Algoritmo para encontrar rutas óptimas para vehículos eléctricos

Tomas Atehortua Ceferino Universidad Eafit Colombia tatehortuc@eafit.edu.co	Mauricio Toro Universidad Eafit Colombia mtorobe@eafit.edu.co

RESUMEN

El objetivo de este proyecto es diseñar un algoritmo que encuentre rutas óptimas a los vehículos eléctricos, en este caso específico para hacer más eficiente la forma de distribuir mercancías. Esto con el fin de solucionar el problema que se presenta en las baterías de los coches por tener un rango de durabilidad limitado y un tiempo de carga muy largo.

La solución que se dio a este problema fue una versión modificada del vecino más cercano, haciendo que camiones recorrieran el mapa y generaran las rutas necesarias. Esto permitió llegar a la solución generando varios camiones, cada uno con una ruta diferente hasta lograr que todos los clientes sean visitados en el menor tiempo posible.

Palabras clave

Optimización - Grafos - Vehículos eléctricos – Rutas - Algoritmo

Palabras clave de la clasificación de la ACM

Theory of computation → Design and analysis of algorithms → Graph algorithms analysis → Shortest paths

Theory of computation → Design and analysis → Approximation algorithms analysis → Routing and network design problems.

Applied computing → Operations research → Trasnportation.

1. INTRODUCCIÓN

En el siguiente informe se pretende solucionar la problemática que tienen las empresas a la hora de entregar productos con camiones de carga eléctricos, para que puedan entregarlos de la manera más eficiente, teniendo en cuenta las horas de trabajo, número de autos y la vida útil de los Batería de coche. Resolver este problema permitiría mejorar las entregas y asegurar que las empresas utilizarán coches eléctricos, incluso con la necesidad de cargarlos, permitiendo una reducción de las emisiones de carbono que liberan estos camiones de carga.

2. PROBLEMA

El problema radica en la baja eficiencia que tienen las baterías de los autos eléctricos para distribuir la mercadería, pues demoran mucho en hacer los recorridos debido a que la batería se descarga constantemente. Esto tiene un alto impacto en la sociedad porque si no se soluciona, los transportistas se verían obligados a utilizar otros vehículos más dañinos para el planeta, por eso para evitar que eso suceda es necesario solucionar este problema.

3. TRABAJOS RELACIONADOS

3.1 The Traveler Salesman Problem

Propuesta por William Hamilton y Thomas Kirkman, consiste en buscar el mejor camino que se pueda realizar en un conjunto de n nodos, pasando solo una vez por cada nodo hasta volver al primero. A lo largo de los años se han propuesto varias soluciones a este problema de optimización. Uno de ellos es el algoritmo de Christofides, que consiste en elegir y reemplazar aristas para mantener menos distancia. Es un algoritmo heurístico ya que su funcionamiento busca una ruta aproximada con los pesos que se plantean en los datos [1].

3.2 Vehicle Routing Problem (VRP)

El VRP es la distribución del Problema del vendedor viajero a diferentes vehículos. "Decide qué vehículo maneja qué solicitud en qué secuencia para que todas las rutas del vehículo puedan ejecutarse de manera factible" [2]. En este caso, una solución viable es el algoritmo de ahorro de Clark y Wright, que consiste en combinar dos rutas posibles, tomando los ahorros que se producen de cada una. Así, de forma aproximada, es posible atribuir el mejor recorrido para cada vehículo.

3.3 Electric Vehicle Routing Problem (E-VRP)

“A medida que las corporaciones se vuelven más conscientes del medio ambiente y los costos de externalidad asociados, los vehículos comerciales eléctricos están ganando terreno en las empresas que entregan productos” [3]. Bajo esta premisa, se creó una variante del problema de enrutamiento de vehículos, el E-VRP, que se enfoca en encontrar una estrategia de enrutamiento óptima con un costo mínimo de tiempo de viaje. costo de energía y vehículos eléctricos despachados. Hacer un enfoque heurístico como un algoritmo inspirado en una búsqueda aleatoria o una solución de programación dinámica como Held Karp (con

limitaciones de recursos) podría ser la mejor manera de abordar el EVRP.

