Resolución Autónoma del Puzzle 8 usando Búsqueda Informada y No Informada

Autonomous Resolution of the 8-Puzzle Using Informed and Uninformed Search

Autor: Martínez, María.

Docente: Paniccia, Manuel.

Pregado de Ingienería en Informática

Universidad Nacional Experimental de Guayana

mariamartinezdev22@gmail.com

Resumen— Este trabajo presenta el desarrollo de un sistema automatizado para la resolución del Puzzle 8, empleando los algoritmos de búsqueda no informada Breadth-First Search (BFS) y búsqueda informada A* con heurísticas. Se diseñó una arquitectura de agentes que permite ejecutar y visualizar el comportamiento de cada algoritmo en tiempo real mediante una interfaz desarrollada con Pygame. El objetivo principal es analizar el rendimiento de ambos enfoques en términos de eficiencia, cantidad de nodos explorados, tiempo de ejecución y longitud de la solución. Los resultados experimentales muestran que A*, al incorporar heurísticas como la distancia de Manhattan, supera significativamente a BFS en eficiencia computacional, aunque a costa de una mayor complejidad de implementación. Este estudio permite comprender mejor las ventajas y limitaciones de cada estrategia de búsqueda aplicada a problemas de planificación y resolución de rompecabezas.

Palabras claves: Puzzle 8, BFS, A*, Búsqueda informada, Búsqueda no informada, Heurísticas, Inteligencia Artificial, Resolución de problemas, Agentes, Comparación de algoritmos.

Abstract— This work presents the development of an automated system for solving the 8-Puzzle using the uninformed search algorithm Breadth-First Search (BFS) and the informed search algorithm A* with heuristics. An agent-based architecture was designed to execute and visualize each algorithm's behavior in real time using a Pygame-based interface. The main goal is to analyze the performance of both approaches in terms of efficiency, number of explored nodes, execution time, and solution length. Experimental results show that A*, by incorporating heuristics such as the Manhattan distance, significantly outperforms BFS in computational efficiency, albeit at the cost of greater implementation complexity. This study provides a clearer understanding of the

strengths and limitations of each search strategy when applied to planning and puzzle-solving problems.

Keywords: 8-Puzzle, BFS, A*, Informed search, Uninformed search, Heuristics, Artificial Intelligence, Problem solving, Agents, Algorithm comparison.

I. INTRODUCCIÓN

La resolución de problemas es una de las áreas fundamentales de la inteligencia artificial (IA), donde los algoritmos de búsqueda desempeñan un papel crucial. El Puzzle 8, un juego clásico de deslizamiento de fichas, se ha convertido en un entorno ideal para estudiar y comparar diferentes estrategias de búsqueda debido a su complejidad combinatoria y su representación clara de estados y transiciones. Este trabajo tiene como objetivo desarrollar un sistema automatizado capaz de resolver el Puzzle 8 utilizando dos enfoques distintos: Búsqueda en Amplitud (Breadth-First Search, BFS) como ejemplo de algoritmo no informado, y A* como ejemplo de algoritmo informado, que emplea heurísticas para guiar la búsqueda hacia la solución de manera más eficiente. Para ello, se implementó una arquitectura basada en agentes, donde cada agente representa un enfoque de resolución autónomo. El sistema fue desarrollado en Python, integrando una interfaz gráfica con Pygame que permite visualizar en tiempo real los pasos que realiza cada agente en la búsqueda de la solución. Este trabajo no solo describe el diseño e implementación del sistema, sino que también compara el rendimiento de ambos algoritmos con base en métricas como el número de nodos explorados, el tiempo de ejecución y la longitud de la solución encontrada. Esta comparación permite evaluar las ventajas y limitaciones de cada estrategia en términos de eficiencia computacional y aplicabilidad en problemas similares.

