FACULTAD DE CIENCIAS
INTELIGENCIA ARTIFICIAL

Algoritmo A^*

Equipo: `Skynet Scribes`

Número de practica: `04`

Sarah Sophía Olivares García
318360638

Marco Silva Huerta
316205326

Carlos Daniel Cortés Jiménez
420004846

Laura Itzel Tinoco Miguel
316020189

Luis Enrique García Gómez
315063880

Fernando Mendoza Eslava
319097690

Profesora: Cecilia Reyes Peña
Ayudante teoría: Karem Ramos Calpulalpan
Ayudante laboratorio: Tania Michelle Rubí Rojas

Semestre 2024-2

Fecha de entrega:
13 de Marzo del 2024

1. Algoritmo A^*

Introducción

El algoritmo A^*, concebido en 1968 por Peter Hart, Nils Nilsson, y Bertram Raphael, se erige como un pilar en la búsqueda de caminos dentro del vasto dominio de la inteligencia artificial. Este algoritmo trasciende la simpleza de métodos tradicionales, como la Búsqueda en Anchura (BFS) y la Búsqueda en Profundidad (DFS), mediante la incorporación de una función heurística. Esta heurística orienta la exploración hacia el objetivo de manera eficiente, optimizando el trayecto en términos de coste y distancia. Su versatilidad le permite adaptarse a una amplia gama de aplicaciones, desde la conducción autónoma y la robótica hasta la generación de rutas en videojuegos y aplicaciones de mapeo geográfico.

¿Cómo funciona?

Algoritmo A^* vs BFS

A diferencia de la Búsqueda en Anchura (BFS), que adopta un enfoque no ponderado y equitativo en su exploración expandiendo todos los nodos vecinos con igual prioridad, el algoritmo A^* introduce una estrategia de búsqueda ponderada. Esta estrategia se centra en minimizar una función de coste $f(n) = g(n) + h(n)$, donde $g(n)$ representa el coste exacto desde el nodo inicial hasta el nodo n, y $h(n)$ es la heurística que estima el coste mínimo desde n hasta el objetivo. Esta dualidad permite a A^* explorar de forma selectiva aquellos caminos que parecen más prometedores, reduciendo significativamente el volumen de cálculo y garantizando la identificación del trayecto óptimo.

Ventajas

- **Optimalidad y Complejidad:** A^* garantiza encontrar la ruta más corta hacia el objetivo, siempre que la heurística empleada sea admisible y consistente. Esta característica lo distingue como un algoritmo de búsqueda óptima.
- **Eficiencia:** Al priorizar nodos basándose en el coste total estimado $f(n)$, A^* es capaz de descartar rutas menos prometedoras de manera precoz, agilizando la consecución de su meta.
- **Flexibilidad Heurística:** La posibilidad de adaptar la heurística $h(n)$ según las particularidades del problema permite una optimización específica del rendimiento de búsqueda.

Desventajas

- **Dependencia Heurística:** La eficacia del algoritmo está intrínsecamente ligada a la calidad de la heurística $h(n)$. Una heurística pobre puede resultar en una exploración ineficiente y un mayor consumo de recursos.
- **Requerimientos de Memoria:** El mantenimiento de las listas abierta y cerrada, especialmente en espacios de búsqueda vastos, puede conducir a un elevado consumo de memoria, superando en ocasiones a alternativas más simples como BFS.

2. Distancia Manhattan

La *distancia de Manhattan* es una métrica que mide la distancia entre dos puntos en un espacio euclidiano multidimensional, y esta basada en líneas paralelas a los ejes de coordenadas en lugar de una línea recta, como puede ser un plano cartesiano, se emplea en inteligencia artificial y diversos campos (que incluyen la teoría de grafos, geometría computacional y optimización).

En el ámbito de la inteligencia artificial, se usa con frecuencia en algoritmos heurísticos como A^* (A-estrella) para resolver problemas de búsqueda de caminos. El uso principal de esta medida es en situaciones donde el desplazamiento entre dos ubicaciones se limita a movimientos verticales u horizontales, como sucede en la planificación de rutas.

