TEC | Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

IC-6200 – Inteligencia Artificial

Tarea Corta 1 – Búsquedas

Estudiantes:

Pablo Mauricio Mesén Alvarado – 2023071259
Samir Fernando Cabrera Tabash – 2022161229
Luis Gerardo Urbina Salazar – 2023156802

Profesor:

Kenneth Roberto Obando Rodríguez

25 de marzo de 2025

II Semestre 2025

Índice

1. Introducción

El presente informe documenta la resolución de dos problemas clásicos mediante algoritmos de búsqueda en inteligencia artificial: *Peg Solitaire* y *Timbiriche* (también conocido como *Dots and Boxes*). Estos juegos de mesa representan desafíos interesantes para el diseño de agentes que exploran espacios de estados, dado que ambos poseen reglas simples pero un crecimiento combinatorio elevado en las posibles configuraciones.

En el caso de **Peg Solitaire**, el objetivo es eliminar todas las piezas del tablero hasta dejar una sola, idealmente en el centro. Cada movimiento consiste en saltar una ficha sobre otra adyacente, eliminándola en el proceso. Este problema se abordó utilizando el algoritmo de búsqueda **A***, que combina el costo acumulado $g(n)$ con una heurística $h(n)$ para estimar el costo restante hasta el estado meta. La clave de la eficiencia radica en diseñar una heurística *admisible* y *consistente*, capaz de guiar la exploración hacia soluciones óptimas sin explorar innecesariamente todos los estados posibles.

Por otra parte, **Timbiriche** es un juego competitivo para dos jugadores, donde cada turno se traza una línea entre puntos de una cuadrícula con el objetivo de completar cuadros. El jugador que completa más cuadros gana la partida. Para este problema se implementó el algoritmo **Minimax** con poda alfa–beta, el cual permite simular decisiones alternadas entre dos jugadores racionales (MAX y MIN), evaluando estados terminales y utilizando una función heurística para estimar la ventaja relativa en estados no terminales. De esta manera, se logra un agente capaz de tomar decisiones estratégicas más allá del turno inmediato.

Ambos algoritmos ilustran el potencial de las técnicas de búsqueda en inteligencia artificial: mientras que A* se enfoca en encontrar soluciones óptimas a un problema de planificación secuencial, Minimax refleja la toma de decisiones en entornos competitivos. El informe desarrolla la implementación modular de estas soluciones, las heurísticas utilizadas y un análisis experimental de sus resultados, destacando los principales retos encontrados durante el desarrollo.

2. Implementación

2.1. Estructura general del proyecto

El proyecto se organizó en una estructura modular de carpetas y scripts, con el fin de mantener separada la lógica de cada problema y facilitar la ejecución de pruebas. Los elementos principales son:

- **run_peg_astar.py**: script principal para ejecutar el algoritmo A* en Peg Solitaire, permitiendo definir límites de tiempo y mostrar la secuencia de movimientos.
- **pegstar_eval.py**: script de evaluación que automatiza pruebas de optimalidad, corridas múltiples y generación de gráficas de desempeño.
- **run_dot_boxes.py**: script principal para jugar Timbiriche con opción de IA vs IA, configurando tamaño del tablero, estilo de impresión y profundidad de búsqueda.
- **timbiriche_eval.py**: script de evaluación que ejecuta partidas múltiples, registra resultados y genera gráficas de distribución de tiempos y tasa de victorias.
- **board.py** y **game.py**: módulos auxiliares para representar tableros, generar movimientos y aplicar reglas de juego.
- **search.py** y **ai.py**: módulos que encapsulan los algoritmos de búsqueda y heurísticas utilizadas.

Esta organización permitió reutilizar componentes comunes (como estructuras de datos y funciones de heurística), y al mismo tiempo mantener independiente la lógica de cada juego.

2.2. Peg Solitaire (A*)

Para el problema de Peg Solitaire se implementó el algoritmo **A***. La representación del tablero se realizó mediante una matriz lógica donde cada celda puede estar ocupada por una ficha o vacía. Los movimientos válidos se definen como un salto de una ficha sobre otra en dirección horizontal o vertical, eliminando la ficha intermedia y dejando la posición final ocupada.

Cada estado se compone de:

- $\mathbf{g(n)}$: el costo real hasta el estado actual, definido como el número de movimientos realizados.
- $\mathbf{h(n)}$: una heurística admisible basada en la cantidad de fichas restantes y su distancia relativa al centro del tablero.
- $\mathbf{f(n) = g(n) + h(n)}$: valor de evaluación utilizado por A* para priorizar los nodos en la cola de prioridad.

