NOBILIS
Revista de Ciencias Sociales

Revista Científica de Ciencias Sociales, NOBILIS
https://nobilis.ube.edu.ec/index.php/nobilis/

COMPARACIÓN DE ESTRATEGIAS DE BÚSQUEDA NO INFORMADA Y HEURÍSTICA EN LA RESOLUCIÓN DE PROBLEMAS DE OPTIMIZACIÓN DE RUTAS: UN ESTUDIO EXPERIMENTAL

COMPARISON OF UNINFORMED AND HEURISTIC SEARCH STRATEGIES IN SOLVING ROUTE OPTIMIZATION PROBLEMS: AN EXPERIMENTAL STUDY

Ing. Harold Ordaz Valdes
Instituto Superior Universitario Bolivariano de Tecnología. Guayaquil, Ecuador.
hordaz@itb.edu.ec

María Magdalena Castro Cañarte, MSc.
Instituto Superior Universitario Bolivariano de Tecnología. Guayaquil, Ecuador.
mcastro@itb.edu.ec

Manuel Eduardo Millán Rodríguez, Mgs.
Instituto Superior Universitario Bolivariano de Tecnología. Guayaquil, Ecuador.
mmillan2@bolivariano.edu.ec

Luisa María Toyo Sánchez, MSc.
Instituto Superior Universitario Bolivariano de Tecnología. Guayaquil, Ecuador.
lmtoyo@itb.edu.ec

RESUMEN

En este estudio se exploran y comparan diferentes estrategias de búsqueda, tanto no informadas como heurísticas, en la resolución de problemas de optimización de rutas. Se evaluaron los algoritmos de Búsqueda en Anchura (BFS), Búsqueda en Profundidad (DFS) y el algoritmo A* con heurísticas de distancia Manhattan y Euclídea. Los experimentos se realizaron en un entorno de simulación que emula mapas (laberintos) de diferentes tamaños y complejidades. Los resultados obtenidos muestran que el algoritmo A*, con ambas heurísticas de distancia Manhattan y Euclídea, fue el óptimo en términos de costo de la solución, superando a BFS y DFS en la mayoría de los escenarios. El algoritmo BFS demostró encontrar la ruta más corta (longitud menor), independientemente del coste de la solución, pero a costa de un mayor tiempo de búsqueda y consumo de memoria. El algoritmo DFS, aunque rápido en encontrar una solución minimizando el gasto computacional, frecuentemente resultó en rutas no óptimas, independientemente del coste de los movimientos dentro del mapa, y no garantizó la optimalidad de la solución.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial 4.0 Internacional .

En resumen, el algoritmo A* con heurísticas admisibles se destaca como la mejor opción para la resolución de problemas de optimización de rutas, ofreciendo un equilibrio óptimo entre eficiencia y costo de la solución.

Palabras clave: Estrategias; Búsqueda no informada; Heurística; Resolución de problemas; Optimización de rutas.

ABSTRACT

In this study, we explore and compare different search strategies, both uninformed and heuristic, in solving route optimization problems. The Breadth-First Search (BFS), Depth-First Search (DFS) and the A* algorithm with Manhattan and Euclidean distance heuristics were evaluated. The experiments were performed in a simulation environment that emulates maps (mazes) of different sizes and complexities. The results obtained show that the A* algorithm, with both Manhattan and Euclidean distance heuristics, was the optimal one in terms of solution cost, outperforming BFS and DFS in most scenarios. The BFS algorithm was shown to find the shortest path (shortest length), regardless of the solution cost, but at the cost of increased search time and memory consumption. The DFS algorithm, although fast in finding a solution by minimizing computational expense, frequently resulted in non-optimal paths, regardless of the cost of movements within the map, and did not guarantee the optimality of the solution. In summary, the A* algorithm with admissible heuristics stands out as the best option for solving route optimization problems, offering an optimal balance between efficiency and cost of the solution.

Key words: Strategies; Uninformed search; Heuristics; Problem solving; Route optimization.

1. INTRODUCCIÓN

En el ámbito de la inteligencia artificial (IA), los agentes inteligentes juegan un papel fundamental. Estos agentes son sistemas autónomos capaces de percibir su entorno, tomar decisiones y realizar acciones para alcanzar objetivos específicos. La capacidad de un agente inteligente para interactuar de manera efectiva con su entorno y adaptarse a cambios dinámicos es crucial para el éxito de muchas aplicaciones de IA, incluyendo la automatización de rutas para la entrega de paquetes.

Un agente inteligente se define como una entidad que percibe su entorno a través de sensores y actúa sobre ese entorno mediante actuadores. Los agentes inteligentes se caracterizan por varias propiedades clave:

- **Autonomía:** Los agentes pueden operar sin intervención humana directa, tomando decisiones basadas en su percepción y conocimientos.
- **Adaptabilidad:** Pueden adaptarse a cambios en el entorno y ajustar sus acciones en consecuencia.
- **Reactividad:** Responden a estímulos en su entorno de manera oportuna y adecuada.

- **Proactividad:** Los agentes pueden tomar la iniciativa para alcanzar sus objetivos, planificando y ejecutando acciones.
- **Comunicación:** En entornos multiagente, los agentes pueden comunicarse y colaborar con otros agentes para lograr objetivos comunes.

Los agentes inteligentes encuentran aplicación en una amplia variedad de dominios, incluyendo la robótica, sistemas de recomendación, la automatización industrial, las finanzas, el sector de la logística y la distribución, entre otros.

