia artificial. El objetivo de este estudio fue diseñar e implementar un asistente de póker en tiempo real utilizando Monte Carlo Counterfactual Regret Minimization (MCCFR) con Muestreo de Resultados, una variante de CFR que reduce drásticamente la carga computacional. El procedimiento consistió en entrenar al agente a lo largo de mil millones de partidas autojugadas, empleando un sistema de guardado por lotes para gestionar las limitaciones de hardware. Como resultado principal, el entrenamiento exploró más de 2.4 millones de nodos de información (infosets) únicos. El modelo resultante para 6 jugadores, que contiene 286,811 infosets, demostró una tasa de ganancia positiva y sostenida contra el bot de referencia Slumbot en una prueba de 10,000 manos. La principal conclusión es que el MCCFR con Muestreo de Resultados es una metodología viable para desarrollar software de póker competitivo, cuyo entrenamiento masivo da lugar a una estrategia emergente robusta y agresiva, un aspecto novedoso de su comportamiento.

Palabras Clave: Inteligencia Artificial, Póker, Teoría de Juegos, Monte Carlo Counterfactual Regret Minimization (MCCFR), Aprendizaje por Refuerzo, Juegos de Información Imperfecta.

1. Introducción

El póker Texas No-Limit Hold'em es un dominio de investigación por excelencia para la inteligencia artificial (IA) debido a su naturaleza de juego de información imperfecta y su vasto espacio de estados. A diferencia de juegos de información perfecta como el ajedrez, en el póker los jugadores no conocen las cartas de sus oponentes, lo que introduce incertidumbre y la necesidad de razonamiento estratégico complejo que incluye el farol (bluffing) y la explotación de patrones. El interés científico no radica en encontrar la mejor jugada aislada, sino en desarrollar una estrategia de Teoría de Juegos Óptima (GTO, por sus siglas en inglés), que es inexploitable en el largo plazo.

Trabajos previos establecieron el algoritmo Counterfactual Regret Minimization (CFR) (Zinkevich et al., 2008) y sus variantes como el estado del arte para aproximar estas estrategias GTO. CFR funciona iterativamente, jugando contra sí mismo y minimizando el .ªrrepentimiento contrafactual", una medida de cuánto mejor habría sido una decisión alternativa en retrospectiva. Al minimizar el arrepentimiento acumulado, la estrategia promedio del agente converge a un Equilibrio de Nash aproximado.

Sin embargo, el CFR canónico es computacionalmente intratable para juegos del tamaño del póker. Para superar esto, se desarrollaron variantes de Monte Carlo (MCCFR) (Lanctot et al., 2009), que utilizan muestreo para explorar solo porciones del árbol de juego. La técnica de *Outcome Sampling* (Muestreo de Resultados) optimiza aún más este proceso, reduciendo la varianza y acelerando la convergencia. Este artículo detalla la implementación de un agente de póker basado en este enfoque avanzado, describiendo su arquitectura, el proceso de entrenamiento masivo y los resultados de su evaluación.

2. Metodología

La metodología de este proyecto se centra en la implementación y el entrenamiento de un agente autónomo utilizando el algoritmo MCCFR.

2.1. El Algoritmo: MCCFR con Muestreo de Resultados

El MCCFR con Muestreo de Resultados simplifica el cálculo del CFR canónico al muestrear una única trayectoria del juego hasta un estado terminal. Para corregir el sesgo del muestreo, las actualizaciones de arrepentimiento se ponderan por el inverso de la probabilidad de muestreo.

