UNIVERSIDAD CENTROCCIDENTA

“LISANDRO ALVARADO”

DECANATO DE CIENCIAS YTECNOLOGÍA

Resumen Ejecutivo: Implementando A Epsilon

Introducción a la Inteligencia Artificial

Sección 1

V-25.401.656, Colmenárez Luis

Profesora: Pérez María Auxiliadora

Barquisimeto, noviembre de 2024

1. Planteamiento del Problema

En una ciudad como Barquisimeto, Venezuela (aunque el concepto puede aplicarse en cualquier ciudad), la recolección de basura urbana plantea un desafío logístico considerable. Optimizar este proceso requiere trazar una ruta que minimice tanto el tiempo como los costos operativos, ya que se deben cubrir numerosos puntos de recolección dispersos.

Este informe tiene como objetivo mejorar las rutas de un vehículo de recolección para visitar los diferentes puntos asignados en la ciudad, implementando el algoritmo A-Epsilon. Esta variante del conocido algoritmo A* permite ajustar el peso de la heurística, lo que genera una solución más flexible y adaptada a las necesidades del problema. Con A-Epsilon, el sistema logra reducir el tiempo total de recolección y optimizar el uso de los recursos, logrando un equilibrio entre precisión de la ruta y eficiencia operativa.

2. Implementación

Se utiliza el algoritmo A-Epsilon (extensión del A*), para permitir cierta flexibilidad al explorar rutas alternativas, gracias al parámetro adicional epsilon, se ajustará el peso de la heurística en la decisión de qué nodos visitar primero, lo que puede reducir el tiempo de cálculo o mejorar la adaptabilidad de la ruta en ciertos escenarios. A continuación se explica cada paso de la implementación:

2.1. Configuración Inicial y Estructuras de Datos

En archivo de python. Importamos la librería “**import heap**” el cuál se utiliza para crear y manejar colas de prioridad eficientes.

Luego procedemos a crear el **grafo**, el cuál es un objeto, que contiene como **key** el nombre del punto (como "Plaza Bolívar", "Avenida Lara", "El Obelisco"), y como **value**, una lista de nodos vecinos, y cada conexión (o arista) tiene un peso que representa la distancia o el costo para el vehículo de recolección.

2.2. Puntuaciones g-score y f-score: Se utilizan dos diccionarios para almacenar los costos:

- **g_score**: Que representa el costo real acumulado desde el inicio hasta un punto.
- **f_score**: Que combina **g_score** con la heurística, multiplicada por **epsilon**, para estimar el costo total al destino.

2.3 Función Heurística

En esta implementación la función heurística es una estimación que utiliza la diferencia de longitud entre los nombres de los puntos (simulando la proximidad entre ellos). Aunque es una aproximación, permite que el algoritmo priorice puntos más cercanos.

```
def heuristic(point_a, point_b):
    return abs(len(point_a) - len(point_b))
```

2.4 Parámetro Epsilon

El parámetro epsilon ajusta la importancia de la heurística en el cálculo de **f_score**. ***Con valores más altos de epsilon, el algoritmo prioriza la heurística y explora más nodos rápidamente, lo cual es útil cuando el tiempo es más crítico que la precisión de la distancia.*** Esto también permite al algoritmo ser menos conservador, buscando rutas alternativas en caso de posible congestión.

2.5. Algoritmo A-Epsilon

El algoritmo sigue los pasos estándar de A* con el ajuste de epsilon. A continuación, cada una de las etapas:

2.5.1 Inicialización

Se agrega el punto de inicio en **open_set**, con un **g_score** de 0 y un **f_score** inicial basado en la heurística y epsilon.

2.5.2. Exploración de Nodos

Para la exploración de nodos, están las siguientes condiciones:

- Mientras haya nodos en **open_set**, se selecciona el nodo con el **f_score** más bajo como el nodo actual.
- Si el nodo actual es el destino, se reconstruye la ruta óptima.
- Para cada vecino del nodo actual, se calcula el **g_score** tentativo (costo acumulado para llegar a ese vecino desde el inicio):

  tentative_g_score = g_score[current] + grafo[current][neighbor]

- Si este **g_score** es menor que cualquier valor previo registrado para el vecino, se actualizan los **g_score** y **f_score** de ese vecino, y se le agrega a **open_set** si no estaba incluido.

2.6. Actualización de f-score

La puntuación **f_score** para cada vecino se calcula como:

f_score[neighbor] = tentative_g_score + epsilon * heuristic(neighbor, objetivo)

En este línea de código, es donde utilizamos A-Epsilon (para explorar rutas con cierta flexibilidad y encontrar alternativas de costo más bajo).

2.7 Reconstrucción de la Ruta

Una vez que se encuentra el destino, se reconstruye la ruta en orden inverso, desde el destino hasta el punto de inicio usando el diccionario **came_from**.

Código completo en archivo LuisColmenárez_ejercicioEpsilon.py

3. Pruebas y Resultados

Se realizaron prueba en un conjunto de puntos representativos de la ciudad. El algoritmo A-Epsilon generó rutas optimizadas al ajustar el factor epsilon en función de la prioridad del tiempo o la distancia, mostrando las siguientes características:

Con Epsilon bajo: Se obtuvo la ruta más corta en distancia, ideal para minimizar combustible.

Con Epsilon alto: Se optimizó el tiempo total, seleccionando rutas que evitan zonas potencialmente congestionadas.

4. Conclusiones

La implementación del algoritmo A-Epsilon en Barquisimeto demostró ser eficaz para reducir costos operativos y mejorar la eficiencia en la logística urbana. Gracias a la flexibilidad del parámetro Epsilon, el algoritmo puede priorizar entre minimizar la distancia recorrida o reducir el tiempo de viaje, ajustándose a la variabilidad del tráfico y la densidad de puntos de recolección. Con valores bajos de Epsilon, se optimiza el consumo de combustible; con valores altos, se minimiza el tiempo de viaje, ideal en situaciones críticas. Además, el algoritmo se puede mejorar con datos en tiempo real, adaptándolo a distintas regiones sin cambios significativos en su estructura. En conclusión, el algoritmo A-Epsilon es una herramienta flexible y potente para la gestión eficiente y sostenible de rutas de recolección de residuos urbanos.