En el sector de la logística y la distribución, la optimización de rutas es crucial para mejorar la eficiencia operativa y reducir costos. Las principales razones por las que la optimización de rutas es tan importante, se enuncian a continuación:

- **Reducción de costos operativos,** en consumo de combustibles y el mantenimiento de vehículos del parque motor. Cualquier ahorro en estos aspectos puede tener un impacto considerable en el presupuesto de la empresa.
- **Mejora de la eficiencia del tiempo,** en la entrega puntual a los clientes y en la gestión del tiempo de los conductores evitando demoras innecesarias aumentando su productividad.
- **Incremento de la satisfacción del cliente**, mediante entregas rápidas y confiables y, a su vez, puedan ser flexibles y adaptables a cambios de última hora.
- **Impacto ambiental**, en la reducción de emisiones de CO2 y otros contaminantes al recorrer menor distancia optimiza el uso de combustibles fósiles, prácticas de sostenibilidad que demuestran el compromiso de responsabilidad social corporativa y el cuidado del medio ambiente.
- **Gestión de recursos,** en la utilización eficiente de la flota, cada vehículo esté en uso de la manera más eficiente posible, y la optimización de personal mejorando la programación y la distribución de tareas entre los empleados.

En el contexto de la automatización de rutas de entrega para una empresa de paquetería, los agentes inteligentes, también, son fundamentales para mejorar la eficiencia operativa y reducir costos. Las furgonetas automáticas, actuando como agentes inteligentes, deben ser capaces de encontrar rutas óptimas entre su ubicación y los puntos de recogida y entrega. Para lograr esto, los agentes utilizan diversas estrategias de búsqueda, tanto no informadas como heurísticas, para navegar en el mapa de la ciudad.

Las estrategias de Búsqueda No Informada incluyen algoritmos como Búsqueda en Anchura (BFS) y Búsqueda en Profundidad (DFS), que exploran todas las posibles rutas sin información adicional sobre el entorno. En las estrategias de Búsqueda Heurística se utilizan información adicional, como estimaciones de distancia, para guiar la búsqueda hacia la ruta óptima de manera más eficiente. El algoritmo A* es un ejemplo destacado en esta categoría, combinando heurísticas con la búsqueda de costo uniforme.

Este artículo analiza, mediante varios experimentos, cómo las estrategias de búsqueda no informada (**BFS y DFS**) y heurística (**A* Euclídea y Manhattan**) pueden ser aplicadas para automatizar las rutas de entrega de una flota de furgonetas automáticas en una ciudad y, determinar bajo diferentes entornos (simplificando el problema considerando el espacio dividido en una matriz rectangular) cuál estrategia es más eficiente maximizando su medida de rendimiento.

2. MATERIALES Y MÉTODOS

El problema de buscar caminos entre puntos de un mapa es una tarea común en diversas aplicaciones, como la planificación de rutas para vehículos autónomos, sistemas de navegación GPS, juegos de video y sistemas de logística y distribución. Este problema puede ser modelado usando grafos, donde los puntos en el mapa se representan como nodos y las conexiones entre ellos (caminos, calles, etc.) como aristas.

2.1 Modelo de Grafo

Un grafo es una estructura matemática utilizada para representar relaciones entre objetos. En el contexto de la búsqueda de caminos, un grafo $\boldsymbol{G}$ se define como $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ donde:

- $\boldsymbol{V}$ es un conjunto de nodos (o vértices) que representan los puntos en el mapa.
- $\boldsymbol{E}$ es un conjunto de aristas (o bordes) que representan los caminos entre los puntos.

Cada arista puede tener un peso asociado que representa la distancia, el tiempo de viaje o algún otro costo relevante.

El objetivo del problema es encontrar la ruta más eficiente desde un nodo inicial $\boldsymbol{s}$ hasta un nodo objetivo $\boldsymbol{t}$ en el grafo. La eficiencia de la ruta puede definirse en términos de distancia mínima, tiempo mínimo, costo mínimo, etc.

2.2 Estrategias de Búsqueda no informada e informada

Búsqueda en Anchura (BFS - Breadth-First Search)

Es una estrategia de búsqueda no informada que explora el espacio de estados de manera sistemática y exhaustiva. En BFS, todos los nodos a un nivel de profundidad son explorados antes de pasar a los nodos en el siguiente nivel. Este método asegura que la solución encontrada es la más corta en términos del número de pasos, siempre que todos los pasos tengan el mismo costo. El algoritmo **BFS** utiliza una cola (FIFO - First In, First Out) para mantener el orden de los nodos a explorar.

A continuación, se describe el algoritmo:

I. ***Inicialización***:
   i. Crear una cola vacía y añade el nodo inicial.
   ii. Marcar el nodo inicial como visitado.

II. ***Bucle de exploración***:

iii. Mientras la cola no esté vacía:

1. Sacar el nodo del frente de la cola.
2. Si este nodo es el nodo objetivo, terminar la búsqueda y devolver la solución.
3. De lo contrario, para cada nodo hijo (nodo vecino no visitado) del nodo actual:
   a. Anadir el nodo hijo al final de la cola.
   b. Marcar el nodo hijo como visitado.
4. Si la cola se vacía y no se encuentra el nodo objetivo, la búsqueda termina sin encontrar una solución.