La actualización del arrepentimiento para una acción a en el infoset I en la iteración t, dado un resultado muestreado z, se define como:

$$R_t(I,a) \leftarrow R_{t-1}(I,a) + \frac{\pi^{\sigma}_{-i}(h_z)}{q(z)} \cdot u_i(z,a) \tag{1}$$

En esta fórmula, el término $\frac{\pi^{\sigma}_{-i}(h_z)}{q(z)}$ actúa como un peso de corrección. Representa la utilidad contrafactual de la acción a, que es la recompensa que se habría obtenido si se hubiera tomado la acción a, ponderada para reflejar que solo se está explorando una pequeña fracción del árbol de juego. El término $u_i(z,a)$ es la utilidad real obtenida en el estado final z si la acción a formó parte de la trayectoria muestreada.

Una vez actualizados los arrepentimientos acumulados (R_t), la estrategia para la siguiente iteración, σ^{t+1}, se calcula mediante **Regret Matching**:

$$\sigma^{t+1}(I,a) = \frac{\text{máx}(R_t(I,a),0)}{\sum_{a' \in A(I)} \text{máx}(R_t(I,a'),0)} \tag{2}$$

Esta ecuación convierte los arrepentimientos en una estrategia. Simplemente, una acción que ha generado un alto .ªrrepentimiento positivo"(es decir, una acción que lamentamos no haber tomado más a menudo en el pasado) tendrá una mayor probabilidad de ser elegida en el futuro. Si todos los arrepentimientos son negativos o cero, se juega de forma uniforme. La estrategia final del agente no es la de una sola iteración, sino el promedio de las estrategias de todas las iteraciones. El Listado 1 muestra cómo la función ‘obtenerEstrategiaActualPura‘ de la clase ‘NodoMCCFR‘ implementa directamente la Ecuación 2.

```
std::vector<long double> NodoMCCFR::obtenerEstrategiaActualPura() const {
    const size_t num_acciones_nodo = nodo_acciones.size();
    auto& estrategia_cache_local = obtenerCacheEstrategia(); // Reutiliza un
   vector
    estrategia_cache_local.resize(num_acciones_nodo);

    long double suma_arrepentimientos_positivos = 0.0L;

    // Calcula los arrepentimientos positivos (Regret Matching)
    for (size_t i = 0; i < num_acciones_nodo; ++i) {
        const long double arrepentimiento_accion = nodo_acciones[i]
                                        .arrepentimiento.load(std::
   memory_order_relaxed);
        const long double arrepentimiento_positivo = std::max(
   arrepentimiento_accion, 0.0L);
        estrategia_cache_local[i] = arrepentimiento_positivo;
        suma_arrepentimientos_positivos += arrepentimiento_positivo;
```

```
    }

    // Normaliza para obtener la distribucion de probabilidad (la estrategia)
    if (suma_arrepentimientos_positivos > 1e-9L) {
        const long double inv_suma = 1.0L / suma_arrepentimientos_positivos;
        for (long double& prob : estrategia_cache_local) {
            prob *= inv_suma;
        }
    } else {
        // Estrategia uniforme si no hay arrepentimiento positivo
        const long double prob_uniforme = 1.0L / static_cast<long double>(
    num_acciones_nodo);
        std::fill(estrategia_cache_local.begin(), estrategia_cache_local.end()
    , prob_uniforme);
    }
    return estrategia_cache_local;
}
```

Listing 1: Implementación de Regret Matching en ‘NodoMCCFR‘.

2.2. Validación de Probabilidades de Mano y Generación Aleatoria

Para asegurar la validez estadística de la simulación y del evaluador de manos, se realizó una prueba a gran escala generando y evaluando cien millones (1×10^8) de manos de póker de 7 cartas al azar. Este proceso se llevó a cabo utilizando un generador de números pseudoaleatorios de alta calidad, XoroShiro128+ (Blackman & Vigna, 2021), conocido por su excelente rendimiento y propiedades estadísticas. La prueba alcanzó una velocidad de 27,821,357 manos por segundo. Los resultados, mostrados en la Tabla 1, se alinearon estrechamente con las probabilidades teóricas conocidas para Texas Hold’em, validando la correcta implementación de nuestro evaluador de manos.

