Análisis y estimación de mejores rutas de un vehículo con menos giros con A*, distancia de Manhattan y Euclidiana

Denisse Baldivieso, Nicole Góngora, Llubitza Linares

Universidad Privada Boliviana

Resúmen

En el presente trabajo se estudiará el algoritmo A para el cálculo de búsqueda de información en forma de nodos, así como las fórmulas de distancia entre nodos que utiliza: Heurística de Manhattan y Heurística Euclidiana. Se realizará un experimento a base de la creación de la ruta desde un punto bi-dimensional a otro en 8 pruebas, cada 4 correspondiendo a una de las fórmulas, observando y verificando su rendimiento y su entorno más óptimo*

Key Words: *Algoritmo A*, Heuristica de Manhattan, Heuristica Euclidiana, calculo heuristico, inteligencia artificial.*

1. Introducción

Los algoritmos de búsqueda son un conjunto de funciones diseñadas para trabajar con diversos conjuntos de datos, tomando como sujetos de prueba uno o varios puntos de información específica y como objetivo un camino óptimo para acceder a estos. Sus enfoques son variados: desde árboles ordenados de nodos que indican relación predecesor-actual entre ellos, hasta grafos o conjunto de puntos en el que cada uno, aparte de un valor, contienen un peso de relevancia que los distingue del resto; pero todos tienen un mismo objetivo: Encontrar la mejor ruta que relacione ciertos sectores de información, sino todos, con el menor costo de tiempo y/o recorrido que sea posible.

La motivación que impulsa a la elaboración y trabajo en este proyecto no sólamente es obtener conocimiento avanzado de un algoritmo de búsqueda bastante utilizado: se desea verificar su utilidad y rendimiento a partir de dos funciones de cálculo que son parte esencial de su ejecución: La función de distancia Euclidiana, enfocada a varios ángulos de búsqueda, y la de Manhattan, con un camino más limitado pero en algunos casos el más rápido.

El proyecto quiere verificar cuál de ambas opciones ha de trabajar mejor por el tiempo al ejecutarse y la distancia que se debe tomar en cuenta.

Para esto, se ha establecido un caso a resolver, con una problemática común en el mundo actual: Hallar la mejor ruta desde un punto de la localización de un sujeto (persona o auto, entre otros) hacia un destino establecido, considerando no sólamente los posibles obstáculos que necesite evitar, sino también el tiempo y los pasos que le tome realizar su recorrido. A partir de esta consigna, se

establecen los objetivos de verificar los entornos y situaciones en el que cada una de las dos funciones de distancia del algoritmo A* se ejecuta mejor, tanto por velocidad como por calidad.

2. Antecedentes o Estado del Arte

*A. Algoritmo A**

A* o A-star es un algoritmo especializado en la búsqueda de una ruta óptima entre dos nodos o puntos clave ubicados en un grafo computacional, este conceptualizado en una matriz bidimensional (Wikipedia, s.f.). Fué creado en 1963 por Nils Nilsson, Bertram Raphael, Charles Rosen y Peter Hart entre otros estudiantes del Stanford Research Institute; y diseñado para calcular estimaciones heurísticas entre los puntos clave en los que el autómata Shakey pudiera desplazarse de la manera más óptima en un entorno específico, utilizando los puntos como nodos y generando siempre un árbol de búsqueda resultante menos extenso (Nilsson, 2009).

Parte de la base del algoritmo Dijsktra, el cual se especializa en el uso de estructuras de cola, en la que los elementos son extraídos en el mismo orden que se almacenaron, para almacenar la información de los nodos vecinos que el punto actual de la ruta, inicial o no, tenga a su disposición, y decidir el mejor de estos a conectar para intentar llegar al objetivo a partir de un valor de costo o peso que se va sumando al costo acumulado por ruta; esta cifra siempre iniciando en 0 y apuntando al menor valor. (Martell & Sandberg, 2016)

El algoritmo sigue las siguientes reglas de ejecución:

1. Inicializar dos colas de nodos: una abierta (nodos disponibles a explorar) y cerrada (para nodos que ya han sido explorados en la búsqueda).