Búsqueda en Profundidad (DFS - Depth-First Search)

Es una estrategia de búsqueda no informada que explora el espacio de estados de manera exhaustiva, profundizando en los nodos hijos antes de retroceder y explorar otros caminos. DFS utiliza una estructura de pila (LIFO - Last In, First Out) para mantener el seguimiento de los nodos que se están explorando. El algoritmo **DFS** puede implementarse de manera recursiva o iterativa.

A continuación, se describe el algoritmo:

1. ***Inicialización***:

i. Crear una pila vacía y añade el nodo inicial.

ii. Marcar el nodo inicial como visitado.

2. ***Bucle de exploración***:

iii. Mientras la pila no esté vacía:

1. Sacar el nodo del tope de la pila.
2. Si este nodo es el nodo objetivo, terminar la búsqueda y devolver la solución.
3. De lo contrario, para cada nodo hijo (nodo vecino no visitado) del nodo actual:
   a. Anadir el nodo hijo al tope de la pila.
   b. Marcar el nodo hijo como visitado.
4. Si la pila se vacía y no se encuentra el nodo objetivo, la búsqueda termina sin encontrar una solución.

Búsqueda A* (A-star Search)

Es una estrategia de búsqueda informada que combina las ventajas de la búsqueda de costo uniforme y la búsqueda heurística para encontrar caminos eficientes en un espacio de estados. A* utiliza una función heurística para guiar la búsqueda hacia el objetivo de manera más eficiente, lo que la hace especialmente adecuada para problemas de planificación de rutas y juegos de estrategia.

El algoritmo A* utiliza una cola de prioridad (normalmente implementada como un heap) para mantener los nodos a explorar, priorizando aquellos que parecen más prometedores en base a una función de costo total $\boldsymbol{f(n) = g(n) + h(n)}$, que es la suma de dos componentes:

- $\boldsymbol{g(n)}$: El costo del camino desde el nodo inicial hasta el nodo n.
- $\boldsymbol{h(n)}$: Una estimación heurística del costo del camino desde el nodo n hasta el nodo objetivo.

A continuación, se describe el algoritmo:

I. ***Inicialización***:
   i. Añadir el nodo inicial a una cola de prioridad (también conocida como "frontera" o "lista abierta") con un costo $\boldsymbol{f}$ a $\boldsymbol{h}$ (ya que $\boldsymbol{g}$ es 0 en el inicio).
   ii. Crear un conjunto vacío para los nodos ya explorados (conocido como "lista cerrada").

II. ***Bucle de exploración***:
   iii. Mientras la cola de prioridad no esté vacía:
      1. Sacar el nodo con el menor valor de $\boldsymbol{f}$ de la cola.
      2. Si este nodo es el nodo objetivo, reconstruir y devolver el camino desde el nodo inicial hasta el nodo objetivo.
      3. De lo contrario, añadir el nodo a la "lista cerrada".
      4. Por cada nodo hijo del nodo actual:
         a. Calcular el valor $\boldsymbol{g}$ y $\boldsymbol{f}$ del nodo hijo.
         b. Si el nodo hijo no está en la "lista cerrada" o si encuentra un camino más barato al nodo hijo, añadirlo a la cola de prioridad y actualizar su valor $\boldsymbol{g}$.

III. **Repetir hasta que se encuentre la solución o la cola de prioridad esté vacía.**

Las heurísticas juegan un papel crucial en la **Búsqueda A*,** ya que guían el algoritmo hacia el objetivo de manera eficiente. Una heurística es una estimación del costo restante desde un nodo actual hasta el nodo objetivo. Para que la **Búsqueda A*** sea óptima, la heurística debe ser admisible, es decir, nunca debe sobreestimar el costo real.

Propiedades de una buena heurística:

- **Admisibilidad**: La heurística nunca sobreestima el costo real desde el nodo actual hasta el nodo objetivo.
- **Consistencia**: Para cada nodo $\boldsymbol{n}$ y sus vecinos $\boldsymbol{n'}$, la heurística cumple la condición $\boldsymbol{h(n) \leq c(n, n') + h(n')}$, donde $\boldsymbol{c(n, n')}$ es el costo del paso desde $\boldsymbol{n}$ hasta $\boldsymbol{n'}$.

Funciones heurísticas

Ejemplos de funciones heurísticas que se pueden aplicar en la **Búsqueda A*** tenemos la Distancia Euclídea, Distancia Manhattan, Distancia Chebyshev y Distancia Hamming.

La **heurística de Distancia Euclídea** es una medida de distancia basada en la geometría euclidiana que se utiliza comúnmente en algoritmos de búsqueda informada, como A*.

Es especialmente útil en entornos donde los movimientos son libres en todas las direcciones, como en un espacio continuo (2D o 3D). Su fórmula matemática es $\boldsymbol{h(n)} = \sqrt{(\boldsymbol{x_{goal}} - \boldsymbol{x_n})^2 + (\boldsymbol{y_{goal}} - \boldsymbol{y_n})^2}$ que calcula la distancia entre dos puntos, sean estos $(\boldsymbol{x_n}, \boldsymbol{y_n})$ ubicación inicial de un nodo $\boldsymbol{n}$ y, $(\boldsymbol{x_{goal}}, \boldsymbol{y_{goal}})$ ubicación del nodo destino u objetivo, para un plano bidimensional.

