

Lightsout automatizado

Jesús Villalba Fernández

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Sevilla, España

jesvilfer@alum.us.es GDT3477

Javier Ruiz Fernández

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Sevilla, España

javruifer1@alum.us.es Javruifer1 forjavi22@gmail.com

Resumen—En este trabajo se ha resuelto el juego Lightsout utilizando técnicas de planificación automática y algoritmos de búsqueda. En primer lugar, se emplea un planificador automático basado en el algoritmo *enhsp* [1]. Este planificador explora de manera ordenada el espacio de estados, pero al ser un planificador para problemas numéricos, decidimos resolverlo con el planificador *Fast Downward* [2], el cual mejoró la eficiencia de la búsqueda al emplear heurísticas avanzadas para optimizar el proceso. Además, se implementó una búsqueda en anchura, la cual fue llevada a cabo de dos maneras: la primera, partiendo de cero, sin utilizar los archivos PDDL generados para resolver el problema mediante *Fast Downward*; y la segunda, parseando los archivos PDDL generados anteriormente, extrayendo los estados iniciales, las relaciones de vecindad entre las celdas y el número total de celdas.

Los resultados nos muestra que los cuatro métodos evaluados son capaces de resolver tableros de tamaño 3x3, 4x4 y 5x5. Sin embargo, se observan diferencias claras en eficiencia. *Fast Downward* destaca por ofrecer planes óptimos, aunque en tableros grandes puede aumentar el tiempo de resolución. *Enhsp* encuentra soluciones válidas con tiempos variables debido a su exploración aleatoria. Por otro lado, la búsqueda en anchura, tanto con como sin parseo, presenta tiempos de ejecución más bajos en general, especialmente en tableros pequeños y medianos, aunque puede generar soluciones menos óptimas si el parámetro *n* no se ajusta correctamente. En conjunto, todos los métodos son eficaces, pero el equilibrio entre rapidez y calidad de la solución depende del enfoque utilizado.

Palabras clave—*Enhsp*, espacio de estados, Lightsout, *Fast Downward*, PDDL, búsqueda en anchura, espacio de estados.

I. INTRODUCCIÓN

La planificación automática es un área de la inteligencia artificial que se ocupa de encontrar una secuencia de acciones que transformen un estado inicial en un estado objetivo, sin intervención humana. Esta disciplina se ha aplicado con éxito en la resolución de juegos lógicos. Uno de los entornos en dónde vamos a estudiarlo es en el juego Lightsout, un rompecabezas que, a pesar de parecer simple, presenta bastante complejidad a nivel computacional.

En este trabajo nos centramos en la resolución del juego Lightsout utilizando distintos enfoques. Concretamente, se han explorado dos enfoques principales: por un lado, la modelización del problema mediante la librería *Unified Planning* [3] y su resolución a través de los planificadores *enhsp* y *Fast*

Downward, y por otro lado, se ha implementado una búsqueda heurística en anchura que, con ayuda de un parámetro *n*, prioriza los estados más prometedores, y que ha sido aplicada desde cero a partir del parseo de archivos PDDL.

El objetivo de este proyecto es analizar y comparar el rendimiento de estos métodos, tanto en términos de eficiencia (tiempo de ejecución) como de calidad de las soluciones (número de pasos), aplicados a tableros de tamaño 3x3, 4x4 y 5x5. Para ello, se han ejecutado los distintos algoritmos, registrando y analizando los resultados obtenidos.

Este documento se estructura de la siguiente manera: en la Sección 2 se revisan los técnicas empleadas y los trabajos relacionados. La Sección 3 describe la metodología seguida para modelar el problema y ejecutar los algoritmos. La Sección 4 presenta los resultados obtenidos y su análisis. En la Sección 5 se exponen las conclusiones del trabajo, así como posibles mejoras futuras. En la sección 6 explicamos la influencia de la IA en este trabajo, y finalmente, se incluyen las referencias bibliográficas.

II. PRELIMINARES

Para resolver el problema planteado, la principal tarea ha sido encontrar una secuencia de movimientos que nos permita pasar de un tablero con todas las luces apagadas a uno en el que todas estén encendidas. Para lograrlo, hemos representado utilizando flujos y acciones, lo que nos ha permitido describir el estado de cada celda y cómo las acciones afectan tanto a la celda pulsada como a sus vecinos. Para la resolución del problema, utilizamos dos enfoques distintos: la planificación automática y la búsqueda en anchura.

