DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 3

Modelando problemas problemas con grafos

9 de diciembre de 2018 — Algoritmos y Estructura de Datos III

Integrante	LU	Correo electrónico
Buceta, Diego	001/17	diegobuceta35@gmail.com
Springhart, Gonzalo	318/17	glspringhart@gmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

http://www.exactas.uba.ar

1. Introducción al problema

El VRP problem o problema de ruteo de vehículos es un nombre que se le ha dado a toda una serie de problemas en las que se busca diseñar rutas para una flota de una clase de vehículos para satisfacer las demandas de un conjunto de clientes. La resolución de este problema es central para el reparto de bienes y servicios alrededor del mundo. Dentro de este problema, existen diferentes variantes que dependen de una serie de cosas tales como: los bienes a transportar, la calidad del servicio que se busca brindar, el tipo de clientes a satisfacer, el tipo de vehículos, la extensión física del servicio que se busca brindar, entre otras cosas.
Alguna de las complicaciones que pueden existir a la hora de diseñar estas rutas consisten en clientes sólo disponibles en determinados rangos horarios, planificación de rutas en múltiples días, incompatibilidad de un tipo de vehículo con algún cliente, diferentes cantidades de vehículos en cada depósito o punto de partida, etc.
Sin embargo, en todos los casos se busca resolver el problema, cualquiera sea su caso particular, al menor costo, donde el costo puede ser representado mediante alguna función objetivo que expresa el parámetro importante a minimizar al resolver el problema. En algunas situaciones esto podría ser el costo de k vehículos realizando rutas de entrega. En otras podría llegar a necesitarse minimizar la cantidad de vehículos que realizan las entregas y no el costo de las rutas en particular. Es decir, estamos ante un problemas que puede particularizarse para muchas situaciones en general.

Sin embargo, uno de los mas estudiados de esta familia de problemas es el de Ruteo de Vehículos con Capacidad (CVRP por sus siglas en inglés). En este caso, tenemos un conjunto de vehículos del mismo tipo, situados en un deposito y se necesita encontrar un conjunto de rutas para cada uno que permitan satisfacer la demanda de un conjunto de clientes con demandas. Cada vehículo solo puede realizar una ruta y tiene una capacidad de demandas, de modo que sólo puede ir a un subconjunto de clientes cuya demanda sea inferior.

En este trabajo, recorreremos diferentes heurísticas para la resolución del CVRP problem.

2. Descripción formal del problema

Sea G un grafo no dirigido, G = (V,E), donde: V = $v_1, v_2, \cdots, v_n$, donde v_1 representa el depósito y todos los demás vértices representan clientes y E es el conjunto de aristas, las cuales tendrán asignado un costo no negativo, representando la distancia euclídea entre el par de vértices adyacentes.
Sea además H el conjunto de vehículos de un mismo tipo con una capacidad asociada.
Definimos una ruta como un circuito R que empieza y termina en v_1 y el costo de R como la suma de los costos de las aristas incidentes a los vértices de R, que además no supera a la capacidad asociada a los vehículos de H. Cada vehículo de H está asignado a sólo una ruta y cada ruta tiene asignado un vehículo diferente, de modo que el mapeo entre el conjunto de vehículos y rutas tiene el comportamiento de una función biyectiva. Cada Ruta r, excepto por el depósito, no tiene vértices repetidos, y la unión de todas las rutas contiene a todos los vértices de G(V).

Si hacemos un recordatorio del TSP problem o problema del viajante de comerci otro problema muy conocido, podremos encontrar una 'reducción' de uno de los problemas que surgieron de CVRP, que consiste en encontrar una determinada ruta.
El problema consiste en dada una lista de ciudades (interpretemos como vértices) y las distancias entre ellas (interpretemos como el valor de una arista, que representa la distancia entre sus vértices adyacentes) hay que encontrar la ruta más corta que sólo visita exactamente una vez cada ciudad y que finaliza en la misma ciudad que comenzó.
Este es un problema que pertenece a la categoría de NP-Hard y por lo tanto no tenemos una algoritmo que lo resuelva en tiempo polinomial. Sin embargo, podemos ver que si partimos de nuestro problema inicial y lo atacamos de determinada manera llegamos a un punto en donde existen una gran variedad de heurís ticas con las cuales atacar este nuevo y bien conocido problema.

3. Técnicas y soluciones

Dado que es un problema ampliamente conocido por su rol dentro de los problemas que son de gran interés en la actualidad, recorreremos diferentes aproximaciones para atacar el problema. Cabe mencionar que como forma parte del conjunto de problemas NP-Hard, no existe un algoritmo exacto que permita resolver el problema en un tiempo computacional razonable, de modo que gran parte de lo que sigue son heurísticas y metaheurísticas que son una de las opciones a lo que podemos apuntar en términos de solución para grandes instancias.

3.1. Heurísticas

En este terreno comenzamos a encontrarnos con algoritmos aproximados, ya que las heurísticas son métodos o formas de atacar un problema buscando encontrar una 'buena' solución en un tiempo razonable pero que dado que suelen recorrer un espacio búsqueda 'limitado' de soluciones no necesariamente encontramos la solución exacta o mejor.

Dentro de este grupo podemos nombrar otros grupos

3.1.1. Constructivas

Consisten en ir construyendo una solución a partir de un criterio inicial. Alguno de las más conocidas son: Savings: Clark and Wright; Matching Based, etc.

3.1.2. Algoritmo de dos fases

Consiste en dividir el problema en otros dos: clusterizar el conjunto de clientes, de modo que cada cluster contenga un subconjunto de clientes que puedan ser recorridos por un vehículo de forma factible (la suma de demandas es menor o igual a la capacidad del vehículo) y posteriormente construir una ruta para cada cluster. También puede realziarse en orden inverso.
Algunos ejemplos: Cluster first: route-second algorithm; Fisher and Jaikumar; The Sweep algorithm; Router-first:cluster-second; etc.