2. Agregar el punto o nodo inicial de la búsqueda en la cola abierta, con un costo de viaje 0 y heurística bajo.

3. Extraer el nodo de la cola, siendo el de menor costo de viaje, y examinar los vecinos disponibles que tuviera, es decir, rutas transitables o fuera de la cola de celdas cerradas (Cuevas Guzmán, 2013). A continuación, colocar este nodo en la cola cerrada, siendo el que ya no se va a examinar, y a los vecinos en la cola abierta.

4. Por cada vecino encontrado calcular su valor de costo de viaje, mostrado en la siguiente ecuación.

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

donde *g(n)* es el costo acumulado desde el punto actual hasta el vecino y *h(n)* el costo heurístico desde este último hacia el punto final. Este último se calcula a partir de una fórmula de distancia entre puntos, siendo la más común la Euclidiana.

Durante la iteración por cada vecino encontrado, se dará prioridad a aquel que haya proporcionado el valor de costo más bajo.

5. Aquel nodo vecino con el valor deseado se convierte en actual: Se extrae de la cola abierta y se coloca en la cerrada, con la consideración de que tal vez no sea el indicado para la solución.

6. Repetir el proceso con los vecinos disponibles del nuevo nodo. Si uno de ellos ya está en la lista abierta antes del proceso, se verifica su costo para determinar si cambiar su ruta predecesora o no.

7. El programa se detiene cuando ha fallado en encontrar un camino óptimo del nodo inicial al final, o cuando se ha llegado al nodo objetivo de manera exitosa, guardando el camino utilizado (Martell & Sandberg, 2016).

```
Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
while the OPEN list is not empty {
  Take from the open list the node node_current with the lowest
       f(node_current) = g(node_current) + h(node_current)
  if node_current is node_goal we have found the solution; break
  Generate each state node_successor that come after node_current
  for each node_successor of node_current {
    Set successor_current_cost = g(node_current) + w(node_current, node_successor)
    if node_successor is in the OPEN list {
      if g(node_successor) ≤ successor_current_cost continue (to line 20)
    } else if node_successor is in the CLOSED list {
      if g(node_successor) ≤ successor_current_cost continue (to line 20)
      Move node_successor from the CLOSED list to the OPEN list
    } else {
      Add node_successor to the OPEN list
      Set h(node_successor) to be the heuristic distance to node_goal
    }
    Set g(node_successor) = successor_current_cost
    Set the parent of node_successor to node_current
  }
  Add node_current to the CLOSED list
}
if(node_current != node_goal) exit with error (the OPEN list is empty)
```

Figura 1. Pseudocódigo del algoritmo A*

Una de las mayores ventajas que este algoritmo proporciona es la garantía de que todos los puntos del recorrido van a ser evaluados de manera equitativa gracias a tener un costo uniforme de manera indirecta entre ellos, es decir, no se declaran los pesos de cada nodo al inicio, sino se toma sólamente las distancias calculadas entre estos. Así mismo, a diferencia del algoritmo Dijkstra, no toma los nodos de manera inmediata para generar el camino, utilizando la estimación de distancia para así evitar recorrer rutas innecesarias y tener un rendimiento rápido (University of Cambridge, s. f.).

Sin embargo, la mayor desventaja de A* es la cantidad de memoria que debe ocupar su ejecución, ya que debe memorizar cada nodo disponible y mantenerlo almacenado en la cola (Konakalla, 2014). Así mismo, si bien cualquier fórmula heurística puede ajustarse a este algoritmo, su velocidad puede variar de acuerdo al método seleccionado (University of Cambridge, s. f.).