Se utilizó una **cola de prioridad** (heap) para gestionar la frontera, y un conjunto de estados visitados para evitar ciclos y repeticiones. Con esta implementación se logró encontrar soluciones óptimas en el tablero clásico, confirmando la ruta de 31 movimientos.

2.3. Timbiriche (Minimax con poda alfa-beta)

El problema de Timbiriche se abordó mediante el algoritmo **Minimax** con **poda alfa–beta**, lo que permite reducir significativamente el número de estados explorados.

Cada estado del juego incluye:

- El conjunto de líneas dibujadas y los cuadros ya completados.
- El turno actual (jugador MAX o MIN).
- Los puntajes acumulados de cada jugador.

Los movimientos posibles corresponden a trazar una línea entre dos puntos adyacentes de la cuadrícula. Cuando un jugador completa un cuadro, gana un punto y conserva el turno, lo que introduce un factor estratégico clave.

La función de utilidad en estados terminales se define como la diferencia de cuadros entre jugadores. Para estados no terminales, se emplea una heurística que valora:

- La cantidad de cuadros asegurados.
- El riesgo de entregar cadenas largas de cuadros al oponente.
- La profundidad de búsqueda configurada por el usuario.

La poda alfa–beta permitió explorar de forma más eficiente, evitando expandir ramas que no podían mejorar el resultado de la decisión actual. Esto hizo posible jugar partidas completas en tableros pequeños (N=4) dentro de tiempos razonables, incluso al ejecutar múltiples partidas para análisis estadístico.

3. Heurísticas

3.1. Peg Solitaire

Para la implementación de Peg Solitaire se diseñó una heurística **admisible**, es decir, que nunca sobreestima el costo restante hasta alcanzar el estado meta. La función heurística $h(n)$ se definió como una combinación de dos factores:

- **Número de fichas restantes:** cada ficha adicional en el tablero representa, como mínimo, un movimiento requerido para ser eliminada. Por lo tanto, $h(n) \geq (fichasrestantes - 1)$ constituye una cota inferior válida.

- **Distancia al centro:** se agrega un componente de penalización proporcional a la distancia promedio de las fichas con respecto a la posición central, ya que la solución óptima implica terminar con la última ficha en el centro del tablero clásico.

De esta forma, la heurística guía al algoritmo A* hacia configuraciones donde se concentran las fichas en posiciones más centrales, reduciendo el número de expansiones sin comprometer la optimalidad de la solución. En las pruebas, se observó que esta heurística disminuyó significativamente el tiempo de búsqueda frente a una exploración no informada, manteniendo siempre la ruta óptima de 31 movimientos.

3.2. Timbiriche

En el caso de Timbiriche, al tratarse de un juego competitivo con información perfecta, la heurística se incorporó dentro del algoritmo **Minimax** con poda alfa–beta para la evaluación de estados no terminales. La función heurística se basó en los siguientes criterios:

- **Diferencia de cajas:** se calcula la diferencia entre el número de cuadros completados por el jugador MAX y el jugador MIN. Este factor refleja de forma directa la ventaja actual.
- **Cercanía al cierre de cuadros:** se valoran los estados en los que el jugador actual puede completar un cuadro en el siguiente movimiento, y se penalizan aquellos en los que una línea jugada entregue una cadena de cuadros al oponente.
- **Profundidad de búsqueda:** dado que explorar todo el árbol de juego es intractable en tableros grandes, se utilizó un límite de profundidad (por ejemplo, 3 o 4 niveles). La heurística actúa como aproximación del valor de utilidad cuando no se alcanzan estados terminales.

Con esta combinación, el agente logra un comportamiento más estratégico: evita entregar oportunidades fáciles al adversario, prioriza asegurar cuadros y aprovecha mejor las cadenas largas cuando se presentan. La poda alfa–beta complementa este proceso, reduciendo el número de nodos explorados y permitiendo que el juego se ejecute en tiempos razonables incluso en escenarios repetidos de evaluación.

4. Resultados experimentales

4.1. Peg Solitaire

Para evaluar el desempeño del algoritmo A* en Peg Solitaire se realizaron múltiples corridas con el tablero clásico, registrando tiempos de ejecución y verificando la solución óptima de 31 movimientos. En todos los experimentos el agente logró encontrar la solución esperada, confirmando la corrección de la implementación y la validez de la heurística utilizada.