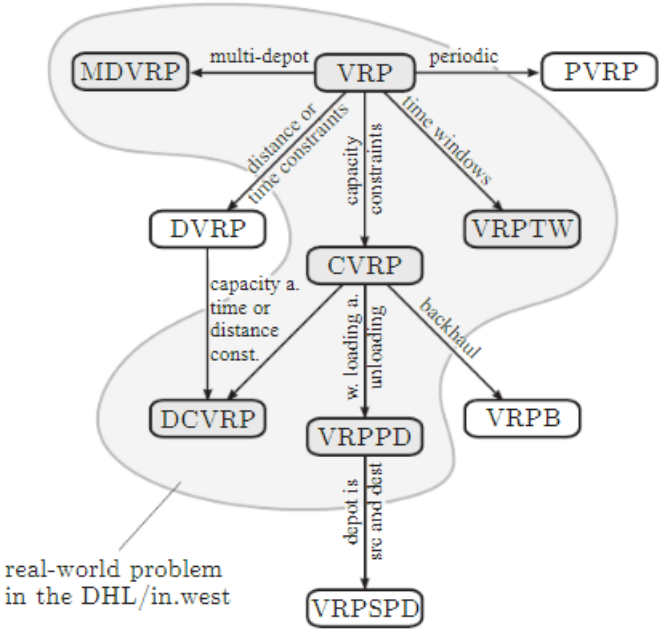
3.2. Metaheurísticas

El objetivo en este tipo consiste en un análisis o recorrido profundo por el espacio de soluciones, que por algún criterio, es el mejor para recorrer. Es razonable pensar que en este

tipo se consiguen mejores resultados que en las heurísticas nombradas, básicamente porque podemos pensar como que dada una solución se trata de buscar otras mejores (optimizar) incursionando o buscando en zonas donde puede haber otras mejores.
Un criterio para recorrer otras soluciones puede ser ir a una que tenga un mejor valor objetivo, donde valor objetivo es una función que expresa lo interesante que puede ser una solución (depende del problema). En general suele denominarse a estos como búsqueda local. Y algunos ejemplos: Grasp, Simulated Annealing; Tabu Search; etc.
Por otro lado tenemos lo que son algoritmos genéticos y que a grandes rasgos buscan someter a las soluciones a determinados procesos 'evolutimos', similares, y por eso el nombre, a los procesos de mutación de la evolución biológica, combinaciones genéticas, procesos de selección (según algún criterio), y en función de los que 'sobreviven' son considerados los más aptos mientras que las demás son descartadas. Algunos ejemplos son: Ant Algorithm; Genetic Algorithm.

4. Problemas de la vida real

Hemos mencionado la importancia del CVRP problem debido a la cantidad de problemas de la vida real que pueden modelarse utilizándolo.
Naturalmente en la vida real los problemas tienden a ser más difíciles de resolver debido a la cantidad de diferentes variables que intervienen en él. El VRP problem tiene muchos derivados y es que en la vida real las empresas suelen necesitar resolver varios tipos de este problema simultaneamente. Imaginemos una empresa de logística y distribución como DHL, UPS, etc, es claro que tienen más de un depósito o lugar desde el que distribuyen; también utilizan diferentes tipos de vehículos (trenes, camiones, aviones, barcos), tienen clientes de diferentes categorías (desde una persona que envía un paquete a otra a una empresa que hace un envío de mercadería hacia otro país, etc.
Es por esto que imaginamos que un diagrama de su logística de distribución podría verse como esto:



Es decir, de alguna manera tienen que subdividir con algún criterio su problema y encontrar la manera de utilizar la instancias del VRP que más se adapte a cada 'zona'.

Dentro del campos de la robótica, aunque para la construcción de circuitos en general, si se necesita en unir ciertas componentes de la placa y se quiere reducir la cantidad de buses que van a unirlos, o la cantidad componentes que unir, se podría modelar el problema con alguna modificación del VRP.

Si pensamos en las agencias de turismo vemos que cuando un cliente solicita un paquete para recorrer determinas ciudades, participar de determinados eventos, etc, estamos frente a problemas fácilmente modelables con el VRP. Por un lado, recorrer N ciudades siendo un turísta podría ser casi el mismo problema que TSP, dado que el turísta probablemente quiera aprovechar al máximo los dias y no volver a ciudades que ya conocio para ir a otra. Además, podríamos tener variantes donde por determinadas razones hay más de un transporte a utilizar, o al contrario sólo uno, etc.

También podría pensare en tareas a resolver y la necesidad de encontrar una forma de resolverlas de forma secuencial reduciendo el costo. Podríamos pensar que el funcionamiento de una máquina podría ser muy costo y varias tareas necesitan de ella. O que determinadas tareas requieren de mayor personal. Es decir, es claramente un problema modelable como un TSP (una instancia particular del VRP, donde sólo hay un vehículo).

Las empresas que transportan personas, por ejemplo micros escolares o micros que llevan gente al trabajo es otra de las situaciones de la vida real que puede modelarse con VRP. El micro sale de una estación y tiene que ir hacia diferentes puntos recogiendo personas. En este caso la estación inicial podría no coincidir con la de llegada o mismo el costo de los viajes podría ser diferente a la mañana que en otra parte del día. Claramente no es un problema sencillo pero es modelable con VRP.

Otra situación podría ser los censos. Podemos modelar a las manzanas como vértices y las personas como vehículos. Habría varios depósitos ya que cada barrio o comuna tendría como un lugar de salida, pero seguro que al final del día las personas que realizaron el censo deben volver hacia el depósito desde el que salieron a entregar los datos relevados.

5. Algoritmos presentados

5.0.1. Algoritmo de cluster-first, route second: mediante alternativa de sweep algorithm

En este tipo de técnicas la forma de atacar el problema es primero clusterizar (con algún criterio) los puntos en el espacio y posteriormente aplicar un algoritmo de resolución de TSP. Si clusterizamos en función de los nodos más cercanos entre sí y que además la suma de las demandas no supere la capacidad del camión, entonces ese es un cluster válido. Si logramos encapsular de esta manera todos los vértices y además tratando de alcanzar el mínimo número de clusters entonces estaremos encontrando soluciones de buena calidad. Para resolver cada uno de los cluster simplemente es aplicar algún algoritmo de resolución de TSP. Empecemos la explicación de la clusterización:

La idea es utilizar un método de agrupamiento llamado k-means, que tiene como finalidad particionar un conjunto en k conjuntos y donde se cumple

que los elementos que están en un determinado grupo cumplen ser los que están más cercanos al valor medio de ese conjunto (de ahora en más cluster).

Para esto vamos a definir a k como el número que resulta de dividir la demanda total de los vértices por el promedio de las demandas y será la primera cantidad de camiones con la que intentaremos realizar los ruteos. Los pasos que realiza nuestro algoritmo son:

