Resolviendo el CARP con variantes del Path Scanning

Nicolas Robayo Pardo[1]

[1] Universidad de los Andes
Cra 1 Nº 18A - 12, 111711, Bogotá, Colombia
n.robayo10@uniandes.edu.co

Resumen

El problema de limpieza de calles con sal es una de las aplicaciones del problema CARP (Problema de ruteo de arcos capacitado). El objetivo de este problema es reducir la distancia recorrida por rutas capacitadas que surten la demanda de varios arcos partiendo y regresando de un único nodo. Esta tarea implementa y utiliza varios algoritmos heurísticos constructivos para el problema de CARP usando variantes de método de solución especifico "Path Scanning" para obtener una respuesta cercana al óptimo que mejore el Path Scanning original. Para establecer su utilidad en un contexto real se usó un set de instancias clásicas que demuestran una mejora importante sobre la heurística original del "Path Scanning".

1 Introducción

Partimos del problema de limpieza de calles con sal donde con un set de vehículos con capacidad igual limitada se quiere cubrir la demanda de sal de un set de calles conectadas entre si empezando y volviendo a un nodo de inicio o *depot*. Esto se quiere lograr minimizando el costo por recorrer las calles y respetando la capacidad de los vehículos disponibles. Este problema fue llamado por Golden y Wong (1981) [4] como el CARP o problema de ruteo de arcos capacitado. Este problema también es aplicable en varios problemas del mundo real como recolección de basuras, limpieza de nieve en las calles, inspección de tuberías, redes eléctricas, etc y a pesar de no ser tan famoso como el VRP o ruteo de nodos es un problema que ha sido abordado por distintos autores de forma exacta y aproximada. Dado que el CARP puede ser conceptualizado como un VRP se puede concluir que es de tipo NP-duro [4] y por lo tanto el uso de heurísticas se hace atractivo para llegar a una solución buena en un tiempo corto.

Una heurística constructiva es aquella que empieza con una solución vacía (infactible) y a través de un set de instrucciones construye una solución paso a paso hasta llegar a una solución factible. Golden y DeArmon propusieron en su trabajo de 1981 [3] tres heurísticas constructivas para llegar a una solución del CARP: Construct-Strike, Augment-Merge y Path Scanning. Estas heurísticas fueron probadas a través de instancias generadas de forma aleatoria por los mismos autores llamadas instancias GDB. Su trabajo ha sido ampliado por diversos autores que han adaptado estas heurísticas y las han adaptado creando nuevos algoritmos que han mejorado el estado del arte del problema. Las heurísticas constructivas son ampliamente investigadas para el CARP debido a que sus aplicaciones requieren de un tiempo de solución rápido y no necesitan de una afinación de parámetros adaptándose fácilmente a las variaciones del CARP [3].

Esta tarea está dividida en las siguientes secciones: Primero se realizará una descripción formal matemática del problema CARP junto con una revisión de literatura de heurísticas de tipo constructivo que han sido utilizadas para resolver este problema. Luego se procederá a enunciar el algoritmo usado para resolver este problema que mezcla soluciones de la literatura y posteriormente se usaran las instancias GDB para medir su rendimiento frente al óptimo. Finalmente se hará un análisis del performance del algoritmo y se enunciaran las conclusiones de la tarea.

2 Descripción

2.1 Definición del CARP

El problema de ruteo de arcos capacitados está definido a continuación:

Para un grafo no dirigido $G(A, N)$ con costos no negativos $c(n_i, n_j)$ y demandas no negativas $d(n_i, n_j)$ asignadas a cada arco $(n_i, n_j) \in A$. Hay un set de arcos $A_R \subseteq A$ con demandas positivas que contiene los arcos requeridos y una flota de vehículos con capacidad limitada D que es usada para atender la demanda de estos arcos. Si un vehículo pasa por un arco (n_i, n_j) este incrementa el costo de la solución por $c(n_i, n_j)$ y puede, si tiene la capacidad, atender la demanda del arco reduciendo la capacidad por $d(n_i, n_j)$.

Una ruta es una trayectoria de un vehículo en K, representada por la secuencia de arcos recorridos que empieza y termina en un nodo específico n_0 llamado el depósito. Un set de rutas es definido como factible si la suma de las demandas atendidas por cada vehículo no excede D y si la demanda de cada arco es atendida por exactamente un vehículo. El objetivo es encontrar un set de rutas factibles con mínimo costo.