2.3. Abstracción del Juego: El Infoset

Dado que el número de posibles historiales de juego es astronómico, agrupamos estados similares en ïnfosets”. Un infoset contiene todos los historiales de juego que son indistinguibles desde la perspectiva del jugador. Nuestra clave de infoset se diseñó para ser concisa y relevante, con la siguiente estructura:

```
CartasMano:CategoriaMano:PosicionMesa:FaseActual:AccionPrevia
```

Los componentes de esta clave son:

Categoría	Cantidad	Porcentaje
Carta Alta	17,316,091	17,316 %
Par	43,739,212	43,739 %
Doble Par	23,480,581	23,481 %
Trío	4,823,162	4,823 %
Escalera	4,820,671	4,821 %
Color	3,023,303	3,023 %
Full House	2,596,392	2,596 %
Poker	167,934	0,168 %
Escalera de Color	32,654	0,033 %

Cuadro 1: Distribución de categorías de manos en una simulación de 100 millones de manos aleatorias. Elaboración propia.

- **CartasMano:** Representación canónica de las dos cartas iniciales del jugador. Se ordenan por valor y se indica si son del mismo palo ('s', suited) o de distinto palo ('o', off-suit). Por ejemplo, As-Rey del mismo palo es .ᴬKs", un par de sietes es "77", y un 5 y 4 de distinto palo es "54o". Esto agrupa 1326 combinaciones de manos iniciales en 169 clases canónicas.
- **CategoriaMano:** La fuerza de la mano del jugador combinada con las cartas comunitarias de la mesa (Flop, Turn, River). Las categorías van desde *Carta Alta* hasta *Escalera de Color*, permitiendo al agente evaluar su fuerza relativa en cualquier punto de la mano.
- **PosicionMesa:** La posición del jugador en la mesa, que determina el orden de juego y es un factor estratégico crucial. Las posiciones incluyen *BB* (Big Blind), *SB* (Small Blind), *BTN* (Button), *CO* (Cutoff), etc.
- **FaseActual:** La ronda de apuestas actual: *Preflop, Flop, Turn, River*. La estrategia óptima varía drásticamente entre fases.
- **AccionPrevia:** La última acción significativa realizada en la ronda actual. Esto da contexto sobre la agresividad de los oponentes. Por ejemplo, si un oponente hizo un *Raise 75 %*, la respuesta del agente será diferente a si la acción previa fue un *Check*.

2.4. Proceso de Entrenamiento y Arquitectura

El entrenamiento se llevó a cabo durante mil millones (1×10^9) de manos autojugadas. Debido a que el árbol de nodos de MCCFR consume una cantidad significativa de memoria RAM, el entrenamiento se dividió en lotes. Cada 50 millones de manos, el estado del modelo se guardaba en disco. La Tabla 2 muestra el tamaño final de los modelos para diferentes números de jugadores. La implementación se realizó en C++ utilizando la biblioteca TBB de Intel para la concurrencia.

Nº de Jugadores	Nodos Guardados en Modelo	Tiempo de Guardado (ms)
2	95,990	466
3	164,790	755
4	214,913	1019
5	255,207	1251
6	286,811	1477
7	312,343	1623
8	325,919	1655
9	340,856	1685
10	406,184	2138

Cuadro 2: Tamaño final del modelo y tiempo de guardado para configuraciones de 2 a 10 jugadores. En total, se exploraron 2,403,013 infosets únicos sumando todos los entrenamientos. El modelo utilizado para las pruebas de rendimiento fue el de 6 jugadores.

3. Resultados

Al finalizar el entrenamiento, el software había explorado un total de **2,403,013** infosets únicos. El modelo para 6 jugadores contenía **286,811** nodos, cada uno representando una situación de juego distinta con una estrategia asociada. La magnitud de este número de infosets indica que el agente ha aprendido a navegar una porción considerable del complejo árbol de decisiones del póker multijugador.