- Lo primero es definir las posiciones de los k cluster como posiciones aleatorias delimitadas por los valores máximos de posición que toman los vértices. La posición de un cluster se llamará centroide y será lo que usemos para comparar las distancias con los vértices. Complejidad: O(k)
- Para cada par de los k cluster se iterará sobre los vértices buscando asignar cada uno al cluster más cercano. En caso de que el cluster no permita por capacidad ingresar otro nodo se analizará si el nodo (perteneciente) más lejano al cluster está más distante que el nodo que queremos ingresar y además si sacar ese nodo permite (por demanda) que ingrese el nuevo. En caso afirmativo sacamos ese nodo y guardamos el otro. En caso negativo nos queda libre el nodo. En ambos casos queda un nodo sin asignar. Complejidad: $\approx$ O(k^2 * cantidad-nodos) $\cdot$ O(TamCluster).
- Después de iterar cada nodo con cada cluster, se recorren los cluster y se re-calcula su centroide en función del promedio de posiciones de sus integrantes. En caso de que la nueva posición coincida con la actual y esto además se cumpla con todos los clusters entonces pasamos a la siguiente fase. En caso de que esto no pase se repite el paso anterior. Además tenemos un flag llamado tope-intentos-cluster-particular que está regulado por el usuario y permite fijar un número máximo de iteraciones con un mismo cluster antes de ir a la siguiente fase. Complejidad: O(k).
- Una vez finalizada una clusterización (ya sea porque se cumplieron la coincidencia de los centroides o porque se llegó al tope de iteraciones) verificamos si la clusterización es válida chequeando que todos los nodos estén asignados y que las demandas de los clusters están acotadas por la capacidad del camión. Complejidad: O(k).
- En caso de ser una clusterización inválida se repite el proceso desde el primer paso (es decir, considerando nuevos k clusters y asignando de nuevo posiciones aleatorias); en caso de ser válida se aplica TSP-GRASP a cada cluster, se analiza con la solución actual (si hay) y se actualiza en consecuencia. Después, en dependiendo de lo que permitan los flags, se repite desde el primer paso. Complejidad: O($cantidad - nodos^3$). Más adelante vamos a detallar TSP-GRASP y ahí queda mejor explicada la complejidad.

En los pasos significativos se aclara cuál es la complejidad para facilitar la comprensión del algoritmo y además de la complejidad total. Como observación a nivel implementación hay cosas que pueden mantenerse actualizadas constantemente en lugar de calcularlas cuando se las necesite. Un ejemplo de esto es para ver cuántos nodos están asignados en cada cluster o la suma de las

demandas de un cluster: no hace falta recorrerlos a todos para calcular esto sino que mantenemos actualizado su valor. Sin embargo sí se recorrerá el cluster para buscar el nodo más lejano debido a la propia dificultad de mantener actualizado este campo por el funcionamiento del algoritmo.

5.0.2. Complejidad

El proceso de definir la complejidad de este algoritmo es algo complicado debido a las ideas sobre las que está construído: hay varios refinamientos controlados por el usuario y gran parte de la solución está basada sobre clustering, que como vimos en el trabajo práctico 2, la idea entre la percepción del ser humano respecto de puntos en el espacio y las decisiones que puede tomar un algoritmo en función de ciertos parámetros no siempre se correlacionan. Adicional a esto, este es un problema aún más complejo, debido a que la construcción de los clusters está definida además por las demandas de los clientes y las cargas que pueden soportar los clusters, de modo que al final del día son muchas las cosas a tener en cuenta.

Entonces como no podemos saber si para un determinado k <n vamos a poder encontrar solución, la manera en que vamos a establecer la complejidad es haciendo mención a cómo fijamos determinados parámetros de control y tomando como peor caso que lo único que podemos afirmar con total seguridad es que dados n clientes, cada uno con capacidad menor o igual a la capacidad de los cluster hay solución con k = n. Claramente esto es un resultado sin sentido a nivel práctico, pero es la forma de establecer la complejidad en peor caso.

Si definimos a los flags de control como constantes del problema y calculando la complejidad en peor caso, donde peor caso $\approx$ k = n. Las complejidades intermedias quedan:

- Repartir al cluster más cercano O(k^2 * cantidad-nodos) y contemplar las posibles extracciones de nodos ya asignados: O(TamCluster), total : O(k^2 * cantidad-nodos) * O(TamCluster)
- Aplicar TSP a la solución es: O($(cantidad - nodos)^3$)

Observemos que TSP se realiza con cada uno de los cluster y que la sumatoria de todos ellos es la cantidad total de vértices (cantidad-nodos). La complejidad entonces de aplicar TSP a todos es : O($(cantidad - nodos)^3$) dado que el tamaño de cada cluster es menor a cantidad-nodos y podemos acotarlo. Total: O(k^2 * cantidad-nodos) * O(TamCluster) + O($(cantidad - nodos)^3$)

Si estamos en el caso en que no se encuentra solución para k <n en particular queda: O($(cantidad - nodos)^4$). Para clarificar más acerca de los flags de control:

1. tope-intentos-sin-aumentar-cluster: determina cuantas veces se puede repetir el proceso descrito arriba de armado de los cluster(que consiste particularmente en generar k cluster de posición aleatorias, iterar cada nodo para cada par de cluster y asignar por cercanía y verificar si es un cluster válido)
2. tope-intentos-mismo-cluster: determina cuantas veces se puede repetir el proceso de asignación de nodos a cluster más cercano mientras no

se consiga que todos los cluster tengan su centroide coincidente con el promedio de posiciones de sus integrantes.

3. tope-intentos-optimizar: determina cuantas soluciones encontrar antes de terminar.
4. seguir-buscando-sol-con-mayor-k: una vez que encontramos solución podríamos seguir buscando otra si el flag tope-intentos-optimizar lo permite. En caso de que se alcance el tope de intentos sin aumentar cluster, ahora el algoritmo estaría buscando nuevas soluciones pero con más cluster. Este flag permite o no que esto suceda.

5.0.3. Algoritmo de resolución de problema de TSP

Este problema lo vamos a resolver mediante la técnica de vecinos más cercanos. Sin embargo le vamos a agregar un refinamiento. Básicamente utilizamos un método de búsqueda local de recorrido de soluciones conocido como GRASP. Nuestra solución inicial será una formada por procedimientos estricamente golosos y para las siguientes vamos a agregar a nuestro vecino más cercano una cuota de probabilidad, donde con probabilidad p se elegirá el vecino más cercano y con probabilidad (1-p) se elegirá otro nodo, dentro de lo que podríamos definir un algoritmo goloso aleatorizado, tomando a veces la mejor decisión y a veces otra cosa.

El valor de la probabilidad inicialmente será 1 y para las demás será producto de restarle 0.1 a p mientras sea mayor a 0.5.

Para la búsqueda local de soluciones utilizaremos una técnica de optimización conocida como 2-opt. La idea de este algoritmo consiste en dada una solución y dos nodos i y j del camino, se modificará la solución hacia una que cumpla: entre i y j el orden del sub-camino se invierte mientras que para los caminos de posición entre 0 e i y entre j+1 y el último nodo se mantienen igual. Esto lo vamos a repetir entre cada posición del camino solución y comparar la solución obtenida con este método con la mejor actual y actualizar en consecuencia.

El funcionamiento será: cada vez que se termine de construir una clusterización correcta se aplicara el algoritmo de resolución de TSP a cada uno de los cluster. Sea V la lista de nodos de un cluster en particular. Sea p el valor de la probabilidad y shifter = 0.1 el valor con el que se reducirá p.