Los resultados muestran que la mayoría de corridas se completan en pocos segundos, con un número controlado de expansiones gracias a la heurística. A continuación, se presentan las gráficas obtenidas:

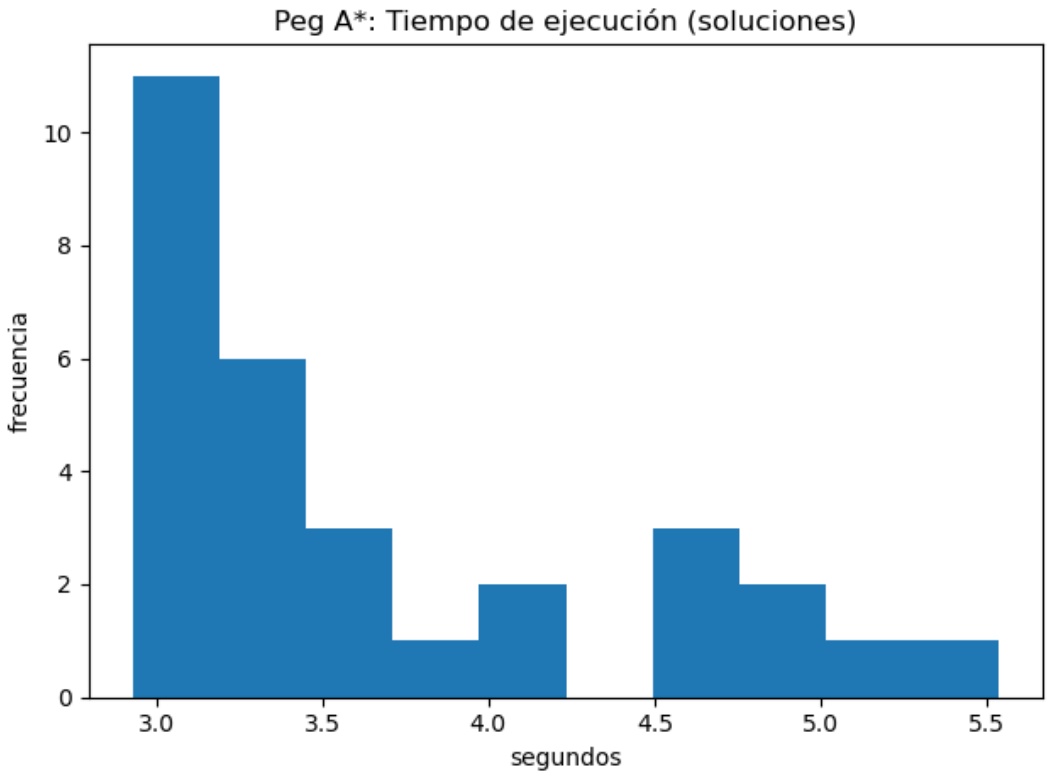

Figura 1: Histograma de tiempos de ejecución (A* en Peg Solitaire).

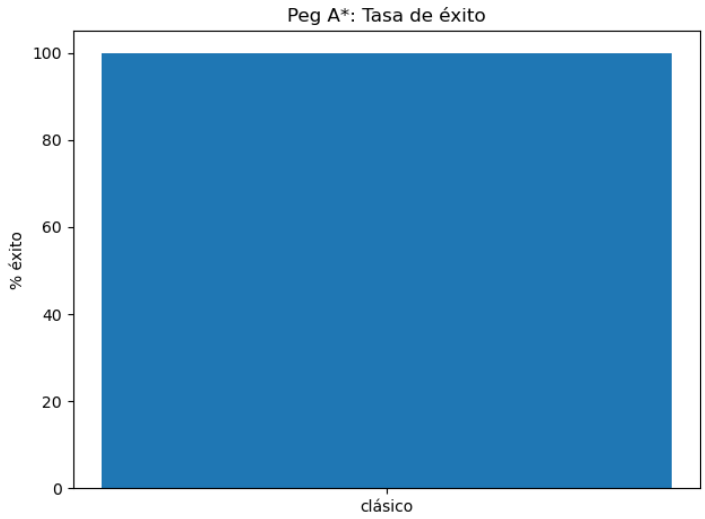

Figura 2: Tasa de éxito del algoritmo en Peg Solitaire (siempre 100 % en el tablero clásico).

