RUTEO DE VEHÍCULOS USANDO GRAFOS

Sebastián Giraldo Gómez
Universidad Eafit
Colombia
sgiraldog@eafit.edu.co

Alejandro Cano Múnera
Universidad Eafit
Colombia
acanom@eafit.edu.co

Mauricio Toro
Universidad Eafit
Colombia
mtorobe@eafit.edu.co

RESUMEN

El problema de encontrar un algoritmo que realice un ruteo optimo con un conjunto de datos en un grafo ha sido uno de los mayores retos planteado en la modernidad. La solución efectiva de este problema es muy importante ya que con esta se lograría un gran ahorro en tiempo, dinero, y una gran reducción en la emisión de gases, logrando así un mayor cuidado del medio ambiente y una mayor eficiencia en las empresas.

Los problemas más relevantes relacionados a este tienen que ver generalmente con encontrar la manera mas efectiva de recorrer un grafo para un fin determinado, ya sea encontrar un circuito Hamiltoniano, Euleriano, entre otros, En un tiempo relativamente corto.

1. INTRODUCCIÓN

A lo largo de la historia moderna se han diseñado numerosos algoritmos para recorrer un grafo, aunque algunos algoritmos son más eficientes que otros, no se ha encontrado uno que en un tiempo medianamente corto logre satisfacer las necesidades planteadas, ya que en el tiempo de ejecución son muy demorados. Generalmente se obtiene una respuesta buena y eficiente, pero no se comprueba que es la mejor. En este documento se plantea y se desarrolla un algoritmo que encuentra la ruta óptima para que un conjunto de camiones eléctricos visite un conjunto de clientes, para mejorar el servicio de una empresa.

2. PROBLEMA

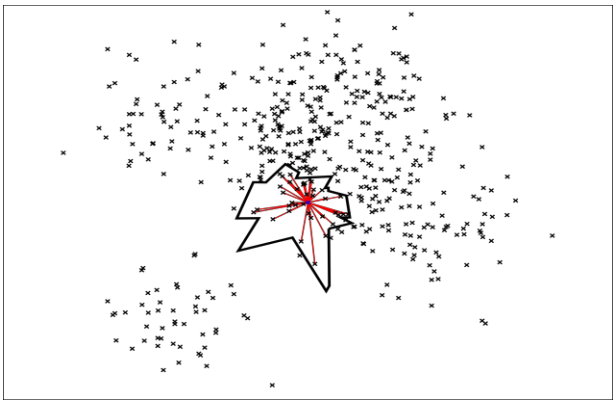
El problema por resolver consiste en diseñar un algoritmo para encontrar las rutas óptimas para que un conjunto de camiones eléctricos visite un conjunto de clientes. La pregunta es la siguiente: Dado una lista de clientes ubicados en un mapa vial bidimensional, un depósito de inicio y fin, y unas restricciones de tiempo y energía ¿cuáles son las rutas para un número libre de camiones eléctricos, para visitar todos los clientes, minimizando el tiempo total, que es la suma del tiempo del recorrido más el tiempo que toman las recargas de batería? [1]. Es importante resolver este problema ya que de esta manera estaríamos ahorrando tiempo, dinero y aportaríamos un mayor cuidado al medio ambiente por la reducción de las emisiones de gases, en la medida de que las compañías implementen algoritmo en sus rutas.

3. TRABAJOS RELACIONADOS

Este problema ha tenido varios intentos y formas de solucionarlo, con diferentes estrategias y formas heurísticas, pero aún no se ha encontrado la solución más óptima a este problema.

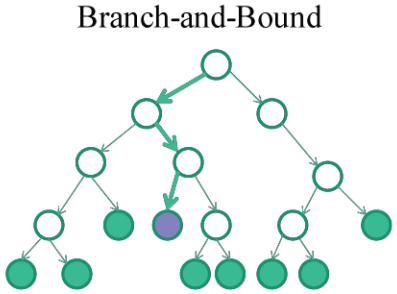
3.1 Vecino más próximo

El Algoritmo del vecino más próximo (NN por sus siglas en inglés) o también llamado algoritmo voraz (greedy) permite al viajante elegir la ciudad no visitada más cercana como próximo movimiento. Este algoritmo retorna rápidamente una ruta corta. Para N ciudades aleatoriamente distribuidas en un plano, el algoritmo en promedio, retorna un camino de un 25% más largo que el menor camino posible.20 Sin embargo, existen muchos casos donde la distribución de las ciudades dada hace que el algoritmo NN devuelva el peor camino.