El algoritmo A* ha tenido varias aplicaciones a partir de su desarrollo e introducción en el campo de informática: La inteligencia artificial para la ubicación y recorrido hacia una localización en mapas GPS, el movimiento de personajes en videojuegos y/o entornos interactivos, revisión de recorridos de consultas en sitios de búsqueda online, entre otros.

B. Heurística de Manhattan

La heurística de la distancia de Manhattan, conocida como la distancia de Minkowski o Geometría del taxista es el cálculo consistente en la suma de las distancias absolutas de las coordenadas de los puntos de un espacio, como se explica en la siguiente ecuación (Sahani, 2021).

$$h(n) = |(n_x - final_x)| + |(n_y - final_y)|$$

Esta fórmula de cálculo heurístico es la más utilizada en A*, su movimiento será siempre en cuatro direcciones, y ocupando menos espacio de exploración a la hora de estimar puntos. Sin embargo, su desventaja consiste en generar un valor de distancia más grande al sólo moverse en líneas rectas y no diagonales, lo cual toma más tiempo en el cálculo (Patel, s. f.).

C. Heurística Euclidiana

La heurística de la distancia Euclidiana o de Pitágoras es el cálculo de la menor distancia entre dos puntos, siendo la raíz cuadrada de la suma de las diferencias entre las coordenadas de los puntos.

$$h(n) = \sqrt{(n_x - final_x)^2 + (n_y - final_y)^2}$$

Si bien la distancia a recorrer entre un punto al otro es menor a la de Manhattan al poder recorrer caminos más accesibles debido a su movimiento múltiple, la desventaja que tiene al ser utilizado en A* es que, debido a que debe examinar todos los ángulos posibles de la grilla, tomará más tiempo analizar las posibles rutas por el menor costo (Patel, s. f.).

3. Propuesta

A. Problemática a plantear

La propuesta del problema a resolver consiste en el análisis de la toma de la ruta más rápida de un punto a otro en cuatro lugares de llegada en un mismo mapa.

Este caso está plasmado en un entorno similar a bloques de calles y avenidas. El sujeto que se utilizará en este ambiente será un camión repartidor que contiene elementos de fragilidad media, en este caso, pasteles y tortas, muchas de más de un nivel como de boda o quinceañeras. Su objetivo es salir desde una de las casas matrices hasta el punto donde se le solicitó el delivery. El análisis de la ruta ideal se basará en la condición de no solo llegar en un tiempo puntual, sino también intentar no dar muchos giros por las calles, ya que, en un entorno realista, haría que en un descuido el camión voltee alguna de las tortas, dañándola.

La hipótesis planteada para este problema es la siguiente: El camino ideal será aquel que, además de llegar en un tiempo puntual (es decir, el menor de los registros de cálculo), tenga la menor cantidad de giros que el camión deba hacer desde la casa matriz hasta el destinatario. En este caso, una ruta calculada con la distancia Euclidiana puede llegar a cumplir los requerimientos.

Sin embargo, ante esta hipótesis existen posibles factores que la puedan refutar:

1. Si la ruta está entre avenidas que estén en un esquema cuadriculado de calle normal, una ruta con giros de 90 grados calculada por Manhattan puede dar mejor velocidad en llegar al destino, siendo menos cambios de direcciones necesarios que la Euclidiana.

2. La ruta por distancia Euclidiana necesita mirar por todos los ángulos posibles a diferencia de la de Manhattan. Si bien la ruta entre los puntos sería la más corta, tal vez tome más tiempo en calcular.

B. Planificación del algoritmo

Para la realización del experimento que resuelva la problemática antes explicada, se utilizó un código escrito en el lenguaje de programación Python, no solamente por su flexibilidad y facilidad de implementación, sino también su amplia gama de librerías gratis cuya importación e implementación ayudan a la realización de códigos más avanzados.