La **heurística de Manhattan**, también conocida como la distancia de Manhattan o distancia de bloques, es una medida de distancia utilizada comúnmente en algoritmos de búsqueda informada, como A* (A-star), en entornos cuadriculados. Es especialmente útil cuando los movimientos están restringidos a direcciones ortogonales (horizontales y verticales), como en una cuadrícula de calles de una ciudad. Su fórmula matemática es $\boldsymbol{h(n)} = |\boldsymbol{x_{goal}} - \boldsymbol{x_n}| + |\boldsymbol{y_{goal}} + \boldsymbol{y_n}|$ donde, $(\boldsymbol{x_n}, \boldsymbol{y_n})$ es la ubicación inicial de un nodo $\boldsymbol{n}$ y, $(\boldsymbol{x_{goal}}, \boldsymbol{y_{goal}})$ ubicación del nodo destino u objetivo.

2.3 Herramientas y entornos de programación

Las herramientas de programación y entornos de ejecución para computadoras utilizados, los que a continuación se describen:

- **Lenguaje de programación *Python***, sitio oficial de la documentación en https://www.python.org/doc/
- **Kit de desarrollo de software (*SDK*) para Python**, sitio oficial para descargar en https://www.python.org/downloads/ y posteriormente instalar.
- **Entorno de Desarrollo Integrado (IDE) de Visual Studio Code**, sitio oficial para descargar desde https://code.visualstudio.com/ y posteriormente instalar.
- ***Extensión* de Python para Visual Studio Code,** sitio oficial para su instalación en https://marketplace.visualstudio.com/items?itemName=ms-python.python
- **Biblioteca *simpleai* para Python** la cual implementa muchos de los algoritmos de inteligencia artificial descritos en el libro “Artificial Intelligence, a Modern Approach", de Stuart Russel y Peter Norvig”, específicamente los algoritmos de interés para la experimentación ***BFS***, ***DFS*** y ***A**** (heurísticas Euclídea y Manthattan). Sitio oficial de la documentación en https://simpleai.readthedocs.io/en/latest/index.html

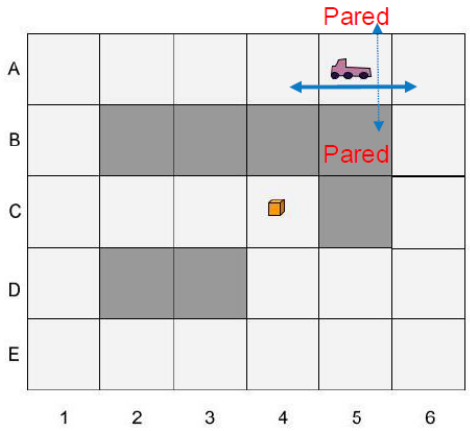
2.4 Descripción del problema

Una empresa de paquetería desea automatizar sus operaciones. Una de las tareas de la flota es entregar una serie de paquetes en distintos puntos de la ciudad. Para ello sus furgonetas automáticas tienen que ser capaces de encontrar caminos entre su ubicación y los puntos de recogida, conociendo el mapa de la ciudad. Simplificamos el problema considerando el espacio dividido en una matriz rectangular, de modo que una furgoneta estará situada en una ubicación identificada por sus coordenadas. La furgoneta puede moverse en sentido horizontal y vertical. El objetivo es llegar hasta la ubicación del paquete.

El mapa base del problema en que se trabajó es el indicado en la figura 1. Las localizaciones se indican mediante filas y columnas: por ejemplo, la furgoneta empieza en A5, y el paquete en C4.

Figura 1

Mapa base para los experimentos, estado inicial A5 y final C4.



2.5 Problema de búsqueda

El problema de búsqueda se define a continuación:

- El estado inicial es el de la Figura 1 y, al final, consiste en que la furgoneta esté en la posición del paquete. El mapa se representa en casillas con paredes que no se pueden atravesar.
- Las acciones permitidas son los movimientos de una casilla en dirección horizontal o vertical (no diagonal).
- El coste del movimiento de la furgoneta es 1 por casilla (experimentos 1, 2 y 4) o bien con una modificación (experimento 3).
- Para A*, en todos los casos se probó como función heurística la distancia Manhattan y distancia Euclídea.

Se realizaron 4 experimentales para comparar los resultados numéricos y la solución obtenida. Se aporta la discusión sobre los experimentos en cuanto a si se obtiene o no la solución de optimalidad, y como varían los resultados entre los diferentes experimentos:

- **Experimento 1**, caso base: El mapa, estado inicial y final de la Figura 1.
- **Experimento 2**: El estado inicial modificado en el que el algoritmo de búsqueda en profundidad obtenga la solución óptima expandiendo menos nodos que el resto de los algoritmos de búsqueda.
- **Experimento 3**: Estado inicial y final de la Figura 1 (caso base), pero cambiando el coste del movimiento hacia arriba a 5, el resto de los movimientos se mantiene en 1(abajo, izquierda y derecha).

- **Experimento 4**: El mapa, estado inicial y estado final modificados, con coste unitario por acción (mismo del caso base) para identificar diferencias entre las dos heurísticas del algoritmo de búsqueda informada A*.

En cada experimento se compararon los algoritmos de búsqueda seleccionados mediante las métricas de evaluación cuantitativas de longitud, coste de la solución, número de nodos expandidos o explorados hasta encontrar la ruta óptima, además de, métricas de evaluación cualitativa como completitud, optimalidad y eficiencia.