II. DESARROLLO

Puzzle 8

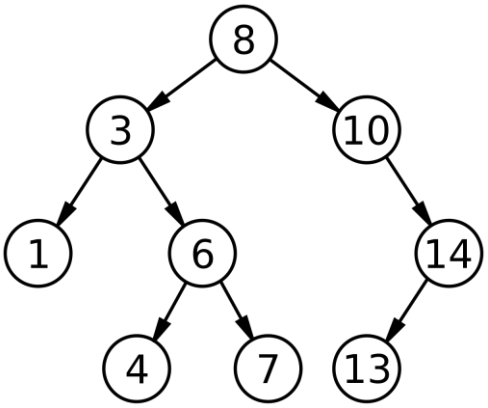
El 8-Puzzle es un rompecabezas deslizante clásico que consta de una cuadrícula de 3x3 con ocho fichas numeradas del 1 al 8 y un espacio vacío. El objetivo es reorganizar las fichas desde una configuración inicial hasta una configuración objetivo específica, generalmente con los números en orden ascendente y el espacio vacío en la esquina inferior derecha. El 8-Puzzle es una versión reducida del 15-Puzzle, inventado en 1874 por Noyes Palmer Chapman, un cartero de Canastota, Nueva York. El 15-Puzzle se popularizó rápidamente en los Estados Unidos en 1880 y se convirtió en una sensación nacional. El 8-Puzzle se utiliza frecuentemente en inteligencia artificial y ciencias de la computación como ejemplo de problemas de búsqueda y planificación.

- Movimientos válidos: Solo se puede mover una ficha adyacente al espacio vacío en las direcciones arriba, abajo, izquierda o derecha. No se permiten movimientos diagonales.
- Objetivo: Alcanzar la configuración objetivo mediante una serie de movimientos válidos.

Árbol de Búsqueda

El árbol de búsqueda es una estructura de datos fundamental en inteligencia artificial utilizada para representar formalmente el espacio de búsqueda de un problema, donde cada nodo del árbol simboliza un estado posible y las ramas representan las acciones o transiciones que conducen de un estado a otro. El nodo raíz representa el estado inicial del problema, y los nodos hijos corresponden a los estados sucesores generados al aplicar acciones válidas desde su nodo padre. Esta estructura permite organizar y explorar sistemáticamente todas las posibles secuencias de acciones que llevan desde el estado inicial hacia un estado objetivo o solución. Los árboles de búsqueda pueden crecer exponencialmente dependiendo del número de acciones

disponibles y la profundidad requerida para alcanzar la solución, lo que genera desafíos tanto en términos de complejidad temporal como espacial. La exploración de este árbol se puede realizar utilizando diferentes estrategias, como búsqueda en amplitud, búsqueda en profundidad, búsqueda informada y otras variantes, que difieren principalmente en el orden en que se expanden los nodos. En el caso del Puzzle 8, el árbol de búsqueda permite modelar todos los posibles movimientos del tablero y las distintas configuraciones resultantes, sirviendo como base para aplicar algoritmos como BFS o A*.



Nodo de Búsqueda

El nodo de búsqueda es una unidad básica del árbol de búsqueda que encapsula información esencial sobre un estado del problema y su relación con otros nodos dentro del espacio de búsqueda. Cada nodo contiene el estado que representa, un puntero al nodo padre desde el cual fue generado, la acción aplicada para alcanzar el estado actual, el costo acumulado del camino desde el estado inicial (usualmente denotado como g(n)), y, en algoritmos informados, una estimación del costo restante hacia la meta, denominada función heurística h(n)). Esta estructura permite no solo reconstruir el camino desde la solución hasta el estado inicial sino también comparar y priorizar nodos según el criterio del algoritmo utilizado. En el contexto del algoritmo A*, por ejemplo, se emplea la suma de $g(n) + h(n)$ como función de evaluación para decidir qué nodo expandir a continuación. La precisión y diseño de los nodos de búsqueda son críticos para la eficiencia de los algoritmos, ya que determinan qué rutas se exploran y cuáles se descartan. En problemas como el Puzzle 8, los nodos representan configuraciones del tablero, y su correcta implementación es clave para evitar ciclos, detectar soluciones y calcular métricas relevantes como número de movimientos y nodos generados.

