República Bolivariana de Venezuela

Universidad centroccidental Lisandro Alvarado

Decanato de ciencias y tecnologías

Algoritmo Epsilon para encontrar la ruta óptima en una red de routers

Integrante:

Douglimar Morles

V-27.737.444

Introducción

Este informe presenta el algoritmo A-epsilon, una variante del algoritmo A* diseñado para encontrar rutas "casi óptimas" en un grafo de red. A-epsilon ofrece una alternativa robusta al algoritmo A*, particularmente útil en escenarios donde los costos de los enlaces entre nodos son dinámicos o propensos a cambios, se busca encontrar diferentes caminos con costos cercanos al mínimo, en lugar de solo la ruta óptima, o se necesita gestionar eficientemente el tráfico en la red y evitar cuellos de botella.

El algoritmo A-epsilon considera un margen de error, representado por el parámetro "epsilon", en la evaluación de rutas alternativas. Esto permite que el algoritmo encuentre soluciones "casi óptimas" de forma más eficiente, explorando un menor número de nodos. Por lo tanto, es especialmente útil en redes con un gran número de routers y rutas posibles.

Definiciones claves

Algoritmo *A:

Es un algoritmo de búsqueda heurística para encontrar la ruta óptima entre un nodo de inicio y un nodo objetivo en un grafo ponderado, donde cada enlace entre nodos tiene un costo asociado. A* utiliza una función de heurística que estima la distancia del nodo actual al nodo objetivo. A* busca caminos con la suma más baja del costo acumulado y la heurística (estimado).

A-épsilon:

Es una variante del algoritmo A* que permite cierto grado de tolerancia en la búsqueda de rutas, en lugar de buscar solo el camino óptimo. En lugar de solo considerar la ruta más corta, A-epsilon busca rutas con costos cercanos al mínimo (dentro de un margen epsilon del costo del camino óptimo).

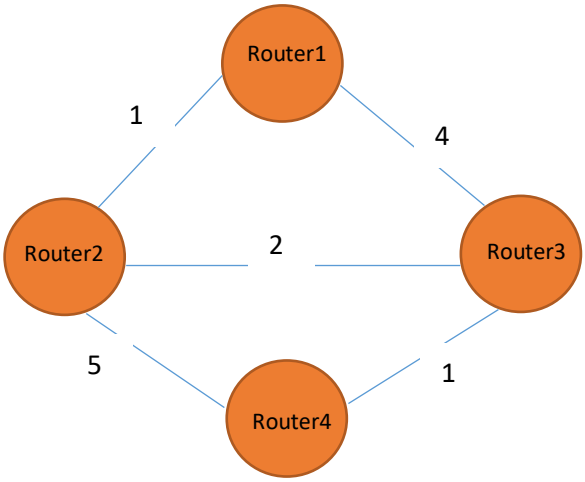
Margen Épsilon (Épsilon):

Es un parámetro positivo del algoritmo que define un rango aceptable para los costos (en tiempo o distancia) de los caminos. A-epsilon buscará rutas que tengan un costo que difiere del costo óptimo en no más que ε.

Problema

La empresa de telecomunicaciones "NetConnect" debe enviar datos esenciales desde el Centro de Control (Router1) hasta el Centro de Atención de Emergencias (Router4). Sin embargo, el tráfico en la red ha aumentado considerablemente, y algunos enlaces se han vuelto más costosos debido a la congestión.Asi, que NetConnect necesita encontrar la ruta más eficiente para enviar los datos desde Router1 hasta Router4, considerando que los costos de los enlaces pueden cambiar en tiempo real, y se busca no solo la ruta óptima, sino también rutas alternativas cercanas al costo mínimo en caso de que algunas conexiones se vuelvan ineficientes.

.



Utilidad del Código

El código proporcionado implementa un algoritmo de búsqueda de rutas conocido como *A Epsilon**, que se utiliza para encontrar caminos eficientes en redes de routers.

A continuación, se detallan las principales utilidades y características de este código:

- **Búsqueda de Rutas Eficientes**:

El algoritmo A* Epsilon permite encontrar rutas que son casi óptimas entre un nodo de inicio y un nodo objetivo en un grafo que representa una red de routers. Esto es especialmente útil en entornos logísticos donde el tiempo de entrega es crítico.

- **Manejo de Costos**:

El algoritmo considera el costo acumulado de los caminos, que en este caso se traduce en el tiempo necesario para recorrer los enlaces entre routers. Esto permite que el algoritmo evalúe diferentes rutas y seleccione aquellas que minimizan el tiempo de entrega.