3. RESULTADOS

Experimento 1:

Consiste en comparar los resultados de aplicar los algoritmos de **BFS, DFS y A*** (heurísticas Euclídea y Manthattan). El mapa con los estados inicial (cuadrícula A5) y final (cuadrícula C4) es el definido por la variable MAP y los costos de movimientos (arriba, abajo, derecha e izquierda con valor 1 todos) por la variable COSTS. Para ver las salidas de los respectivos algoritmos de búsqueda con Python ir al **Anexo I.**

Tabla 1

Resumen de resultados de las métricas de evaluación cuantitativas

Algoritmo	Solución	Longitud	Coste solución	Nodos expandidos	Max. Abierta
BFS	A5 -> A6 -> B6 -> C6 -> D6 -> D5 -> D4 ->C4	8	7	17	4
DFS	A5 -> A4 -> A3 -> A2 -> A1 -> B1 -> C1 -> C2 -> C3 -> C4	10	9	10	3
A* Euclídea	A5 -> A6 -> B6 -> C6 -> D6 -> D5 -> D4 ->C4	8	7	11	5
A* Manhattan	A5 -> A6 -> B6 -> C6 -> D6 -> D5 -> D4 ->C4	8	7	11	5

Experimento 2:

Consiste en modificar el estado inicial del mapa del Experimento 1 para que el algoritmo ***DFS*** obtenga la solución óptima expandiendo menos nodos que el resto de los algoritmos. Se mantienen los costos de movimientos Se aplicaron varias modificaciones en la ubicación del transporte dentro del mapa y, luego de varias pruebas se observó que el algoritmo ***BFS*** siempre trata de iniciar, en movimientos verticales, primero arriba y después abajo y, por último, en movimiento horizontales primero a la derecha y luego a la izquierda. En cambio, el algoritmo ***DFS*** prioriza los movimientos en horizontal, primero a la izquierda y luego a la derecha y, por último, movimientos en la vertical primero abajo y luego arriba. Por las razones anteriormente explicadas determinamos el mapa

actual que nos permite concluir el experimento con éxito. Para ver las salidas de los respectivos algoritmos de búsqueda con Python ir al **Anexo II**.

Figura 2

Mapa modificado del experimento 2, estado inicial A4 y final C4.

Tabla 2

Resumen de resultados de las métricas de evaluación cuantitativas

Algoritmo	Solución	Longitud	Coste solución	Nodos expandidos	Max. Abierta
BFS	A4 -> A5 -> A6 -> B6 -> C6 -> D6 -> D5 -> D4 -> C4	9	8	21	4
DFS	A4 -> A3 -> A2 -> A1 -> B1 -> C1 -> C2 -> C3 -> C4	9	8	9	3
A* Euclídea	A4 -> A5 -> A6 -> B6 -> C6 -> D6 -> D5 -> D4 -> C4	9	8	15	6
A* Manhattan	A4 -> A5 -> A6 -> B6 -> C6 -> D6 -> D5 -> D4 -> C4	9	8	15	6

Experimento 3:

Tomando como mapa el caso base Experimento 1 sin modificar el estado inicial (transporte) y estado objetivo (paquete), únicamente se modifica el costo del movimiento hacia arriba, entre las cuadrículas, con valor de 5. Los restantes movimientos (derecha, izquierda y abajo) se mantiene con costo de 1. Para ver las salidas de los respectivos algoritmos de búsqueda con Python ir al **Anexo III**.

Tabla 3

Resumen de resultados de las métricas de evaluación cuantitativas

Algoritmo	Solución	Longitud	Coste solución	Nodos expandidos	Max. Abierta
BFS	A5 -> A6 -> B6 -> C6 -> D6 -> D5 -> D4 -> C4	8	11	17	4
DFS	A5 -> A4 -> A3 -> A2 -> A1 -> B1 -> C1 -> C2 -> C3 -> C4	10	9	10	3
A* Euclídea	A5 -> A4 -> A3 -> A2 -> A1 -> B1 -> C1 -> C2 -> C3 -> C4	10	9	19	5
A* Manhattan	A5 -> A4 -> A3 -> A2 -> A1 -> B1 -> C1 -> C2 -> C3 -> C4	10	9	19	5

Experimento 4:

Se amplió el tamaño del mapa tanto en ancho como en altura. Se modificaron las posiciones del transporte (estado inicial) y del paquete (estado objetivo), además se incorporaron más parades u obstáculos. Los costos de movimientos horizontales y verticales se mantienen en valor de 1 como en el caso base Experimento 1. En este experimento se diseñaron dos mapas distintos, caso (4.1) y caso (4.2). Al mapa caso (4.2) se le disminuyen los obstáculos en comparación al caso (4.1), permitiendo así, establecer una mejor comparación entre las dos heurísticas del algoritmo de Búsqueda informada **A***: **Distancia Euclídea** y **Distancia Manhattan**. Para ver las salidas de los respectivos algoritmos de búsqueda con Python ir al **Anexo IV.1** y **IV.2** respectivamente.

