DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 3

Modelando problemas problemas con grafos

22 de noviembre 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 lo único a lo que podemos apuntar en términos de solución para grandes instancias.

3.1. Branch and Bound

En esta aproximación se utiliza la técnica de divide and conquer para particionar el problema en subproblemas y optimizar cada uno de ellos por separado. La parte de 'Branch' se corresponde con que se trata de conseguir subconjuntos del espacio total que se busca rutear. Cada uno de estos subconjuntos unidos forman el conjunto total y se juntan dentro de una lista de candidatos, próximos a analizar. Posteriormente se selecciona uno de ellos y se procesa. Los resultados pueden ser desde intercambiar la mejor solución por la actual, eliminarla porque no sirve o subdividir el subproblema actual y actualizar la lista de candidatos. En general el algoritmo termina después de haber vaciado toda la lista de candidatos del problema. Como se puede ver, es una solución del tipo fuerza bruta, en general combinada con ciertos requisitos para acercarse a una solución del tipo backtracking, pero en la teoría es una solución de tiempos computacionales no razonables.

3.2. 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

3.2.1. 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.

3.2.2. Algoritmo de dos fases

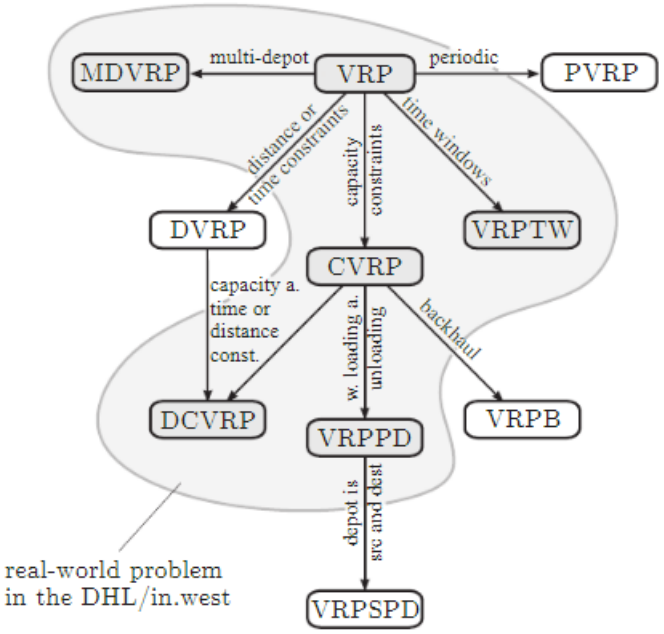
Consiste en dividir el problema en otros dos: clusterizar el conjunto de clientes, de modo que cada cluster contenga un subconjunto de clientes que puedan ser recorridos por un vehículo de forma factible (la suma de demandas es menor o igual a la capacidad del vehículo) y posteriormente construir una 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.

3.3. 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 detemrinados 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 descartadasl. 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

Antes de comenzar la explicación formal del algoritmo describiremos de forma práctica qué hace.

Cuando recibimos un conjunto de puntos en el espacio, representando un depósito y clientes lo primero que es importante hacer es conseguir la mayor cantidad de información de los datos que ya tenemos disponible, esto es, cuáles son los rangos de máximos de las posiciones, la demanda de todos los clientes, el promedio de demanda por cliente y la capacidad del camión. Con esto atacamos el problema de la siguiente manera: definimos que la cantidad de camiones que se necesitarán está definido por la relación entre la cantidad de clientes que hay y la cantidad de clientes que puede satisfacer aproximadamente un camión. Es decir, si cada camión podría visitar k clientes y hay q $\cdot$ k clientes en total, se necesitarán aproximadamente q camiones. Hablamos de aproximaciones porque trabajamos con promedios y los promedios dan una información acotada, debido a que contextos con grandes desviaciones podrían generar problemas en nuestro algoritmo. Sin embargo, estamos definiendo nuestra propia heurística y, como tal, demostrará su 'calidad' en la posterior experimentación.

Con esta información podemos aproximar la clusterización mediante un método de agrupamiento conocido: k-means. Es interesante este método porque habiendo definido nuestro valor k (la cantidad de camiones) ya podemos utilizarlo. La idea es agrupar cierta cantidad de observaciones en grupos con el valor medio más próximo, donde el valor medio en nuestro caso sería las distancias entre los puntos en el plano.

En este punto surge un inconveniente y es que nosotros necesitamos que cada grupo cumpla que la suma de las demandas de los integrantes no supere a la capacidad total y esto implicará cierto proceso de intentos o prueba y error para funcionar.

Sea C el conjunto de clusters, |C| = k, la cantidad de clusters, y n la cantidad de clientes, y definimos ($\forall$ i $\in$ 1$\cdots$ k)($\forall$ j $\in$ 1$\cdots$ k) : c_i y c_j, i != j, c_i, c_j $\in$ C, una posición aleatoria inicial.

Ya tenemos las posiciones de lo que llamaremos 'centroide' de un cluster, y en función a ellos definiremos qué cliente esta más cerca.