1. Calculamos solución golosa con la técnica de vecinos más cercanos usando V y p = 1 y sea C el camino retornado. Complejidad: O($|V|^2$).
2. Dada esta solución inicial realizamos lo siguiente, mientras p >= 0.5
   - Iteramos sobre cada par de posibles posiciones de C sin considerar la primera y la última que son el depósito. Sobre cada uno de ellas aplicamos la técnica 2-opt. O($|V|^2$ * 2-opt) $\approx$ O($|V|^2$ * |V|) $\approx$ O($|V|^3$).
   - Verificamos si la solución nueva obtenida aplicando 2-opt es mejor que la actual y en ese caso la actualizamos.
   - p = p - shifter
   - calculamos otra solución golosa (ahora randomizada) con el p actualizado.

- Verificamos si esta nueva solución es mejor que la actual y en ese caso actualizamos la actual.
- Repetimos desde (2).

3. Finalmente tenemos el camino y el costo de haber ruteado el cluster y haberlo optimizado.

La complejidad de todo esto es O($|V|^3$). Notar que |V| = |C|, dado que las variaciones entre la primera solución golosa y las demás no sacan o agregan nodos sino los mueven de lugar.

El funcionamiento del algoritmo de vecinos más cercanos es muy intuitivo: el primero en guardarse en el camino C es el depósito que además lo marcamos como el nodo actual y a partir de ahí mientras haya nodos realiza:

- Mientras haya nodos
  - Buscar nodo más cercano al actual que no haya sido visitado. Además en este mismo recorrido por los nodos guardamos uno no visitado para usar como opción aleatoria. En caso de que no se encuentre ninguno porque están todos visitados ponemos en true un flag de todos visitados.
- Calculamos valor aleatorio entre 0 y 1 y si es mayor que p y todos visitados es false, entonces guardamos en el camino el nodo anteriormente almacenado para esto y lo marcamos como visitado y como nodo actual. Si es mayor que p pero todos visitados es verdadero marcamos en false el flag de hay nodos, agregamos al camino el nodo actual y termina. En caso de que el valor aleatorio no sea mayor a p, haríamos lo equivalente pero con el vecino más cercano buscado en el paso anterior y seguiríamos sujetos a lo que nos permitan los flags.

La complejidad de esto entonces es cuadrática respecto del tamaño de la lista de nodos de entrada (tamaño del cluster).
El procedimiento de 2-opt es lineal en el tamaño de la lista de entrada porque simplemente mueve algunos elementos de lugar.

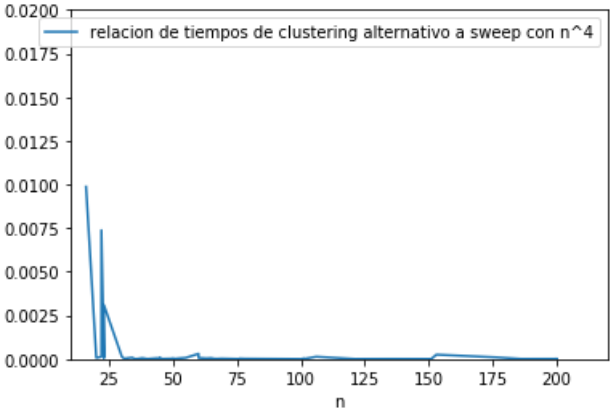
5.1. Experimentación: Heurística constructiva de cluster-first, route-second: mediante alternativa a sweep algorithm

Como mencionamos durante la explicación del algoritmo fijaremos los flags de control en constantes fijas para los grupos de instancias. A priori el flag de seguir buscando solución con más cluster la vamos a deshabilitar, la cantidad de soluciones que queremos obtener para optimizar es 1, la cantidad de iteraciones hasta aumentar de cluster la fijamos en 15 y la cantidad de iteraciones para intentar formar un cluster en 5. Proyectamos experimentar con casos de entre 20 a 200 nodos y nos parece adecuada la proporción entre el tope de repeticiones fijado y los tamaños de los sets aunque en busca de experimentar acerca de la funcionalidad de estos flags los iremos modificando a lo largo de la experimentación. Los casos de test que utilizaremos son un subconjunto de los de A, B, E, F, M , P y X tratando de conseguir una distribución lo más uniformemente posible entre los valores de n.

Algunas de las preguntas que nos surgen son: cuán alejado de la solución óptima esta nuestra solución; cuánto afecta la relación entre la cantidad de clientes y la cantidad de

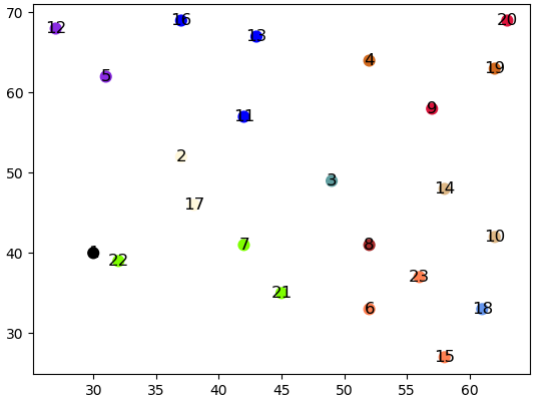
cluster que se necesitarán; cómo afecta el promedio de demandas y su relación con la capacidad máxima son algunas de ellas.

Comenzamos con un gráfico que muestra la relación de complejidad calculado anteriormente.



Podemos notar que en el primer intervalo de n hay determinados saltos hasta que termina por asentarse. Analizando qué podía producir esto encontramos que los casos problemáticos eran:P-n16-k8, P-n22-k8, P-n23-k8. En los tres casos el algoritmo no pudo encontrar el ruteo con la cantidad de vehículos óptimo. Incluso para el caso n22-k8 necesitó de dos más. A su vez encontramos que en estos casos las demandas de los clientes están cercanas a las capacidades de los vehículos [0,31] y 31;[0,2500] y 3000;[0,30] y 40 respectivamente.