3.2 Metodo de branch and bound.

El método de branch and bound (ramificación y poda), nos proporciona una solución óptima del problema del agente viajero, calculando mediante el algoritmo simplex la solución del modelo. A medida que aumente el tamaño de la red el método puede tardar gran cantidad de tiempo en resolverse, sin embargo, para redes de mediano tamaño es una excelente alternativa.



Each node in branch-and-bound is a new MIP

¿De qué trata el Algoritmo Simplex?

El Método Simplex es un método iterativo que permite ir mejorando la solución en cada paso. La razón matemática de esta mejora radica en que el método consiste en caminar del vértice de un poliedro a un vértice vecino de manera que aumente o disminuya (según el contexto de la función objetivo, sea maximizar o minimizar), dado que el número de vértices que presenta un poliedro solución es finito siempre se hallará solución.

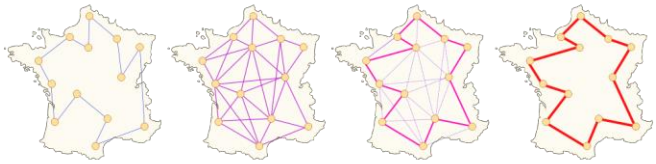
3.3 Método de fuerza bruta

El método de la fuerza bruta no implica la aplicación de ningún algoritmo sistemático, tan solo consiste en explorar todos los recorridos posibles. Es el menos óptimo y el más demorado de todos, debido a que al probar todas las posibilidades sin importar condiciones, ejecuta un sin número de acciones que ralentizan el proceso y lo pueden hacer casi infinito.

3.4 Optimización por Colonia de Hormigas

El investigador de Inteligencia artificial, Marco Dorigo describió en 1997 un método que genera heurísticamente "buenas soluciones" para el TSP usando una simulación de una colonia de hormigas llamada ACS (Ant Colony System).

ACS envía un gran número de hormigas (agentes virtuales) para explorar las posibles rutas en el mapa. Cada hormiga elige probabilísticamente la próxima ciudad a visitar basada en una heurística, combinando la distancia a la ciudad y la cantidad de feromonas depositadas en la arista hacia la ciudad. La hormiga exploradora, deposita feromonas en cada arista que ella cruce, hasta que complete todo el camino. En este punto la hormiga que completó el camino más corto deposita feromonas virtuales a lo largo de toda la ruta recorrida (actualización del camino global). La cantidad de feromonas depositadas es inversamente proporcional a la longitud del camino: el camino más corto, tiene más cantidad de feromonas.



1 2 3 4

4. VECINO MÁS CERCANO MEJORADO

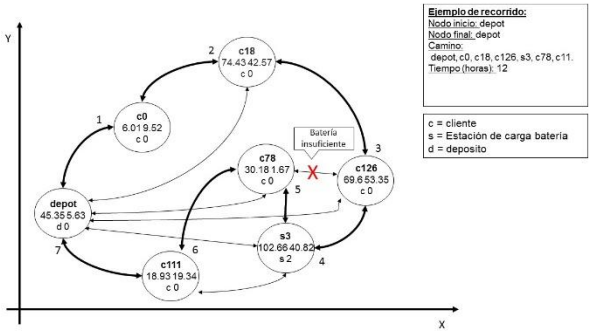
A este problema se le dará solución usando el método del vecino más cercano ya que a pesar de que no todas las veces arroja el resultado óptimo, es un algoritmo bastante rápido.

La idea es implementarlo como base y agregar otras condiciones ya que se tiene una batería y unos nodos estación de carga, lo que le agrega más dificultad y hace más lento el algoritmo.

4.1 Algoritmo Del Vecino Más Cercano

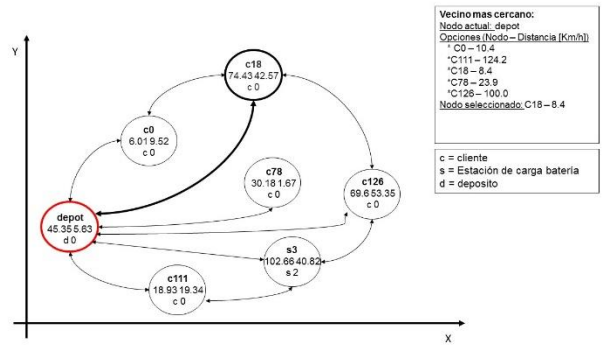
El algoritmo del vecino más cercano funciona de manera muy simple: recorre un grafo en el cual en cada nodo decide a cuál ir, dependiendo de la distancia a la que esté, esto con el fin de realizar el camino más corto, pero a esto también hubo que agregarle una funcionalidad adicional debido a la condición de baterías y estaciones de carga.