La razón por la que se llama *Manhattan* a esta distancia es porque, en una cuadrícula de calles de ciudades como Manhattan, para ir de un punto a otro hay que moverse siguiendo las manzanas delimitadas por las calles horizontales y verticales en ángulos rectos. Entonces la distancia de *Manhattan* es una medida básica en inteligencia artificial, especialmente para resolver problemas de búsqueda de rutas y planificación, gracias a su simpleza y eficacia en escenarios con movimientos restringidos a líneas rectas horizontales y verticales.

Planear rutas en entornos urbanos, especialmente en ciudades con calles ortogonales como Nueva York, es un ejemplo de uso real de la distancia de Manhattan como heurística. Por ejemplo: Se puede usar en una app de mapas para hallar la ruta más corta desde donde estás hasta un punto específico en una ciudad con calles que se cruzan formando cuadras. *distancia de Manhattan* puede servir como una guía para calcular la distancia que queda hasta llegar a tu destino.

O imagina que estás manejando en tu carro y tienes que ir desde un cruce A hasta otro cruce B en una ciudad con calles dispuestas en forma de cuadrícula. ¿Puedes determinar la *distancia de Manhattan* entre A y B sumando el número de cuadras que necesitas recorrer en dirección horizontal y vertical para llegar de A a B, sin considerar los obstáculos como edificios o restricciones de tráfico?

La distancia de Manhattan proporciona una estimación precisa del recorrido necesario en automóvil, ya que las rutas siguen generalmente un patrón de calles y avenidas en ángulos rectos. A pesar de que en la práctica puedan existir variaciones debido a las limitaciones del tráfico o la disposición exacta de las calles, la distancia de Manhattan aún es una buena aproximación y puede ayudar a los sistemas de mapas para calcular rutas eficientes.

Así que, la distancia de Manhattan es empleada en aplicaciones de mapas y navegación como una heurística para estimar el tiempo y la distancia restante hasta un destino en entornos urbanos donde las calles siguen una disposición de cuadrícula ortogonal. Ayuda a los conductores a tomar decisiones informadas sobre sus rutas y permite que las aplicaciones de navegación calculen rutas óptimas de manera más eficiente.

La fórmula para calcular la distancia de Manhattan entre dos puntos $A(x_1, y_1)$ y $B(x_2, y_2)$, en un espacio bidimensional es:

$$\mathrm{D} = |x_1 - x_2| + |y_1 - y_2|$$

Donde $|x_1 - x_2|$ representa la diferencia en la coordenada x entre los dos puntos y $|y_1 - y_2|$ representa la diferencia en la coordenada y entre los dos puntos.

En pocas palabras la elección de la distancia Manhattan como heurística en la implementación de A^* se fundamenta en su capacidad para estimar costes de manera coherente y admisible en entornos cuadriculados, donde los movimientos están restringidos a las direcciones cardinales. Esta simplicidad y eficacia la convierten en una heurística ideal para muchos escenarios de búsqueda en laberintos y grillas.

Pseudocódigo

```
función distancia_manhattan(nodo_actual, nodo_objetivo):
    diferencia_x = abs(nodo_actual.x - nodo_objetivo.x)
    diferencia_y = abs(nodo_actual.y - nodo_objetivo.y)
    return diferencia_x + diferencia_y
```

3. Otras heurísticas

Otros dos ejemplos de heurísticas que se pueden usar en el algoritmo A^* son:

- Distancia Euclidiana: Se utiliza como métrica para calcular la separación entre puntos en un espacio euclidiano, ya sea en el plano o en el espacio tridimensional. Se utiliza el teorema de Pitágoras para calcular la longitud del segmento de línea recta que une los dos puntos. La distancia euclidiana es una heurística admisible, no obstante, puede no ser uniforme en todos los casos, especialmente en ambientes con obstáculos donde la ruta más directa puede no ser una línea recta.

  La fórmula para la distancia euclidiana entre dos puntos $A(x_1, y_1)$ y $B(x_2, y_2)$, en un espacio bidimensional es:

$$\text{D} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Distancia de Chebyshev:

  - La distancia de Chebyshev se usa como métrica para calcular la distancia entre dos puntos en un espacio de rejilla, donde el movimiento está permitido en todas las direcciones (horizontal, vertical y diagonal).
  - La distancia de Chebyshev en entornos de rejilla es una heurística que cumple con la propiedad de consistencia, lo que la hace admisible.
  - Estas reglas proporcionan estimaciones del costo restante para alcanzar el objetivo desde cualquier nodo dado en un grafo de búsqueda, lo que asiste al algoritmo A^* a dirigir su búsqueda hacia el objetivo de forma más eficaz. Depende del problema específico y de las características del espacio de búsqueda la elección de la heurística.

