OPTIMIZACIÓN DE RUTEO PARA VEHÍCULOS DISTRIBUIDORES ELÉCTRICOS

Alejandro Arroyave Bedoya
Universidad EAFIT
Colombia
Aarroyaveb@eafit.edu.co

Santiago Castrillón Galvis
Universidad EAFIT
Colombia
Scastrillg@eafit.edu.co

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

Luis Javier Palacio Mesa
Universidad EAFIT
Colombia
Ljpalaciom@eafit.edu.co

Palabras clave
TSP, grafos, algoritmo voraz, ruteo de vehiculos, estructura de datos, vehículos eléctricos con batería (VEB), K-means.

Palabras claves de la clasificación de la ACM
CCS → Theory of computation → Design and analysis of algorithms → Graph algorithms analysis → Shortest paths.

1. INTRODUCCIÓN

Con la creciente globalización, la manera eficiente para el transporte de productos surge como una necesidad ante la progresiva demanda de los mismos. Se deben diseñar rutas para que vehículos de carga distribuyan en el menor tiempo posible, y es por esto que constantemente vemos nuevos modelos de vehículos, con mayor capacidad de carga. Sin embargo, al existir un número altísimo de clientes que complican la solución manual, es evidente que la velocidad y capacidad de carga no son los únicos factores que determinan la efectividad en el tiempo de las entregas. Por ello es que la ruta obtiene tanta relevancia, porque se debe calcular el orden de entrega de los productos de manera que se visite a todos los clientes y se consuma el menor tiempo posible.

No obstante, dado que el petróleo es un recurso limitado, pensando en el futuro, se deben buscar soluciones a este problema con un nuevo detalle: Se deben usar vehículos eléctricos. Esto presenta un reto puesto que los mismos tiene poca capacidad de autonomía si se compara con un vehículo que use petróleo.

2. PROBLEMA

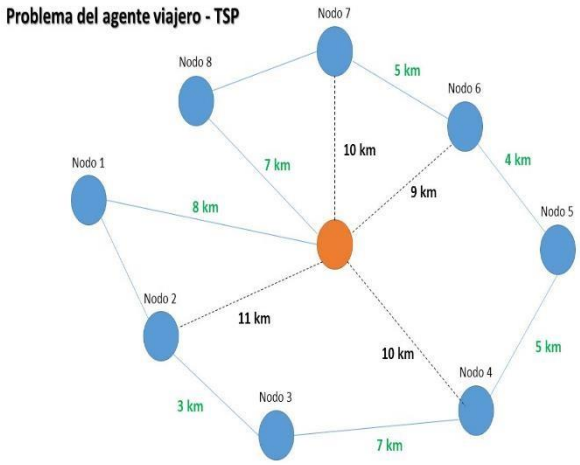
Cuando hacemos un envío de varios productos a diferentes puntos, o queremos recorrer varios lugares determinados en el menor tiempo posible, se vuelve necesario trazar la ruta más corta que nos optimice el tiempo de viaje, pero también debemos tener en cuenta que los medios de transporte utilizan combustible o energía, por lo que se hace necesario tener en cuenta el tiempo de recarga y el desplazamiento hacia este punto. Así que es necesario determinar bajo estos factores, cual es la ruta más corta que podemos escoger para lograr el menor tiempo posible.

3. TRABAJOS RELACIONADOS

3.1 Problema del agente viajero (TSP) El problema del agente viajero consiste en encontrar la mejor ruta (minimizando distancia recorrida o tiempo recorrido) para un vehículo de reparto que pase por todos los vértices de un grafo(clientes) sin repetir, y volviendo al punto de origen.

Solución exacta: Una solución exacta sería hacer todas las posibles permutaciones del grafo (fuerza bruta), sin embargo, la complejidad es O(n!), el algoritmo es muy ineficiente para un grafo mediano o más grande (más de 15 nodos).