3.4 Problema de las colonias de hormigas

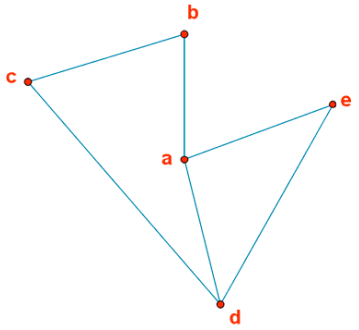
El algoritmo basado en las colonias de hormigas es utilizado para resolver varios problemas que requieren elegir la solución más corta entre varias posibles soluciones. [4] Lo que hace este algoritmo es construir las posibles soluciones e ir escogiendo cual es la óptima, para de esta manera, actualizar el peso o la distancia que hay entre cada nodo.

4. PRIMERA SOLUCIÖN DISEÑADA

A continuación, explicamos la estructura de datos y el algoritmo.

4.1 Estructura de datos

Diseñen la estructura de datos para resolver el problema y grafíquenla. No usar gráficas extraídas de internet



	a	b	c	d	e
a	0	1	0	1	1
b	1	0	1	0	0
c	0	1	0	1	0
d	1	0	1	0	1
e	1	0	0	1	0

Gráfica 1: Grafo representado como una matriz de adyacencia

4.2 Operaciones de la estructura de datos

Buscar vértices:

	0	1	2	3
0	-1	2	3	-1
1	6	-1	3	-1
2	-1	3	-1	5
3	4	6	2	-1

Gráfica 2: Dados dos nodos, retornar el vértice (peso), que hay entre los dos nodos, En este caso los que están de color amarillo corresponden al peso que hay para ir de un nodo al otro.

Buscar los vecinos:

	0	1	2	3
0	-1	2	3	-1
1	6	-1	3	-1
2	-1	3	-1	5
3	4	6	2	-1

Gráfica 3: Desde un nodo inicial encontrar todos los nodos a los que se pueden llegar, en este caso para el nodo 0 los vecinos son el 1 y el 2.

Buscar un vertice:

	0	1	2	3
0	-1	2	3	-1
1	6	-1	3	-1
2	-1	3	-1	5
3	4	6	2	-1

Gráfica 4: De todos los nodos a los que se puede llegar, retornar el peso entre esos dos nodos, en este caso para el nodo 0 y el nodo 1 el peso es 2.

4.3 Criterios de diseño de la estructura de datos

Elegí representar el grafo con una matriz de adyacencia debido a que su complejidad en el acceso a los datos es de O(1). Con esta complejidad el algoritmo se hace mucho más eficiente a la hora de realizar las operaciones que requieren acceder a los pesos de las aristas de nuestro grafo. Además, teniendo en cuenta que los datasets van desde 320 hasta 360 nodos, la cantidad de memoria que se utiliza en esta matriz no es demasiada comparada con la eficiencia en tiempo que ganamos.

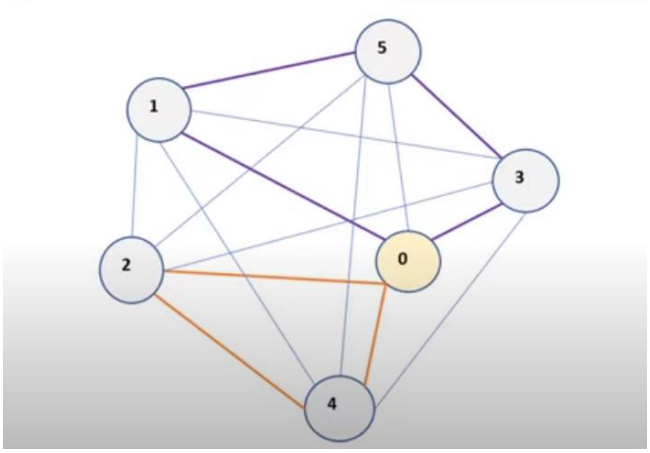
4.4 Análisis de Complejidad

La complejidad de los métodos es la siguiente, en la cual n corresponde al número de nodos

Método	**Complejidad**
Buscar un vertice	O(1)
Buscar vecinos	O(n)
Creación de la matriz	O(n^2)