1. Librerías

- La librería Numpy, con las funciones matemáticas necesarias para el cálculo de raíces y la estructuración de matrices
- La librería Matplotlib.pyplot, con herramientas y funciones para la graficación de los valores matriciales y las coordenadas de los puntos de inicio, llegada y las rutas calculadas
- La librería Heapq, con funciones y una estructura similar a la de una cola de prioridad, en la cual el nodo padre o primero a extraer será aquel del menor valor del grupo

- La librería Time, para el cálculo del tiempo de ejecución de un algoritmo.

2. *Estructuras:*

- Conjuntos/Diccionarios: Los puntos de origen, valores g y f de distancia y costo.
- Listas: Movimientos posibles para ir a nodos vecinos, nodos cerrados, la ruta final calculada.

4. Resultados

Con las iniciativas planteadas para la resolución del experimento, se estableció el mapa en forma de una matriz de valores: 1 indica bloques de edificios y 0 una ruta libre, como se muestra en la Figura 2.

```
nmap = np.array([
    [0,0,1,1,1,1,0,1,1,1,0,0,0,0,0,1,1,1,1,1],
    [1,0,1,1,1,1,0,1,1,1,0,0,1,1,0,1,1,1,1,1],
    [1,0,1,1,1,1,0,1,1,1,0,0,1,1,0,1,1,1,1,1],
    [1,0,1,1,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0],
    [1,0,1,1,1,1,0,1,1,0,1,1,1,1,1,1,1,0,1,1],
    [1,0,1,1,1,1,0,1,1,0,1,1,1,1,1,1,1,0,1,1],
    [0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1],
    [0,1,1,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,1,1],
    [1,1,1,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,0,0],
    [1,1,1,0,0,0,1,1,0,0,1,1,1,1,0,1,1,1,0,0],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0],
    [1,0,1,1,1,1,0,1,1,1,0,0,0,0,0,1,1,1,0,0],
    [1,0,1,1,1,1,0,1,1,1,0,0,1,1,0,1,1,1,0,0],
    [0,0,0,0,0,0,0,1,1,1,0,0,1,1,0,0,0,0,0,0],
    [1,1,0,1,1,1,0,0,0,0,0,0,1,1,0,1,1,1,0,0],
    [1,1,0,1,1,1,1,1,1,0,1,1,1,1,0,1,1,1,0,0],
    [1,1,0,1,1,1,1,1,1,0,1,1,1,1,0,0,0,0,0,0],
    [1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,1,1,1,1],
    [1,1,0,1,1,1,1,1,1,0,1,1,1,1,0,0,1,1,1,1],
    [1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,1,1,1,1]])
```

Figura 2. Matriz de valores qué servirá de mapa

Para los puntos de partida y llegada, se establecieron los siguientes:

	Salida	Llegada
Primera	(0,0)	(16,19)
Segunda	(16,19)	(19,2)
Tercera	(19,2)	(0,13)
Cuarta	(0,13)	(19.14)

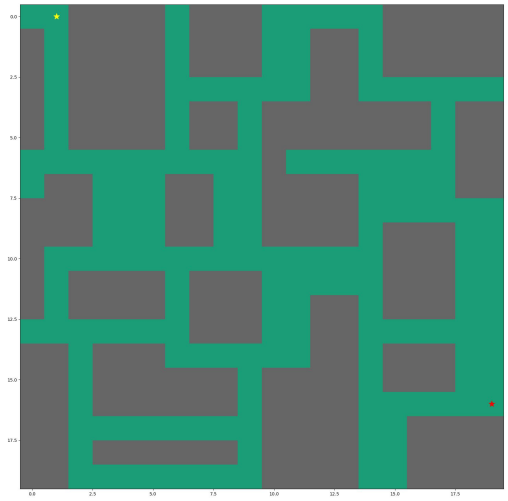

Figura 3. Ilustración del mapa generado por pyplot

Se ejecutó la función de hallazgo de ruta seis veces, las cuatro rutas y los dos métodos de distancia, generando caminos accesibles, como se muestra en las Figuras 4 a 9.