Esta funcionalidad funciona de la siguiente manera: Antes de hacer el movimiento al posible vecino más cercano calcula si es posible hacer este movimiento y movimientos posteriores sin pasar por una estación de carga, en el caso de que no, inicia una ruta hacia la estación de carga más cercana.

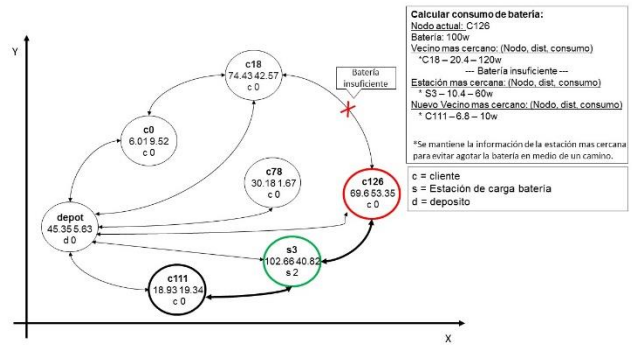


4.2 Operaciones realizadas

La primera operación más importante es la de identificar el vecino más cercano; una vez se está en un nodo, se busca el nodo vecino hacia el cual tiene menos distancia y una vez identificado se mueve hacia dicho nodo.



La segunda operación más importante es la de identificar si es posible seguir sin ir a una estación de carga; una vez se está en un nodo, se identifica el vecino más cercano y se identifica si es viable o no ir, en caso de que no, se busca la estación de carga más cercana.



4.3 Criterios de diseño del algoritmo

La primero que se buscó fue que fuera un algoritmo que tuviera una complejidad no muy alta y que consumiera lo menos posible, ya que ahí radica la dificultad de solucionar este problema; de por sí ya existen varias soluciones, pero si son aplicadas a problemas más grandes y de la vida real, dejan de funcionar correctamente o pueden demorarse tiempos demasiado altos, cosa que no sería útil. Debido a esto, el principal criterio para la selección de este algoritmo fue su viabilidad respecto a la complejidad, tiempos de ejecución y gasto de memoria.

4.4 Análisis de complejidad

Método	Complejidad
Vecino más cercano	O(n)
Consumo barrería	O(n)
Cargar nodos	O(m)

Tabla 1: Tabla para reportar la complejidad en el peor de los casos, en esta tabla n representa el número de nodos que se pueden visitar desde un nodo en particular, y m representa el número total de nodos en el grafo.

4.5 Tiempos de ejecución

La siguiente tabla muestra el tiempo promedio en milisegundos que le tomo a la estructura de datos realizar las operaciones de buscar el vecino más cercano y calcular el consumo de batería con los diferentes conjuntos de datos.

	Conjunto de datos 1 (345 nodos)	Conjunto de datos 2 (359 nodos)	Conjunto de datos 3 (359 nodos)
Vecino más cercano	1000 ms	1100 ms	1500 ms
Consumo barrería	1000 ms	1000 ms	1200 ms

Tabla 2: Tiempos de ejecución de las operaciones de la estructura de datos con diferentes conjuntos de datos en su caso promedio.

4.6 Memoria

La siguiente tabla muestra la cantidad de memoria en MB que le tomo a la estructura de datos realizar las operaciones de buscar el vecino más cercano y calcular el consumo de batería con los diferentes conjuntos de datos.

	Conjunto de datos 1 (345 nodos)	Conjunto de datos 2 (359 nodos)	Conjunto de datos 3 (359 nodos)
consumo de memoria	10 MB	12 MB	12 MB

Tabla 3: Consumo de memoria de la estructura de datos con diferentes conjuntos de datos

REFERENCIAS

[1] Montoya A., Guérethttps C. [En línea] Disponible en: https://hal.archives-ouvertes.fr/hal-01245232/document [Accedido el 03 de Marzo de 2018]

1. Wikipedia, El Problema del Viajante. https://es.wikipedia.org/wiki/Problema_del_viajante

2. Ingeniería Industrial Online, El Problema del Agente Viajero-TSP. https://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/investigaci%C3%B3n-de-operaciones/problema-del-agente-viajero-tsp/

3. Ingeniería Industrial Online, El Método Simplex. https://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/investigaci%C3%B3n-de-operaciones/m%C3%A9todo-simplex/