Búsqueda en Amplitud (BFS)

El algoritmo Breadth-First Search (BFS) es una técnica de búsqueda no informada que explora el espacio de estados de manera sistemática nivel por nivel, comenzando desde el nodo raíz y visitando todos los nodos vecinos antes de continuar a los nodos del siguiente nivel de profundidad. Esta estrategia se implementa utilizando una estructura de datos tipo cola FIFO (First-In, First-Out), en la cual los nodos recién descubiertos se agregan al final y se procesan en orden de llegada. BFS es particularmente efectivo para encontrar la solución óptima en términos del menor número de pasos en entornos donde todas las transiciones de estado tienen el mismo costo, como es el caso del Puzzle 8. Sin embargo, debido a su naturaleza exhaustiva, BFS tiende a consumir grandes cantidades de memoria, ya que mantiene todos los nodos generados en la frontera, lo que limita su escalabilidad en espacios de búsqueda más amplios. A pesar de esta desventaja, su simplicidad y garantía de optimalidad la hacen útil en entornos controlados o con restricciones de profundidad. En el contexto del Puzzle 8, BFS asegura encontrar la solución más corta en términos de movimientos, aunque su tiempo de ejecución puede incrementarse significativamente a medida que aumenta la complejidad del estado inicial. El funcionamiento básico de este algoritmo puede representarse con el siguiente pseudocódigo:

```
func BFS(inicio, objetivo):
    abrir ← cola con nodo inicio
    cerrados ← conjunto vacío
    mientras abrir no está vacío:
        actual ← sacar primero de abrir
        si actual es objetivo:
            retornar camino desde inicio a actual
        agregar actual a cerrados
        para cada vecino de actual:
            si vecino no en cerrados y no en abrir:
                agregar vecino a abrir
                establecer padre de vecino como actual
    retornar fallo
```

El pseudocódigo ilustrativo del algoritmo BFS, refleja su estrategia sistemática y su estructura secuencial. A pesar de su simplicidad, BFS es una herramienta fundamental en ciencias de la computación y proporciona una base conceptual sólida para algoritmos de búsqueda más complejos y eficientes.

Algoritmo A*

El algoritmo A* es una técnica de búsqueda informada que combina las ventajas de la búsqueda de costo uniforme y la búsqueda voraz, utilizando una función de evaluación definida como $f(n) = g(n) + h(n)$, donde $g(n)$ representa el costo acumulado desde el nodo inicial hasta el nodo actual n, y $h(n)$ es una heurística que estima el costo restante hasta el objetivo. A* garantiza encontrar la solución óptima siempre que la heurística sea admisible, es decir, que nunca sobreestime el costo real al objetivo. En problemas como el Puzzle 8, A* se destaca por su eficiencia al explorar solo los nodos que tienen mayor probabilidad de llevar a una solución, lo que lo hace significativamente más rápido que algoritmos no informados como BFS en espacios de búsqueda grandes. El algoritmo mantiene una cola de prioridad (generalmente implementada como una min-heap) donde los nodos son seleccionados según el menor valor de $f(n)$. A medida que expande nodos, actualiza los caminos más cortos conocidos y evita repetir estados ya explorados mediante una estructura de conjunto cerrado. La eficacia de A* depende en gran parte de la heurística utilizada; si esta es consistente, A* no volverá a visitar nodos ya evaluados. Su funcionamiento puede representarse con claridad mediante el siguiente pseudocódigo:

```
func A_estrella(inicio, objetivo):
    abrir ← cola de prioridad con nodo inicio
    cerrados ← conjunto vacío
    mientras abrir no está vacío:
        actual ← nodo con menor f(n) en abrir
        si actual es objetivo:
            retornar camino desde inicio a actual
        mover actual de abrir a cerrados
        para cada vecino de actual:
            si vecino en cerrados:
                continuar
            calcular g(n) y h(n) para vecino
            si vecino no en abrir o nuevo g(n) es menor:
                actualizar padre de vecino y su f(n)
                agregar vecino a abrir si no está
    retornar fallo
```

La función de evaluación que guía la búsqueda se representa formalmente como:

$$f(n) = g(n) + h(n)$$

Gracias a su capacidad de encontrar soluciones óptimas de manera eficiente, A* se ha convertido en un estándar en problemas de planificación, navegación y resolución de rompecabezas.