2.2 Modelamiento Matemático

Se adjunta el modelamiento matemático expuesto por Kiilerich y Wøhlk (2017) [6].

2.2.1 Variables de decisión

$$y_{ij}^k = \begin{cases} 1 & si\ el\ arco\ (i,j) \in A_R\ es\ atendido\ por\ el\ vehículo\ k \in K \\ 0 & dlc \end{cases}$$

$$x_{ij}^k = \begin{cases} 1 si\ el\ arco\ (i,j) \in A\ \ es\ atravesado\ por\ el\ vehículo\ k \in K \\ 0 & dlc \end{cases}$$

$$x_{ji}^k = \begin{cases} 1 si\ el\ arco\ (i,j) \in A\ \ es\ atravesado\ por\ el\ vehículo\ k \in K \\ 0 & dlc \end{cases}$$

2.2.2 Función Objetivo

Se busca minimizar la distancia total recorrida.

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}\left(x_{ij}^k + x_{ji}^k\right)$$

2.2.3 Restricciones

(1) Todos los arcos con demandas son atendidos por exactamente por un vehículo.

$$\sum_{(i,j) \in A} y_{ij}^k = 1 \forall (i,j) \in A_R$$

(2) No se puede sobrepasar la capacidad del vehículo.

$$\sum_{(i,j) \in A} d_{ij} y_{ij}^k \leq D\ \forall k \in K$$

(3) Si un vehículo atiende un arco debe pasar por el.

$$x_{ij}^k + x_{ji}^k \geq y_{ij}^k \forall k \in K, (i,j) \in A_R$$

(4) Restricción de continuidad de ruta.

$$\sum_{j \in N:(i,j) \in A} \left(x_{ij}^k - x_{ji}^k\right) = 0\ \forall k \in K, i \in N$$

(5) Restricción de Eliminación de Subtours.

Esta restricción fuerza que todas las rutas empiecen y terminen en el Depot. Se generan con cada corrida de la optimización al detectarse subtours.

(6) Naturaleza de las variables.

$$x_{ij}^k, x_{ji}^k \in \{0,1\} \forall k \in K, (i,j) \in A$$
$$y_{ij}^k \in \{0,1\} \forall k \in K, (i,j) \in A_R$$

3 Revisión de Literatura

Como se menciono en la introducción, Golden et al. [3] introdujeron 3 heurísticas constructivas: Construct-Strike, Augment-Merge y Path Scanning. El Construct-Strike se basa en la remoción de ciclos del grafo basado en las restricciones del problema construyendo rutas que luego son conectadas al nodo inicial. Esto se repite hasta el que el grafo quede vacío y todos los nodos queden atendidos. El Augment-Merge inicializa todos los arcos con demanda con su propia ruta y luego iterativamente va uniendo rutas basado en las restricciones del problema y en las ganancias posibles en la función objetivo. Finalmente, el Path Scanning crea rutas seleccionando de forma golosa nodos con demanda adyacentes para añadirlos a la ruta basados en reglas de selección.

Posteriormente Pearn [7] introdujo una modificación al Path Scanning que aleatoriza la selección de nodos usada en el Path Scanning al escoger una regla al azar. Este algoritmo se corre m veces y se devuelve la mejor solución. Wølh (2005) [10] creo dos nuevos algoritmos: el primero llamado duplicación de nodos está basado en el algoritmo de limite inferior propuesto por Hirabayashi en el 92 y se basa en la construcción de rutas de Euler. El segundo algoritmo se llama el "Double Outer Scan Heuristic" el cual mezcla elementos del Path Scanning y el Augment-Merge creando rutas en las partes más alejadas del grafo utilizando el PS para unir nodos que lo lleven al depósito y luego usa el Augment Merge para unir rutas de forma que el problema se vuelva factible.

Otro tipo de modificación al PS propuesto por Santos et al. (2009) [9] que consiste en añadir al algoritmo una "regla de elipse" la cual se ejecuta cuando la capacidad remanente del vehículo baja a una cantidad fijada. Esta regla restringe la adición de nuevos arcos a la ruta a un subset de los arcos dentro de la elipse de arcos más cercanos. También existe otra modificación similar con una regla de eficiencia propuesta por Arakaki [1] que solo considera arcos "costo-eficientes" dentro de una elipse. Estas modificaciones resultan en una mejora frente a los algoritmos anteriormente planteados.