En cuanto a la resolución, inicialmente empleamos un planificador automático basado en el algoritmo *enhsp*, el cual explora exhaustivamente el espacio de estados para encontrar una secuencia de acciones que nos permita alcanzar el objetivo desde el estado inicial. Sin embargo, al ser un planificador orientado a problemas numéricos, decidimos sustituirlo por *Fast Downward*, que proporcionó un rendimiento más eficiente en la búsqueda. Además, implementamos la búsqueda en anchura de dos formas. La primera consistió en una búsqueda desde cero, mientras que la segunda utilizó el parseo de los archivos PDDL del problema y dominio, extrayendo la

información necesaria para representar el estado, los vecinos y las relaciones de vecindad.

A. Métodos empleados

En este trabajo, abordamos la resolución del *Lightsout* utilizando las técnicas de planificación automática y búsqueda mencionadas anteriormente. A continuación, se describen detalladamente estos métodos.

- 1) **Planificación Automática con enhsp:** El primer enfoque que empleamos fue la planificación automática utilizando el planificador **enhsp**. Este planificador explora el espacio de estados paso a paso, generando secuencias de acciones para transformar el estado inicial, donde todas las luces están apagadas, en el estado final donde todas las luces están encendidas. El algoritmo de enhsp realiza una búsqueda exhaustiva, asegurando que no se pase por alto ninguna solución. Sin embargo, aunque este método asegura que no se nos escape ninguna solución, nos dimos cuenta de que puede tardar bastante cuando el tablero es más grande, porque el número de posibles configuraciones crece mucho.
- 2) **Planificación Automática con Fast Downward:** Como mencionamos previamente, al ser enhsp un planificador para problemas numéricos, decidimos utilizar el planificador **Fast Downward**. A diferencia de enhsp, Fast Downward emplea un enfoque de búsqueda más optimizado, utilizando heurísticas avanzadas para reducir el número de configuraciones que se exploran. El planificador no solo realiza la búsqueda del espacio de estados, sino que también optimiza la selección de los estados más prometedores, priorizando aquellos que tienen más probabilidades de llegar al objetivo. Esta característica de Fast Downward permite que la búsqueda se realice de manera más rápida y óptima, lo que lo convierte en una opción ideal cuando se trata de tableros más grandes.
- 3) **Búsqueda en Anchura (sin parseo):** En este enfoque, implementamos una búsqueda en anchura básica que explora con precisión todo el espacio de estados para encontrar la solución óptima en el menor número de pasos. En este caso, se generan todos los sucesores posibles de un estado y se expanden sucesivamente hasta alcanzar el estado objetivo. Aunque este método garantiza encontrar la solución más corta, puede volverse bastante ineficiente en cuanto a tiempo de ejecución, especialmente al trabajar con tableros de mayor tamaño, debido a la gran cantidad de configuraciones posibles que se deben explorar.
- 4) **Búsqueda en Anchura (con parseo de archivos PDDL):** Finalmente, implementamos una versión mejorada de la búsqueda en anchura que utiliza los archivos PDDL generados por el planificador. Estos archivos contienen la descripción del dominio y del problema,

detallando las celdas del tablero y sus relaciones de vecindad. Al parsear estos archivos, pudimos extraer los estados iniciales, los vecinos de cada celda y el número total de celdas en el tablero. Esto nos permitió optimizar la búsqueda, centrándonos solo en los estados relevantes. Con este enfoque, la búsqueda en anchura se vuelve más eficiente, reduciendo el número de estados que se deben explorar, pero sin perder la precisión en la solución. Este método es especialmente útil en tableros 3x3, 4x4 y 5x5, aunque para tableros más grandes tiene el mismo problema que los demás.

En resumen, usamos varios métodos para resolver el juego *Lightsout*. Primero, implementamos planificación automática con enhsp y Fast Downward, que nos permitió encontrar soluciones correctas a través de la exploración del espacio de estados. A continuación, implementamos la búsqueda en anchura, tanto desde cero como optimizada con el parseo de archivos PDDL. Al analizar estos archivos, pudimos extraer los estados iniciales, las relaciones de vecindad y el número de celdas del tablero, lo que hizo que la búsqueda fuera más eficiente.