Heurística de Manhattan

La heurística de Manhattan, también conocida como City Block Distance o L1 Distance, es una función heurística ampliamente utilizada en problemas de búsqueda informada como A*, especialmente en entornos cuadriculados como laberintos o el Puzzle 8. Esta heurística estima el costo restante para alcanzar el objetivo como la suma de las distancias horizontales y verticales entre cada ficha y su posición final, sin tener en cuenta obstáculos. En el contexto del Puzzle 8, proporciona una estimación optimista del número de movimientos necesarios para colocar cada ficha en su ubicación correcta, ya que asume desplazamientos libres en el eje horizontal y vertical. Dado que nunca sobreestima el costo real, la heurística de Manhattan es admisible, y al respetar la desigualdad triangular, también es consistente, lo que la hace adecuada para garantizar la optimalidad en algoritmos como A*. Es preferida frente a otras

heurísticas simples como el conteo de fichas fuera de lugar debido a su mayor precisión, aunque puede ser menos precisa que variantes más complejas como la heurística de Manhattan con conflictos lineales. Su implementación es sencilla y computacionalmente eficiente, lo que contribuye a su popularidad. La función heurística se define formalmente como:

$$h(n) = \sum_{i=0}^{N} (|x_i - x_i^*| + |y_i - y_i^*|)$$

donde (x_i, y_i) representa la posición actual de la ficha i , y (x_i^*, y_i^*) su posición objetivo. En términos computacionales, se puede expresar en pseudocódigo como sigue:

```
func heuristica_manhattan(tablero):
    total ← 0
    para cada ficha i en tablero:
        si ficha i no es espacio vacío:
            x, y ← posición actual de ficha i
            x_obj, y_obj ← posición objetivo de ficha i
            total ← total + abs(x - x_obj) + abs(y - y_obj)
    retornar total
```

III. DISEÑO DEL DISTEMAD

El sistema propuesto para la resolución automatizada del Puzzle 8 fue concebido como una plataforma modular que permite la comparación y visualización de diferentes enfoques de búsqueda, tanto informados como no informados. La arquitectura fue diseñada para facilitar la separación de responsabilidades entre los distintos componentes funcionales, garantizando la escalabilidad y la facilidad de mantenimiento. El diseño se centra en una lógica central encargada de definir el entorno del problema (el tablero del Puzzle 8), agentes que representan algoritmos de búsqueda y un sistema de interfaz gráfica basado en Pygame para permitir la interacción visual y comprensión del proceso de resolución. La interacción entre estos módulos está orquestada desde un archivo principal que actúa como punto de entrada, permitiendo la ejecución secuencial de los agentes sobre un mismo tablero generado aleatoriamente, asegurando condiciones equitativas para la comparación de resultados. La elección de una interfaz gráfica no solo mejora la interpretación de los resultados, sino que también aporta valor educativo al facilitar el entendimiento del comportamiento interno de cada algoritmo.

IV. ESTRUCTURA DEL SISTEMA

La estructura del sistema está organizada en un directorio principal denominado `resolucion-automatizada-puzzle8`, el cual contiene subdirectorios y archivos distribuidos por funcionalidad. El subdirectorio `agents/` agrupa los algoritmos de búsqueda implementados como agentes autónomos. En particular, `bfs_agent.py` contiene la implementación del algoritmo de Búsqueda en Amplitud (BFS), un enfoque no

informado que expande nodos en orden creciente de profundidad, mientras que `a_star_agent.py` implementa el algoritmo A*, el cual combina el costo acumulado del camino con una heurística para optimizar la búsqueda. La lógica del puzzle se encuentra centralizada en `utils/puzzle.py`, que define la representación del tablero, las operaciones permitidas sobre él, y la generación de configuraciones válidas. Asimismo, `utils/heuristics.py` contiene la implementación de la heurística de Manhattan y funciones auxiliares como la verificación de resolubilidad del tablero inicial. El módulo de visualización, ubicado en `ui/display.py`, utiliza la librería Pygame para renderizar el estado del tablero, los movimientos del agente y las métricas de evaluación como tiempo de ejecución, cantidad de nodos explorados y longitud de la solución. El archivo `main.py` constituye el núcleo de control, donde se genera una instancia aleatoria del tablero, se ejecutan los agentes y se visualizan los resultados. Finalmente, `requirements.txt` lista las dependencias necesarias para ejecutar el sistema, mientras que `README.md` proporciona documentación básica sobre el uso y configuración del proyecto. Esta organización modular facilita la extensión del sistema para incluir nuevos agentes o heurísticas, y permite reutilizar componentes de forma eficiente.