Tabla 1: Tabla para reportar la complejidad

4.5 Algoritmo de Clarke-Wright



Gráfica 3: Ejemplo de los caminos (ahorros) que son encontrados por el algoritmo de Clarke-Wright

4.6 Cálculo de la complejidad del algoritmo

La complejidad del algoritmo para el peor de los casos, el mejor de los casos y el caso promedio

Sub problema	Complejidad
Determinar si un nodo se puede visitar	O(n)
Calcular el camino para cada vehiculo	O(n^3)
Incluir un nodo en un camino	O(1)
Encontrar los ahorros	O(n^2)
Complejidad Total	O(n+n^2+n^3 + 1) = O(n^3)

Tabla 2: complejidad de cada uno de los subproblemas que compoenne el algoritmo. Donde n es la cantidad de nodos del dataset,

4.7 Criterios de diseño del algoritmo

Después de analizar diferentes soluciones al problema, se decidió por implementar una solución basada en el algoritmo de Clarke-Wright. Este algoritmo ayuda bastante para resolver problemas en los que el número de vehículos no es concreto. En esta solución, se procura encontrar una ruta eficiente con limitaciones de tiempo y batería, ya que son vehículos eléctricos. Es bueno mencionar que este algoritmo no siempre encuentra la mejor solución, pero aun así es eficiente en tiempo y da una respuesta suficientemente valida.

4.8 Tiempos de Ejecución

Calculen, (I) El tiempo de ejecución que consume el programa para varios ejemplos de dataset es el siguiente:

	Conjunto de Datos 1	***Conjunto de Datos 2***	***Conjunto de Datos 3***
Mejor caso	4 sg	3 sg	7 sg
Caso promedio	12 sg	10 sg	13 sg
Peor caso	30 sg	19 sg	25 sg

Tabla 3: Tiempos de ejecución del algoritmo con diferentes conjuntos de datos

4.9 Memoria

La memoria que consume el programa para varios ejemplos de dataset es el siguiente:

	Conjunto de Datos 1	***Conjunto de Datos 2***	***Conjunto de Datos 3***
Consumo de memoria	0,74 MB	0,83MB	0,84 MB

Tabla 4: Consumo de memoria del algoritmo con diferentes conjuntos de datos

4.10 Análisis de los resultados

Los resultados obtenidos son los siguientes:

Dataset	Tiempo de ejecución	Numero de clientes	Numero de vehiculos	Total tiempo de la ruta
1	12 s	320	33	2.1500 h
2	11 s	320	27	1,5995h
3	13 s	320	29	1,9882h
4	16 s	320	33	2,6587h

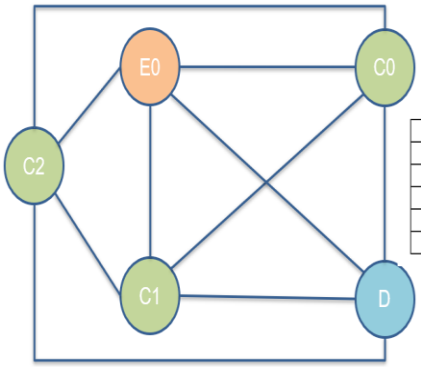
Tabla 5: Análisis de los resultados obtenidos con la implementación del algoritmo

Gráfica 5: Imagen de una operación getWeight del grafo, en la cual se retorna el peso que hay entre C1 y C0

5. SOLUCIÓN FINAL DISEÑADA

A continuación, explicamos la estructura de datos y el algoritmo.