B. Trabajo Relacionado

En la inteligencia artificial, la aplicación de técnicas de planificación automática y búsqueda heurística en puzzles y juegos de lógica ha sido ampliamente estudiada. Algunos trabajos destacados son:

- **Sistema FF de Hoffmann y Nebel:** Utiliza una búsqueda heurística basada en la relajación del problema para generar planes de manera rápida y eficiente. Este sistema ha servido de base para muchos otros planificadores en dominios complejos [4].
- **Planificador Fast Downward de Helmert:** Es un planificador ampliamente utilizado que trabaja con problemas definidos en el lenguaje PDDL, aplicándose con éxito en juegos y escenarios con grandes espacios de estados [5].
- **Librería Unified Planning (Palacios y Segovia-Aguas):** Proporciona una interfaz flexible para definir y resolver problemas de planificación, facilitando la integración con diferentes planificadores y técnicas [6].
- **Análisis algebraico de LightsOut (Sutner, 1992):** Estudio que aborda la estructura matemática del juego usando álgebra lineal sobre campos finitos para caracterizar soluciones posibles de forma eficiente [7].
- **Técnicas de aprendizaje automático para mejorar heurísticas (Silver et al. - AlphaGo):** Aunque enfocado en otros juegos de tablero, este trabajo establece bases importantes para el uso de redes neuronales en la mejora de planificadores y algoritmos heurísticos [8].

A pesar de los avances, estos métodos presentan limitaciones para tableros grandes o variantes complejas de Lightsout. Por ello, la investigación sigue enfocándose en desarrollar heurísticas más avanzadas y optimizar algoritmos para mejorar la escalabilidad.

Nuestro trabajo se sitúa en esta línea, aportando una implementación práctica que combina planificación automática, búsqueda heurística y búsqueda en anchura para Lightsout, con el objetivo de evaluar su rendimiento y poder llevar a cabo futuras mejoras.

III. METODOLOGÍA

En esta sección se describe detalladamente la metodología empleada para resolver el juego, donde a lo largo de su desarrollo se utilizaron los métodos mencionados anteriormente: *planificación automática con enhsp*, *planificación automática con Fast Downward*, *búsqueda en anchura sin parseo* y *búsqueda en anchura con parseo de archivos PDDL*.

A continuación, se explica cada metodología con detalle, incluyendo su modelado, funcionamiento y proceso de resolución, ofreciendo una visión clara de las técnicas aplicadas y las características principales de cada uno.

A. Planificación Automática con enhsp

Este enfoque se basa en la planificación automática usando el planificador *enhsp*. El objetivo es encontrar una secuencia de acciones que transforme el estado inicial, con todas las luces apagadas, en uno donde estén todas encendidas.

1) **Modelado del problema:** El problema lo hemos modelado mediante la librería *Unified Planning* en Python:

- **Objetos:** cada celda del tablero se representa como un objeto del tipo *Celda*, identificado por su coordenada $[i, j]$.
- **Fluents:** se definen dos fluents: *estado(c)*, que indica si una celda está encendida, y *vecino(c1, c2)*, que define las relaciones de vecindad.
- **Acciones:** una acción *pulsar(c)* que invierte el estado de la celda seleccionada y el de sus vecinos.
- **Estado inicial y objetivo:** todas las celdas están inicialmente apagadas; el objetivo es que todas estén encendidas.

2) **Resolución con enhsp:** Al utilizar *enhsp* como planificador, exploramos de forma sistemática el espacio de estados generado a partir del modelado. Aunque este enfoque garantiza encontrar una solución si existe, presenta limitaciones en eficiencia al aumentar el tamaño del tablero, debido a la explosión combinatoria. A continuación, vemos el pseudocódigo.

resolver_con_enhsp(tamaño)

Entrada: tamaño del tablero $tamaño \times tamaño$

Salida: solución del problema mediante el planificador *enhsp*

```

1 problema ← crear_problema_lightsout(tamaño)
2 seleccionar planificador: OneshotPlanner (name="enhsp")
3 planificador ← ONESHOTPLANNER()
4 resultado ← PLANIFICADOR.SOLVE(problema)
5 si resultado.plan existe entonces
6     mostrar plan paso a paso
7 sino mostrar: "No se encontró solución"
crear_problema_lightsout(tamaño)