Tabla 4

Resumen de resultados de las métricas de evaluación cuantitativas del caso (4.1)

Algoritmo	Solución	Longitud	Coste solución	Nodos expandidos	Max. Abierta
A* Euclídea	Si encuentra el camino	38	37	63	18
A* Manhattan	Si encuentra el camino	38	37	62	18

Tabla 5

Resumen de resultados de las métricas de evaluación cuantitativas del caso (4.2)

Algoritmo	Solución	Longitud	Coste solución	Nodos expandidos	Max. Abierta
A* Euclídea	Si encuentra el camino	36	35	94	26
A* Manhattan	Si encuentra el camino	36	35	69	30

4. DISCUSIÓN

Para interpretar correctamente los resultados se tuvo en cuenta las siguientes consideraciones:

- Entender bien la naturaleza de los algoritmos de búsqueda (**BFS, DFS y A***):
    - **Qué partes del algoritmo dependen del problema:** las funciones que generan los sucesores, la función que detecta la meta, la función que calcula la heurística y la que calcula el coste.
    - Un algoritmo se programa sin tener en cuenta un problema u otro. Por ejemplo, hay que definir un orden entre operadores que puede ser bueno o malo, pero sin heurística no lo sabemos.
    - Entender cómo funciona la heurística y justificar si una heurística es admisible o no.
- Comparar los algoritmos de búsqueda (**BFS, DFS y A***) entre sí:
    - Distinguir entre **completo** (encuentra solución si la hay), **óptimo** (encuentra la solución con menor coste, o número de acciones si no hay coste) y **eficiente** (expande pocos nodos o tarda poco tiempo).
    - Comparar numéricamente los valores relevantes.

Respecto a los resultados obtenidos del **Experimento 1** podemos concluir que los 4 algoritmos encuentran la solución de búsqueda de una ruta; esto no querría decir automáticamente que sean completos, pero sabemos que todos ellos lo son (porque evitan ciclos y mantienen la lista abierta para hacer backtracking). El algoritmo ***DFS*** no es óptimo (no encuentra la mejor solución) con un coste de 9 (de la rama solución de su árbol de expansión), contra un coste de 7 en los restantes algoritmos (***BFS y A****, este último con cualquiera de las dos heurísticas); es decir hay tres que encuentran el óptimo, pero esto no quiere decir que estos algoritmos "sean" óptimos siempre, porque esto no está garantizado (sabemos por la teoría que en este experimento estos tres lo son). Sin embargo, el algoritmo de ***DFS*** reduce notablemente el gasto computacional con la menor complejidad en tiempo (menos iteraciones) y en espacio (menor cantidad de nodos expandidos).

En este sentido el algoritmo de mayor gasto computacional fue ***BFS***. El algoritmo A*, con cualquiera de las heurísticas, expande menos nodos que BFS, por lo tanto, es más eficiente; el motivo es que descarta explorar el camino de la izquierda (la función heurística suma una cantidad creciente, y eso hace que sea peor ir en esa dirección que ir a la derecha).

Respecto a los resultados obtenidos del **Experimento 2** podemos concluir que se encontró un escenario favorable en que DFS sea el más eficiente, es decir, encuentra la misma solución de coste 8, pero expandiendo menos nodos (9 contra 21 de BFS y 15 de A*). Bastó con mover el punto de inicio un poco a la izquierda (casilla A4 del mapa), y el camino que encuentra pasa a ser el óptimo, previo a entender la naturaleza de dicho algoritmo.

Respecto a los resultados obtenidos del **Experimento 3** podemos concluir que el algoritmo de ***BFS*** obtiene la misma solución del experimento (1), es decir, la menor longitud, pero con el mayor costo en la solución debido a que le es indiferente el costo de 5 del movimiento hacia arriba último (para encontrar el objetivo), es decir no optimiza al mínimo posible el costo de movimientos, por tanto, no cumple con la característica de optimalidad. El algoritmo informado ***A****, con sus dos heurísticas, si cambió su solución en comparación al experimento base, priorizando minimizar el coste de los movimientos de la ruta a encontrar por encima de la longitud mínima posible de la misma. Sin embargo, coincide (con el experimento 2) que el algoritmo ganador fue ***DFS***, obtuvo los mismos resultados del algoritmo ***A****, pero mejorando el gasto computacional en tiempo y espacio, ya que expande menos nodos (producto de su naturaleza en el orden de expansión de sucesores del nodo visitado y orden de apilado de nodos expandidos en la frontera). Como se mencionó en la idea anterior, el algoritmo ***A**** también encuentra el camino óptimo porque la heurística es admisible (ambas lo son); para conseguirlo tiene que expandir muchos más nodos ya que, por su naturaleza, el camino hacia la izquierda no puede descartarse. Por último, es importante mencionar que los dos algoritmos de búsqueda no informada (***BFS y DFS***) no tienen en cuenta el factor costo variable de los movimientos, es decir, no buscan minimizar el costo en movimientos de la solución final, sino la longitud del camino solución dentro del árbol de expansión generado en el proceso de búsqueda, debido a sus respectivas particularidades de expansión de nodos y como añadir a la lista de fronteras.

Respecto a los resultados obtenidos del **Experimento 4** podemos concluir que el camino óptimo es el resuelto aplicando la heurística Distancia de Manhattan (distancia taxi). En cada experimento (4.1) y (4.2), las longitudes y coste de la solución encontrada, son iguales en ambas heurísticas. La cantidad de nodos expandidos en el primer experimento (4.1) es ligera la diferencia si aumentamos los obstáculos del mapa, pero en el caso del experimento (4.2) donde se facilita el camino con menos obstáculos, notablemente la heurística Distancia Manhattan toma la ventaja expandiendo menos nodos. En resumen, ambas heurísticas son capaces de encontrar la solución de optimalidad, pero la heurística Distancia de Manhattan obtiene el menor gasto computacional en tiempo y espacio, por tanto, es más eficiente.