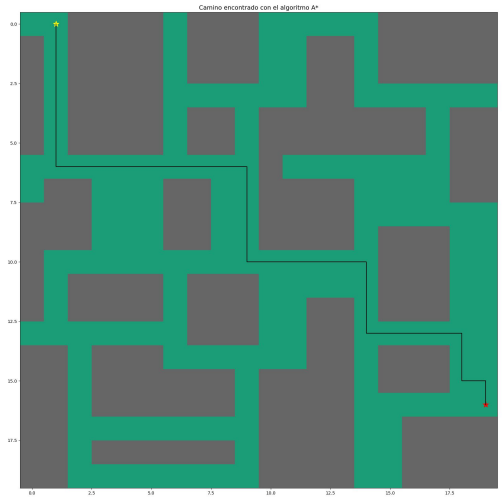

Figura 4. Primera ruta por distancia de Manhattan

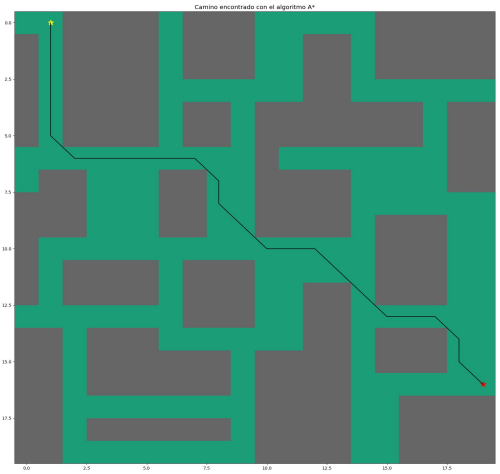

Figura 5. Primera ruta por distancia Euclidiana

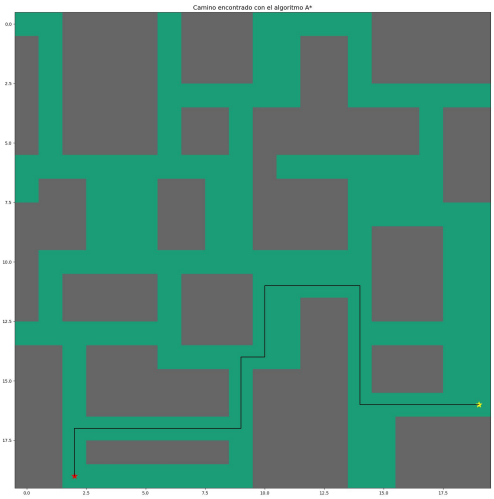

Figura 6. Segunda ruta por distancia de Manhattan

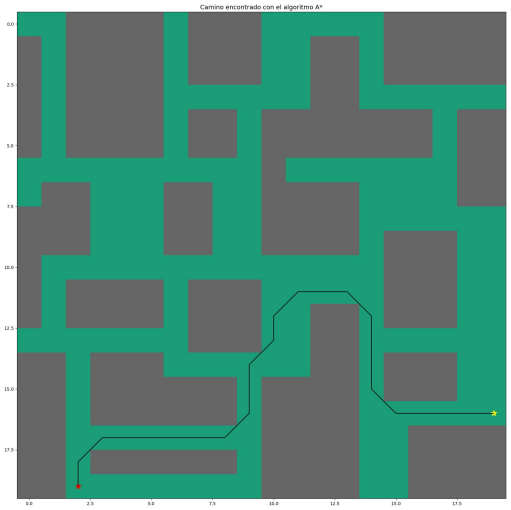

Figura 7. Segunda ruta por distancia Euclidiana

Figura 8. Tercera ruta por distancia de Manhattan

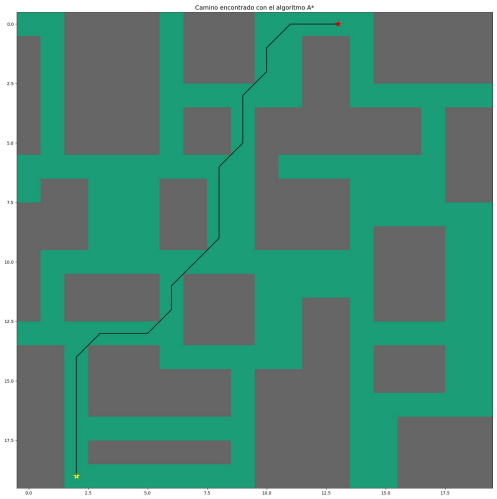

Figura 9. Tercera ruta por distancia Euclidiana

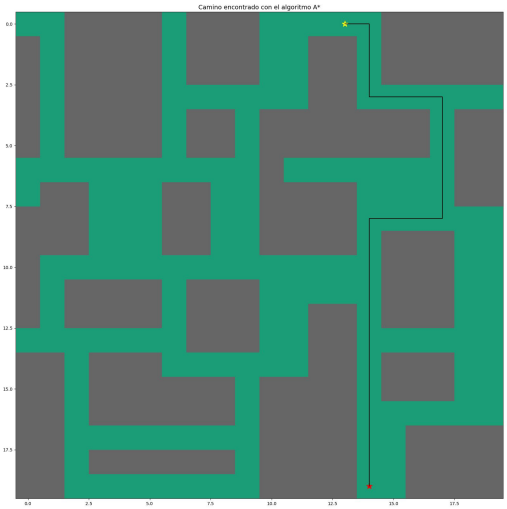

Figura 10. Cuarta ruta por distancia de Manhattan

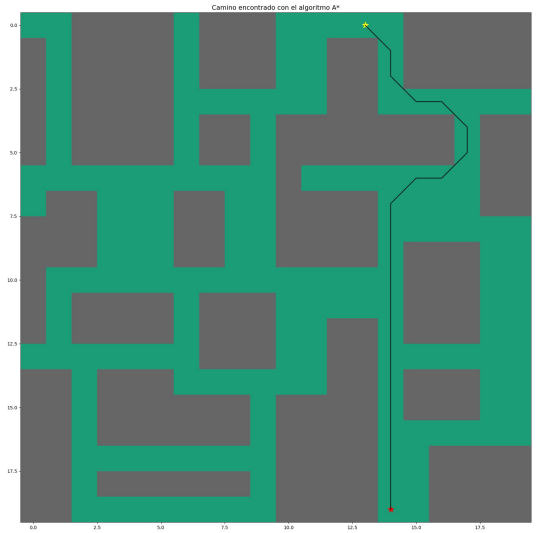

Figura 11. Cuarta ruta por distancia Euclidiana

La ejecución del código planteado devolvió dos valores, la longitud de la ruta calculada, y la velocidad en milisegundos que le tomó. Así mismo, se contaron las curvas por gráfica, obteniendo así la siguiente tabla:

Ruta	Velocidad (ms)	Pasos del camino	# de curvas
1 M	**1.9969**	35	**8 (90°)**
1 E	1.9977	**26**	11 (45°)
2 M	2.9921	31	**7 (90°)**
2 E	**2.0022**	**25**	12 (45°)
3 M	**3.9975**	31	**11 (90°)**
3 E	0.9989	**24**	12 (45°)
4 M	**2.0015**	27	**5 (90°)**
4 E	2.0112	**22**	9 (45°)

A partir de los datos obtenidos, se puede interpretar que:

- La primera, tercera y cuarta ruta son más óptimas con Manhattan. Su

distancia es mayor a la que harían con giros de 45 grados, pero llegan a una mejor velocidad y no requieren de realizar muchas curvas.

- La segunda ruta es más óptima con la distancia Euclidiana. Si bien le toma realizar varias curvas en 45 grados para llegar al destino, tiene menos distancia a recorrer y llega más rápido.

