Prescriptive Analytics: Heuristics for Decision Making

Universidad de los Andes

Wilmar Calderón – 201630701

2/03/2025

Tarea 2: Ruteo en arcos capacitado

1. Descripción del Problema y propuesta metodológica

El *Capacitated Arc Routing Problem (CARP)* es un problema que consiste en encontrar rutas óptimas para vehículos con capacidad limitada que deben atender demandas ubicadas en los arcos (calles o caminos) de una red. No obstante, este es un problema NP-Hard, lo que implica que encontrar la mejor ruta exacta en un tiempo razonable es muy difícil, especialmente para redes grandes. Por este motivo, se plantea el uso de heurística para encontrar soluciones aproximadas. Más específicamente, para este caso se busca establecer un método heurístico constructivo inicial que permita tener una solución factible inicial para el CARP.

Como aplicación de la heurística planteada, se realizará una evaluación para las 23 instancias de Golden *et al.* (1983) para evaluar el desempeño de la heurística planteada. Este ejercicio entonces buscara identificar el grafo descrito en la instancia (tome como ejemplo la Figura 1) y posteriormente, encontrar una solución factible inicial para el problema descrito.

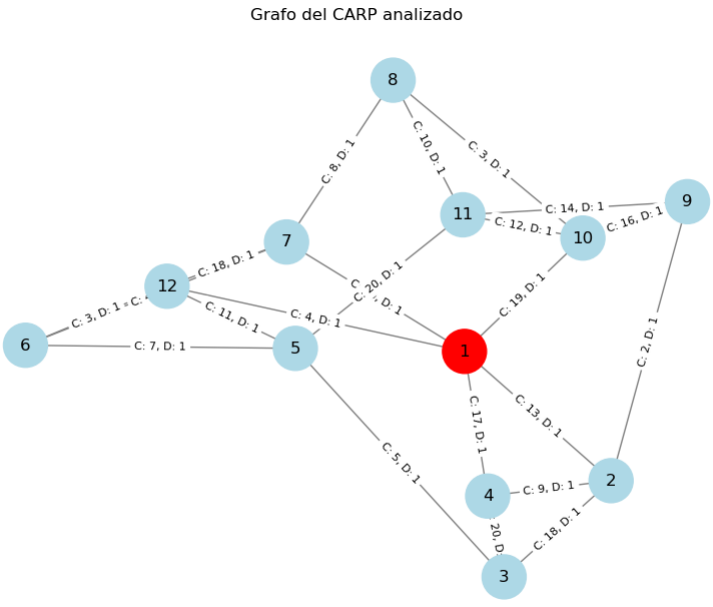

Figura 1 Ejemplo de un grafo a evaluar (caso de la instancia 1 de Golden et al. (1983)

2. Heurística

La heurística planteada para resolver el método CARP considera el uso de metodologías de estudio de grafos (como el algoritmo de Dijkstra) y la exploración del grafo de cada instancia para obtener una solución factible. La heurística planteada es una heurística constructiva que se puede describir mediante el siguiente pseudocódigo:

```
def implementación_CARP:
    G← Grafo de la red evaluada
    A← Matriz de adyacencia del grafo G (con los costos de los arcos)
    D← Matriz de adyacencia del grafo G (con las demandas almacenadas)
    por_visitar← arcos(G)
    bandera ←0 (bandera útil para recorrer otra ruta si los arcos obtenidos por una ruta más corta ya fueron accedidos)
    While len (por_visitar)>0, Do:
        b=|Di| (calcule cuanta demanda llega a cada nodo con la cardinalidad de los arcos aferentes a este nodo)
        nodoi= argmax(b) (Seleccione el arco que atiende más demanda)
        si (bandera==0):
            rutai← Dijkstra(depot, nodoi, depot)
        si no:
            rutai ←Dijkstra(depot, por_visitar[0], depot) (haga la ruta del depot, que pase por el arco que no se ha recorrido y vuelva al depot)
        for arco in rutai:
            si arco está en por_visitar:
                por_visitar.remove(arco)
                D[(arco)]=0 (actualiza la matriz de adyacencia con demandas para que en la siguiente iteración el algoritmo busque otro nodo)
                G[(arco)]["costo"]*=4 (actualiza la matriz de costos para que el arco visitado no sea más parte de la ruta más corta y así el algoritmo de Dijkstra busque una ruta por otros arcos (diferentes a los marcados))
            si no:
                bandera==1 (hecho para que busque en la próxima iteración marcar un arco como visitado)
```

Es importante notar que esta implementación considera las siguientes limitaciones/condiciones:

i. Todos los vehículos inician en un nodo marcado como **depósito** (*depot*) y vuelven a terminar en este nodo.
ii. Si un arco es recorrido una vez (por la $ruta_i$) y atiende la demanda, pero después, otra ruta ($ruta_{i+1}$) utiliza este arco para atender su demanda, el costo de recorrer el arco se le asigna a esta ruta $ruta_{i+1}$).
iii. Si un nodo solo se puede conectar por un arco con el depósito, la ruta tendrá el costo de atender la demanda y de volver al origen.

iv. Como primera implementación, no se considera la capacidad de los vehículos en la ruta.

Este algoritmo se implementó en Python y el cuaderno de Jupyter se adjunta a este reporte.

3. Análisis de resultados

Después de ejecutar el algoritmo planteado para las 23 instancias, los resultados de la implementación se comparan con las soluciones del problema desarrolladas por Belenguer & Benavent (2003) y se resumen en la Tabla 1.

Tabla 1 Resultados obtenidos de la implementación del CARP.

Belenguer & Benavent (2003)			**Implementación Propia**			**Comparación**	
Instancia	Costo	Tiempo CPU (s)	Instancia	Costo	Tiempo CPU (s)	Gap Costo (%)	Mejora Computacional (%)
gdb1	316	0.11	1	660	0.00	109%	-96%
gdb2	339	0.08	2	803	0.01	137%	-87%
gdb3	275	0.1	3	610	0.00	122%	-95%
gdb4	287	0.09	4	636	0.01	122%	-94%
gdb5	377	0.15	5	818	0.00	117%	-98%
gdb6	298	0.08	6	680	0.00	128%	-94%
gdb7	325	0.14	7	773	0.00	138%	-99%
gdb8	348	0.27	8	784	0.01	125%	-96%
gdb9	303	0.42	9	741	0.01	145%	-98%
gdb10	275	0.08	10	565	0.00	105%	-94%
gdb11	395	0.54	11	952	0.01	141%	-98%
gdb12	458	0.11	12	1179	0.00	157%	-96%
gdb13	538	0.16	13	871	0.00	62%	-97%
gdb14	100	0.05	14	192	0.00	92%	-94%
gdb15	58	0.04	15	82	0.00	41%	-95%
gdb16	127	0.08	16	214	0.00	69%	-94%
gdb17	91	0.04	17	121	0.01	33%	-87%
gdb18	164	0.06	18	415	0.00	153%	-92%
gdb19	55	0.04	19	80	0.00	45%	-100%
gdb20	121	0.13	20	207	0.01	71%	-96%
gdb21	156	0.08	21	238	0.01	53%	-93%
gdb22	200	0.06	22	307	0.01	54%	-90%
gdb23	233	0.09	23	369	0.01	58%	-90%

Con respecto a los resultados obtenidos, se observa que en todos los casos se obtienen soluciones factibles, así mismo, el método propuesto se asegura que todos los arcos sean recorridos. Con respecto a los resultados, se observa que, en promedio, las soluciones obtenidas por implementaciones propias están considerablemente alejadas de los óptimos de cada instancia. Sin embargo, se observa que el tiempo computacional de la

implementación propia es más rápida que la reportada previamente. En cuanto a los órdenes de magnitud de las diferencias, la implementación propia arroja en promedio soluciones con un valor doble al óptimo reportado por Belenguer y Benavent (2003), sin embargo, la implementación propia se demora la mitad que lo reportado en la literatura.

Considerando estos valores, se identifican conclusiones clave como son:

1. **El método propuesto encuentra una solución factible.** Debido a que este método se implemento como una solución constructiva, una modificación a la implementación se puede llevar a cabo para mejorar los valores obtenidos.
2. **Las respuestas encontradas sirven para obtener cotas superiores** de los resultados, por lo cual la modificación del algoritmo (o adición) puede tener un punto de comparación para garantizar una mejora.
3. **La implementación es eficiente computacionalmente**, el uso de operaciones matriciales y la modelación del problema como un grafo, permite reducir el costo computacional de la solución, lo cual permitió tener un mejor desempeño computacional que el previamente reportado.
4. **Ajustes adicionales:** La implementación realizada considera ciertas limitaciones (mencionadas arriba), una segunda versión de este algoritmo deberá atender esas limitaciones para resolver de manera más completa el CARP.
5. **Búsqueda de Óptimos Locales:** sería el siguiente paso a esta implementación para mejorar las soluciones encontradas.

Bibliografía

Belenguer J. & Benavent E., 2003, A cutting plane algorithm for the capacitated arc routing problem, *Computers & Operations Research*, Volume 30, Issue 5, https://doi.org/10.1016/S0305-0548(02)00046-1