Heurística para ruteo por arcos con capacidades

Lina Reyes1

1 Universidad de Los Andes  
Bogotá, Colombia  
[l.reyesa@uniandes.edu.co](mailto:l.reyesa@uniandes.edu.co)

Resumen

Este documento presenta una heurística para solucionar el problema de ruteo de vehículos sobre arcos, con restricciones de su capacidad. El algoritmo propuesto consta de dos fases en la primera se utiliza el algoritmo Dijkstra para calcular la distancia más corta entre cada par de nodos y en la segunda un algoritmo constructivo, que encuentra el vecino más cercano con menor costo para barrerlo y encontrar la ruta con menor costo hasta cumplir la capacidad del vehículo. Para probar la eficiencia del método propuesto se analizan algunas instancias propuestas por (Golden & Wong, 1981). El análisis de los resultados muestra que la heurística en promedio presento un GAP del 32% frente a los resultados previos.

# Introducción

Los problemas de ruteo se pueden plantear como problemas de optimización combinatoria. Los procedimientos heurísticos son el desarrollo, entre los métodos aproximados, para resolver este tipo de problemas, ya que son un área prometedora para acercarse a encontrar soluciones factibles.

El problema de ruteo de vehículos sobre arcos, con restricciones de su capacidad (CARP- por sus siglas en inglés) es uno de los muchos problemas combinatorios recurrentes en la investigación operativa. En este problema, una serie de arcos, con diferentes demandas y costos, deben ser atendidos con una flota de vehículos capacitados. Como se ha mencionado anteriormente, la naturaleza combinatoria del problema supone un reto ya que su complejidad es NP-Hard (Golden & Wong, 1981).

Teniendo en cuenta esto, la idea del problema es buscar trazar las rutas de los vehículos, de tal modo que cada uno de ellos tenga su inicio y fin en el depósito, que cada arista sea atendida por un número de vehículos determinado, teniendo en cuenta que la cantidad de oferta no supere la capacidad de estos, minimizando el costo total de las rutas trazadas.

Este artículo presenta la descripción de la heurística constructiva implementada para el problema CARP aplicando el procedimiento de métodos constructivos en dos fases, el cual la estrategia principal es “divide y vencerás” donde se construye un recorrido que pase por todos los arcos obligatorios y se define a través de la metodología del vecino más cercano, ya en la segunda fase, el recorrido se divide en cinco rutas viables dependiendo de la capacidad de los vehículos. Teniendo en cuenta esto, el objetivo será evaluar los resultados de la heurística propuesta con las primeras siete instancias propuestas por (Golden & Wong, 1981) y otros autores.

# Descripción del problema

El CARP se define de la siguiente manera: Sea un grafo no dirigido, donde *V* es un conjunto de nodos, *E* es un conjunto de aristas y , con una demanda no negativa y un costo no negativo asignada a cada arista , (Golden & Wong, 1981).

A continuación, se presenta la formulación propuesta de programación matemática para el CARP. (Golden & Wong, 1981)

|  |  |
| --- | --- |
|  | (1) |

Sujeto a,

|  |  |  |
| --- | --- | --- |
|  | *p* = 1, …, *K,* | (2) |
|  |  | (3) |
|  | *p* = 1, …, *K,* | (4) |
|  |  | (5) |
|  |  | (6)  (7) |

La variable indica si el arco es atravesado por el vehículo . La variable indica si ya realizó el servicio en el arco por el vehículo. La es una variable de flujo que solo puede tomar valores positivos si

La Ecuación (1) muestra la función objetivo, la cual es minimizar la distancia total recorrida. La Ecuación (2) garantiza la continuidad de la ruta, la Ecuación (3) indica que cada arco con demanda positiva es atendido exactamente una vez, la Ecuación (4) garantiza que el arco puede ser atendido por el vehículo si solo cubre el arco . La capacidad del vehículo es restringida por la Ecuación (5). La Ecuación (6) prohíbe la formación de subrutas. Las restricciones de integralidad se dan en la Ecuación (7).

# Metodología de solución

# La metodología de solución propuesto está basada en un algoritmo heurístico llamado métodos constructivos en dos fases, que está planteado por (Dror, 2000) , el cual es un método constructivo que sirve para determinar rutas factibles en un grafo no dirigido con demanda en sus arcos. Para probar la eficiencia de este algoritmo se utilizan las instancias propuestas por (Golden & Wong, 1981) donde la medida de desempeño esta propuesta por el tiempo computacional y la calidad de la solución (brecha entre optimalidad y solución encontrada).

La codificación del algoritmo se divide en dos fases; algoritmo Dijkstra para calcular la distancia más corta entre cada par de nodos y el algoritmo constructivo que encuentra el vecino más cercano con menor costo para barrerlo y encontrar la ruta con menor costo hasta cumplir la capacidad del vehículo.

**Heurístico Dijkstra-VMS**

1. Nodos finalizados
2. Distancia = {}
3. Visitados = {}
4. Ruta {}
5. **For** i = 1 **to** nodos:
6. índice del nodo no finalizado con la distancia mínima
7. agregar nodo a conjunto de nodos finalizados
8. actualizar la distancia a los nodos adyacentes al nodo actual
9. nodo = **True**
10. **For** j = 1 **to** nodos:
11. **If** la distancia > 0 and nodo = false and dist[j] > dist[i] + inf
12. dist[j] = Distancia
13. **Return** Distancia
14. Copia\_Distancia = Distancia
15. **While not**
16. **If** capacidad = 5
17. Regrese camión al deposito
18. **Else**
19. **If nodo no es adyacente**
20. **RUTA =** Nodo actual + Distancia mínima (nodo siguiente) + arco no visitado
21. **Else**
22. **RUTA** = Nodo actual + arco adyacente
23. Return **Ruta**

**Algoritmo ruta más corta entre dos nodos mediante Dijkstra.**

1. En las primeras líneas se definen tres vectores: el primero es el vector de distancias el cual se inicializa en infinito, el segundo vector es el nodo fuente que permite que el parámetro de la función sea cero para poder encontrar el vector de las distancias menores a cada uno de los nodos y por último se crea un vector que guarde la información de si el nodo ya ha sido visitado y va cambiando a medida que cada nodo es visitado.
2. En la línea 12 se implementó un ciclo FOR para todos los vértices (este valor depende de la información que tenga cada instancia) donde se agrega el nodo que menor distancia tenga del nodo de partida que aún no haya sido visitado. La función **minDistance** va a estar compuesta por tres condiciones: el vector distancias, el vector que guarda la información si fue visitado el nodo y el número total de nodos. Para comprobar si la distancia del primer nodo es menor a la distancia actual y verificar que el nodo que no ha sido visitado se ejecuta una condición en la línea 15.
3. Actualiza la distancia mínima, y me retoma el índice del nodo que tiene la distancia más corta al nodo actual, guarda la información que este nodo ya fue visitado.
4. Una vez tiene esta información, se actualiza la distancia mínima del nuevo nodo y se establecen tres nuevas condiciones:

* Si en el grafo se encuentra un número que es mayor a cero, significa que es un nodo adyacente (conexión directa entre nodos).
* Que el arco no ha sido visitado.
* La distancia que existe hasta un arco que no ha sido visitado sea menor a la distancia que hay del nodo actual más la distancia de ese arco, si esto se cumple se puede actualizar los valores.

1. El algoritmo de Dijkstra arroja dos matrices, una de distancias mínimas entre cada par de nodos, y la segunda muestra el costo de los arcos con nodos adyacentes; en caso de que no haya una conexión directa entre cada par de nodos se agrega un cero a la solución de esta matriz.

**Algoritmo vecino más cercano (VMS).**

Este algoritmo va a realizar todo el recorrido de los arcos y después entregara la solución con los arcos de demanda suplida en grupos menores o iguales a la capacidad del vehículo, para regresar al depósito.

1. Se utiliza la copia de la matriz obtenida por el algoritmo Dijkstra mencionada anteriormente.
2. Si los arcos agregados no han suplido la demanda se suman a la capacidad del vehículo hasta un máximo de cinco, lo que quiere decir que ya recorrido las cinco calles y vuelve al depósito y suma cuánto cuesta ir del nodo actual al depósito a descargar.
3. Si aún no se cumple la capacidad del vehículo se busca entre los arcos adyacentes al nodo que aún no han suplido la demanda hasta que se hayan suplido todos los arcos de lo contrario busque en la otra copia de la matriz que contiene la información de la distancia mínima entre cada par de nodos en donde se va a dirigir al nodo más cercano agregando solo el costo adicional y se repita nuevamente la búsqueda de arcos por suplir.
4. Para identificar los nodos visitados se asigna el valor de cero a la copia de la matriz de Dijkstra y los elimina para continuar con el recorrido.
5. En la solución se agrega el valor del siguiente nodo al vector de visitado, se concatena al nodo actual y se le asigna el valor obteniendo como resultado el costo y las rutas que suplen la demanda y capacidad del vehículo.

# Experimentos computacionales y análisis de resultados

Las instancias con las cuales se evaluó el desempeño del algoritmos son las presentadas y utilizadas por (Golden & Wong, 1981). El algoritmo propuesto se implementó en Python y se ejecutó en una máquina Mac Book Air 10.1, chip M1, CPU de ocho núcleos con cuatro núcleos de rendimiento y cuatro de eficiencia.

Los resultados obtenidos se resumen en la Tabla 1. Se toma como referencia la mejor solución encontrada por Belenguer y Benavent (2003) y Golden y Wong (1981), se calcula el tiempo de cómputo en milisegundos para cada una de las instancias evaluadas y el porcentaje del gap del algoritmo propuesto.

*Tabla 1. Resultados obtenidos por el algoritmo Dijkstra- VMS*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | (Belenguer & Benavent, 2003) | (Golden & Wong, 1981) | Heurística  Propuesta | Tiempo  Heurística  Propuesta (ms) | Gap  Óptimo | Gap *Golden Wong* |
| gdb1 | 316 | 351 | 346 | 5,33 | 9% | -1% |
| gdb2 | 339 | 394 | 721 | 4,7 | 53% | 45% |
| gdb3 | 275 | 316 | 461 | 5,26 | 40% | 31% |
| gdb4 | 287 | 316 | 389 | 3,45 | 26% | 19% |
| gdb5 | 377 | 429 | 609 | 3,35 | 38% | 30% |
| gdb6 | 298 | 340 | 441 | 1,84 | 32% | 23% |
| gdb7 | 325 | 325 | 449 | 3,77 | 28% | 28% |

A continuación se presenta la ruta generada en la instancia número uno (gdb1). Donde el orden de visita de los arcos esta entendido como el paso de un nodo a otro, es decir como se muestra |S’| que representa el arco que utilizo para ir de un nodo a otro.

|S’| = [1, 12, 6, 7, 8, 10, 1, 2, 9, 11, 8, 6, 1, 4, 2, 3, 5, 6, 1, 7, 12, 5, 11, 10, 1, 10, 9, 4, 3, 1]

Ya que la capacidad del camión no debe superar cinco unidades se determina que arcos fueron atendidos y cuales se usaron de transición. Se utilizó un vector llamado “AT”, donde el 1 representa que se suple una demanda y el 0 que solo se usó como arco de transición, el tamaño de estos vectores va a estar relacionado en que |S’ |-1=|A| para este caso el arco (1, 10) fue suplido, por tanto, en la primera posición del vector A se ve reflejado con un 1.

| AT | = [1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,1, 0, 1, 1, 0, 1, 0]

El orden de visita de los arcos es importante para minimizar los costos asociados al paso de los arcos, entre menos se utilicen arcos de transición mejores soluciones se obtendrán ya que solo se recorren los arcos para suplir su demanda.

# Conclusiones y trabajos futuros

En este trabajo se propone un algoritmo para el Problema de Enrutamiento de Arcos Capacitados (CARP), los resultados obtenidos no supera los de los métodos presentados por otros autores, pero genera soluciones factibles a las instancias en un tiempo computacional razonable. En promedio presento un GAP de 32%. Este podría mejorar si se definen variables de decisión mucho más complejas y específicas.

En trabajos futuros, una de las cosas que podría mejorar el porcentaje del GAP, es que dentro del algoritmo se definan algunas decisiones aleatorias para tener un campo de soluciones más grande. Por otro lado, se puede definir un procedimiento de limitación de las rutas en cada paso de la iteración, es decir, optar por la ruta con mayor beneficio y rendimiento en cuanto la función objetivo.

Referencias

Alvarez Nuñez, M. F. (2013). Teoría de grafos.

Belenguer, J. M., & Benavent, E. (2003). A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 30(5), 705-728.

Dror, M. (Ed.). (2012). *Arc routing: theory, solutions, and applications*. Springer Science & Business Media.

Eiselt, H. A., Gendreau, M., & Laporte, G. (1995). Arc routing problems, part II: The rural postman problem. *Operations research*, 43(3), 399-414.

Golden, B. L., & Wong, R. T. (1981). Capacitated arc routing problems. *Networks*, *11*(3), 305-315.