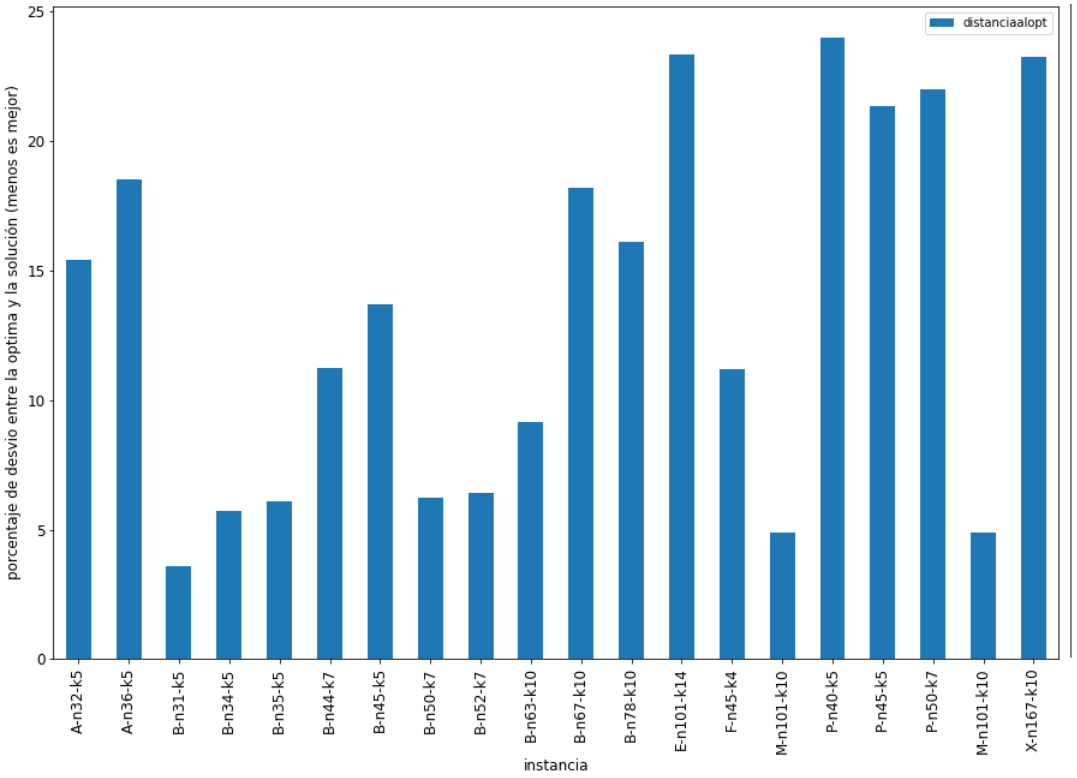
Clusterización de P-n23-k8



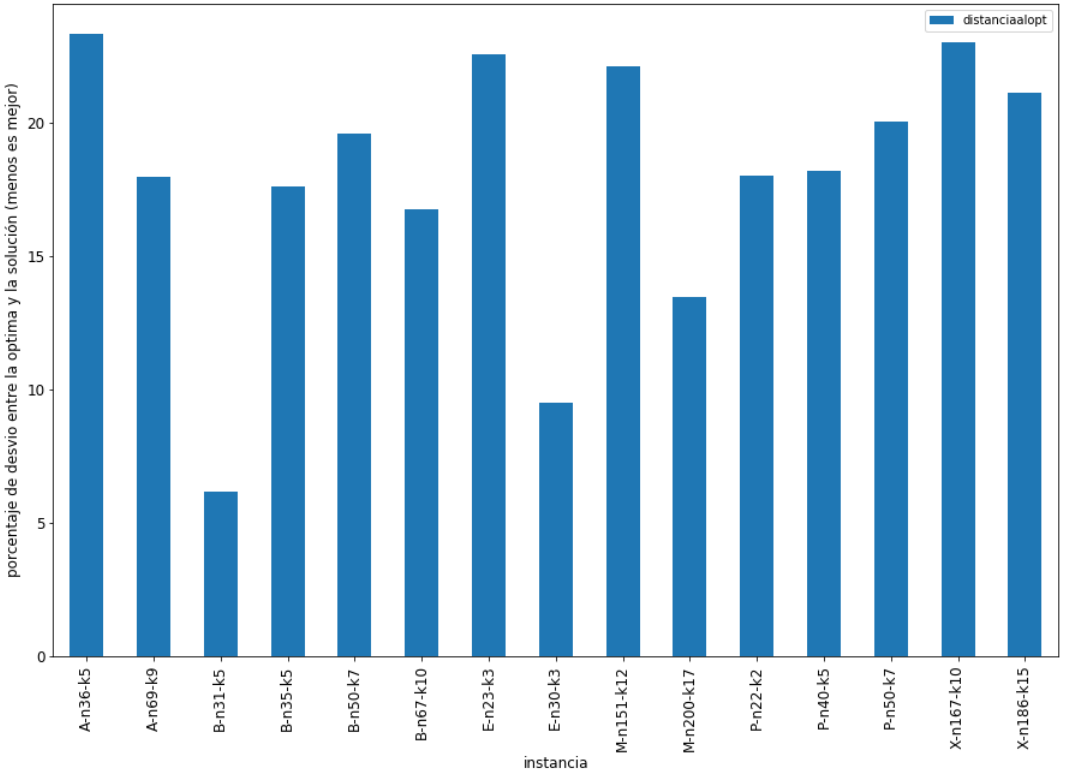
Si tenemos en cuenta que los flags de control y que la cantidad de clusters en estos casos están muy cercanos al valor de n y que gráficamente los puntos están distribuidos de forma que no ayuda a la clusterización estamos frente a casos en donde aplicar este tipo de técnicas no brinda soluciones de calidad.

Centrémonos en las soluciones de buena calidad producidas por el algoritmo. Primero definamos que vamos a tomar como buena calidad: debe haberse ruteado con la cantidad de camiones óptima y no superar en 25 % el costo total del ruteo.

Algoritmo de alternativa de Sweep



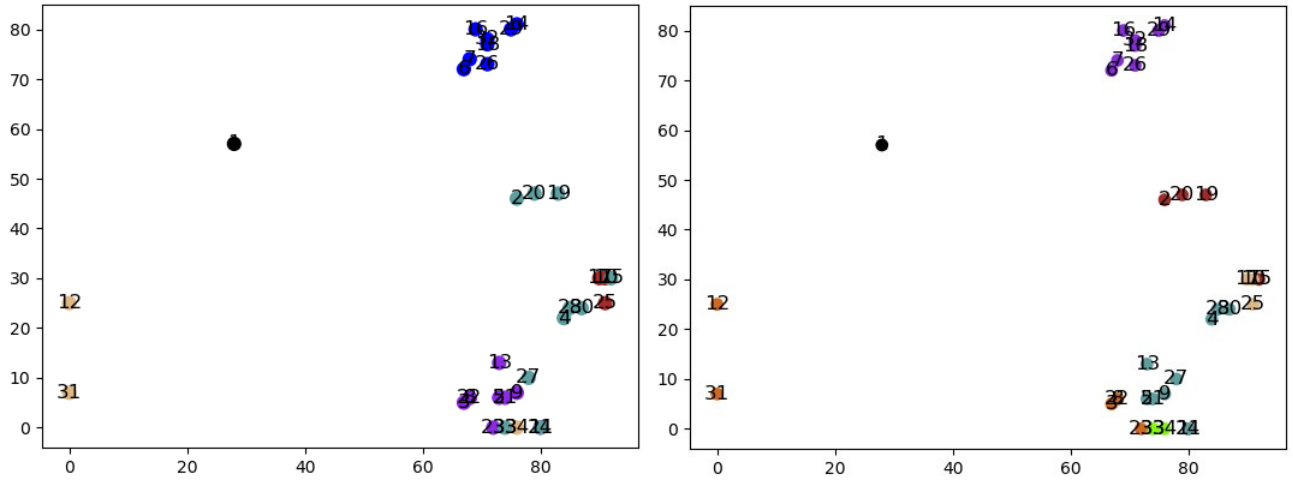
Algoritmo de Sweep



Los anteriores gráficos de barras describen para cada instancia el porcentaje de desvío que tiene respecto de la solución óptima, es decir, cuánto más 'costoso' son los caminos ruteados. En este gráfico menos es mejor. La información relevante que brinda son