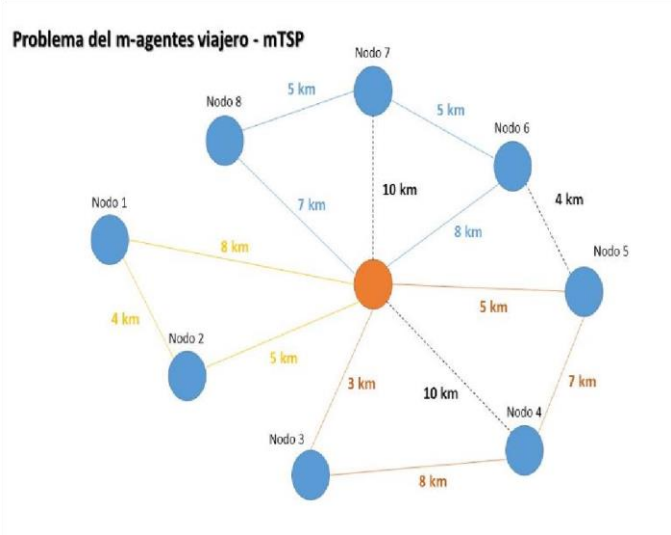
Solución heurística: Una solución heurística sería la llamada el "vecino más cercano", que consiste en escoger para el nodo inicial, el nodo más próximo, y a su vez, de manera recursiva, este último, a su más cercano, excluyendo al ya visitado. Cuando todos los nodos hayan sido visitados, se debe agregar el camino hacía el nodo inicial.



Gráfica 1: Representación gráfica del problema del agente viajero - TSP

3.2 Problema del m-agentes viajeros (m-TSP) Esta es una variación del problema tradicional del agente viajero. En este caso se tienen m vehículos de reparto, y un nodo inicial y final visto como la bodega de una empresa, se deben realizar m rutas para los m vehículos y se debe garantizar

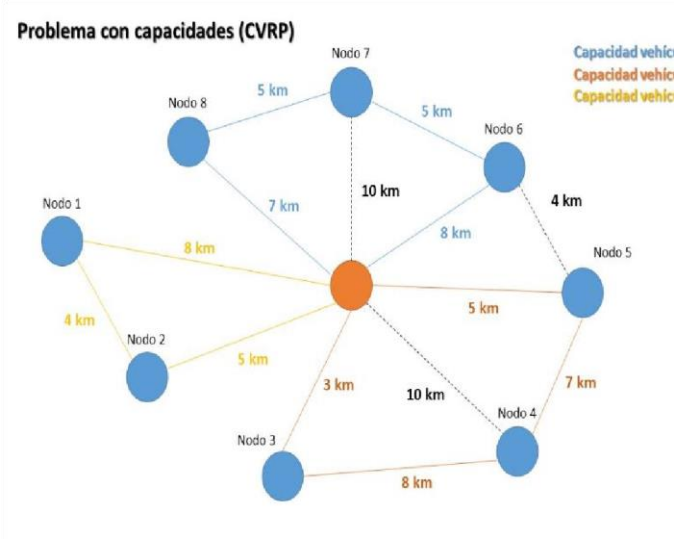
que cada cliente sea visitado exclusivamente una vez por uno sólo de estos vehículos, encontrando las rutas que generen el menor tiempo o distancia posible.



Gráfica 2: Representación gráfica del problema del m-agentes viajero - mTSP

3.3 Problema con capacidades (CVRP)

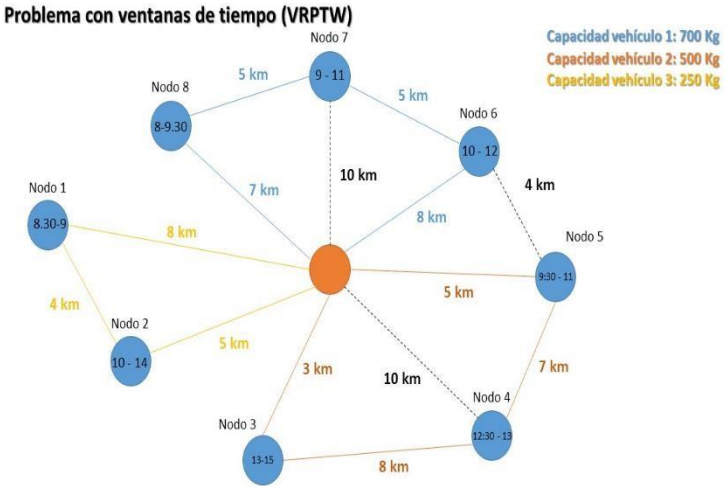
Este es una derivación del problema m-agentes viajeros, en las que se introducen dos nuevos conceptos. El primero es la demanda de cada cliente, y se refiere a la cantidad de bienes que cada uno requiere, y el segundo es la capacidad de los bienes de cada vehículo de reparto.



Gráfica 3: Representación gráfica del problema del agente viajero con capacidades - CVRP

3.4 Problema con ventanas de tiempo (VRPTW)

En este problema, además de tener capacidades y demandas (CVRP), hay un tiempo de visita válido para cada cliente, en el que se restringe las horas en las que cada camión puede llevar el pedido a cada uno de ellos. Además, cada cliente tarda un tiempo determinado para ser despachado.



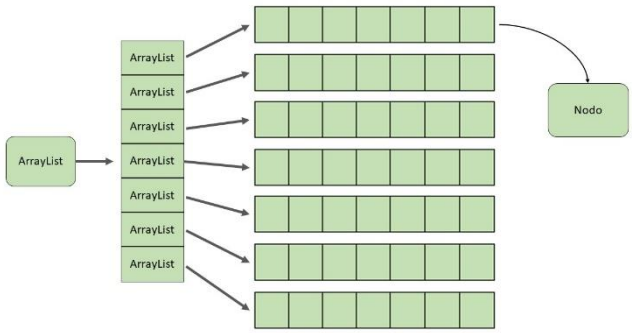
Gráfica 4: Representación gráfica del problema del agente viajero con ventanas de tiempo - VRPTW

4. TÍTULO DE LA PRIMERA SOLUCIÓN DISEÑADA

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

4.1 Estructura de datos

La estructura de datos principal utilizada es un grafo



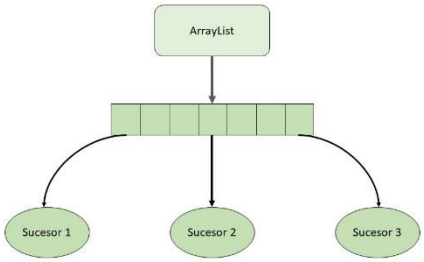
Gráfica 5: Grafo representado por listas de adyacencia

Dentro de los algoritmos del proyecto se encuentran otras estructuras de datos que son:

- Arreglos

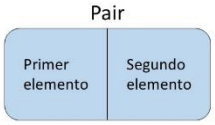
Gráfica 6: Representación gráfica de arreglo

- ArrayList



Gráfica 7: Representación gráfica de ArrayList

- Pair

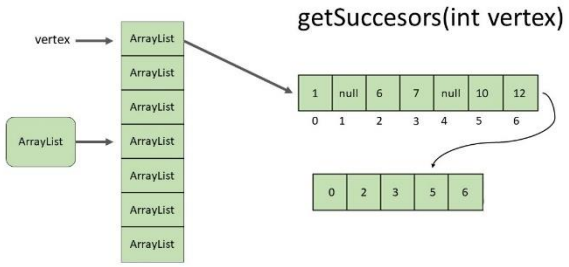


Gráfica 8: Representación gráfica de pair

4.2 Operaciones de la estructura de datos

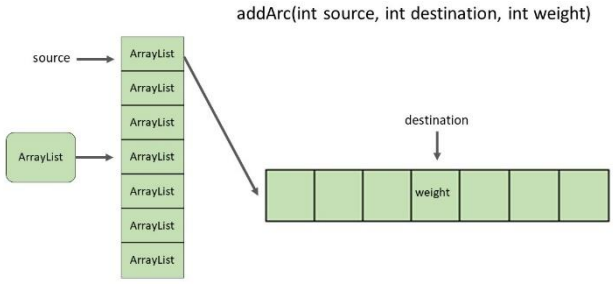
Las operaciones de la estructura de datos principal son:

- getSuccesors (int vertex): Esta operación devuelve un arreglo con los sucesores del nodo vertex.



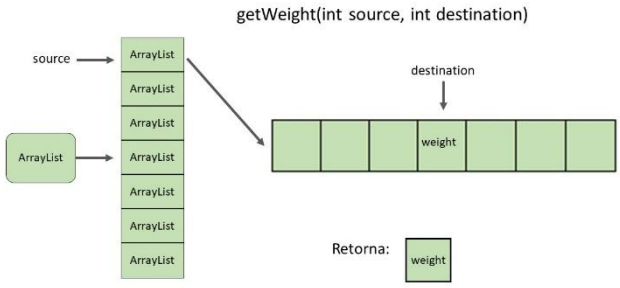
Gráfica 9: Método getSuccesors (int vertex)

- addArc (int source, int destination, int weight): Esta operación inserta un arco desde el nodo source al nodo destination con un peso weight.



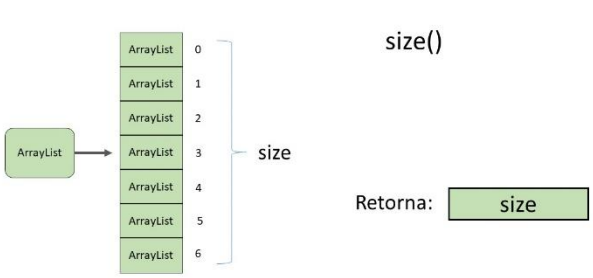
Gráfica 10: Método addArc (int source, int destination, int weight)

- getWeight (int source, int destination): Esta operación devuelve el peso del arco que hay entre source y destination.



Gráfica 11: Método getWeight(int source, int destination)

- size(): Esta operación retorna el tamaño del grafo que tenemos.



Grafica 12: Método size()

4.3 Criterios de diseño de la estructura de datos

Diseñamos el algoritmo utilizando listas de adyacencia porque ahorra un porcentaje grande memoria, además de tener una buena complejidad de las operaciones del grafo, además de tener más fácil manejo de los datos almacenados allí.

4.4 Análisis de Complejidad

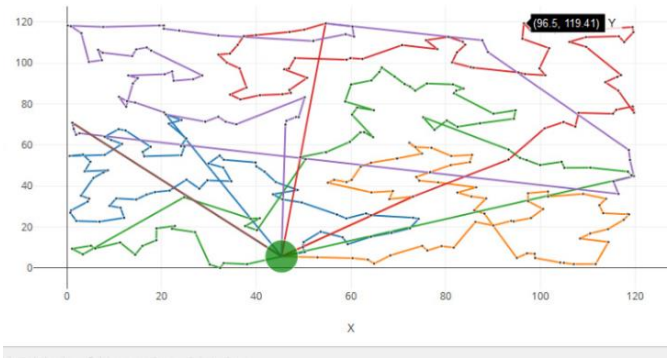
Método	Complejidad
getSuccesors	O(nLogn)
addArc	O(n)
getWeight	O(n)
Size	O(1)

Tabla 1: Tabla para reportar la complejidad, donde n es el número de nodos que hay en el grafo.

4.5 Algoritmo

En este primer algoritmo se busca generar varias rutas, las cuales ninguna supere una duración de "Tmax" horas. Para esto se usa el algoritmo del vecino más cercano con una condición que lo detiene cada que se va a superar esta constante del problema. De esta manera se obtienen n rutas que corresponden a n buses. Este algoritmo implica que las últimas rutas incluyan puntos bastante lejanos del depósito, y serán éstas las que determinan el tiempo de solución. No se tiene en cuenta las estaciones de carga ni la batería de los autos.

A continuación, un gráfico que muestra el algoritmo:



Grafica 13: Solución al algoritmo

4.7 Criterios de diseño de la estructura de datos

Diseñamos el algoritmo utilizando listas de adyacencia porque ahorra un porcentaje grande memoria, además de tener una buena complejidad de las operaciones del grafo, además de tener más fácil manejo de los datos almacenados allí.

4.8 Tiempos de ejecución

	Mejor caso (en ms)	Caso promedio (en ms)	Peor caso (en ms)
Conjunto de datos 1	144	171	172
Conjunto de datos 2	257	317	350
Conjunto de datos 3	44	60	64
Conjunto de datos 4	59	67	71
Conjunto de datos 5	46	52	60
Conjunto de datos 6	25	34	46
Conjunto de datos 7	28	37	38
Conjunto de datos 8	29	32	35
Conjunto de datos 9	24	37	39
Conjunto de datos 10	32	37	42
Conjunto de datos 11	33	36	52
Conjunto de datos 12	56	70	67

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

4.9 Memoria

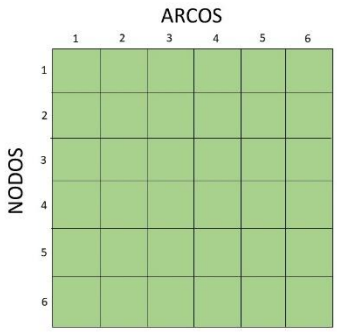
	Memoria (en MB)
Conjunto de datos 1	30
Conjunto de datos 2	17
Conjunto de datos 3	44
Conjunto de datos 4	34
Conjunto de datos 5	23
Conjunto de datos 6	28
Conjunto de datos 7	37
Conjunto de datos 8	31
Conjunto de datos 9	27
Conjunto de datos 10	43
Conjunto de datos 11	24
Conjunto de datos 12	19

5. TÍTULO DE LA SOLUCIÓN FINAL DISEÑADA

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

5.1 Estructura de datos

La estructura de datos principal utilizada es un grafo



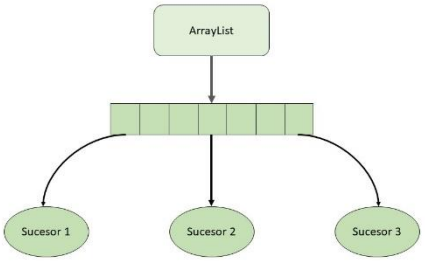
Gráfica 14: Grafo representado por matriz de adyacencia

Dentro de los algoritmos del proyecto se encuentran otras estructuras de datos que son:

- Arreglos

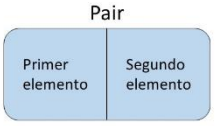
Gráfica 15: Representación gráfica de arreglo

- ArrayList



Gráfica 16: Representación gráfica de ArrayList

- Pair

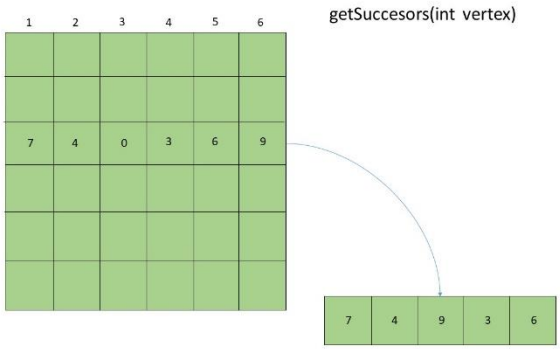


Gráfica 17: Representación gráfica de pair

5.2 Operaciones de la estructura de datos

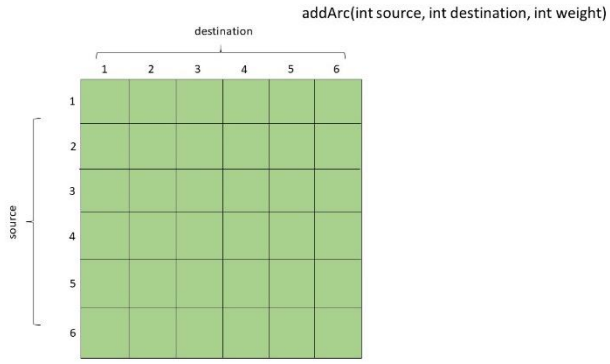
Las operaciones de la estructura de datos principal son:

- getSuccesors (int vertex): Esta operación devuelve una lista con los sucesores del nodo vertex.



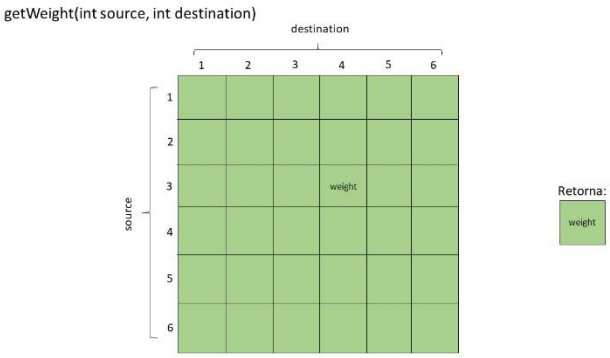
Gráfica 18: Método getSuccesors (int vertex).

- addArc (int source, int destination, int weight): Esta operación inserta un arco desde el nodo source al nodo destination con un peso weight.



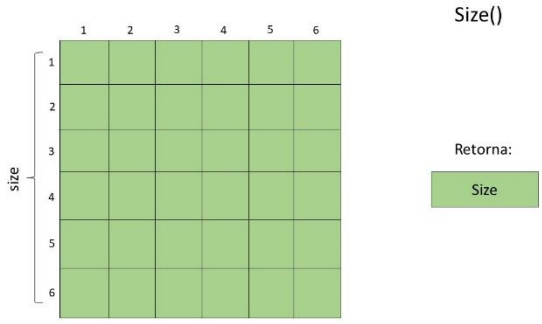
Gráfica 19: Método addArc (int source, int destination, int weight).

- getWeight (int source, int destination): Esta operación devuelve el peso del arco que hay entre source y destination.



Gráfica 20: Método getWeight(int source, int destination)

- size(): Esta operación retorna el tamaño del grafo que tenemos.



Grafica 21: Método size()

5.3 Criterios de diseño de la estructura de datos

Se decidió implementar el algoritmo usando un grafo implementado con una matriz de adyacencia, ya que al ser un grafo completo no dirigido se ocuparía la misma memoria que al utilizar listas de adyacencia, y al hacer este cambio se optimizan todas las operaciones de esta estructura de datos, quedando la mayoría en O(1).

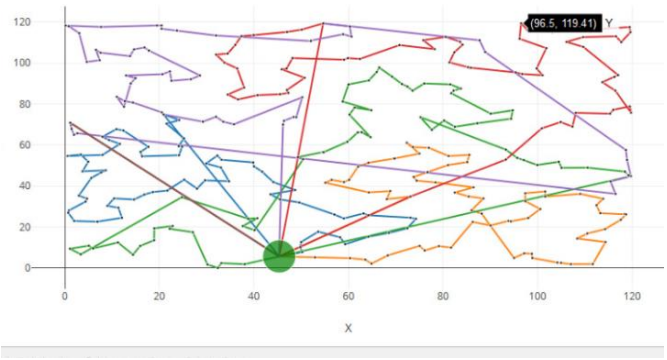
5.4 Análisis de Complejidad

Método	**Complejidad**
getSuccesors	O(n)
addArc	O(1)
getWeight	O(1)
Size	O(1)

Tabla 4: Tabla para reportar la complejidad, donde n es el número de nodos que hay en el grafo.

5.5 Algoritmo

El algoritmo inicia buscando el vecino más cercano, pero antes de moverse, analiza las restricciones, es decir, que el vehículo tenga suficiente batería como para desplazarse a ese punto y que le sobre para poder volver a la base o cargar y moverse a otro punto dependiendo del tiempo que esto le tome, de esta forma se va desplazando y va trazando rutas de manera que abarque todos los puntos, pero si en algún momento infringe alguna condición de las antes mencionadas, desecha las rutas almacenadas e intenta recorrer los puntos de una forma diferente hasta obtener una solución



Grafica 22: Solución al algoritmo

5.6 Cálculo de la complejidad del algoritmo

Subproblema	**Complejidad**
Leer archivo	O(n)
Calcular la batería	O(n.u)
Calcular el tiempo	O(1)
Total	**$O(n^2)$**

Tabla 5: Donde n es el número de nodos en el archivo,

5.7 Criterios de diseño de la estructura de datos

Se decidió implementar el algoritmo usando un grafo implementado con una matriz de adyacencia, ya que al ser un grafo completo no dirigido se ocuparía la misma memoria que al utilizar listas de adyacencia, y al hacer este cambio se optimizan todas las operaciones de esta estructura de datos, quedando la mayoría en O (1).

5.8 Tiempos de ejecución

	Mejor caso (en ms)	**Caso promedio (en ms)**	**Peor caso (en ms)**
Conjunto de datos 1	390	478	759
Conjunto de datos 2	111	144	245
Conjunto de datos 3	123	132	169
Conjunto de datos 4	78	98	112
Conjunto de datos 5	79	84	84
Conjunto de datos 6	73	116	117
Conjunto de datos 7	75	99	113
Conjunto de datos 8	79	85	130
Conjunto de datos 9	74	68	80
Conjunto de datos 10	78	82	87
Conjunto de datos 11	76	77	86
Conjunto de datos 12	75	107	113

Tabla 6: Tiempos de ejecución del algoritmo con diferentes conjuntos de datos.

5.9 Memoria

	Memoria (en MB)
Conjunto de datos 1	100
Conjunto de datos 2	115
Conjunto de datos 3	97
Conjunto de datos 4	86
Conjunto de datos 5	108
Conjunto de datos 6	79
Conjunto de datos 7	125
Conjunto de datos 8	110
Conjunto de datos 9	90
Conjunto de datos 10	97
Conjunto de datos 11	83
Conjunto de datos 12	108

Tabla 7: Tabla donde se presenta la memoria consumida por el algoritmo.

5.10 Análisis de resultados

Estructuras de datos	**Matriz de adyacencia**	**Listas de adyacencia**
Mejor tiempo de ejecución	73	24
Mejor memoria utilizada	79	17
Mejor complejidad en las operaciones de la estructura de datos	O(1)	O(n)

Tabla 8: Comparación de los resultados de las estructuras de datos utilizadas para este problema.

6.CONCLUSIONES

- El ruteo de vehículos eléctricos depende de muchos factores y métodos cuantitativos y por esto aún no se conoce la mejor forma de solucionarlo, pero en este proyecto han acontecido algunas aproximaciones a una solución dentro de lo que conforma el problema principal (selección de rutas), y se ha logrado que la implementación de las restricciones usadas para seleccionar rutas y las operaciones en general, sean lo más eficientes posibles en lo que respecta a tiempo de ejecución y memoria.

- Los resultados finales del proyecto fueron positivos debido a que se logró desarrollar las múltiples rutas necesarias para solucionar el problema de ruteo, además de esto, el algoritmo no consume mucha memoria y se ejecuta rápidamente, lo que permite en un futuro poder optimizar la solución incluyendo mayor cantidad de variables y factores que influyen en el desempeño al momento de recorrer todos los puntos.

- En la solución final se tuvo en cuenta la batería de los vehículos, se pasó de usar arreglos a usar matrices de adyacencia para guardar los grafos, y se conservó el algoritmo del vecino más cercano para trazar la ruta al igual que sucede con los trabajos relacionados, pero esta vez se tuvieron en cuenta varias restricciones enfocadas a acortar el tiempo total de la solución y escoger rutas más optimas que permitieron dar un mejor resultado.

6.1 Trabajos futuros

En el futuro es bueno aprovechar la eficiencia de la solución del problema base para aumentarle la cantidad de restricciones y usar el tiempo sobrante, teniendo en cuenta las situaciones que acontecen en la vida real, dejando a un lado las situaciones idealistas y usando condiciones de la vida real, como lo son la variación del tiempo de carga según el porcentaje de la batería, las diferentes superficies que tiene que atravesar el vehículo para llegar a los puntos, las condiciones físicas que afectan el tiempo del envío(el tipo de vehículo, las distintas velocidades que toma dentro del recorrido, subidas y bajadas o irregularidades del trayecto, clima, etc.) y las condiciones del tráfico. También es posible usar como entrada para el algoritmo, coordenadas extraídas de una base de datos con puntos de entrega que se necesitan en un sector específico.

AGRADECIMIENTOS

Esta investigación fue soportada parcialmente por institución ICETEX y programa ser pilo paga, beca ANDY EAFIT y beca SUIZA.

Nosotros agradecemos por la ayuda con la presentación e información sobre la dinámica del problema a José Alejandro Montoya, profesor del Departamento de Ingeniería de Producción de la Universidad EAFIT por los comentarios que nos hizo para entender la situación problema y encontrar una solución.

REFERENCIAS

1. Alfredo Olivera. Heurísticas para Problemas de Ruteo de Vehículos. Universidad de la República, Montevideo, Uruguay. Agosto 2004.

https://www.fing.edu.uy/inco/pedeciba/bibliote/reptec/TR0408.pdf

2. Bryan Salazar López. PROBLEMA DEL AGENTE VIAJERO – TSP. www.ingenieriaindustrialonline.com. Abril 2016.

https://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/investigaci%C3%B3nde-operaciones/problema-del-agente-viajero-tsp/

3. Wikipedia. Problema del viajante. Wikipedia. Enero 2018. https://es.wikipedia.org/wiki/Problema_del_viajante