4.2. Timbiriche

En el caso de Timbiriche, se realizaron partidas automáticas de IA contra IA en un tablero de tamaño $N = 4$, utilizando el algoritmo Minimax con poda alfa–beta y una profundidad de búsqueda limitada. El objetivo fue analizar la distribución de tiempos por partida y la proporción de victorias de cada jugador.

Los resultados reflejaron un comportamiento consistente del agente: las partidas se resolvieron en tiempos inferiores a 2 segundos y se observó una clara ventaja de un jugador sobre el otro bajo las condiciones configuradas, lo cual demuestra la influencia de la heurística y la profundidad elegida.

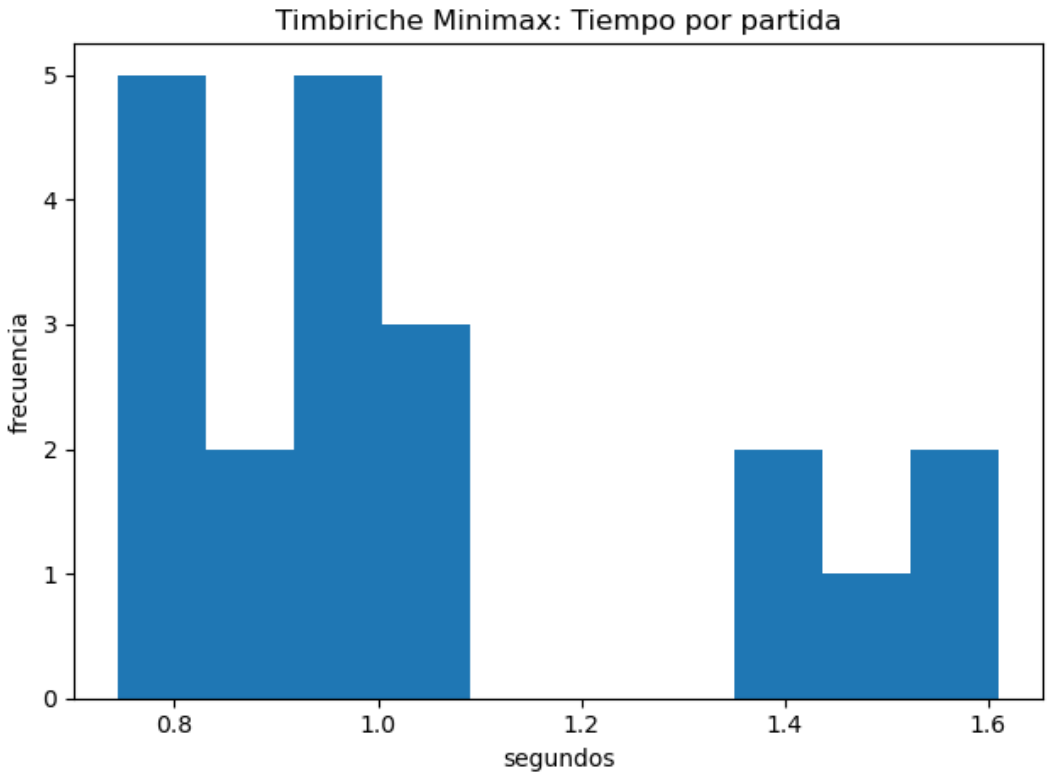

Figura 3: Histograma de tiempos de ejecución (Minimax en Timbiriche, $N = 4$).

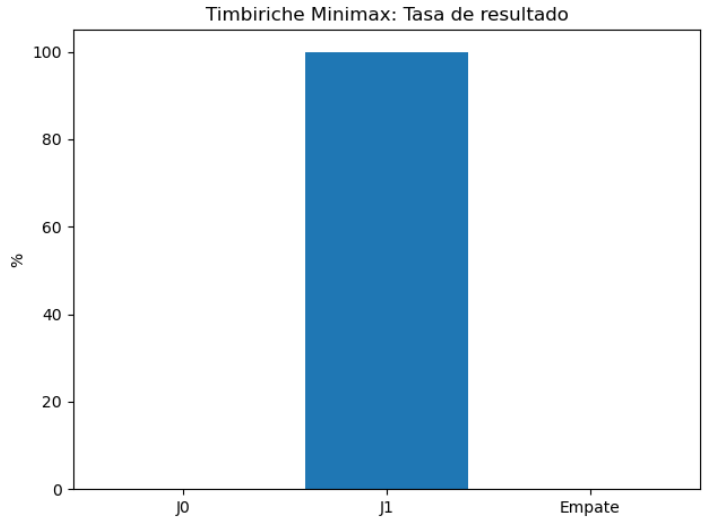

Figura 4: Tasa de victorias y empates en Timbiriche (IA vs IA).