  La fórmula para la distancia euclidiana entre dos puntos $A(x_1, y_1)$ y $B(x_2, y_2)$, en un espacio bidimensional se calcula como la máxima de las diferencias absolutas en las coordenadas *x* y *y*

$$\text{D} = max(|x_2 - x_1|, |y_2 - y_1|)$$

4. Documentación

Pseudocódigo A^*

Es importante mencionar que el algoritmo que la ayudante Tania presentó en su clase grabada fue el que utilizamos para implementar el algoritmo.

Entrada: Posición inicial, posición objetivo

Salida: Camino desde la posición inicial hasta la posición objetivo

Variables:

`lista_cerrada`: Lista de nodos ya examinados
`lista_abierta`: Lista de nodos por examinar
`nodo_actual`: Nodo actual que se está evaluando
`vecino`: Cada uno de los vecinos del nodo actual

Algoritmo:

1. **Inicializar:**
   - `lista_cerrada` vacía
   - Agregar nodo inicial a `lista_abierta`
2. **Mientras** `lista_abierta` no esté vacía:
   a) **Obtener** *nodo_actual* con el menor valor de f de `lista_abierta`
   b) **Eliminar** `nodo_actual` de `lista_abierta`
   c) **Agregar** `nodo_actual` a `lista_cerrada`
   d) **Obtener vecinos** de `nodo_actual`
   e) **Para cada vecino:**
   - **Si** vecino es la posición objetivo:
     - **Retornar camino** desde `nodo_actual`
   - **Calcular** g del vecino
   - **Calcular** h del vecino
   - **Calcular** f del vecino
   - **Si** vecino ya está en `lista_abierta` y f del vecino en `lista_abierta` es mayor:
     - **Actualizar** f del vecino en `lista_abierta`
     - **Actualizar** padre del vecino en `lista_abierta`
   - **Si** vecino ya está en `lista_cerrada` y f del vecino en `lista_cerrada` es mayor:
     - **Eliminar** vecino de `lista_cerrada`
     - **Agregar** vecino a `lista_abierta`
     - **Actualizar** padre del vecino
   - **Si** vecino no está en ninguna lista:
     - **Agregar** vecino a `lista_abierta`
     - **Actualizar** padre del vecino

Fin del algoritmo

5. Resultados obtenidos

Ejemplos usados

Observaciones

Referencias

[HNR68] Peter E Hart, Nils J Nilsson y Bertram Raphael. «A formal basis for the heuristic determination of minimum cost paths». En: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), págs. 100-107. URL: https://ieeexplore.ieee.org/document/4082128 (visitado 05-03-2024).

[RN16] Stuart Russell y Peter Norvig. *Inteligencia Artificial Un Enfoque moderno*. 2nd. Pearson Prentice Hall, 2016.

[Ecu] EcuRed. *Algoritmo de Búsqueda Heurística A**. URL: https://www.ecured.cu/Algoritmo_de_B%C3%BAsqueda_Heur%C3%ADstica_A* (visitado 05-03-2024).

[Gee] GeeksforGeeks. *A* Search Algorithm*. URL: https://www.geeksforgeeks.org/a-search-algorithm/ (visitado 05-03-2024).

[jar] jariasf. *ALGORITMO DE BÚSQUEDA: BREADTH FIRST SEARCH*. URL: https://jariasf.wordpress.com/2012/02/27/algoritmo-de-busqueda-breadth-first-search/.

[UNAa] TECNOLOGIA EDUCATIVA UNAN. *Algoritmo A estrellas*. URL: https://aeia.home.blog/algoritmo-a-estrellas-a/ (visitado 11-03-2024).

[UNAb] TECNOLOGIA EDUCATIVA UNAN. *Heuristic Search Strategies*. URL: https://www.cs.unm.edu/~luger/ai-final2/4-BeyondSearch/4-Beyond_search.html (visitado 05-03-2024).