5.1 Estructura de datos



	C0	C1	C2	E0	D
C0	-1	1	1	1	1
C1	1	-1	1	1	1
C2	1	1	-1	1	1
E0	1	1	1	-1	1
D	1	1	1	1	-1

Gráfica 4: Grafo representado como una matriz de adyacencia, en el cual se representan los puntos del mapa (con peso 1), los cuales pueden ser tipo deposito "D", cliente **"C"** o estación **"E"**

5.2 Operaciones de la estructura de datos

Diseñen las operaciones de la estructura de datos para solucionar el problema eficientemente. Incluyan una imagen explicando cada operación

addArc:

	C0	C1	C2	E0	D
C0	-1	-1	-1	-1	-1
C1	-1	-1	-1	-1	-1
C2	-1	-1	-1	-1	-1
E0	-1	-1	-1	-1	-1
D	-1	-1	-1	-1	-1

	C0	C1	C2	E0	D
C0	-1	1	-1	-1	-1
C1	1	-1	-1	-1	-1
C2	-1	-1	-1	-1	-1
E0	-1	-1	-1	-1	-1
D	-1	-1	-1	-1	-1

Gráfica 6: Imagen de una operación addArc del grafo, en la cual se crea en arco entre C0 y C1

getWeight:

	C0	C1	C2	E0	D
C0	-1	1	-1	-1	-1
C1	1	-1	-1	-1	-1
C2	-1	-1	-1	-1	-1
E0	-1	-1	-1	-1	-1
D	-1	-1	-1	-1	-1

getSuccesors:

	C0	C1	C2	E0	D
C0	-1	1	1	-1	-1
C1	1	-1	-1	-1	-1
C2	1	-1	-1	-1	-1
E0	-1	-1	-1	-1	-1
D	-1	-1	-1	-1	-1

Gráfica 7: Imagen de una operación de getSucceros del grafo, en la cual se deben retornar los vecinos de C0, en este caso serían C1 y C2

5.3 Criterios de diseño de la estructura de datos

Elegí representar el mapa como un grafo con una matriz de adyacencia debido a que su complejidad en el acceso a los datos es de O (1). Con esta complejidad el algoritmo se hace mucho más eficiente a la hora de realizar las operaciones que requieren acceder a los pesos de las aristas de nuestro grafo, mejorando así el tiempo de ejecución porque para poder saber cuál es el vecino más cercano se debe consultar constantemente los pesos de cada uno los nodos. Además, teniendo en cuenta que los datasets van desde 320 hasta 360 nodos (no son extremadamente grandes), la cantidad de memoria que se utiliza en esta matriz no es demasiada comparada con la eficiencia en tiempo que ganamos a la hora de la ejecución y búsqueda de la solución al problema.

5.4 Análisis de Complejidad

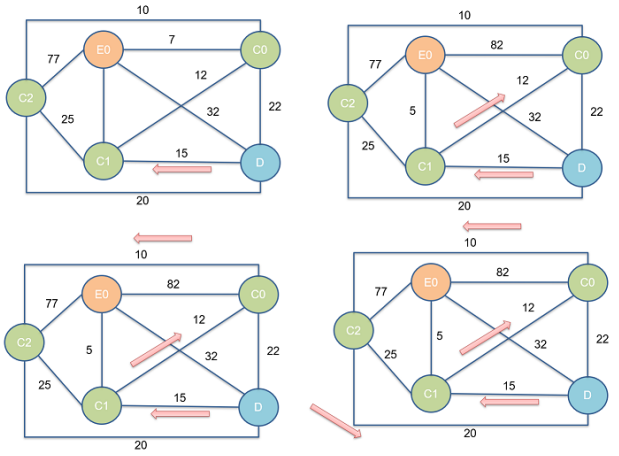
Calculen la complejidad de las operaciones de la estructura de datos para el peor de los casos. Vean un ejemplo para reportarla:

Metodo	Complejidad
Crear la matriz	O(n^2)
AddArc	O(1)
getWeight	O(1)
getSuccersors	O(n)

Tabla 6: Tabla para reportar la complejidad

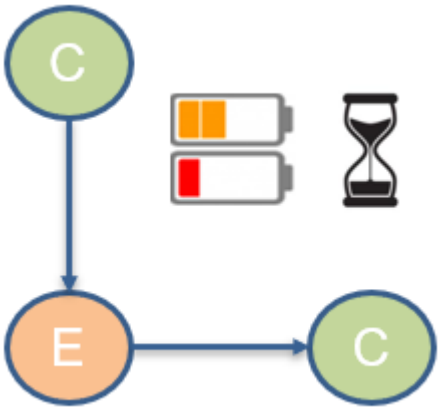
5.5 AGENTE VIAJERO, VECINO MÁS CERCANO

El algoritmo diseñado consiste en buscar el vértice más cercano al que se está ubicado en el momento (siempre teniendo en cuenta la batería y el tiempo), esto con el fin de encontrar una ruta óptima al problema. En caso de tenga tiempo y batería en una condición óptima, se dirigirá a cliente más cercano, en caso tal de tener tiempo, pero no batería se dirigirá a la estación más cercano y por otro lado si se está quedando sin tiempo, se debe dirigir al deposito para dar salida a otro camión y continuar con la solución del problema.