```

Entrada: tamaño del tablero $tamaño \times tamaño$

Salida: problema de planificación codificado en Unified Planning

```

1 definir tipo Celda
2 definir fluents: estado(c) y vecino(c1, c2)
3 definir acción pulsar(c)
4 para cada celda [i, j] hacer
5     crear objeto Celda con nombre [i, j]
6     establecer vecinos arriba, abajo, izquierda, derecha
7     si c2 es vecino de c1 entonces
8         vecino(c1, c2) = True
9     añadir efectos en pulsar para celdas vecinas
10    establecer estado inicial: todas apagadas
11    establecer objetivo: todas encendidas
12    devolver problema

```

Fig. 1. Resolución utilizando el planificador Enhsp

B. Planificación Automática con Fast Downward

Este segundo enfoque también utiliza *Unified Planning*, pero en lugar de usar *enhsp*, se exportan los archivos PDDL del dominio y del problema para ser resueltos externamente con el planificador *Fast Downward* en la terminal mediante el siguiente comando:

```
python fast-downward.py Dominio.pddl
Problema.pddl --search "astar(blind())"
```

1) **Modelado del problema:** El modelado es idéntico al explicado en el apartado anterior, con objetos tipo *Celda*, fluents, acciones y definición de estado inicial y objetivo.

2) **Resolución con Fast Downward:** Mediante la clase *PDDLWriter*, se generaron los archivos *Dominio.pddl* y *Problema.pddl*. Estos archivos se introdujeron manualmente en *Fast Downward* a través de la terminal. Este planificador utiliza técnicas más avanzadas y suele encontrar planes más rápido y de manera más óptima que *enhsp*, sobre todo para tableros medianos como 4x4 o 5x5.

A continuación, se presenta el pseudocódigo que describe el proceso de modelado y exportación del problema *Lightsout* utilizando la librería *Unified Planning*. Este modelado genera los archivos PDDL necesarios que posteriormente son resueltos con el planificador *Fast Downward* a través de la terminal.

```

resolver_lightsout(tamaño)
Entrada: tamaño del tablero  $\text{tamaño} \times \text{tamaño}$ 
Salida: archivos PDDL generados para Fast Downward
1  $\text{problema} \leftarrow \text{crear\_problema\_lightsout}(\text{tamaño})$ 
2 escribir Dominio.pddl y Problema.pddl con PDDLWriter
3 ejecutar desde terminal

crear_problema_lightsout(tamaño)
Entrada: tamaño del tablero  $\text{tamaño} \times \text{tamaño}$ 
Salida: problema de planificación en Unified Planning
1 definir tipo Celda
2 definir fluents: estado(c) y vecino(c1, c2)
3 definir acción pulsar(c)
4 para cada celda  $[i, j]$  hacer
5     crear objeto Celda con nombre  $[i, j]$ 
6     establecer vecinos arriba, abajo, izquierda y derecha
7     si  $c2$  es vecino de  $c1$  entonces
8         vecino(c1, c2) = True
9     añadir efectos en pulsar para celdas vecinas
10    establecer estado inicial: todas apagadas
11    establecer objetivo: todas encendidas
12    devolver problema

```

Fig. 2. Modelado y exportación del problema para *Fast Downward*

C. Búsqueda en Anchura desde Cero

Este enfoque consiste en una implementación del algoritmo de búsqueda en anchura sin apoyarse en los archivos PDDL.

1) Modelado del problema:

- *Objetos:* cada celda está representada por su posición en el tablero $[i, j]$.
- *Vecindad:* se construye un diccionario donde cada celda está asociada a sus vecinos inmediatos (arriba, abajo, izquierda, derecha).
- *Estado inicial y objetivo:* el estado inicial tiene todas las celdas apagadas, y el objetivo es encenderlas todas.

2) *Resolución mediante búsqueda:* Se utiliza una función cola para recorrer los estados. En cada iteración, se exploran los sucesores generados al pulsar cada celda, y se priorizan los estados con mayor número de luces encendidas. Para limitar el crecimiento del espacio de búsqueda, se seleccionan únicamente los n sucesores más prometedores en cada paso. Se evita repetir estados ya visitados usando *frozenset*, una versión inmutable de un conjunto. Este método es claro, aunque poco eficiente para tableros grandes si n no se ajusta adecuadamente.