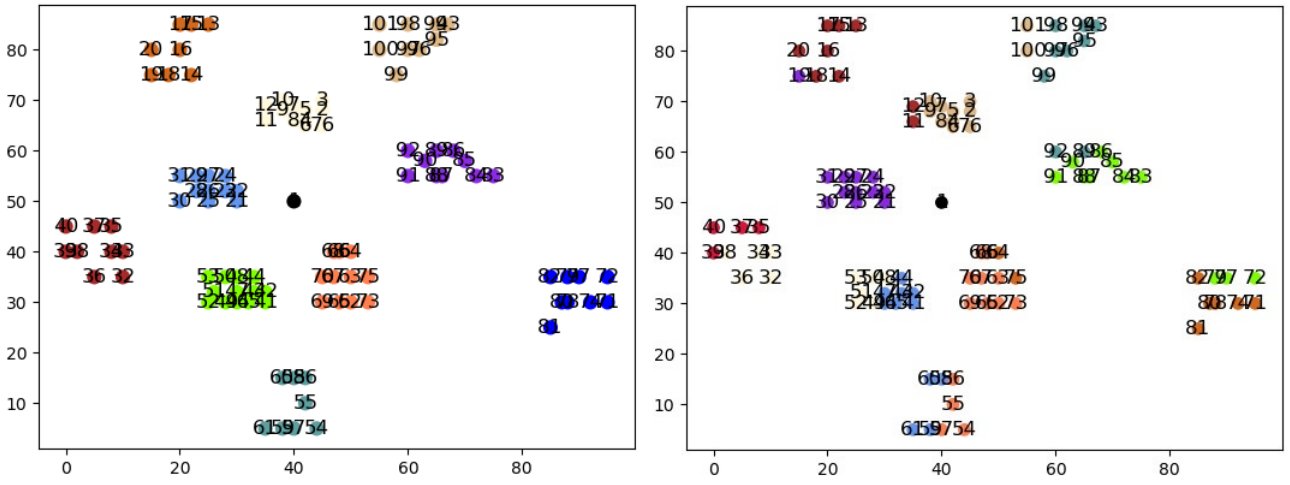
aspectos que podemos reconocer: si bien corrimos instancias hasta alrededor de 200 clientes las instancias de solución óptima del gráfico sólo alcanzan hasta alrededor de 100 clientes, además para el mismo set de instancias Sweep consiguió un porcentaje de 36 % contra 40 % del algoritmo alternativa de efectividad para resolver el problema de forma óptima. Por otro lado no parece haber una relación estricta entre mayor cantidad de clientes y mayor desvío dado que el test M-n101-k10 tiene un n considerablemente más grande que la demás instancias y sin embargo se ruteó de forma cercana a la óptima. Por ejemplo el test B-n34-k5 Sweep no logra rutearlo con 5 vehículos y en cambio lo hace con 6.

Clusterización con algoritmo alternativa a Sweep a la izquierda y Sweep a la derecha.

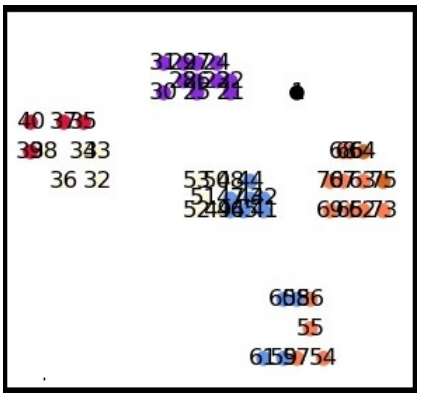


En este tipo de casos al no tener un refinamiento entre nodos asignados y nodos por asignar como sí tiene el algoritmo alternativa sucede que no se consigue completar lo máximo posible las demandas de los vehículos. En estos casos sucede que se necesitan agregar nuevas rutas para satisfacer clientes que en caso de haber hecho algún tipo de pasada adicional o intentado mover clientes previamente asignados se hubiera descubierto que no era necesario.

Uno de los casos que parece interesante analizar es el M-n101-k10. La alternativa de Sweep lo resolvio de forma óptima mientras que Sweep necesitó un vehículo más. Al anaizar en detalle encontramos lo siguiente:



Naturalmente este caso forma parte de los que intuitivamente es bueno para utilizar este tipo de algoritmo: los clientes están organizados en grupos idénticos a los que podrían clusterizarse para resolver el problema con TSP. Sin embargo Sweep no consigue clusterizar de la mejor manera y por lo tanto requiere incrementar los vehículos. La idea de picking de vértices de Sweep podría explicarlo: estamos frente a un algoritmo que trata de 'barrer' con un mismo cluster todos los nodos que queden a su alcance mientras las demandas lo permitan. La propia variación de demandas de los clientes podría generar una 'desincronización' entre el momento óptimo para cerrar un cluster por capacidad o por lejanía de los nodos.



Esa porción de la clusterización condice bastante con la explicación dada y con lo verificado paso a paso en la corrida del test por Sweep: cuando termina de armar el cluster blanco es porque no puede ingresar ningún nodo (de los que posteriormente serán rojos) dado que excederían la capacidad. Los rojos pasan a estar en este nuevo cluster que queda medio vacío debido a que los nodos violetas ya habían sido barridos y por lo tanto no queda nada que agregar.

Respecto de la alternativa a Sweep la clusterización es idéntica a la percepcion humana y es probable que estemos consiguiendo la mejor forma de organizar a los clientes y las variaciones en los costos respecto de la solución óptima se deban a la heurística que utilizamos para TSP.