Por último, tenemos dos versiones aleatorizadas del PS. Reghioui et al. (2007) [8] usa el PS con una heurística tipo GRASP para resolver el CARP con ventanas de tiempo donde en cada iteración un arco es seleccionado aleatoriamente de una lista de candidatos restrictos factibles. Gonzalez Martín et al. (2011) [5] usa el PS asignando un peso en cada paso a los arcos seleccionados por el criterio actual con una distribución geométrica y luego procede a escoger uno aleatoriamente.

4 Metodología de solución

Usaremos como base el algoritmo de Path Scanning propuesto por Golden et al. [3] el cual consiste en construir una ruta a la vez usando un set de reglas que optimizan la selección de nodos. La ruta se forma escogiendo el nodo adyacente que tenga más promesa dada una regla hasta que la capacidad del vehículo se agote. Después de esto, una ruta más corta es trazada de vuelta al nodo inicial. Se hace una iteración para cada regla y se escoge las rutas hechas por la regla que tenga el menor coste total. Los autores propusieron 5 reglas de selección de arcos:

Dada una ruta cuyo último nodo es i, y capacidad es la capacidad remanente en cada paso:

1. El arco (n_i, n_j) cuyo $\frac{c(n_i,n_j)}{capacidad}$ es minimizado.
2. El arco (n_i, n_j) cuyo $\frac{c(n_i,n_j)}{capacidad}$ es maximizado.
3. La distancia $c(n_j, n_{depot})$ $es\ minimizada$
4. La distancia $c(n_j, n_{depot})$ $es\ maximizada$
5. Si $\frac{D}{2} \leq capacidad$, seguir la regla 4, de lo contrario seguir la regla 3.

Las reglas propuestas están diseñadas para alcanzar buenas soluciones tomando ciertas decisiones razonables. La primera busca maximizar la demanda escogida incrementando los beneficios a corto pazo mientras que la segunda se incurre en costos tempranamente para no tenerlos en cuenta en pasos futuros. La tercera obtiene rutas más pequeñas mientras que la cuarta crea rutas grandes que podría tener menores costos. La última regla tiene en cuenta la solución actual y trata de optimizar el tipo de demandas a incluir en la ruta.

Una forma de aleatorizar este algoritmo es el propuesto por Pearn (1898) [7] en el cual el PS se corre m veces y en cada selección de nodos, en vez de tener una regla de selección fija, se escoge una regla al azar con igual probabilidad seleccionando el mejor arco según esta regla para el set de arcos adyacentes al nodo actual. Esto permite explorar varios lugares del espacio de solución a través de criterios que son ellos mismos muy buenos. Pearn en su articulo afirma que con m=30 se obtienen soluciones similares al PS por lo que en esta tarea usaremos m=50 para ver su efecto en la solución sin que esto incremente demasiado el tiempo computacional con respecto a Pearn.

Los autores del algoritmo original omitieron explicar que hacer en la situación cuando no hay nodos adyacentes cuya demanda pueda ser servida. En este caso, según Pearn [7], se crea el camino más corto al arco con la demanda más pequeña (en el caso que pueda ser atendida, sino se devuelve al Depot). Aunque, si en ese camino encontramos un arco con demanda que puede ser atendida, se atiende y se vuelve a iterar por la regla de selección de nodos. Según el autor el algoritmo planteado de esta manera tiene complejidad $\mathcal{O}(n^3)$.

Otro camino propuesto en la situación donde no hay nodos adyacentes es la propuesta por Wøhlk [10] llamada Modified Path-Scanning Heuristic. En este algoritmo se mantiene el Path Scanning original excepto por la forma de encontrar un arco no adyacente. Cuando se llega a un nodo donde sus arcos adyacentes no puedan ser servidos se traza una ruta más corta al arco con demanda suficientemente pequeña para ser atendido y se atiende. Luego se continua con el método de selección de arcos original.

El Path Scanning se puede dividir entonces en dos partes fundamentales: la regla de selección de nodos y la regla de búsqueda de arcos no adyacentes. Para esta tarea probaremos variaciones con los métodos descritos anteriormente: usaremos las reglas originales del PS junto con su versión aleatorizada para la selección de nodos y para cada uno de estos veremos su desempeño con la regla de búsqueda de nodos no adyacentes (o estrategia de terminación) original y modificada.