A continuación, se presenta el pseudocódigo del algoritmo de búsqueda en anchura implementado desde cero.

```

busqueda_anchura(tamaño, n)
Entrada: tamaño del tablero  $\text{tamaño} \times \text{tamaño}$ , nº de sucesores  $n$ 
Salida: plan de acciones para encender todas las luces
1  $\text{celdas} \leftarrow$  lista de todas las coordenadas
2  $\text{vecinos} \leftarrow \text{crear\_vecinos}(\text{tamaño})$ 
3  $\text{estado\_inicial} \leftarrow$  todas las celdas apagadas
4  $\text{cola} \leftarrow$  cola con  $(\text{estado\_inicial}, [])$ 
5  $\text{visitados} \leftarrow \emptyset$ 
6 mientras  $\text{cola}$  no esté vacía hacer
7      $\text{estado}, \text{plan} \leftarrow$  extraer de  $\text{cola}$ 
8     si  $\text{estado} \in \text{visitados}$  entonces continuar
9     añadir  $\text{estado}$  a  $\text{visitados}$ 
10    si  $\text{es\_objetivo}(\text{estado})$  entonces devolver  $\text{plan}$ 
11     $\text{sucesores} \leftarrow []$ 
12    para cada celda  $\in \text{celdas}$  hacer
13         $\text{nuevo\_estado} \leftarrow \text{aplicar\_accion}(\text{estado}, \text{celda}, \text{vecinos})$ 
14         $\text{nuevo\_plan} \leftarrow \text{plan} + [\text{pulsar celda}]$ 
15        añadir  $(\text{nuevo\_estado}, \text{nuevo\_plan})$  a  $\text{sucesores}$ 
16    para cada  $(s, p) \in \text{sucesores}$  hacer
17        si  $\text{es\_objetivo}(s)$  entonces devolver  $p$ 
18    ordenar  $\text{sucesores}$  según  $\text{similitud}(\text{estado})$ , descendente
19    añadir los  $n$  mejores a  $\text{cola}$ 
20 devolver None

```

Fig. 3. Algoritmo de búsqueda en anchura para Lightsout sin parseo

D. Búsqueda en Anchura con Parseo de Archivos PDDL

Finalmente, se desarrolló una variante de la búsqueda en anchura que aprovecha la información contenida en los archivos PDDL generados anteriormente para Fast Downward.

1) *Extracción de información del problema:* Parseamos los archivos `Dominio.pddl` y `Problema.pddl` utilizando la clase `PDDLReader`, lo que permite extraer:

- El estado inicial de cada celda.
- Las relaciones de vecindad definidas en el fluent `vecino`.

2) *Resolución del problema:* Con esta información, se reconstruye el grafo de estados y se aplica una búsqueda en anchura similar a la anterior, pero partiendo de datos ya definidos y más estructurados. Esto permite mejorar la eficiencia y reduce errores al automatizar la lectura del problema.

A continuación, presentamos el pseudocódigo que describe este método:

busqueda_heuristica_pddl(problema, n)

Entrada: problema PDDL parseado, número máximo de sucesores n

Salida: plan que enciende todas las luces o *None*

```

1  vecinos ← get_vecinos_from_problem(problema)
2  estado_inicial ← estado_inicial_from_problem(problema)
3  celdas ← lista de claves de vecinos
4  crear cola con (estado_inicial, [])
5  visitados ← {}
6  mientras la cola no esté vacía hacer
7      (estado, plan) ← sacar siguiente de la cola
8      estado_hash ← frozenset de estado
9      si estado_hash ∈ visitados entonces continuar
10     añadir estado_hash a visitados
11     si es_objetivo(estado) entonces devolver plan
12     sucesores ← []
13     para cada celda c ∈ celdas hacer
14         nuevo_estado ← aplicar_accion_estado(estado, c, vecinos)
15         nuevo_plan ← plan + ["pulsar " + c]
16         añadir (nuevo_estado, nuevo_plan) a sucesores
17     para cada (est, p) ∈ sucesores hacer
18         si es_objetivo(est) entonces devolver p
19     ordenar sucesores por similitud(estado) descendente
20     añadir los n mejores sucesores a la cola
21 devolver None

```

Fig. 4. Búsqueda en anchura con parseo de archivos PDDL

IV. RESULTADOS

En esta sección se describen los experimentos que hemos llevado a cabo, los resultados que hemos obtenido y el análisis que hemos hecho de ellos, con el objetivo de evaluar cómo funcionan los métodos que hemos implementado para resolver nuestro juego.