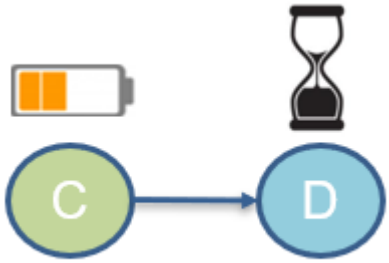


Gráfica 8: Representación gráfica de la selección del vecino más cercano

Gráfica 9: Representación gráfica del movimiento del camión, En caso de tener batería y tiempo



Gráfica 10: Representación gráfica del movimiento del camión, no tener batería y si tener tiempo



Gráfica 10: Representación gráfica del movimiento del camión, no tener batería y no tener tiempo

5.6 Cálculo de la complejidad del algoritmo

Calculen la complejidad del algoritmo para el peor de los casos, el mejor de los casos y el caso promedio

Sub-problema	Complejidad
Leer cada archivo	O(n)
Crear grafo de matriz de adyacencia	O(V^2)
Calcular la distancia entre todos los nodos	O(V^2)
Verificar el tiempo y verificar la energía	O(1)

Encontrar el vecino más cercano	O(V^2)
Encontrar las posibles rutas	O(R)
Complejidad total:	O(V^2+V^2+R+1) = O(V^2)

Tabla 7: Complejidad de cada uno de los sub-problemas que componen el algoritmo utilizado, Donde V es el tamaño o número de nodos que hay en el grafo, y R el conjunto de clientes que han sido ya visitados por cada ruta, donde equivale a recorrer el tamaño del nodo V menos la cantidad de estaciones

5.7 Criterios de diseño del algoritmo

Para la creación de este algoritmo que da una posible solución al problema planteado se tuvo en cuenta la eficiencia tanto del tiempo como de la memoria empleada por este, y como la razón entre estos varía dependiendo de los algoritmos que usamos. Además de encontrar una solución óptima y que se adapte a los requisitos funcionales y no funcionales dados en el problema. Después de analizar las diferentes aplicaciones de las soluciones que se planteó para el problema, podría decir que lo más optimo, teniendo en cuenta también la razón entre el tiempo y la memoria fue usar un algoritmo voraz, específicamente la heurística del vecino más cercano; sin embargo, para adaptarlo al problema tuve que modificar el algoritmo, creando nuevas técnicas para dar con el funcionamiento correcto de este. La implementación del vecino más cercano me ayudó a optimizar la solución ya que puede encontrar rutas optimas en donde se pueda emplear menos tiempo y energía y visitar varios clientes, utilizando más de un camión. Una solución simple, eficiente y eficaz.

5.8 Tiempos de Ejecución

	Conjunto de Datos 1	***Conjunto de Datos 2***	***...Conjunto de Datos 3***
Mejor caso	16 ms	15ms	50ms
Caso promedio	200 ms	65ms	78ms
Peor caso	389 ms	250ms	550ms

Tabla 8: Tiempos de ejecución del algoritmo con diferentes conjuntos de datos

5.9 Memoria

	Conjunto de Datos 1	***Conjunto de Datos 2***	***Conjunto de Datos 3***
Consumo de memoria	8.5 MB	8.4 MB	7.4 MB

Tabla 9: Consumo de memoria del algoritmo con diferentes conjuntos de datos

5.10 Análisis de los resultados

Como bien sabemos el algoritmo implementado, basado en el algoritmo del vecino más cercano, tiene teóricamente una complejidad en el peor de los casos de $O(n^2)$, De lo que se puede concluir que, aunque la complejidad no sea la más optima, los resultados siguen siendo óptimos para la solución al problema planteado relativo a otros algoritmos gracias a los datasets utulizados, ya que el algoritmo se ejecuta rápidamente, para cada uno de los casos de prueba, manteniendo la eficiencia en cuanto al tiempo y memoria, aun cuando toma valores grandes.