- **Flexibilidad ante Congestión**:

Al incorporar un parámetro de **epsilon**, el algoritmo permite cierta flexibilidad en la búsqueda de rutas. Esto significa que puede aceptar caminos que no son estrictamente óptimos, pero que son robustos ante condiciones de congestión y cambios en el tráfico, lo que es crucial en un centro logístico dinámico.

- **Visualización de Resultados**:

El código incluye una función para visualizar el grafo y los caminos evaluados utilizando la biblioteca **Matplotlib** y **NetworkX**. Esto facilita la comprensión de cómo se distribuyen los paquetes en la red y cuáles son las rutas seleccionadas, lo que es valioso para la toma de decisiones y la optimización de procesos logísticos.

- **Almacenamiento de Caminos Evaluados**:

A medida que el algoritmo evalúa diferentes caminos, almacena información sobre los caminos evaluados. Esto no solo ayuda a reconstruir la ruta final, sino que también proporciona datos sobre las decisiones tomadas durante la búsqueda, lo que puede ser útil para análisis posteriores y mejoras en el algoritmo.

- **Aplicaciones Prácticas**:

Este tipo de algoritmo es aplicable en diversas áreas, como la gestión de redes de transporte, la planificación de rutas de entrega en empresas de logística, y la optimización de redes de comunicación, donde la eficiencia y la adaptabilidad son esenciales.

Impacto de los Cuellos de Botella:

Los estudios han demostrado que en grandes centros de logística, la congestión causada por la inflexibilidad de los algoritmos de ruta puede provocar:

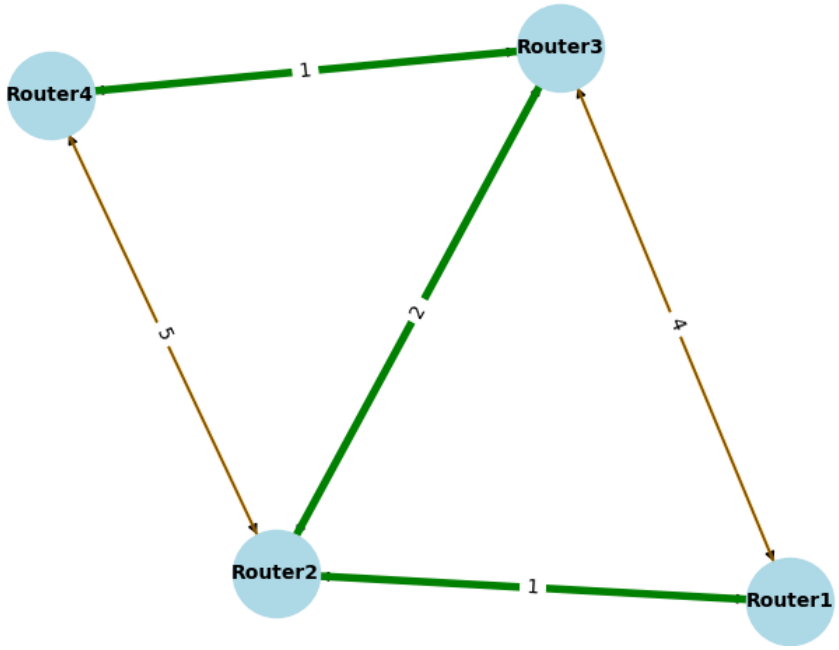
  - Un 15% a 20% de reducción en la velocidad de procesamiento de los paquetes.
  - Un aumento en el tiempo de entrega hasta en un 25%, provocando retrasos en los envíos.
  - Aumento en los costos logísticos.

- Ejemplo: En un centro logístico que procesa miles de paquetes diarios, el algoritmo de ruta tradicional puede asignar la misma ruta (a través de una zona específica) a varios paquetes.

- La congestión en esa zona afecta negativamente a la velocidad de transporte de todos los paquetes que utilizan ese camino, causando retrasos en cascada.

Ejemplo de Uso

El código incluye un ejemplo práctico donde se define una red de routers y se busca la ruta más eficiente desde Router1 hasta Router4, utilizando un valor de epsilon de 1. Al ejecutar el algoritmo, se obtiene la ruta encontrada y el costo asociado, lo que permite a los usuarios evaluar la efectividad del algoritmo en un contexto real.