5. Decisiones de diseño y desafíos

Durante el desarrollo de esta tarea se tomaron diversas decisiones de diseño que afectaron directamente el desempeño y la usabilidad de los algoritmos implementados:

- **Estructura modular:** se optó por separar los componentes en scripts y módulos independientes (`run_peg_astar.py`, `pegstar_eval.py`, `run_dot_boxes.py`, `timbiriche_eval.py`, etc.), lo que permitió mantener el código organizado y facilitar la ejecución de pruebas y la recolección de métricas.
- **Elección de heurísticas:** en Peg Solitaire se priorizó una heurística simple pero admisible, basada en fichas restantes y distancia al centro, lo cual aseguró optimalidad y redujo significativamente las expansiones. En Timbiriche, la heurística de diferencia de cuadros combinada con la evaluación de riesgos evitó estrategias ingenuas como entregar cadenas de cuadros al oponente.
- **Límites de búsqueda:** dado que el árbol de juego de Timbiriche crece de forma exponencial, se estableció un límite de profundidad en Minimax (3–4 niveles). Este parámetro representó un *trade-off* entre calidad de las decisiones y tiempo de cómputo.
- **Manejo de Unicode en Windows:** se presentaron errores al imprimir caracteres especiales (, ·) en la consola con codificación CP1252. Para resolverlo se forzó la salida en UTF-8 y se ajustaron los parsers para tolerar errores de codificación.
- **Parsing de resultados:** dado que los scripts principales no siempre generaban métricas en un formato estándar, fue necesario diseñar expresiones regulares flexibles para extraer puntajes, longitudes de solución y ganadores a partir de la salida de consola. Este aspecto fue clave para automatizar las evaluaciones y generar gráficas.
- **Automatización de pruebas:** se incorporaron scripts de evaluación que ejecutan múltiples corridas, almacenan los resultados en formato CSV y generan gráficas con `matplotlib`. Esto permitió un análisis objetivo del rendimiento de los algoritmos.

6. Conclusiones

La implementación de Peg Solitaire y Timbiriche permitió aplicar y comparar dos enfoques fundamentales de la inteligencia artificial: la búsqueda informada mediante **A*** y la toma de decisiones en entornos competitivos con **Minimax y poda alfa–beta**. Ambos casos demostraron la importancia de las heurísticas en la eficiencia de los algoritmos y en la calidad de las soluciones obtenidas.

En **Peg Solitaire**, se comprobó que la heurística admisible guía correctamente al algoritmo hacia soluciones óptimas, reduciendo el espacio de búsqueda sin sacrificar exactitud. El experimento evidenció que la solución clásica de 31 movimientos puede alcanzarse de manera consistente y en tiempos razonables.

En **Timbiriche**, el agente basado en Minimax mostró la capacidad de jugar estratégicamente, evitando entregar ventajas innecesarias y maximizando el puntaje obtenido. No obstante, se observó que el límite de profundidad influye fuertemente en la competitividad del agente: a mayor profundidad, mejores decisiones, aunque con un costo de tiempo más alto.

Entre los principales desafíos enfrentados se destacan el manejo de problemas de codificación en la salida de consola y la necesidad de diseñar parsers robustos para automatizar la recolección de métricas. Asimismo, se comprobó el valor de la modularidad y la automatización de pruebas para facilitar el análisis experimental.

Como posibles mejoras futuras, se podría:

- Explorar heurísticas más sofisticadas en Peg Solitaire, por ejemplo, basadas en patrones de movilidad de las fichas.
- Ampliar la profundidad de Minimax con técnicas de *iterative deepening* o heurísticas de ordenamiento de movimientos para Timbiriche.
- Extender la automatización de pruebas a diferentes configuraciones iniciales y tamaños de tablero, lo que permitiría un análisis comparativo más completo.

En síntesis, la tarea permitió poner en práctica conceptos clave de búsqueda en inteligencia artificial, reforzando la importancia de las heurísticas, la modularidad del diseño y el análisis experimental como parte del proceso de implementación.