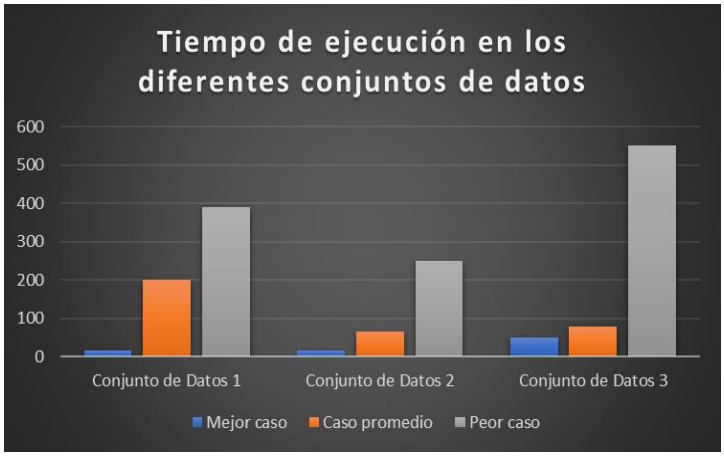


Tabla 10: Análisis de los resultados obtenidos con la implementación del algoritmo

6. CONCLUSIONES

Para dar la solución a un problema se debe pensar muy bien en el diseño del algoritmo y la estructura de datos, para dar con una solución eficiente en todos los criterios. El algoritmo diseñado cumple con algunos requisitos de optimización como lo son la memoria y el tiempo de ejecución, sin embargo, solución que genera este algoritmo no es la más optima en el ruteo generado.

Aun así, en comparación con la primera versión del algoritmo que se diseñó (implementando el algoritmo de Clarke-Wright) esta versión es mucho mejor y a su vez se implementó la batería de cada de los camiones, que no se había implementado en el primer diseño. Pero aun así sin el primer diseño de este algoritmo no hubiera podido llegar a la solución final, gracias a que el problema fue dividido en etapas fue mucho más fácil entenderlo y poder llegar a una de sus muchas soluciones.

El algoritmo aún se puede mejorar, se puede implementar un mejor manejo de la memoria y de la eficiencia en tiempo pensando en más aspectos que también son importantes (como que las rutas ya creadas no choquen entre sí) y así poder llegar resultados mejores para dar una mejor respuesta a este tipo de problemas

6.1 Trabajos futuros

Al algoritmo me gustaría implementarle una forma de poder visualizar las rutas que genera. Que de alguna manera se pueda proyectar todo lo que se está generando, a su vez me gustaría mejorar la forma en que realiza todo el ruteo, sé que la forma en que genera las rutas mi algoritmo no es el más optimo y por lo tanto el tiempo total tampoco lo es, me gustaría encontrar la forma más optima de hacer todo este ruteo para dar con la mejor solución al problema.

AGRADECIMIENTOS

Agradezco a Eafit y al profesor Mauricio Toro, gracias a ellos esta investigación fue lograda y parcialmente terminada.

REFERENCIAS

[1] Bernal, J., Hontoria, E. and Alekvsovski, D., 2015. El problema del viajante de comercio: Búsqueda de soluciones y herramientas asequibles. ASEPUMA, 16(2), pp.117-133.

[2] Irnich, S., Toth, P., and Vigo, D., 2014. Vehicle Routing Problems, Methods, and Applications. Philadelphia: Society for Industrial and Applied Mathematics.

[3] Jane Lin, Wei Zhou, Ouri Wolfson, Electric Vehicle Routing Problem, Transportation Research Procedia, Volume 12, 2016, 508-521, https://doi.org/10.1016/j.trpro.2016.02.007.

[4] Sancho, F. Algoritmos de hormigas y el problema del viajante. Recuperado Noviembre 17, 2016, del Dpto. de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla: http://www.cs.us.es/~fsancho/?e=71