Se creó entonces un programa que ejecuta el algoritmo en su forma tradicional y pseudo-aleatorizada y que además se pueda escoger la estrategia de búsquedas de arcos no adyacentes entre la tradicional y la modificada. Si un problema al ser resuelto de la forma tradicional (determinista) resulta ser infactible se vuelve a solucionar con el modo aleatorizado. Esto resulta ser algo novedoso entre las investigaciones revisadas. Si ya estaba siendo resuelto de manera aleatorizada se declara infactible.

4.1 Pseudocódigo

Input: G(A,N): Grafo de la instancia; D: Capacidad de los Vehículos; c: vector distancias; d: vector demandas; SP: matriz de costo de camino mas corto. RAND: Si se ejecuta el modo aleatorio. EstrategiaModificada: Si la estrategia de salida es la modificada

Output: bestSol: La mejor Solución.

Start
bestSol ← ∅
if rand **then**: k = 300, **else**: k = 5
for it = 1 to k **do**
  sol(i) ← ∅;
  $A_R \leftarrow A|\ d(i) > 0 \forall\ i \in A$
  for vehiculo = 1 **to** Vehiculos
    $R \leftarrow \emptyset; libre \leftarrow D$
    $R + v_0;\ v_h \leftarrow v_0$
    while espacio:
      Determinar arcos adyacentes $F \subseteq A_R | d(a) \leq libre\ \forall a \in A_R$ //
      if $F = \emptyset$ **then**
        if $A_R = \emptyset$ or $\{i|d(i) \leq libre, i \in A_R\} = \emptyset$ **then**
          sol(i) ← $sol(i) \cup (R \cup rutaMasCorta\{v_0\})$ $Fin\ de\ la\ ruta$
          espacio ← False
        else
          if estrategiaModificada **then**
            Seleccionar $(n_i, n_j)^* \in A_R$ más cercano y añadir a R
            $libre \leftarrow libre - d(v_i, v_j); A_R \backslash (n_i, n_j)$
          else
            Seleccionar $(n_i, n_j)^* \in A_R$ con menor demanda
            if rutaMasCorta$(n_h, (n_i, n_j)^*)$ pasa por arco i con demanda **then**
              Añadir i a R
              libre ← $libre - d(i)\ ; A_R \backslash i$
            else
              Añadir $(n_i, n_j)^*$ a R
              libre ← $libre - d(v_i, v_j)$
      else
        if RAND **then:** criterio = rand(1,5), **else:** criterio = it
          escoger mejor arco i∈F según criterio y añadir a R
          libre ← $libre - d(i)\ ; A_R \backslash i$
  if (cost(sol(i))<cost(bestSol)) or bestSol =∅) and sol is factible **then**
    bestSol ← sol

if bestSol =∅ **then**
  if random == True **then: Error** Problema Infactible # End
    rand ← True
    GO TO Start
return (bestSol)
End

4.2 Detalles de Implementación

Se implementaron los algoritmos en Python usando una conceptualización de clases para priorizar el ahorro de memoria. La implementación recibe como parámetros el archivo de la instancia a solucionar, si se desea usar el modo aleatorio (1, 0 d.l.c) y la estrategia de terminación a usar donde 1 es la modificada y 2 la original. Por defecto se soluciona de forma determinística con la estrategia de terminación modificada. Se debe correr en una terminal de la siguiente forma:

```
# python HW1.py “nombre_de_la_instancia.dat” aleatorización? Estrategia_terminacion VERBOSE(T/F)
# para correr el modo aleatorizado con estategia de terminación modificada sin imprimir las rutas se usaría:
$ python HW1.py “gdb/gdb1.dat” 1 1 0
```

Para calcular los caminos más cortos se utilizo el código de David Eppstein (2002) [2] el cual implemento un algoritmo de Dijkstra usando diccionarios de prioridad.

También se genero un archivo ejecutable para equipos que no tengan Python instalado para equipos tipo Linux. Las instancias se pueden correr con un archivo .sh para correr todos los experimentos acá presentados. Se incluyo un README en los soportes de esta tarea que indican como ejecutar estos archivos.