5. CONCLUSIONES

A partir de los análisis de resultados experimentales obtenidos, de comparación entre los algoritmos de búsqueda no informada (***BFS y DFS***) e informada con heurísticas (***A**** con distancia ***Euclídea y Manthattan***), podemos concluir que, El algoritmo ***BFS*** trata de encontrar el óptimo en cuanto a la longitud de la solución obtenida y expande muchos nodos. El algoritmo ***DFS*** trata de ser más eficiente en minimizar la cantidad de nodos expandidos (menor gasto computacional) para la solución a encontrar (no tiene que ser necesariamente la óptima en longitud). Los algoritmos BFS y DFS no buscan optimizar el costo de la solución final en caso de que los movimientos, dentro del mapa, tengan costes no uniformes.

El algoritmo ***A**** encuentra el óptimo en costes de la solución tanto para costes uniformes o no uniformes de los movimientos posibles dentro del mapa. Para costes uniformes de los movimientos del mapa busca optimizar la cantidad de nodos expandidos, pero para costes no uniformes, de dichos movimientos, expande muchos más nodos que ***BFS y DFS*** buscando encontrar la solución de coste óptimo.

La heurística Manthattan y Euclídea encuentran la solución de optimalidad deseada, pero la Manthattan es más informada que Euclídea porque genera un gasto computacional menor al expandir menos nodos, es decir, el algoritmo ***A**** con heurística Manthattan es más eficiente que con Euclídea. Las estrategias heurísticas anteriormente mencionadas, particularmente el algoritmo A*, son más eficientes que las búsquedas no informadas para la optimización de problemas de ruta.

6. REFERENCIAS BIBLIOGRÁFICAS

Ghallab, M., Nau, D., & Traverso, P. (2004). Automated planning: Theory and practice. Morgan Kaufmann.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100-107. https://doi.org/10.1109/TSSC.1968.300136

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence, 27(1), 97-109. https://doi.org/10.1016/0004-3702(85)90084-0

Nilsson, N. J. (1980). Principles of Artificial Intelligence. Morgan Kaufmann.

Pearl, J. (1984). Heuristics: Intelligent search strategies for computer problem solving. Addison-Wesley.

Russell, S. J., & Norvig, P. (2020). Artificial intelligence: A modern approach (4th ed.). Pearson.

7. ANEXOS

Anexo I

Datos de entrada del algoritmo de Python:

```
MAP = """
########
#    T #
# #### #
#   P# #
# ##   #
#      #
########
"""
MAP = [list(x) for x in MAP.split("\n") if x]

COSTS = {
    "up": 1.0,
    "down": 1.0,
    "right": 1.0,
    "left": 1.0,
}
```

Salidas de resultados de correr las funciones (librería simpleai) <<breadth_first>> (BFS), <<depth_first>> (DFS) y <<astar>> (A *):

BFS	DFS	A* Euclídea	A* Manthattan
######## # T·# # ####·# # P#·# # ##···# # # ######## Total length of solution: 8 Total cost of solution: 7.0 max fringe size: 4 visited nodes: 17 iterations: 17	######## #····T # #·#### # #···P# # # ## # # # ######## Total length of solution: 10 Total cost of solution: 9.0 max fringe size: 3 visited nodes: 10 iterations: 10	######## # T·# # ####·# # P#·# # ##···# # # ######## Total length of solution: 8 Total cost of solution: 7.0 max fringe size: 5 visited nodes: 11 iterations: 11	######## # T·# # ####·# # P#·# # ##···# # # ######## Total length of solution: 8 Total cost of solution: 7.0 max fringe size: 5 visited nodes: 11 iterations: 11

Anexo II

Datos de entrada del algoritmo de Python:

```
MAP = """
########
#    T  #
# #### #
#    P# #
# ##    #
#       #
########
"""
MAP = [list(x) for x in MAP.split("\n") if x]

COSTS = {
    "up": 1.0,
    "down": 1.0,
    "right": 1.0,
    "left": 1.0,
}
```

Salidas de resultados de correr las funciones (librería simpleai) <<breadth_first>> (BFS), <<depth_first>> (DFS) y <<astar>> (A *):

BFS	DFS	A* Euclídea	A* Manthattan
######## # T··# # ####·# # P#·# # ##···# # # ######## Total length of solution: 9 Total cost of solution: 8.0 max fringe size: 4 visited nodes: 21 iterations: 21	######## #···T # #·#### # #···P# # # ## # # # ######## Total length of solution: 9 Total cost of solution: 8.0 max fringe size: 3 visited nodes: 9 iterations: 9	######## # T··# # ####·# # P#·# # ##···# # # ######## Total length of solution: 9 Total cost of solution: 8.0 max fringe size: 6 visited nodes: 15 iterations: 15	######## # T··# # ####·# # P#·# # ##···# # # ######## Total length of solution: 9 Total cost of solution: 8.0 max fringe size: 6 visited nodes: 15 iterations: 15

Anexo III

Datos de entrada del algoritmo de Python:

```
MAP = """
########
#    T #
# #### #
#   P# #
# ##   #
#      #
########
"""
MAP = [list(x) for x in MAP.split("\n") if x]

COSTS = {
    "up": 5.0, #costo del movimiento hacia arriba modificado a 5
    "down": 1.0,
    "right": 1.0,
    "left": 1.0,
}
```