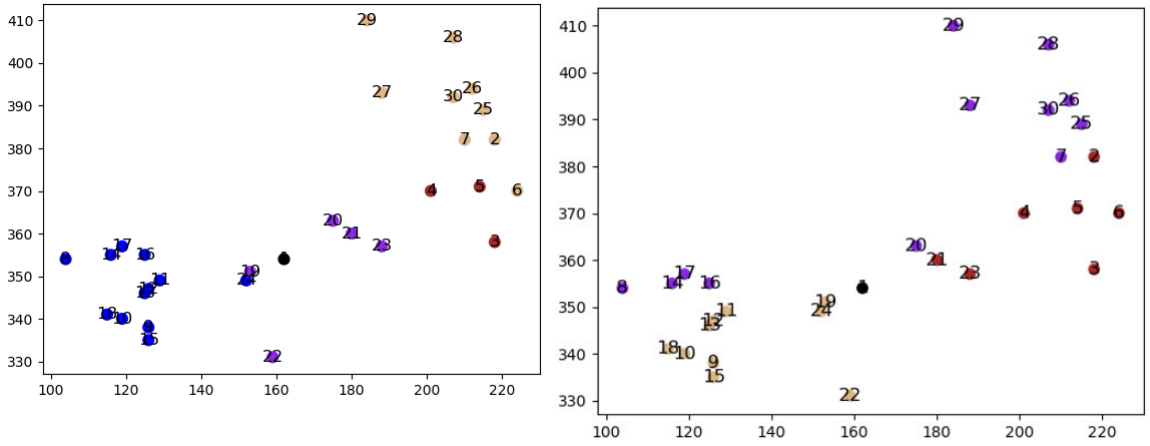
Otra de las observaciones que podemos hacer es que nuestros algoritmos se ven afectados en gran medida cuando los casos de test tienen intervalos de demandas de clientes cercanas a las capacidades del camión y sobre todo cuando hay muchos de estos puntos problemáticos (demandas iguales o muy cercanas a la capacidad). Sobre todo en la alternativa a Sweep los tiempos ejecución son mucho más elevados y las soluciones no suele consiguen rutear con la misma capacidad de vehículos ni un costo cercano. Y en relación con lo anterior mencionado si bien distribuciones de clientes en grupos similares a una clusterización ayudan al algoritmo aún ayuda más que no existan este tipo de puntos problemáticos y que la cantidad de clusters sea considerablemente más baja que n.

El hecho de que existan puntos con demandas tan cercanas al tope genera que el algoritmo necesite una cantidad de clusters cercana al valor de n y por otro lado que mientras se asigna los nodos a los clusters es altamente probable que las asignaciones necesiten sacar uno de los nodos del cluster para poder hacerse y esto genera más iteraciones para poder armar los clusters, mayor cantidad de recorridos de los cluster, mayor cantidad de clusters, etc. Por este motivo este es un caso que este algoritmo no podrá dar soluciones de calidad.

Para visibilizar esta problemática presentamos los siguientes casos: B-n35-k5 y E-n30-k3. Son instancias de casi la misma cantidad de nodos y clusters sin embargo el primero tiene una distribución de demandas de [0,69] con un tope de 100 y el tercero de [0,3100] y 4500

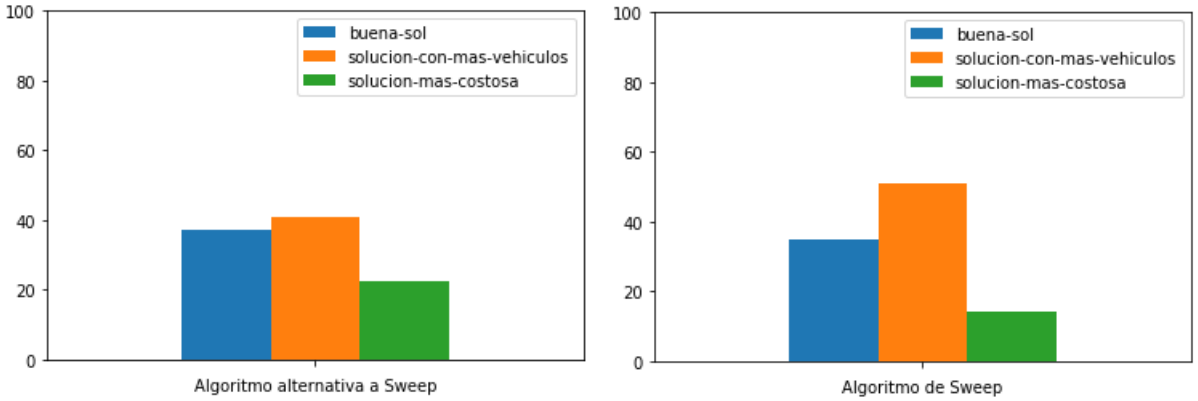
de tope. Con alternativa a Sweep conseguimos un resultado de 5 % de diferencia respecto del costo óptimo y con la misma cantidad de vehículos y en el segundo sólo conseguimos rutearlo con un vehículos más con un 14 % de desvío. Con Sweep logramos resolver a ambos con la cantidad de vehículos óptima pero con un desvío mayor: 17 % y 9 % respectivamente.

Veamos por qué sólo Sweep consigue solución de buena calidad en E-n30-k3.



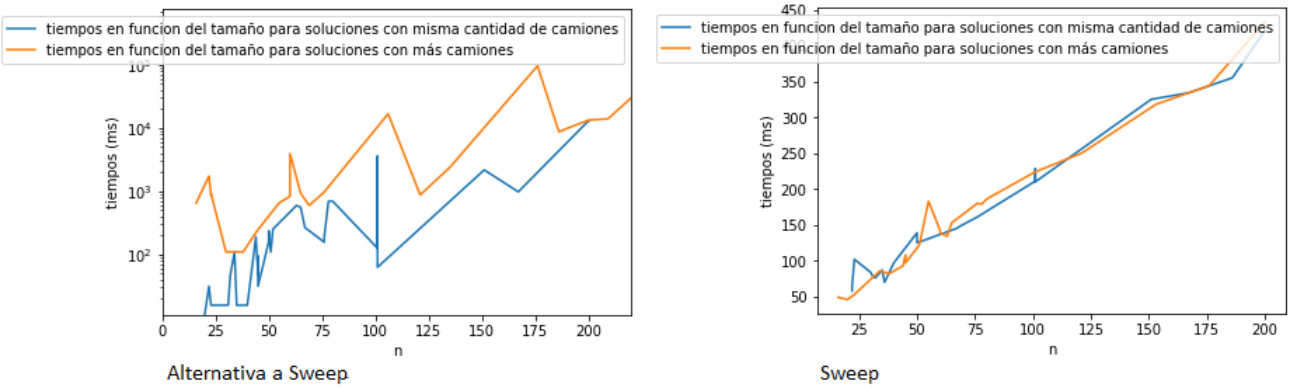
Lo interesante de este caso es que si bien anticipamos que casos con puntos problemáticos eran un problema para este tipo de técnicas encontramos un caso en donde la distribución de los puntos favorece a Sweep. El barrido que tiene utiliza esta técnica permite que dos grupos de nodos en esquinas opuestas sean del mismo cluster: esto nunca pasaría en la alternativa y en este caso en particular donde hay 3 zonas de mayor convergencia de clientes (y en particular cada zona tiene un cliente con una demanda muy cercana a la capacidad, es decir, un punto problemático) se pone más en evidencia sus falencias. En este caso lo que sucede es que los puntos problemáticos quedan fuera del cluster azul pero lo suficientemente cercanos como para que el cluster azul sólo tenga a ellos como alternativa a intercambiar por nodos propios pero esto nunca pasa porque el nodo más lejano del azul está incluso más cerca que esos nodos. Entonces el problema es que los nodos que quedan fuera del cluster azul no pueden clusterizarse en dos clusters y por lo tanto se necesita uno más. Como idea final podemos decir que cuando tenemos esta distribución de puntos Sweep es posible que se favorezca (recordemos el caso anterior B-n34-k5, de similar distribución de clientes, en donde Sweep no se beneficiaba por la ubicación de los puntos problemáticos). Pensemos que si esta instancia no tendría puntos de ese tipo posiblemente la alternativa habría logrado una solución de buena calidad o si la instancia tuviera puntos más uniformes en el plano aunque existieran no lo afectarían tanto.