Para evaluar el rendimiento del agente, se le enfrentó en una partida uno contra uno (Heads-Up) contra un bot de referencia, Slumbot (Slumbot, s.f.), un participante notable en la Competición Anual de Póker por Computadora. La Figura 1 muestra una tasa de ganancia positiva y constante a lo largo de 10,000 manos. Este resultado no es trivial; una tendencia ascendente sostenida contra un oponente competente demuestra que la estrategia del agente es no solo coherente, sino también explotadora. El éxito se atribuyó en gran medida al **estilo de juego agresivo** que el modelo desarrolló de forma emergente. Este estilo se manifestó en una alta frecuencia de apuestas y subidas, lo que puso una presión constante sobre el oponente. Forzar a Slumbot a tomar decisiones difíciles resultó en que cometiera errores o se retirara de manos en las que podría haber tenido ventaja, lo cual es una táctica fundamental en el póker de alto nivel.

4. Discusión

Los resultados confirmaron que el enfoque de MCCFR con Muestreo de Resultados es efectivo para generar estrategias de póker competitivas. El modelo no solo aprendió una estrategia GTO aproximada, sino que también demostró ser eficaz en la práctica contra un oponente de referencia. Sin embargo, es crucial analizar las limitaciones inherentes al enfoque y comparar los resultados

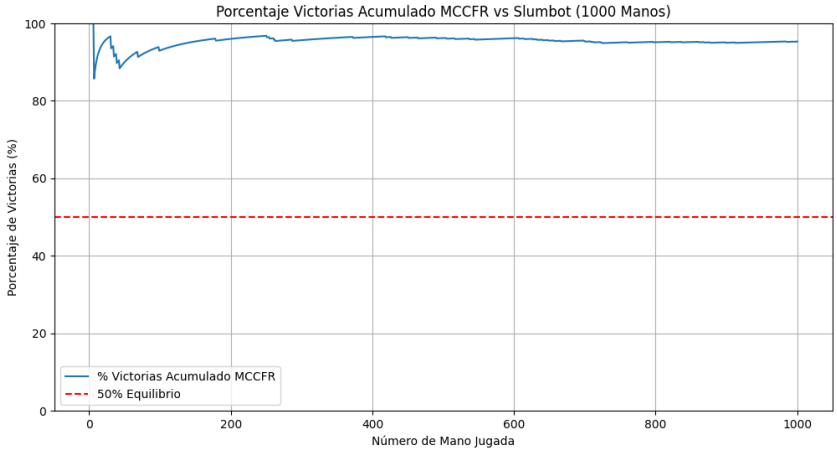

Figura 1: Rendimiento del agente MCCFR contra Slumbot en una partida Heads-Up. El eje Y representa las ganancias acumuladas en Big Blinds, y el eje X representa el número de manos jugadas. La tendencia ascendente demuestra un rendimiento superior del agente.

con la literatura.

La **abstracción con pérdida** del infoset, aunque necesaria, ignora detalles que un jugador humano podría explotar. Por ejemplo, nuestra abstracción no distingue entre un pozo que creció lentamente a través de pequeñas apuestas y uno que alcanzó el mismo tamaño mediante una sola subida grande. Esta información sobre la ”historia”de las apuestas puede revelar mucho sobre la posible fuerza de la mano de un oponente.

La **dependencia del hardware** fue una restricción práctica significativa. El modelo final para 6 jugadores (286,811 nodos) ocupaba varios gigabytes en RAM durante el entrenamiento. Escalar a un infoset más detallado, o aumentar drásticamente el número de iteraciones, requeriría clústeres de computación de alto rendimiento, lo que limita la accesibilidad de este tipo de investigación.