A. Estados iniciales y finales de los tableros

Para ilustrar mejor el problema, a continuación, se muestran imágenes que representan los estados iniciales y finales de los tableros de tamaño 3x3, 4x4 y 5x5. [9]

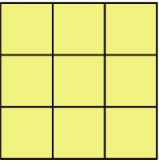
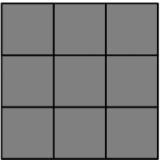
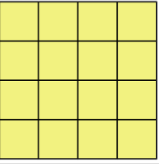
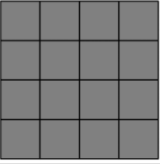
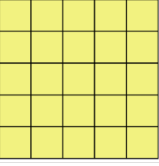
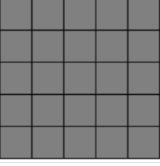
Tamaño	Estado inicial	Estado final (objetivo)
3x3		
4x4		
5x5		

TABLA I

ESTADOS INICIALES Y FINALES DE LOS TABLEROS

B. Descripción de los métodos

Durante la evaluación de cada método que hemos ido explicando a lo largo del documento, registramos:

- Tiempo de ejecución
- Número de pasos hasta la solución (estado objetivo)

A continuación, vamos a los resultados de cada uno de los cuatro enfoques.

1) **Planificación automática con *enhsp***: El método de planificación automática basado en el planificador *enhsp* se evaluó aplicándolo a tableros de tamaño 3x3, 4x4 y 5x5. Este planificador realiza una exploración exhaustiva del espacio de estados para encontrar la secuencia de acciones que lleva del estado inicial al objetivo. Hay que tener en cuenta que con este método las soluciones encontradas son aleatorias. Por lo tanto, cada vez que se reestablezca el kernel y se ejecute, los pasos hasta llegar a la solución pueden variar.

Tamaño	Tiempo (s)	Pasos
3x3	2.0	21
4x4	0.5	8
5x5	8.6	51

TABLA II

RESULTADOS CON *enhsp*

Llevamos a cabo pruebas con tableros de 3x3, 4x4 y 5x5 usando el planificador *enhsp*. Aunque el 4x4 dio mejores resultados que el 3x3 en tiempo y pasos, esto se debe a que la dificultad no siempre depende del tamaño. En general, *enhsp* encuentra soluciones válidas, pero su rendimiento varía según el tablero y no siempre ofrece la solución más corta.

2) **Planificación automática con *Fast Downward***: El método de planificación automática basado en el planificador *Fast Downward* se evaluó también aplicándolo a tableros de tamaño 3x3, 4x4 y 5x5, utilizando los archivos PDDL generados previamente. Este planificador destaca gracias a su optimización, ya que te da las mejores soluciones. Sin embargo, como vemos a continuación, su rendimiento no siempre supera al de *enhsp*, especialmente en tableros grandes, donde el tiempo de ejecución se incrementa significativamente.

Tamaño	Tiempo (s)	Pasos
3x3	0.10	5
4x4	0.30	4
5x5	31.28	15

TABLA III

RESULTADOS CON *Fast Downward*

Los resultados muestran que *Fast Downward* ofrece soluciones más cortas en términos de pasos que *enhsp*. No obstante, en el caso del tablero 5x5, el tiempo de ejecución fue más alto, con más de 30 segundos, ya que realiza una búsqueda más profunda y con más precisión. Por lo tanto, aunque *Fast*

Downward puede generar planes más óptimos, no siempre es la opción más rápida cuando se incrementa la dificultad del problema.

3) **Búsqueda en anchura sin parseo:** Este método aplica búsqueda en anchura a tableros 3x3, 4x4 y 5x5, utilizando una heurística que prioriza los estados con más luces encendidas. El parámetro n controla cuántos sucesores se exploran en cada paso: unos valores bajos reducen el tiempo pero pueden dar soluciones peores, mientras que valores altos mejoran la calidad del plan, aunque incrementan el tiempo de búsqueda.