*** Ruta encontrada: ['Router1', 'Router2', 'Router3', 'Router4'] con costo: 4***

Codigo hecho en python:

```
import heapq
import matplotlib.pyplot as plt
import networkx as nx

def a_epsilon_red(grafo, inicio, objetivo, epsilon):
# Inicialización
    cola = []
    heapq.heappush(cola, (0, inicio))  # (costo acumulado, nodo
actual)
    venido_de = {}
    costo_hasta_ahora = {inicio: 0}
    caminos_evaluados = []

# Para almacenar los caminos evaluados
    while cola:
        costo_actual, nodo_actual = heapq.heappop(cola)

# Si se alcanza el nodo objetivo
        if nodo_actual == objetivo:
            break

        for vecino, peso in grafo[nodo_actual].items():
            nuevo_costo = costo_actual + peso

# Verificar si el nuevo costo es menor o si está dentro del margen
epsilon
            if (vecino not in costo_hasta_ahora or nuevo_costo <
costo_hasta_ahora[vecino]) and nuevo_costo <=
costo_hasta_ahora.get(objetivo, float('inf')) + epsilon:
                costo_hasta_ahora[vecino] = nuevo_costo
                prioridad = nuevo_costo
                heapq.heappush(cola, (prioridad, vecino))
                venido_de[vecino] = nodo_actual
                caminos_evaluados.append((nodo_actual, vecino,
peso))
# Guardar el camino evaluado
# Reconstruir el camino
    camino = []
    if objetivo in venido_de:
        while objetivo is not None:
            camino.append(objetivo)
            objetivo = venido_de.get(objetivo)
        camino.reverse()
```

```
    return camino, costo_hasta_ahora.get(camino[-1], float('inf')),
caminos_evaluados

def dibujar_grafo(grafo, camino, caminos_evaluados):
    G = nx.DiGraph()  # Crear un grafo dirigido

# SE AÑADE NODOS Y ARISTAS AL GRAFO
    for nodo, vecinos in grafo.items():
        for vecino, peso in vecinos.items():
            G.add_edge(nodo, vecino, weight=peso)

# DIBUJAMOS EL GRAFO
    pos = nx.spring_layout(G)  # Posiciones de los nodos
    nx.draw(G, pos, with_labels=True, node_color='lightblue',
node_size=2000, font_size=10, font_weight='bold')

# Se dibuja los caminos evaluados
    for u, v, peso in caminos_evaluados:
        nx.draw_networkx_edges(G, pos, edgelist=[(u, v)], width=2,
alpha=0.5, edge_color='orange')

# SE RESALTA EL CAMINO ENCONTRADO DE COLOR VERDE
    if camino:
        camino_aristas = [(camino[i], camino[i + 1]) for i in
range(len(camino) - 1)]
        nx.draw_networkx_edges(G, pos, edgelist=camino_aristas,
width=4, edge_color='green')

# Etiquetas de los pesos
    edge_labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

    plt.title("Visualización del Grafo y Caminos Evaluados")
    plt.show()

# AQUI TENEMOS LA RED CON LOS NODOS QUE REPRESENTAN LOS ROUTERS
red = {
    'Router1': {'Router2': 1, 'Router3': 4},
    'Router2': {'Router1': 1, 'Router3': 2, 'Router4': 5},
    'Router3': {'Router1': 4, 'Router2': 2, 'Router4': 1},
    'Router4': {'Router2': 5, 'Router3': 1}
}

# EDITAMOS LOS ROUTERS QUE QUEREMOS ENCONTRAR LA MEJOR RUTA Y
EDITAMOS  EL EPSILON AL VALOR QUE NECESITEMOS
```

```
ruta, costo, caminos_evaluados = a_epsilon_red(red, 'Router1',
'Router4', epsilon=1)
print(f"Ruta encontrada: {ruta} con costo: {costo}")

# DIBUJAMOS EL GRAFO
dibujar_grafo(red, ruta, caminos_evaluados)
```

Nota:

- En este segmento de codigo introducimos cualquier diseño de router para generar los nodos.

```
red = {
    'Router1': {'Router2': 1, 'Router3': 4},
    'Router2': {'Router1': 1, 'Router3': 2, 'Router4': 5},
    'Router3': {'Router1': 4, 'Router2': 2, 'Router4': 1},
    'Router4': {'Router2': 5, 'Router3': 1}
}
```

- En este segemento se introducen el nodo de inicio y el nodo final, pero tambien el valor elipson.

```
ruta, costo, caminos_evaluados = a_epsilon_red(red, 'Router1',
'Router4', epsilon=1)
```

- Librerias descargadas:

```
pip install matplotlib
pip install networkx
```

Conclusion

El algoritmo Epsilon proporciona un balance efectivo entre precisión y eficiencia al buscar rutas en redes de routers, particularmente en entornos complejos. Este algoritmo es capaz de descubrir rutas que son casi óptimas de manera más ágil. Además, Epsilon facilita la identificación de trayectorias con tiempos de entrega próximos a los ideales, al mismo tiempo que ofrece una mayor resistencia frente a la congestión y las variaciones en el centro logístico. Como resultado, esto se traduce en mejoras significativas en eficiencia y reducción de costos