Salidas de resultados de correr las funciones (librería simpleai) <<breadth_first>> (BFS), <<depth_first>> (DFS) y <<astar>> (A *):

BFS	DFS	A* Euclídea	A* Manthattan
######## # T·# # ####·# # P#·# # ##···# # # ######## Total length of solution: 8 Total cost of solution: 11.0 max fringe size: 4 visited nodes: 17 iterations: 17	######## #····T # #·#### # #···P# # # ## # # # ######## Total length of solution: 10 Total cost of solution: 9.0 max fringe size: 3 visited nodes: 10 iterations: 10	######## #····T # #·#### # #···P# # # ## # # # ######## Total length of solution: 10 Total cost of solution: 9.0 max fringe size: 5 visited nodes: 19 iterations: 19	######## #····T # #·#### # #···P# # # ## # # # ######## Total length of solution: 10 Total cost of solution: 9.0 max fringe size: 5 visited nodes: 19 iterations: 19

Anexo IV 1

Datos de entrada del algoritmo de Python:

```
MAP = """
##################################
#T            #          #      #
######    ########       #      #
#    #           #       #   #  #
#  ######     ####   ######  #  #
#         ####      #         #  #
#            #  #   #   #### #  #
#     ######    #   #   #       #
#                               P#
##################################
"""
MAP = [list(x) for x in MAP.split("\n") if x]

COSTS = {
    "up": 1.0,
    "down": 1.0,
    "right": 1.0,
    "left": 1.0,
}
```

Salidas de resultados de correr las funciones (librería ***simpleai***) <<astar>> (***A ****):

A* Euclídea	A* Manthattan

<table>
<tr>
<td>
<pre>
##################################
#T·····   #   #   #
·########   #
·   #   #   #
######·   ####   ###### #
·####   #   #
····# #   #   #### #
######···#   #   #
···············P#
##################################
Total length of solution: 38
Total cost of solution: 37.0
max fringe size: 18
visited nodes: 63
iterations: 63
</pre>
</td>
<td>
<pre>
##################################
#T·····   #   #   #
######·   ########   #   #
#····   #   #   #
######·   ####   ###### #
·####   #   #
····# #   #   #### #
######·   #   #   #
·················P#
##################################
Total length of solution: 38
Total cost of solution: 37.0
max fringe size: 18
visited nodes: 62
iterations: 62
</pre>
</td>
</tr>
</table>

Anexo IV.2

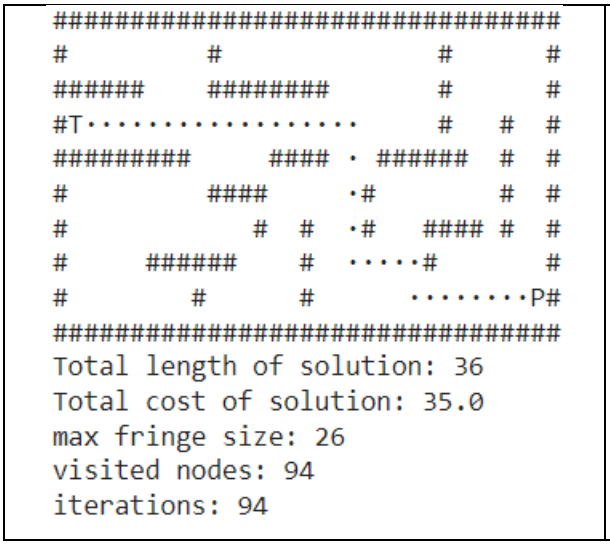
Datos de entrada del algoritmo de Python:

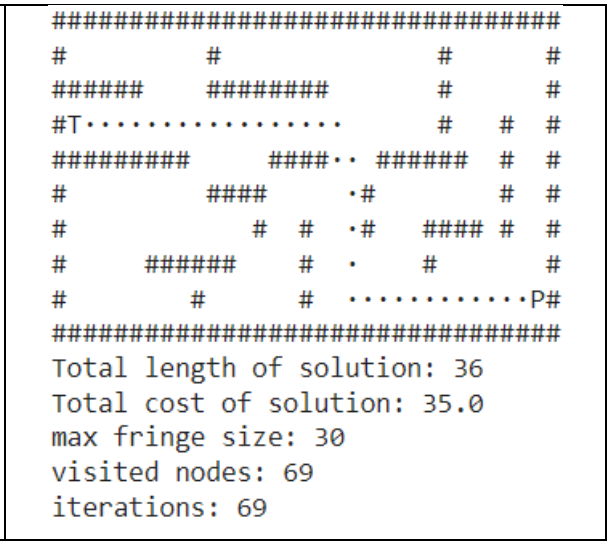
```
MAP = """
##################################
#         #              #      #
######    ########       #      #
#T                       #   #  #
#########     ####   ######  #  #
#         ####      #        #  #
#            #  #   #   #### #  #
#     ######    #       #       #
#        #      #             P#
##################################
"""
MAP = [list(x) for x in MAP.split("\n") if x]

COSTS = {
    "up": 1.0,
    "down": 1.0,
    "right": 1.0,
    "left": 1.0,
}
```

Salidas de resultados de correr las funciones (librería simpleai) <<astar>> (A *):

A* Euclídea	**A* Manthattan**


Total length of solution: 36
Total cost of solution: 35.0
max fringe size: 26
visited nodes: 94
iterations: 94


Total length of solution: 36
Total cost of solution: 35.0
max fringe size: 30
visited nodes: 69
iterations: 69