Con estos resultados, se revisa la hipótesis antes planteada, así como sus posibles refutaciones, y se puede observar que una de ellas ha sido cumplida: En un camino de avenidas y calles de forma recta y cuadrada, Manhattan llevará a un mejor resultado, con una mayor velocidad y, sorpresivamente, menos giros a realizar que la Euclidiana. Sin embargo, se puede ver que, en rutas con mayor espacio de desplazamiento, un camino con giros de 45° será el mejor, ya que indica que es más efectivo un recorrido diagonal que uno escalonado.

5. Conclusiones

Este experimento ha demostrado la variedad qué puede existir en el rendimiento del algoritmo A*, un método con una utilidad grande en el mundo de la programación y ciencia de datos. Este algoritmo, enfocado en considerar su información como puntos en un entorno bidimensional, proporciona un acercamiento no solamente cuantitativo con respecto a la distancia más corta entre ellos, sino también de calidad, garantizando uno de estos atributos, sino los dos: velocidad de cálculo y menos información adicional requerida para llegar a un resultado óptimo.

A partir de este proyecto, se ha observado el potencial de este algoritmo más allá del hallazgo del mejor camino de conexión entre un punto a otro: Gracias a su tendencia a priorizar el menor costo de realizar esta ruta, se pueden generar análisis de información en una base de datos para verificar la relación existente de cada registro, y con esta, sacar conclusiones significativas. Algoritmos de segmentación como K-Means y de regresión en cualquier formato pueden ajustarse a este cálculo, obteniendo los mejores acercamientos de información que ayuden a comprender los datos proporcionados.

Como recomendaciones, para una siguiente implementación del algoritmo, una librería automática de Python que resuelva las distancias Euclidiana, Manhattan, entre otras utilizadas en el algoritmo A* podría facilitar el trabajo si se requiere sólamente estimar puntos fijos y no la grilla entera de opciones; a diferencia de este proyecto que fue a un camino más manual para demostrar la funcionalidad de este método. Así mismo, tomar más información relevante del rendimiento de la función puede dar mejores conclusiones en trabajos más avanzados, como el número de iteraciones necesarias que se realizaron, las veces que una estructura de datos es modificada, entre otros.

6. Bibliografia

Cuevas Guzmán, D. A. (2013). Análisis de Algoritmos Pathfinding [Tesis de Graduación]. PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO.

Martell, V. & Sandberg, A. (2016). Performance Evaluation of A* Algorithms [Tesis]. Blekinge Institute of Technology.

Nilsson, N. J. (2009, 3 septiembre). THE QUEST FOR ARTIFICIAL INTELLIGENCE - A HISTORY OF IDEAS AND ACHIEVEMENTS. En Cambridge University Press. Cambridge University Press. Recuperado 2 de octubre de 2022, de https://ai.stanford.edu/~nilsson/QAI/qai.pdf

Sahani, G. R. (2021, 15 diciembre). Euclidean and Manhattan distance metrics in Machine Learning. Medium. Recuperado 2 de octubre de 2022, de https://medium.com/analytics-vidhya/euclidean-and-manhattan-distance-metrics-in-machine-learning-a5942a8c9f2f

Patel, A. (s. f.). Heuristics. Red Blob Games. Recuperado 2 de octubre de 2022, de http://theory.stanford.edu/%7Eamitp/GameProgramming/Heuristics.html

University of Cambridge. (s. f.). A* search algorithm. Isaac Computer Science. Recuperado 3 de octubre de 2022, de https://isaaccomputerscience.org/concepts/dsa_search_a_star?examBoard=all&stage=all

Wikipedia contributors. (s. f.). A* search algorithm. Wikipedia, the Free Encyclopedia. Recuperado 2 de octubre de 2022, de https://en.wikipedia.org/wiki/A*_search_algorithm

Konakalla, S. V. (2014). A star Algorithm [Paper]. Indiana State University.