V. RESULTADOS

Para evaluar el desempeño de los algoritmos implementados, se seleccionó una configuración meta no tradicional del Puzzle 8, definida por la siguiente disposición:

$$\text{GOAL\_STATE} = \begin{matrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{matrix}$$

Esta configuración introduce un reto adicional al alejarse del objetivo canónico utilizado comúnmente (con el espacio vacío en la esquina inferior derecha), permitiendo una evaluación más completa del comportamiento de los algoritmos. Se realizaron múltiples ejecuciones utilizando tableros iniciales generados aleatoriamente y verificando previamente su resolubilidad. Para cada ejecución, se registraron tres métricas clave: el número de nodos explorados, el tiempo total de ejecución y la longitud del camino solución (es decir, el número de movimientos requeridos para alcanzar el estado objetivo).

Los resultados evidencian diferencias significativas entre los dos enfoques analizados. En general, el algoritmo A* logró encontrar soluciones óptimas con mayor eficiencia en tiempo y número de nodos explorados, gracias a la incorporación de la heurística de Manhattan. Por el contrario, el algoritmo BFS, al no contar con información heurística, incurrió en una exploración exhaustiva del espacio de estados, lo cual se reflejó en una mayor carga computacional y tiempos de respuesta más prolongados. No obstante, ambos algoritmos fueron capaces de resolver el problema satisfactoriamente, retornando caminos válidos hacia la meta en todos los casos.

VI. COMPARACIÓN DE AGENTES

La comparación entre los agentes se realizó en función de las métricas obtenidas: número de nodos explorados, tiempo de ejecución (en segundos) y longitud de la solución. En una ejecución típica, partiendo de un tablero inicial aleatorio válido, los resultados fueron los siguientes:

$$\text{INITIAL\_STATE} = \begin{matrix} 5 & 1 & 0 \\ 6 & 2 & 4 \\ 3 & 8 & 7 \end{matrix}$$

Métrica	Agente BFS	Agente A*
Nodos explorados	158,854	1,143
Tiempo de ejecución	11.52 segundos	0.180 segundos
Longitud de la solución	24 movimientos	24 movimientos

$$\text{INITIAL\_STATE} = \begin{matrix} 8 & 6 & 3 \\ 2 & 7 & 0 \\ 4 & 1 & 5 \end{matrix}$$

Métrica	Agente BFS	Agente A*
Nodos explorados	81,485	631
Tiempo de ejecución	7.502 segundos	0.164 segundos
Longitud de la solución	21 movimientos	21 movimientos

$$\text{INITIAL\_STATE} = \begin{matrix} 4 & 5 & 1 \\ 8 & 2 & 0 \\ 6 & 3 & 7 \end{matrix}$$

Métrica	Agente BFS	Agente A*
Nodos explorados	137,947	1035
Tiempo de ejecución	10.890 segundos	0.184 segundos
Longitud de la solución	23 movimientos	23 movimientos

$$\text{INITIAL\_STATE} = \begin{matrix} 2 & 7 & 3 \\ 1 & 8 & 0 \\ 5 & 4 & 6 \end{matrix}$$

Métrica	Agente BFS	Agente A*
Nodos explorados	6,486	71
Tiempo de ejecución	1.899 segundos	0.025 segundos
Longitud de la solución	15 movimientos	15 movimientos

$$\text{INITIAL\_STATE} = \begin{matrix} 6 & 2 & 8 \\ 4 & 1 & 0 \\ 3 & 5 & 7 \end{matrix}$$

Métrica	Agente BFS	Agente A*
Nodos explorados	224,594	4,288
Tiempo de ejecución	16.323 segundos	0.722 segundos
Longitud de la solución	27 movimientos	27 movimientos

Estos resultados confirman que A* supera ampliamente a BFS en eficiencia computacional. Aunque ambos algoritmos encontraron soluciones de la misma longitud, A* lo logró explorando una fracción significativamente menor del espacio de estados, lo cual se traduce en un menor uso de recursos y tiempos de ejecución considerablemente más bajos. Esto valida el papel fundamental de las heurísticas bien diseñadas en la optimización de algoritmos informados. La heurística de Manhattan utilizada en A* resultó efectiva para guiar la búsqueda de manera eficiente hacia el objetivo, reduciendo el número de decisiones innecesarias.