Tamaño	n	Tiempo (s)	Pasos
3x3	3	0.0	9
3x3	5	0.0	5
4x4	3	0.0	6
4x4	5	0.0	4
5x5	3	0.5	17
5x5	5	7.3	15

TABLA IV
RESULTADOS DE BÚSQUEDA HEURÍSTICA SIN PARSEO

Los resultados muestran que el parámetro n afecta al equilibrio entre rapidez y calidad de la solución. Con valores bajos, el algoritmo es más rápido pero encuentra planes más largos; con valores altos, mejora la calidad del plan a costa de mayor tiempo, sobre todo en tableros grandes. Ajustar n según el tamaño del tablero y los recursos disponibles es clave para un buen rendimiento.

4) **Búsqueda en anchura con parseo de archivos PDDL:** Este método también aplica búsqueda en anchura sobre tableros 3x3, 4x4 y 5x5, pero en lugar de crear manualmente los datos, se utiliza el parseo de archivos PDDL generados previamente. Se extrae el estado inicial y la relación de vecindad. Al igual que en el método anterior, se priorizan los estados con más luces encendidas, limitando los sucesores con el parámetro n .

Tamaño	n	Tiempo (s)	Pasos
3x3	3	0.4	9
3x3	5	0.3	5
4x4	3	1.0	6
4x4	5	0.9	4
5x5	3	3.0	17
5x5	5	10.4	15

TABLA V
RESULTADOS DE BÚSQUEDA HEURÍSTICA CON PARSEO

Los resultados obtenidos con la búsqueda heurística usando parseo muestran un comportamiento similar al del enfoque sin parseo: al aumentar el parámetro n , mejora la calidad del plan (menos pasos), pero a costa de un mayor tiempo de ejecución. Esta diferencia es mayor en tableros más grandes, como el de 5x5, donde el tiempo puede multiplicarse significativamente.

C. Análisis comparativo de los 4 métodos

Tras aplicar los cuatro métodos podemos extraer varias conclusiones relevantes sobre su rendimiento y utilidad.

En primer lugar, los métodos de planificación automática (*enhsp* y *Fast Downward*) destacan por ofrecer una solución garantizada, es decir, siempre encuentran un plan válido cuando existe. Sin embargo, presentan algunas diferencias claras: *enhsp* suele ser más rápido para tableros pequeños, pero sus soluciones no siempre son óptimas y pueden variar entre ejecuciones. En cambio, *Fast Downward* tiende a encontrar planes más cortos, pero con un coste en tiempo mucho mayor, especialmente en tableros grandes como el 5x5, donde el tiempo de resolución se dispara.

Por otro lado, las búsquedas heurísticas (con y sin parseo) resultan mucho más eficientes en tiempo para tableros pequeños y medianos. Además, permiten cierto control sobre la exploración gracias al parámetro n , que selecciona cuántos sucesores se quiere explorar. La versión sin parseo es algo más rápida, ya que trabaja directamente con estructuras internas. La versión con parseo, aunque más lenta, añade la ventaja de reutilizar descripciones PDDL externas, lo que facilita su integración con planificadores y sistemas más complejos.

La elección del método depende, por tanto, del tamaño del problema, el tiempo disponible y la necesidad de reutilización o integración con otros sistemas.

V. CONCLUSIONES

En este trabajo se ha abordado la resolución del juego *Light-sout* utilizando distintas estrategias basadas en planificación automática y búsqueda heurística. Se implementaron y compararon cuatro métodos: planificación automática mediante los planificadores *enhsp* y *Fast Downward*, y búsqueda en anchura heurística tanto con como sin parseo de archivos PDDL. A lo largo del documento, se ha detallado el modelado del problema, la implementación de cada enfoque, y se han analizado los resultados obtenidos al aplicar cada uno a tableros 3x3, 4x4 y 5x5.

Los resultados muestran que los métodos de planificación automática son útiles cuando se necesita una solución garantizada, aunque su rendimiento puede variar en función del tamaño del tablero y del planificador utilizado. *Fast Downward* ofrece soluciones más óptimas, pero requiere más tiempo, especialmente con tableros grandes. Por otro lado, las búsquedas heurísticas permiten ajustar el equilibrio entre tiempo y calidad de solución mediante el parámetro n , siendo más rápidas en tableros pequeños y medianos.