5 Experimentación y Análisis de Resultados

Para experimentar con los algoritmos propuestos se usaron las instancias GDB propuestos por Golden et al. [3]. Estas fueron corridas en un procesador dual core “Broadwell” 2.7 GHz Intel “Core i5” (5275U) con 8 GB 1866 MHz LPDDR3 SDRAM corriendo macOS Catalina 10.15.6 con Python 3.6. En el caso de la estrategia de selección de nodos pseudoaleatoria se usaron 500 iteraciones para obtener el promedio de las soluciones.

5.1 Instancias Utilizadas

Para determinar la calidad de las soluciones creadas por el algoritmo se utilizaron las instancias usadas por Golden [3] que llamaremos GDB. Estas varían entre 7 y 27 nodos y 11 a 55 arcos y son instancias relativamente pequeñas creadas de forma aleatoria. Estas instancias han sido utilizadas por una gran cantidad de artículos que estudian el CARP dado que fueron de las primeras en estar disponibles y por esto también se encuentra su valor óptimo en la literatura. Se han descargado del sitio web de la Universidad de Valencia[1].

Ilustración 1: Número de arcos requeridos de las instancias GDB

[1] https://www.uv.es/belengue/carp.html

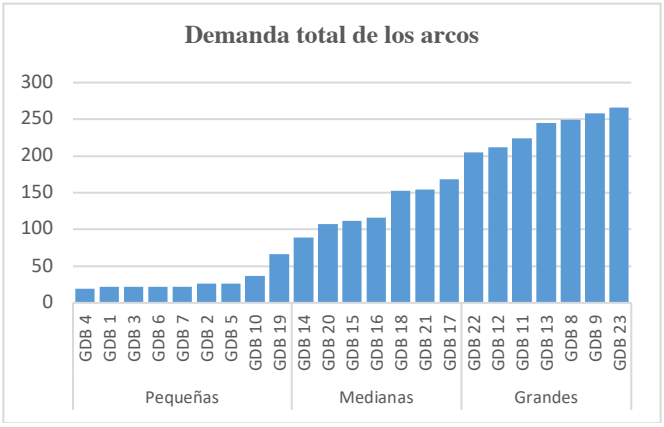

Ilustración 2: Demanda Total de los Arcos de las instancias GDB

Se dividieron las instancias entre pequeñas, medianas y grandes usando como criterio de separación la demanda total en los arcos de cada instancia dado que es el parámetro con mayor varianza entre las instancias.

Instancia	Vértices	Arcos	Demanda	Vehículos	Capacidad	Saturación	Óptimo
GDB 1	12	22	22	5	5	88%	316
GDB 2	12	26	26	6	5	87%	339
GDB 3	12	22	22	5	5	88%	275
GDB 4	11	19	19	4	5	95%	287
GDB 5	13	26	26	6	5	87%	377
GDB 6	12	22	22	5	5	88%	298
GDB 7	12	22	22	5	5	88%	325
GDB 8	27	46	249	10	27	92%	348
GDB 9	27	51	258	10	27	96%	303
GDB 10	12	25	37	4	10	93%	275
GDB 11	22	45	224	5	50	90%	395
GDB 12	13	23	212	7	35	87%	458
GDB 13	10	28	245	6	41	100%	536
GDB 14	7	21	89	5	21	85%	100
GDB 15	7	21	112	4	37	76%	58
GDB 16	8	28	116	5	24	97%	127
GDB 17	8	28	168	5	41	82%	91
GDB 18	9	36	153	5	37	83%	164
GDB 19	8	11	66	3	27	81%	55
GDB 20	11	22	107	4	27	99%	121
GDB 21	11	33	154	6	27	95%	156
GDB 22	11	44	205	8	27	95%	200
GDB 23	11	55	266	10	27	99%	233

Tabla 1: Características de las Instancias GDB

5.2 Resultados de las instancias

Se probo el algoritmo con cada posible combinación de las modificaciones del Path Scanning. A continuación, mostramos los resultados de la solución y el tiempo de corrida. Hay que notar que para la

instancia 13 los algoritmos determinísticos no encontraron una solución factible por lo que se activo el algoritmo aleatorio.