En las siguientes gráficas podemos ver de forma más estructurada los puntos resaltados anteriormente. Lo que se representa son las soluciones de buena calidad (definadas previamente), las soluciones con más vehículos y las soluciones que están ruteadas con la cantidad de vehículos óptima pero con un costo asociado mayor al 25 %.



Ambos gráficos tienen un porcentaje similar de soluciones óptimas y las diferencias más notorias están con las 'no' soluciones óptimas. Es evidente que los refinamientos que no tiene Sweep provocan que tenga un mayor porcentaje de soluciones ruteadas con más vehículos, mientras que la alternativa que sí tiene este tipo de complementos parece evitar esto en algunas instancias pero con un costo asociado: posiblemente la forma en la que termina por agrupar los clientes no le permiten posteriormente crear rutas lo más óptimas posibles.

Hasta ahora estuvimos haciendo un análisis comparativo de calidad de soluciones de ambos algoritmos y las menciones de los tiempo de ejecución no se mencionaron o enfatizaron. Durante el análisis los algoritmos supieron tener marcadas diferencias en sus tiempos: Sweep tiene un crecimiento en relación al tamaño y la alternativa también aunque condicionado con el tipo de instancia que esté evaluando. Al ser un algoritmo con refinamientos definidos por el usuario podría complejizarse (desde el punto de vista de los tiempos) tanto como uno quiera, debido a que sería como agrandar la fuerza bruta que analiza las posibles soluciones. Como introducción a todo esto podemos anticipar que los tiempos de Sweep son mejores y que la alternativa puede obtener soluciones de calidad en instancias considerablemente grandes siempre y cuando la instancia no contenga punto problemáticos o una combinación de muchos clientes con demandas muy chicas respecto de la capacidad (clusters muy grandes) y con una distribución uniforme (similar a un spray) a lo largo de todo el plano (provocando que dado un nodo está muy cerca de muchos clusters y por lo tanto se generan movimientos excesivos de asignaciones y re-asignaciones de nodos).

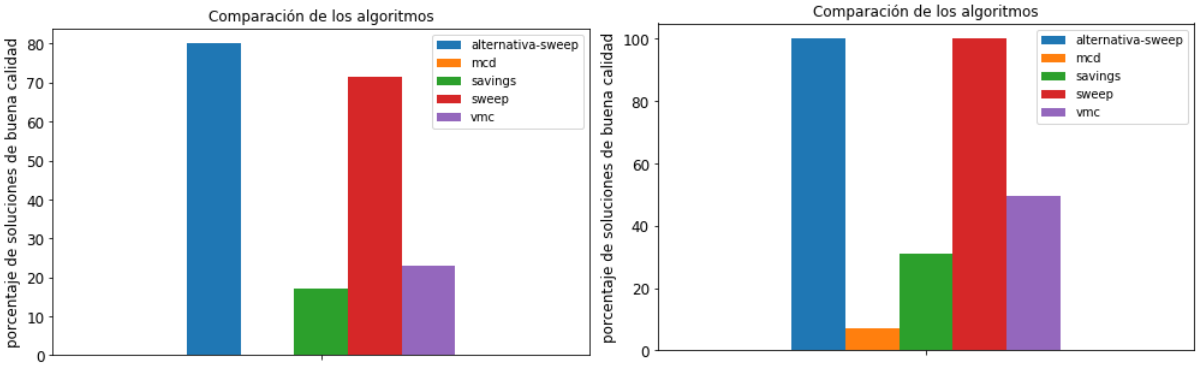


El gráfico anterior ilustra los conceptos que se vienen desarrollando. Lo que hicimos fue

analizar los tiempos de ejecución en función del tamaño de entrada y separando en el tipo de solución conseguida. En naranja están los tiempos en función del tamaño de entrada cuando sólo se pudo conseguir soluciones con una cantidad de camiones mayor a la solución óptima. En azul están las que sí se pudo conseguir la óptima cantidad de camiones. Por razones de escala la alternativa a Sweep está en escala logarítmica.
El algoritmo de Sweep tiene una estructura que depende completamente del tamaño de entrada sus tiempos de cómputo, mientras que la alternativa es un método que depende no sólo de tamaño de la entrada sino también del tipo de instancia. Esto es algo muy visible en los gráficos: saltos pequeños hay en el de Sweep mientras que en el otro hay más saltos y teniendo en cuenta la escala de considerable mayor tamaño. Con esto podemos hablar acerca de lo malo que puede ser la alternativa a Sweep cuando no es una buena instancia: incluso en caso de poder llegar a la solución óptima los tiempos pueden ser mucho mayores a Sweep y estamos experimentando con instancias de menos de 200 clientes.

6. Comparando performances

Las comparaciones entre los algoritmos de cluster-first fueron poniendo la vara alta respecto de la consideración de calidad de solución. Sin embargo para un análisis más general entre los diferentes algoritmos será necesario por lo menos al inicio reducir los criterios de calidad. Resaltemos los puntos importantes reconocidos hasta el momento:



En el gráfico anterior se puede comparar las soluciones brindadas por los algoritmos para un conjunto de instancias de entre 20 y 250 clientes y una calidad de solución definida. En el primero se considera una buena solución cuando el costo obtenido no supera en 30 % al de la solución óptima. Un detalle a considerar es que para el set de instancias ninguna pudo resolver en esta configuración el algoritmo de más cercano al depósito. En la segunda se considera una buena solución a las que no superen en 50 % al costo de la óptima. En este caso se puede apreciar que el algoritmo MCD aparece escena aunque muy distante respecto de los demás.