En conclusión, no existe un único método ideal para todos los casos: los métodos de planificación son útiles para garantizar soluciones, mientras que las búsquedas heurísticas son más adecuadas cuando se requiere eficiencia. La elección dependerá del tamaño del problema y del contexto de uso.

Como posibles mejoras para el futuro, podríamos intentar optimizar el código para que tarde menos tiempo, sobre todo en tableros grandes, o probar con otros métodos de búsqueda

que combinen planificación y heurística. Otra idea sería ajustar el parámetro n de forma automática según el tamaño del tablero, para no tener que elegirlo manualmente.

VI. USO DE INTELIGENCIA ARTIFICIAL GENERATIVA

Durante la elaboración de este trabajo se ha utilizado ChatGPT como herramienta de apoyo para mejorar la redacción del documento, así como para ayudarnos en algunos aspectos del código.

Utilidad de la herramienta

La IA se ha utilizado para:

- Redactar apartados del informe de forma clara y coherente, basándose en ideas y fragmentos escritos por nosotros.
- Corregir textos extensos para hacerlos más comprensibles.
- Generar pseudocódigo a partir de nuestro código, manteniendo el formato que se pedía para este informe.
- Dar formato en latex a las tablas.
- Resolver dudas sobre cómo hacer algunas partes del código y resolvernos ciertos errores de programación.

Entradas proporcionadas al sistema

Las entradas (prompts) fueron las siguientes:

- Conviértete este código en pseudocódigo siguiendo el formato especificado.
- Créame esta tabla hecha en word en formato latex.
- Párrafos escritos por nosotros que queríamos mejorar en redacción o resumir.
- Dime alguna librería en python para parsear archivos pddl.
- Como puedo usar PDDLReader de unified planning?
- Sabes porque enhsp me da soluciones muy largas?
- Sabes de trabajos relacionados con nuestro proyecto?
- Ayúdame a hacer una función cola en python para llevar a cabo una búsqueda en anchura y quedarme con la mejor solución en función de un parámetro n que lo que hace es limitar los estados que queremos explorar en cada paso. Este código es lo que llevo hecho pero no sé como continuar.
- Puedes decirme los pasos de instalación del fast downward y como puedo ejecutar estos archivos pddl. Dime los comandos por favor.
- Teniendo en cuenta que estas son las referencias que quiero poner, pónmelas en el formato que aparece.

Ambos miembros del grupo entendemos perfectamente el contenido generado por esta IA y hemos revisado cada respuesta para asegurarnos de que es correcta, adaptándola a nuestras necesidades para este trabajo.

REFERENCIAS

- [1] ENHSP, “ENHSP - Expressive Numeric Heuristic Search Planner,” [Online]. Available: <https://sites.google.com/view/enhsp/>. Accedido el 8 de junio de 2025.
Página oficial del planificador ENHSP, utilizada para consultar documentación, descargar el software y entender su funcionamiento.
- [2] Fast Downward, “Fast Downward Planning System,” [Online]. Available: <https://www.fast-downward.org/latest/>. Accedido el 8 de junio de 2025.
Sitio oficial del planificador Fast Downward, empleado para obtener información técnica y acceder al sistema de planificación utilizado en nuestros experimentos.
- [3] Unified Planning GitHub, “Basic Example Notebook,” [Online]. Available: <https://github.com/aiplan4eu/unified-planning/blob/master/docs/notebooks/01-basic-example.ipynb>. Accedido el 8 de junio de 2025.
Ejemplo práctico para entender cómo implementar problemas en Unified Planning, usado como referencia para estructurar nuestro código.
- [4] J. Hoffmann y B. Nebel, “The FF Planning System: Fast Plan Generation Through Heuristic Search,” *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.
- [5] M. Helmert, “The Fast Downward Planning System,” *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.
- [6] A. Palacios y J. Segovia-Aguas, “Unified Planning: An API for Classical Planning,” *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2019.
- [7] K. Sutner, “The Lights Out Puzzle,” *Mathematical Intelligencer*, vol. 14, no. 3, pp. 28–32, 1992.
- [8] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, pp. 484–489, 2016.
- [9] GeoGebra, “Lights Out Solver,” [Online]. Available: <https://www.geogebra.org/m/WG8PPmygmateral/CtzDmRDf>. Accedido el 8 de junio de 2025.
Herramienta visual empleada para analizar soluciones óptimas del juego Lights Out y verificar resultados manualmente.