5.2.1 Determinístico con estrategia de salida original

Instancia	Solución	GAP	Tiempo (ms)
GDB 1	337	7%	4,768944
GDB 2	382	13%	5,171957
GDB 3	311	13%	5,31151
GDB 4	309	8%	5,85806
GDB 5	413	10%	6,814444
GDB 6	317	6%	6,137967
GDB 7	365	12%	6,538598
GDB 8	465	34%	27,716075
GDB 9	394	30%	12,349781
GDB 10	314	14%	9,476244
GDB 11	493	25%	9,50455
GDB 12	642	40%	4,029239
GDB 13	**573**	**7%**	**91,785818**
GDB 14	122	22%	3,296234
GDB 15	64	10%	5,817491
GDB 16	137	8%	13,997071
GDB 17	95	4%	4,312832
GDB 18	193	18%	9,131416
GDB 19	57	4%	3,527059
GDB 20	129	7%	3,08566
GDB 21	168	8%	6,040731
GDB 22	209	4%	9,236578
GDB 23	249	7%	63,844941

Tabla 2: Resultados Heurística PS Determinístico con estrategia de salida original

5.2.2 Determinístico con estrategia de salida modificada

Instancia	Solución	GAP	Tiempo (ms)
GDB 1	323	2%	10,773832
GDB 2	353	4%	6,179383
GDB 3	297	8%	6,2708
GDB 4	287	0%	5,232476
GDB 5	413	10%	6,905791
GDB 6	317	6%	7,07578
GDB 7	349	7%	6,966442
GDB 8	450	29%	38,427942
GDB 9	397	31%	79,36792
GDB 10	314	14%	6,188631
GDB 11	446	13%	20,892787
GDB 12	628	37%	10,14583

GDB 13	**594**	**11%**	**181,488542**
GDB 14	121	21%	6,134769
GDB 15	62	7%	4,312433
GDB 16	137	8%	5,29501
GDB 17	93	2%	5,763487
GDB 18	188	15%	8,924478
GDB 19	57	4%	4,569098
GDB 20	123	2%	6,421528
GDB 21	170	9%	7,515023
GDB 22	211	5%	9,543396
GDB 23	249	7%	46,899201

Tabla 3: Resultados Heurística Determinístico con estrategia de salida modificada

5.2.3 Aleatorizado con k =50 con método de salida tradicional

Instancia	Solución Promedio	GAP	Tiempo promedio (ms)
GDB 1	328,84	4%	30,3
GDB 2	362,343333	7%	30,4
GDB 3	292,016667	6%	18,0
GDB 4	311,546667	9%	15,7
GDB 5	399,053333	6%	20,8
GDB 6	320,616667	8%	17,2
GDB 7	346,34	7%	25,0
GDB 8	446,193333	28%	65,1
GDB 9	374,393333	24%	70,9
GDB 10	302,193333	10%	21,2
GDB 11	470,04	19%	44,3
GDB 12	561,826667	23%	23,9
GDB 13	585,123333	9%	21,6
GDB 14	108,66	9%	16,3
GDB 15	60,9266667	5%	15,8
GDB 16	132,25	4%	25,0
GDB 17	91,32	0%	22,0
GDB 18	180,013333	10%	32,1
GDB 19	55,22	0%	15,4
GDB 20	125,493333	4%	57,3
GDB 21	166,843333	7%	63,0
GDB 22	207,043333	4%	55,5
GDB 23	247,38	6%	63,7

Tabla 4: Resultados Heurística Aleatorizado con k =50 con método de salida tradicional

5.2.4 Aleatorizado con k=50 y metodo de salida modificado

Instancia	Solución Promedio	GAP	Tiempo promedio (ms)
GDB 1	322,53	2%	34,9

GDB 2	360,4	6%	38,1
GDB 3	291	6%	26,3
GDB 4	292	2%	14,6
GDB 5	399	6%	20,1
GDB 6	319,85	7%	16,5
GDB 7	334	3%	17,7
GDB 8	422	21%	55,4
GDB 9	362	19%	81,6
GDB 10	290,36	6%	21,5
GDB 11	445	13%	48,8
GDB 12	572	25%	23,8
GDB 13	582	9%	22,7
GDB 14	108,41	8%	16,3
GDB 15	61	5%	18,6
GDB 16	131	3%	24,2
GDB 17	91	0%	21,0
GDB 18	180	10%	33,4
GDB 19	55	0%	14,6
GDB 20	125	3%	35,9
GDB 21	166	7%	31,1
GDB 22	206	3%	39,2
GDB 23	246	6%	50,5

Tabla 5: Resultados Heurística Aleatorizado con k=50 y metodo de salida modificado

5.3 Comparación con el óptimo y análisis de resultados

Al analizar los resultados, notamos que el mejor algoritmo para todos los tipos de instancia (pequeño, mediano, grande) es el PS Aleatorizado con estrategia de terminación modificada. Para las partes fundamentales del algoritmo: la escogencia de nodos y la estrategia de terminación vemos que el modo aleatorizado de la escogencia de nodos domina al modo determinista. De igual forma el método de terminación modificado domina al método original.