En contraste, BFS garantiza la optimalidad de la solución en términos de número de pasos, pero a costa de un consumo considerable de tiempo y memoria, debido a su naturaleza exploratoria exhaustiva. Esto lo hace poco escalable a problemas de mayor tamaño o complejidad.

VII. EFICIENCIA, VENTAJAS Y DESVENTAJAS DE CADA ENFOQUE

Desde una perspectiva de eficiencia, el algoritmo A* demostró ser superior en términos de velocidad, resolviendo el problema en menor tiempo gracias a su enfoque heurístico. Esta característica lo hace especialmente adecuado para problemas donde el tiempo de respuesta es crítico. Además, A* garantiza la optimalidad de la solución siempre que la heurística utilizada sea admisible, como es el caso de la distancia de Manhattan.

Sin embargo, A* puede tener un costo adicional en términos de memoria, ya que necesita almacenar tanto los nodos abiertos como los valores de costo acumulado y heurístico. Además, su desempeño está estrechamente ligado a la calidad de la heurística empleada: si esta no refleja de forma precisa la proximidad al objetivo, el algoritmo puede explorar más nodos de los necesarios.

Por otro lado, BFS tiene la ventaja de ser simple de implementar y garantiza la solución más corta en términos de número de movimientos. No requiere ninguna función heurística y su comportamiento es predecible. No obstante, su mayor desventaja radica en la ineficiencia computacional: explora indiscriminadamente todos los nodos al mismo nivel antes de avanzar, lo que genera un alto consumo de memoria y tiempo, especialmente en espacios de búsqueda grandes o complejos. Esto lo hace poco adecuado para aplicaciones en tiempo real o problemas de mayor escala.

En síntesis, A* ofrece una solución más eficiente y escalable para problemas como el Puzzle 8, siempre que se disponga de una heurística adecuada. BFS, aunque garantiza soluciones óptimas, es más costoso en recursos y menos viable en contextos donde la eficiencia sea prioritaria.

VIII. CONCLUSIÓN

Este trabajo abordó la resolución del Puzzle 8 mediante dos enfoques contrastantes: un agente no informado basado en Búsqueda en

Anchura (BFS) y un agente informado utilizando el algoritmo A* con heurística de Manhattan. Ambos métodos demostraron ser capaces de encontrar soluciones válidas al problema, sin embargo, su comportamiento y eficiencia difirieron significativamente en múltiples aspectos. La elección de una configuración meta no convencional permitió observar con mayor claridad las fortalezas y limitaciones de cada estrategia.

A lo largo de las pruebas experimentales, el algoritmo A* fue consistentemente más rápido que BFS, alcanzando la solución en un tiempo considerablemente menor. Esta superioridad en tiempo se debió a la guía proporcionada por la heurística de Manhattan, que permitió al algoritmo enfocar su búsqueda en las rutas más prometedoras. Sin embargo, un hallazgo relevante fue que A* no siempre utilizó menos nodos que BFS; de hecho, en varios escenarios, exploró una mayor cantidad. Esto puede atribuirse a la necesidad de mantener actualizadas las prioridades en la estructura de datos (cola de prioridad), y a que en algunas configuraciones, la heurística no fue lo suficientemente precisa para reducir el espacio de búsqueda de forma sustancial.

IX. REPOSITORIO

https://github.com/mariadev22/resolucion-automatizada-puzzle8.git

X. BIBLIOGRAFÍA

[1]-Datta, D. (2019, April 24). *8 Puzzle Problem*. GeeksforGeeks. https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/

[2]-dpthegrey. (2019, July 31). *8 Puzzle Problem*. Medium. https://medium.com/@dpthegrey/8-puzzle-problem-2ec7d832b6db

[3]-Varun, V. (2019, October 14). *Solving 8 puzzle using A* Algorithm*. Good Audience. https://blog.goodaudience.com/solving-8-puzzle-using-a-algorithm-7b509c331288

[4]-GamesCrafters. (s.f.). *8 Puzzle Game*. UC Berkeley. https://gamescrafters.berkeley.edu/site-legacy-archive-sp20/games.php?puzzle=8puzzle

[5]-Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall.

[6]-Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

[7]-Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann.

[8]-Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. 3rd ed. MIT Press.

[9]-Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann.