Tarea 2: Implementación de Heurísticas Constructiva

Juan Esteban Aguirre Olarte – 202210404 – je.aguirreo1

Heurística:

Para resolver el problema de ruteo en arcos capacitados, CARP, se diseñó la siguiente heurística. Cabe resaltar que se usa el término ***deadhead*** para referirse a un arco que se atraviesa, es decir, se incluye en la ruta, pero no se sirve (o recoge demanda, según se vea).

Heurística CARP:

1. Cargar datos (ordenados por costo), pre-generar un grafo y pre-correr *Dijkstra*
2. Crear *unserved* con todas las aristas del archivo (previamente cargado y transformado a un grafo)
3. Inicializar *routes* como una lista vacía
4. Mientras existan aristas en *unserved*
    a. *current_load* se inicializa en 0
    b. *total_cost* se inicializa en 0
    c. *current_pos* comienza en el depot
    d. *path_log* se inicializa como vacío
    e. Mientras la ruta sea factible en términos de carga (True)
        i. *candidates* se inicializa como vacío
        ii. Para cada arista en *unserved*
            1. Si la capacidad, al añadir la arista, se excede NO considerar a la arista en *candidates*
            2. Elegir el camino para recorrer la arista "min(*Dijkstra*(*current_pos*, nodo inicial del arco), *Dijkstra*(*current_pos*, nodo final del arco))"
            3. **Criterio Greedy:** relación demanda/costo **"**demanda de la arista/(camino a recorrer + costo)"
            4. Agregar el candidato a *candidates*
        iii. Si no hay *candidates* parar (ir a la línea f)
        iv. Ordenar *candidates* según el C**riterio Greedy** y elegir el primer candidato por mayor relación demanda/costo
        v. Para llegar a dicho candidato se valdrán de caminos *deadhead* para alcanzar el nodo más cercano de dicha arista dados por *Dijkstra*, *total_cost* se actualiza, *path_log* se actualiza con los *deadhead´s* y *current_pos* se actualiza
        vi. *path_log* se actualiza con el arco del inicio de la arista hasta el final de la misma arista, *total_cost* se actualiza, *current_load* se actualiza con el la demanda del arco servido y *current_pos* se actualiza
        vii. Se remueve el arco de *unserved*
        viii. Volver a la línea e
    f. Volver al depósito con *Dijkstra*
    g. Añadir *path_log* a *routes*

Resultados:

Inst.	Costo Heurístico	Óptimo	Gap (%)	Tiempo (s)
gdb1	396	316	**25.32**	0.0121

gdb2	467	339	**37.76**	0.0175
gdb3	371	275	**34.91**	0.012
gdb4	357	287	**24.39**	0.009
gdb5	479	377	**27.06**	0.023
gdb6	332	298	**11.41**	0.012
gdb7	393	325	**20.92**	0.012
gdb8	543	348	**56.03**	0.0831
gdb9	455	303	**50.17**	0.1076
gdb10	332	275	**20.73**	0.019
gdb11	544	395	**37.72**	0.0875
gdb12	797	458	**74.02**	0.018
gdb13	712	536	**32.84**	0.0475
gdb14	134	100	**34**	0.022
gdb15	66	58	**13.79**	0.013
gdb16	139	127	**9.45**	0.0255
gdb17	95	91	**4.4**	0.023
gdb18	197	164	**20.12**	0.039
gdb19	71	55	**29.09**	0.004
gdb20	175	121	**44.63**	0.018
gdb21	199	156	**27.56**	0.0263
gdb22	230	200	**15**	0.063
gdb23	268	233	**15.02**	0.1196

Analisis de Resultados:

La complejidad de la heurística constructiva varía según las características de las instancias.

- El número de nodos afecta en el cálculo de caminos más cortos (usados para deadheading). Esto pues, en la implementación de la heurística se utiliza el algoritmo de Dijkstra en cada iteración para calcular distancias entre nodos. Se podría modificar el algoritmo usado para calcular la ruta más corta, véase Floyd o Bellman.

    - gdb8 tiene un tiempo de 0.0831s.
    - gdb23 tiene un tiempo mayor 0.1196s.

- El número de arcos definen la conectividad del grafo y la cantidad de rutas posibles. Por tanto, a mayor cantidad de arcos el espacio de búsqueda y el costo del preprocesamiento aumenta.

    - gdb23 tiene el tiempo más alto (0.1196s).
    - gdb19 tiene el tiempo más bajo (0.004s).

- Cada arco requerido debe ser servido exactamente una vez. Así pues, más arcos requeridos implican más iteraciones en el bucle principal.

  - gdb12 tiene un tiempo de 0.018s.
  - gdb19 tiene un tiempo de 0.004s.

- Una menor capacidad incrementa el número de rutas necesarias, lo que podría aumentar el tiempo. Aunque existen casos como el siguiente:

  - gdb8 (Q=27) y gdb11(50) tienen tiempos similares (0.0831s vs 0.0875s), lo que sugiere que Q no es determinante individualmente.

- Se encontró que, en promedio, para número de arcos menores a 30 el gap no suele subir de 13%, para número de arcos entre 30 y 45 el gap no sube de 30%, mientras que para más de 45 arcos el gap sube a más de 40%. No obstante, estas aproximaciones varían según los costes asociados a los arcos, pues si son muy homogéneos entre si estás conclusiones no son del todo validas (es decir, varía según la densidad del grafo).

Referencias:

Chen, Y., Hao, J., & Glover, F. (2016). A hybrid metaheuristic approach for the capacitated arc routing problem. European Journal Of Operational Research, 253(1), 25-39. https://doi.org/10.1016/j.ejor.2016.02.015
Letchford, A. N., & Oukil, A. (2008). Exploiting sparsity in pricing routines for the capacitated arc routing problem. Computers & Operations Research, 36(7), 2320-2327. https://doi.org/10.1016/j.cor.2008.09.008