En términos de tiempos computacionales, el método de terminación modificado tiene un tiempo de ejecución casi igual (aproximadamente +3ms). Por otra parte, la selección de nodos pseudoaleatoria incrementa notoriamente los tiempos de ejecución del algoritmo con respecto al modo determinista, aunque entrega mejores resultados. En promedio el modo aleatorizado tiene tiempos de ejecución 4.5 veces más largos que el modo determinista.

Entre los resultados de las instancias pequeñas, medianas y grandes notamos que la diferencia entre el GAP de los métodos de solución se va incrementando al pasar a instancias más grandes. En estas los métodos aleatorizados se tornan más competitivos frente a los deterministas a diferencia de las instancias pequeñas donde no hay mucha diferencia entre las distintas variantes del PS.

Para las primeras siete instancias, el GAP más alto obtenido entre estas fue del 7% y se encontró el óptimo una vez lo cual posiciona el algoritmo implementado como una solución prometedora al problema del CARP que podría ser usada como insumo para una heurística de búsqueda local que encuentre el óptimo. Aunque el modo aleatorizado incremente el tiempo de ejecución de forma importante mejora

la solución en promedio 4 puntos por debajo del GAP de la solución determinista. Esta diferencia podría llegar a ser importante cuando el método constructivo sea el único en ser usado y por el contrario se podría usar la solución determinista para proceder luego con una búsqueda local.