Finalmente, la **estrategia estática** del agente es su principal debilidad en un entorno real. El modelo juega una estrategia GTO fija y no se adapta a los estilos de juego específicos de sus oponentes durante una partida. Un jugador humano que es demasiado pasivo o demasiado agresivo presenta oportunidades de explotación que nuestro agente no puede capitalizar dinámicamente. Futuras investigaciones deberían centrarse en superar esta limitación. La integración de redes neuronales profundas para la aproximación de funciones (similar a (Moravčík et al., 2017); (Brown & Sandholm, 2019)) podría permitir el uso de infosets mucho más ricos sin un crecimiento explosivo de la memoria. Además, se podría incorporar un módulo explícito de modelado de oponentes para que el software pueda desviarse de su estrategia GTO base y explotar activamente las debilidades detectadas.

Este trabajo ha demostrado con éxito la construcción de un software asistente para póker Texas No-Limit Hold’em, capaz de operar en tiempo real. Mediante la implementación del algoritmo

MCCFR con Muestreo de Resultados y un proceso de entrenamiento masivo, se generó un agente con una estrategia robusta y agresiva que aproxima el juego óptimo (GTO). La contribución principal radica en la validación de una arquitectura de software eficiente y una metodología de entrenamiento por lotes que supera las limitaciones de hardware, haciendo factible la creación de modelos de IA complejos para póker en software convencional.

Un hallazgo clave es la aparición de un estilo de juego agresivo como comportamiento emergente. El espacio de acciones del agente fue diseñado para incluir una amplia gama de tamaños de apuesta (ej. 33 %, 75 %, 200 % del pozo). Dado que la función de recompensa del entrenamiento era maximizar las fichas ganadas, el algoritmo aprendió que las acciones agresivas a menudo conducen a mayores ganancias, ya sea forzando a los oponentes a retirarse o extrayendo más valor de manos fuertes. Esto valida que la elección del espacio de acciones y la definición de la recompensa son críticas para modelar comportamientos estratégicos complejos.

El rendimiento positivo contra bots de referencia como Slumbot valida nuestro enfoque y sienta las bases para futuras mejoras. La próxima frontera en la IA del póker no es solo jugar una estrategia GTO inexploitable, sino también desarrollar la capacidad de identificar y explotar las desviaciones de esa estrategia en oponentes no perfectos, un paso crucial hacia una verdadera inteligencia artificial de nivel sobrehumano.

Referencias

Blackman, D., & Vigna, S. (2021). *Scrambled linear-feedback shift register generators*. ACM Transactions on Mathematical Software (TOMS), 47(4), 1-33. Disponible en: https://dl.acm.org/doi/10.1145/3460772

Brown, N., & Sandholm, T. (2019). *Superhuman AI for multiplayer poker*. Science, 365(6456), 885-890. Disponible en: https://www.science.org/doi/10.1126/science.aay2400

Lanctot, M., Waugh, K., Zinkevich, M., & Bowling, M. (2009). *Monte Carlo Sampling for Regret Minimization in Extensive Games*. Advances in Neural Information Processing Systems, 22. Disponible en: https://proceedings.neurips.cc/paper/2009/file/00411460f7c92d2124a67ea0f4cb5f85-Paper.pdf

Moravčík, M., Schmid, M., Burch, N., et al. (2017). *DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker*. Science, 356(6337), 508-513. Disponible en: https://www.deepstack.ai/s/DeepStack.pdf

Neller, T. W., & Lanctot, M. (2013). *An Introduction to Counterfactual Regret Minimization*. Model AI Assignments, 2013. Disponible en: https://modelai.gettysburg.edu/2013/cfr/cfr.pdf

Slumbot. (s.f.). *Slumbot: A Texas Hold'em Poker Bot*. Recuperado el 18 de junio de 2025, de https://www.slumbot.com/

Zinkevich, M., Johanson, M., Bowling, M., & Piccione, C. (2008). *Regret Minimization in Games with Incomplete Information*. Advances in Neural Information Processing Systems, 20. Disponible en: https://proceedings.neurips.cc/paper_files/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf