# GESTIÓN DE RUTAS PARA VEHÍCULOS ELÉCTRICOS

Luisa María Vásquez Gómez
Universidad Eafit
Colombia
lmvasquezg@eafit.edu.co

Juan José Parra Díaz
Universidad Eafit
Colombia
jjparrad@eafit.edu.co

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

**RESUMEN**

El problema a tratar en este proyecto es el problema de enrutamiento de vehículos eléctricos, el cual consiste en el hallazgo de las mejores vías y rutas para este tipo de vehículos, de manera que estos logren llegar a su destino, sin quedarse sin energía y optimizando al máximo los costos de transporte. La importancia de este problema está principalmente en el sector de los envíos en un futuro no muy lejano, cuando se reemplacen las fuentes de energía obsoletas con las renovables, en donde cada vehículo contiene una cantidad de paquetes que debe entregar en puntos específicos de la ciudad, pero, al ser eléctricos, no cuentan con una duración de la batería muy óptima, por lo que tener las rutas ideales trazadas es un aspecto vital para el funcionamiento adecuado de estas. Hoy en día se pueden ver aplicaciones de este problema en el sector de la limpieza, específicamente con los camiones encargados de recoger la basura, cuyo objetivo es recorrer toda la ciudad de la manera más eficiente, sin dejar acumular la basura excesivamente.

**Palabras clave**

Energía renovable, vehículos eléctricos, ruta, carga lineal, minimización, mapa bidimensional, algoritmo, grafo, desplazamiento, eficiencia.

**Palabras clave de la clasificación de la ACM**

- Theory of computation → Design and analysis of algorithms → Graph algorithms analysis → Shortest paths
- Applied computing → Enterprise computing → Operations research
- Human-centered computing → Visualization → Accessibility

## 1. INTRODUCCIÓN

El acelerado desarrollo de las nuevas tecnologías ha traído consigo los problemas que conlleva usarlas de la manera más óptima, sin desperdiciar su potencial y teniendo en cuenta todas las limitaciones que estas traen. La situación con los vehículos eléctricos no es diferente, estos tienen un campo de acción muy grande y traen una mejora significativa, tanto a la sociedad como al medio ambiente, pero estos significan una duración menor de cada viaje debido a las limitaciones que traen el uso de baterías eléctricas en vez de combustibles.

En este documento se tratarán diferentes posibles soluciones que logren hacer que este problema se reduzca lo más posible, a través de algoritmos que analicen de múltiples maneras diferentes estructuras de datos en las que será guardad la información de cada "mapa" entregado.

## 2. PROBLEMA

El problema que será desarrollado en este documento es el siguiente: ¿Cómo encontrar una ruta óptima en la que un conjunto de vehículos eléctricos visite a múltiples clientes diferentes?

## 3. TRABAJOS RELACIONADOS

**3.1 Algoritmo de la Colonia de Hormigas (ACO)** El algoritmo de la colonia de hormigas es un algoritmo que tiene como objetivo imitar el comportamiento de estos insectos, los cuales se desplazan de un nodo inicio a uno fin dejando un camino, que con el tiempo se va desvaneciendo, a seguir por las otras hormigas que lo encuentren. Como el camino va desapareciendo, los caminos más largos desde el nodo inicio hasta el fin serán olvidados con el tiempo, mientras que por los cortos y eficientes, las hormigas lo reforzarán y seguirán usándolo.

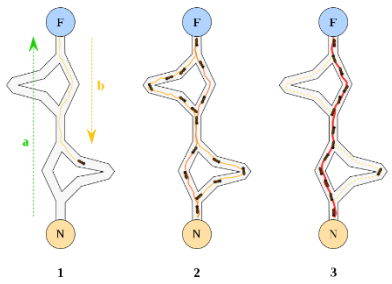



**Gráfico 1:** Explicación del Algoritmo de la Colonia de Hormigas.[1]

**3.2 Algoritmos genéticos**

Los algoritmos genéticos son un tipo de algoritmos que, como lo dice el nombre, "evolucionan" para buscar la mejor solución a un problema. Estos algoritmos funcionan

[1] Tomado de: https://commons.wikimedia.org/wiki/File:Aco_branches.svg

al enviarles un número de entradas, y, a partir de ellas, genera una salida aleatoria. Tras muchas salidas, el algoritmo elige las que hayan proporcionado los mejores resultados, las combina y las altera, para así seguir probando salidas y mezclando soluciones, hasta que se llegue a un resultado suficientemente bueno que proporcione la solución óptima al problema.