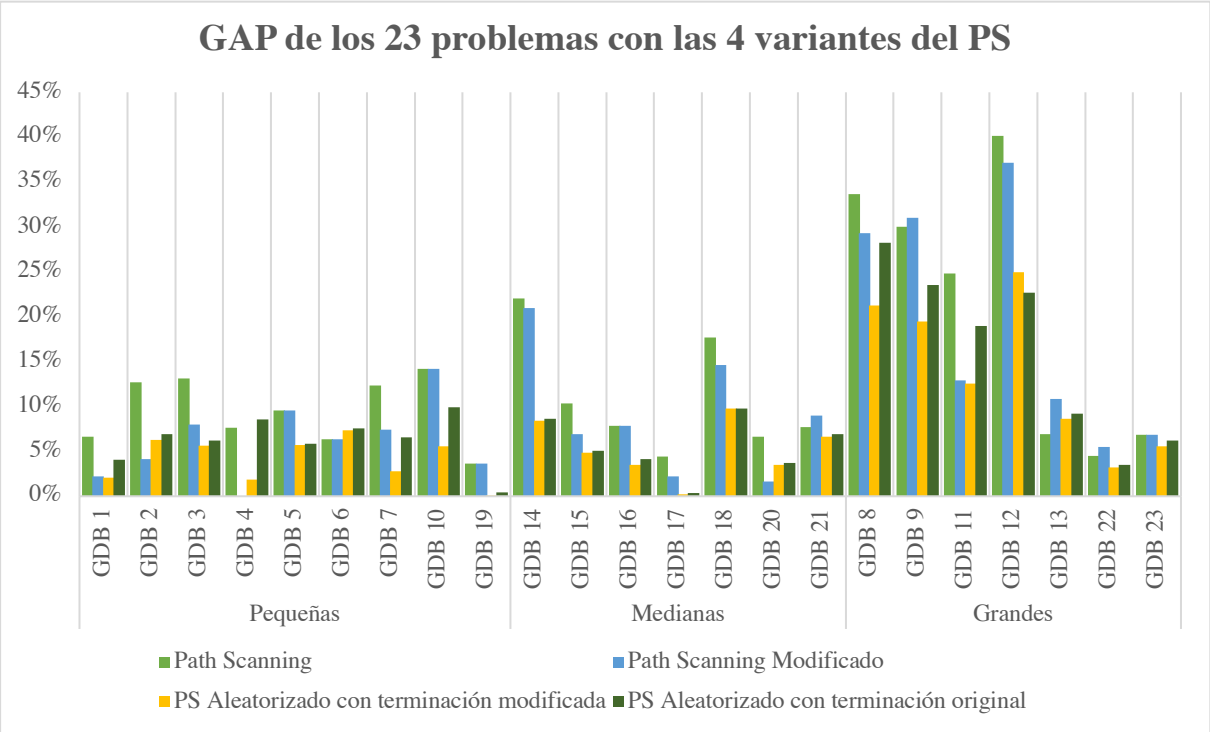

Ilustración 3 GAP por tipo de algoritmo para todas las instancias

GAP (promedio) Instancias	Path Scanning	Path Scanning Modificado	PS Aleatorizado con terminación modificada	PS Aleatorizado con terminación original
Pequeñas	10%	6%	4%	6%
Medianas	11%	9%	5%	6%
Grandes	21%	19%	14%	16%

Tabla 6: GAP de las variantes del PS por tipo de instancia

Tiempo Instancias	Path Scanning	Path Scanning Modificado	PS Aleatorizado con terminación modificada	PS Aleatorizado con terminación original
Pequeñas	5,5ms	9,7ms	22,7ms	21,6ms
Medianas	3,7ms	5,9ms	25,8ms	33,1ms
Grandes	8,8ms	11,6ms	46,0ms	49,3ms

Tabla 7: Tiempo Computacional de las variantes del PS por tipo de instancia

6 Conclusiones

- La diferencia en GAP en las instancias probadas entre el método determinista de selección de nodos y el método aleatorio esta entre el 3 y 5% y su diferencia en tiempo computacional crece de forma no lineal con respecto a la complejidad del problema.
- La diferencia en GAP entre el método original de selección de nodos no adyacentes o estrategia

de terminación original y el método modificado está entre 2 y 3% en las instancias probadas y no incrementa de forma importante el tiempo computacional.

- La mejor modificación del Path Scanning usa el método aleatorio para la selección de nodos y el método modificado para la selección de nodos no adyacentes. A pesar de esto, el tiempo computacional crece de forma no lineal por lo que se recomienda usar esta modificación cuando sea la única en ser utilizada
- La solución retornada por método determinista con estrategia de terminación modificada puede ser usada como insumo para un algoritmo de búsqueda local ya que su tiempo de computo es muy bajo con respecto al método aleatorio y su GAP con respecto a este es solo 2-4% mayor.

Referencias

[1] Arakaki, R., Usberti, F. An efficiency-based path-scanning heuristic for the capacitated arc routing problem. Comput. Oper. Res. 103: 208-295 (2018).

[2] Eppstein. D. Dijkstra's Algorithm For Shortest Paths. https://code.activestate.com/recipes/577343-dijkstras-algorithm-for-shortest-paths/. (2002)

[3] Golden, B., DeArmon, J. Computational Experiments with Algotithms for a Class of Routing Problem. Computer Ops Res 10(1): 47-59 (1983).

[4] Golden, B., Wong, R. Capacitated Arc Routing Problems. Networks 11(3): 305-315 (1981).

[5] González, M., Perez, J., et al. A hybrid algorithm combining path scanning and biased random sampling for the Arc Routing Problem. Proceedings of the 18th RCRA Workshop on, 46–54 (2011).

[6] Kiilerich, L., Wøhlk, S. New large-scale data instances for CARP and new variations of CARP. Information Systems And Operational Research. 56(1): 1-32 (2018).

[7] Pearn, W,L,. Approximate Solutions for the Capacitated Arc Routing Problem. Computers & Operations Research, 16(6): 589–600 (1989).

[8] Reghioui, M., Prins, C., and Labadi, N. Grasp with path relinking for the capacitated arc routing problem with time windows. En Giacobini, M., editor, Applications of Evolutionary Computing. 722–731 (2007).

[9] Santos, L., Coutinho-Rodrigues, J., Current, J.R., An improved heuristic for the capacitated arc routing problem. Comput. Oper. Res. 36 (9): 2632–2637 (2009).

[10] Wøhlk, S. Contributions to Arc Routing. PhD thesis, University of Southern Denmark. (2005).