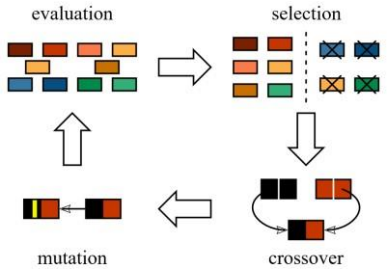



**Gráfico 2:** Explicación del funcionamiento de algoritmos genéticos.[2]

### 3.3 Heurística constructiva

Una heurística constructiva es un algoritmo que va construyendo una solución completa a partir de una solución vacía añadiendo a esta última, en cada iteración, la mejor elección local de un conjunto de posibles elecciones. Este método ha sido usado para solucionar problemas como el problema del vendedor viajero, sin embargo, a pesar de encontrar una solución completa, esta no es la más efectiva. A continuación se muestra un ejemplo de heurística constructiva eligiendo siempre el arco más corto a un nodo no visitado que sale de un nodo:

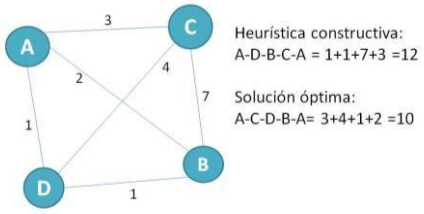



**Gráfico 3:** Recorrido de todos los nodos de un grafo con heurística constructiva.

### 3.4 Búsqueda tabú

Creada por Fred W. Glover, es un método de optimización matemática que genera iterativamente diferentes soluciones y las almacena en una estructura de memoria hasta cumplir determinada condición de parada, y al finalizar, define la solución final como la más óptima de las generadas. Por ejemplo, en el problema del vendedor viajero, se genera una solución a partir de una heurística constructiva anteriormente descrita y a partir de esta, se generan nuevas soluciones intercambiando aleatoriamente el orden en que se visitan las ciudades hasta cumplir con cierto número de iteraciones.

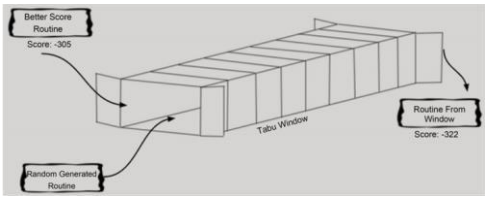
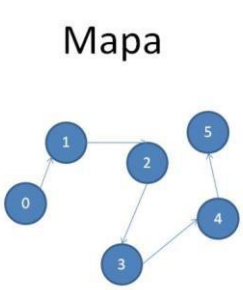



**Gráfico 4:** Representación gráfica de la búsqueda tabú[3]

## 4. SOLUCIÓN 1

### 4.1 Estructura de Datos

Para la resolución del problema, se optó por crear una estructura de datos en forma de grafo, es decir, que cada estación, depósito y cliente sean tipos de nodos diferentes, almacenando además, la información general de manera global, como lo es la velocidad, el tiempo máximo y demás variables. Además se utiliza una tabla de Hash para almacenar que nodos del grafo corresponden a clientes o a estaciones.




Tabla

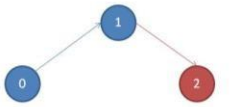
| Nodo | Tipo |
|---|---|
| 1 | Estación 1 |
| 2 | Cliente 1 |
| 3 | Cliente 2 |
| 4 | Estación 2 |

**Gráfico 5:** Estructuras de datos usadas en el algoritmo.

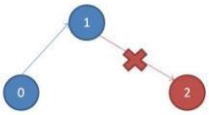
**4.2 Operaciones de la Estructura de Datos** En un grafo es fácil insertar un nodo, simplemente basta con conectarlo a alguno de los ya existentes para que pueda ser accesible, y para que pueda ser tomado en cuenta en la ruta, solo se agrega a la tabla de Hash para ser reconocido, caso en el cual la complejidad es O(1).

[2] Tomado de: http://www.jadecheng.com/au/coalhmm/optimization/

[3] Tomado de: http://file.scirp.org/Html/31730343_73004.htm




| Nodo | Tipo |
|---|---|
| 0 | Estación 1 |
| 1 | Cliente 1 |
| 2 | Estación 2 |




| Nodo | Tipo |
|---|---|
| 0 | Estación 1 |
| 1 | Cliente 1 |
| 2 | Estacion 2 |

**Gráfico 6:** Inserción y eliminación en la estructura de datos.

### 4.3 Criterios de diseño de la estructura de datos

Se optó por elegir un grafo con listas de adyacencia porque esta es una estructura de datos que permite el almacenamiento de diferentes puntos en el espacio, sin necesidad guardar coordenadas precisas, al reemplazar estas con peso entre nodos, ahorrando memoria de esta manera.

Igualmente, consideró que esta estructura era la más adecuada para darle solución a este problema porque esta, al ser basada en puntos y sus distancias entre ellos, es la que permite encontrar de manera más óptima recorridos entre dos puntos.

Al no ser este un problema de búsqueda o de almacenamiento de datos, no es adecuado el uso de árboles o estructuras similares, puesto que estos son pensados para encontrar de manera veloz un dato específico. Al intentar hacer recorridos entre dos puntos en una estructura como estas, se debería buscar en cada una de las ramas del árbol hasta encontrar el nodo siguiente al deseado haciendo que fuera ineficiente su uso.

### 4.4 Análisis de complejidad

La complejidad de las estructuras de datos usadas para las operaciones de inserción y eliminación son O(1), por lo que no consume tiempo adicional al realizar el cálculo de la ruta.

### 4.5 Algoritmo

- Trazar ruta directa del inicio a la estación más cercana.
- Trazar ruta directa de estación a estación hasta haberlas recorrido todas.
- Mirar clientes más cercanos a la ruta "inicioestación1" y agregarlos a la ruta, mirando el gasto de batería.
- Si al llegar a la estación, todavía faltan clientes de los cercanos, ir por ellos y volver a la misma estación.
- Repetir por cada ruta entre estaciones hasta llegar al objetivo

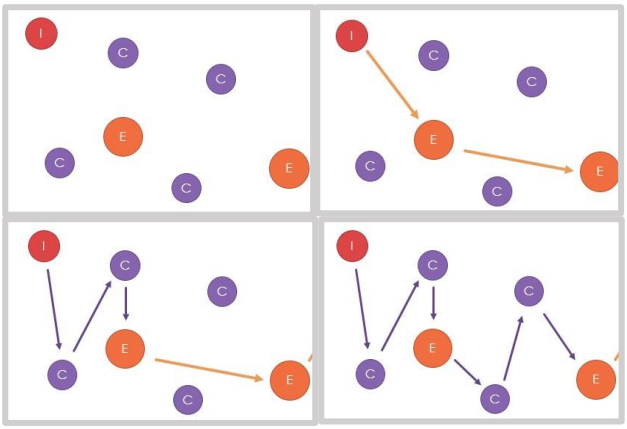



**Gráfico 7:** Representación gráfica del algoritmo

### 4.6 Cálculo de la complejidad del algoritmo

El algoritmo tiene una complejidad, en el peor de los casos de $O(n^3)$, debido a que el agrupamiento de los nodos y la posterior organización de estos requiere que se tengan que mirar todos los nodos varias veces.

### 4.7 Criterios de diseño algoritmo

Para diseñar el algoritmo nos enfocamos principalmente en "divide y vencerás ", de manera que asegurando la ruta más corta entre pequeños grupos de nodos, podemos lograr que la ruta total sea corta, en lugar de generar muchas rutas que pasen por todos los nodos y tomen más tiempo en ser generadas y posteriormente minimizadas.

### 4.8 Análisis de resultados

A pesar de que el algoritmo puede llegar a una ruta óptima, la complejidad causa que se demore demasiado calculándola, por lo que no es una solución efectiva a menos que se busque la manera de agrupar los nodos de una forma mucho más rápida.

## 5. SOLUCIÓN FINAL

### 5.1 Estructura de Datos

Para la resolución del problema, se optó por crear una estructura de datos basada en matrices, en donde toda la información leída se almacena en diferentes matrices que representan bloques de datos diferentes. Por ejemplo, los clientes leídos se almacenaban en una matriz única para estos, en donde cada fila representa un cliente diferente y cada columna un dato específico del mismo, como lo son el id o la estación de carga más cercana. Con las matrices creadas, se obtuvo que las complejidades de búsqueda fueran O(1), pues toda la información estaba estandarizada para todos los clientes.

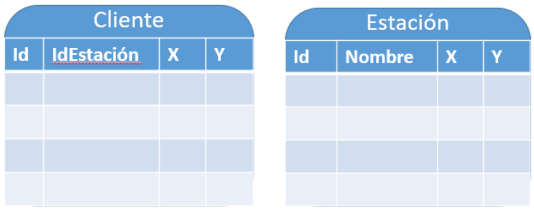



**Gráfico 8:** Representación gráfica de las estructuras de datos

### 5.2 Operaciones de la Estructura de Datos

La estructura de datos que se decidió utilizar tiene operaciones de inserción, eliminación y modificación de datos, las cuales, al tratarse de matrices, con tener el índice del dato que se desea cambiar, se pueden hacer con una complejidad O(1).

### 5.3 Criterios de diseño de la Estructura de Datos

Se optó por utilizar una estructura de datos basada en matrices, puesto a que estas permiten hacen operaciones en O(1), y, al tener que hacer tantas operaciones en este problema, se concluyó que esta era una de las opciones más viables.

De igual manera, esta estructura permite guardar una gran cantidad de información, por lo que al calcular la distancia hasta cualquier punto o cualquier otro tipo de variable que sea considerada oportuna para el cálculo de las rutas, es posible guardarla en la misma matriz, permitiendo así su consulta posterior para facilitar las operaciones realizadas más adelante.

### 5.4 Análisis de complejidad

La estructura de datos tiene operaciones que, al tratarse de matrices, con tener el índice del dato en el que se quiere hacer la operación, se puede hacer con una complejidad O(1).

Si por algún motivo no se sabe cuál es el dato que se desea modificar o no se tiene el índice de dónde se quiere insertar, la complejidad se volvería O(n), pero para evitar eso, se optó por mantener una variable que diga dónde se encuentran los índices pertinentes, por lo que este no sería el caso en esta solución.

### 5.5 Algoritmo

- Trazar ruta directa del inicio a cada estación.
- Trazar ruta directa de estación a cliente hasta haberlos recorrido todos.
- Si la batería no alcanza para visitar el siguiente cliente se devuelve al depósito cercano a recargar.
- Si recargando supera el tiempo máximo, se devuelve al depósito y se crea una nueva ruta para la estación a la que le faltan clientes por visitar.

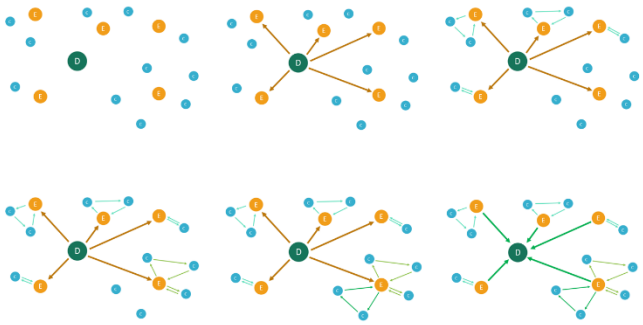


**Gráfico 9:** Representación gráfica del algoritmo.

### 5.6 Complejidad del Algoritmo

| Método | Complejidad (peor de los casos) | Caso promedio |
|---|---|---|
| Lectura y obtención de datos | O(n) | O(n) |
| Agrupación por estaciones | $O(n^2)$ | O(c x s) |
| Rutas por estación | $O(n^2)$ | $O(j^2)$ |
| Rutas totales | $O(n^2)$ | $O(s \times j^2)$ |
| Imprimir rutas | O(n) | O(c) |

**Tabla 1:** Tabla para reportar la complejidad.

En el cálculo de la complejidad "n" representa el total de nodos, "c" el total de clientes, "s" el total de estaciones y "j" el número de clientes por grupo. El cálculo en el peor de los casos está dado cuando solo hay un estación de carga, por lo que el único grupo que hay contiene todos los clientes.

### 5.7 Criterios de diseño del Algoritmo

El algoritmo se pensó algo parecido al algoritmo del vecino más cercano usado para TSP, sin embargo, al ver que la complejidad era de $O(n^2)$ quisimos reducirla en el caso promedio haciendo que para decidir ir a un nodo no haya que recorrer todos los existentes sino solo una parte, aunque en el peor de los casos ( solo una estación de carga) aún sería $O(n^2)$, a medida que aumentan las estaciones de carga disminuyen los nodos por mirar para decidir a donde ir, es decir, que "s" y "j" son inversamente proporcionales entre sí.

### 5.8 Tiempos de ejecución

| Set de datos | Mejor tiempo | Peor tiempo | Tiempo promedio |
|---|---|---|---|
| set0.txt | 47 ms | 82 ms | 61 ms |
| set1.txt | 220 ms | 288 ms | 249 ms |
| set5.txt | 218 ms | 285 ms | 238 ms |
| set8.txt | 200 ms | 309 ms | 237 ms |
| set10.txt | 217 ms | 301 ms | 248 ms |

**Tabla 2:** Tabla para reportar los tiempos de ejecución.

### 5.9 Memoria

| Set de datos | Memoria |
|---|---|
| set0.txt | 3.2 MB |
| set1.txt | 8.6 MB |
| set5.txt | 8.8 MB |
| set8.txt | 9.4 MB |
| set10.txt | 9.7 MB |

**Tabla 2:** Tabla para reportar memoria usada.

### 5.10 Análisis de resultados

Con este algoritmo observamos que, respecto al anteriormente diseñado, evitamos los viajes a la estación de carga que son innecesarios, ahorrándonos tiempo para visitar más clientes y hacer menos rutas, asimismo, logramos que el algoritmo diera respuesta de menos de 1 segundo para todos los sets de datos, lo cual nos da la proyección que en caso de tener cantidades más grandes de datos el tiempo seguiría siendo mínimo. Asimismo logramos optimizar el algoritmo para el caos promedio , permitiendo optimizar la cantidad de rutas según el número de estaciones.

## 6. CONCLUSIONES

El agente viajero es un problema al que se le han desarrollado numerosas soluciones abordadas desde puntos de vista diferentes, de tal manera que cada vez se encuentra una solución mejor, pero nunca la óptima. Este también es el caso de este trabajo, en donde se logró encontrar una solución a este problema, adaptado a los vehículos eléctricos en la ciudad de París, que, si bien no es la óptima, es una eficiente que representa una forma de abordar el problema diferente a las que ya se han hecho.

Para encontrar soluciones a los problemas del agente viajero es muy importante estar abierto a diferentes formas de solución, yendo desde un uso de estructuras de datos basadas en matrices, hasta de listas, sin excluir ninguna de estas soluciones potenciales, pues, al fin y al cabo, cada forma de mirar el problema permite encontrar diferentes caminos a una misma meta.

Por medio de este trabajo se logró llegar a una solución que permite concluir que los problemas de la informática no son solo encontrados en computadores y problemas de maratones, sino se pueden encontrar en la vida cotidiana también, como es el caso del enrutamiento de vehículos eléctricos.

## REFERENCIAS


1. Johann Dréo, 27 mayo, 2006. https://commons.wikimedia.org/wiki/File:Aco_branches.svg

2. Cheng Yu, Jade. Numerical Optimization, Genetic Algorithms. http://www.jadecheng.com/au/coalhmm/optimization/

3. Tanzila Islam, Zunayed Shahriar, Mohammad Anower Perves, Monirul Hasan. University Timetable Generator Using Tabu Search. http://file.scirp.org/Html/31730343_73004.htm

4. Dr Rong Qu. Constructive Heuristic Methods. http://www.cs.nott.ac.uk/~pszrq/files/2AIMconstruct.pdf