Para cada par de clusters diferentes, A y B, se recorren todos los clientes. Se analiza si el cliente actual esta más cerca de A o de B. Para simplificar la explicación supongamos que está más cerca de A, ya que el otro caso es análogo. Posteriormente verificamos si el cliente actual este más cerca de A que de su cluster actual. Si no tiene actual (la distancia a un cluster que no pertenece pensemos como que es un valor muy grande) entonces está más cerca de A. Sabiendo esto tenemos dos opciones: A tiene espacio para albergarlo o A no tiene espacio. Si A tiene espacio es trivial que guardamos el cliente en A y seguimos con el siguiente cliente. Si A no tiene espacio

vamos a buscar cuál es el nodo de A que está más lejos de A. Como cada cluster tiene definido un centroide, cada nodo perteneciente a un cluster tiene una distancia a ese centroide, de modo que esto esta bien definido. Ahora tenemos otro cliente, uno que es el más lejano de A. Comparamos la distancia que tiene nuestro cliente actual con el cliente más lejano de A. Otra vez tenemos dos opciones: si el cliente más lejano de A está más cerca de A que el cliente actual, dejamos el cliente actual en stand by y seguimos con otro cliente (esperando que en proximas iteraciones de cluster esto se solucione). Si el cliente actual efectivamente está más cerca de A que el cliente más lejano de A entonces otra vez hay dos opciones: si meter al cliente actual en A y sacar al cliente más lejano de A me genera una sobre-demanda del cluster A, entonces no lo meto y sigo con otro cliente. Si en cambio, sacar de A al cliente más lejano y meter al cliente actual me sigue manteniendo la carga actual de demandas del cluster A dentro del rango, entonces efectivamente los intercambios, y el cliente lejano pasa a dejar de tener cluster y la iteración sigue con el próximo cliente (esperando que el cliente que acabamos de retirar encuentre nuevo cluster en próximas iteraciones).

Después de recorrer los clientes con cada par de cluster de C, vamos a calcular el promedio de posiciones de cada cluster (promedio de posiciones en eje x y promedio de posiciones en eje y de cada nodo del cluster) y compararlo la posición del centroide del cluster. Si no coincide, le asignamos al centroide estos valores; si coinciden, aumentamos en uno un acumulador. Si después de consultar esto para cada cluster, todos tienen coincidencia entre posicion del centroide y los promedios de posición de sus nodos, entonces dejamos de trabajar sobre esta clusterización. Si existe al menos un cluster que todavia puede refinarse (definir como sus posiciones las del promedio de sus nodos), entonces volvemos recorrer los clientes con cada par de cluster.

Después de este proceso tenemos una clusterización: sin embargo, no necesariamente es correcta. Como antes dijimos, podría llegar a darse de que algunos nodos quedaron sin asignar. En este caso diríamos que hay que repetir proceso anterior. Si la clusterización es correcta, resolvemos cada cluster con un algoritmo que resuelve el problema del TSP, ya que es exactamente el problema que queremos resolver: un mismo lugar de salida y llegada, la suma de las demandas esta dentro del rango y pasar una vez exactamente por cada ciudad y recorrerlas todas. Detallaremos posteriormente cómo esta desarrollado nuestro algoritmo de resolución de TSP.

Haciendo un checkpoint tenemos lo siguiente: generamos un cluster, verificamos si era válido. Si lo era resolvimos TSP para cada cluster y si no volvimos a intentar generar un cluster desde 0. Se podría decir que esto es la idea general del algoritmo y ahora solo restan los detalle, que forma parte del refinamiento de las soluciones del problema.

Este procedimiento tiene varios parámetros modificables por el usuario, y existen como regulador de 'calidad' y 'tipo' de solución y ahora diremos que significa esto.

Los parámetros son cuatro y para enumerarlos directamente diremos su función:

(I) un contador de intentos por clusterización: cuando intentamos refinar una clusterización asignándole a la posición de cada cluster el promedio de sus nodos, esto podría ser un proceso que a priori no termine,

dado que mínimas variaciones de posición por el intercambio de nodos podrían repetirse cíclicamente (no terminaría) o repetirse una cantidad de veces muy grande antes de poder dar una solución de clusterización (sobre todo cuando hay muchos nodos). Con este contador, definimos nosotros mismos cuál es el tope de iteraciones para dejar de refinar una determinada clusterización.

(II) un contador de intentos de clusterización antes de aumentar los cluster: básicamente, si después de haber generado una determinada cantidad de clsuterizaciones vemos que no podemos encontrar una correcta (quedan nodos sin asignar) entonces cortamos los intentos de clusterización con esta cantidad de cluster y aumentamos la cantidad de cluster (agregamos un camión más). Puede parecer confuso este contador con el anterior, pero la diferencia es muy clara: el anterior permite dejar de intentar una clusterización y empezar otra de 0, con la misma cantidad de camiones/cluster; mientras que este acumulador trabaja sobre los clusters ya terminados (bien o no) y en cierta manera define cuan interesados estamos en seguir tratando de buscar posibles soluciones con una cantidad de camiones. Recodemos que al estar asignando al inicio posiciones aleatorias, esto podría tan grande como 'queramos'.

(III) un contador de soluciones encontradas: el algoritmo terminaría de buscar soluciones después de haber encontrado tantas como le hayamos pedido (y devolverá la de menor costo de ruteo).

(IV) Por último, un dato bandera que define si nos interesa seguir buscando soluciones después de haber encontrado 'la mejor' (según nuestro algoritmo) con una cantidad de clusters. Es decir, si nuestro algoritmo alcanzó las iteraciones necesarias para poder aumentar los camiones/clusters, podemos pedir que termine ahí mismo (si ya encontró alguna solución) en caso de que nos interese reducir la cantidad de camiones y no los costos totales).

Por último mencionamos que las posibilidades para considerar una clusterización incorrecta (y repetir todo el proceso) son: que hayan quedado nodos sin asignaro o que se haya excedido el acumulador de intentos antes de aumentar un cluster. Por otro lado, el algoritmo termina si se alcanza el acumulador de soluciones encontradas o si estamos buscando nuevas soluciones con una cantidad de cluster mayor a la cantidad de cluster de una solución ya encontrada y nuestro dato bandera esta en false).

Esto es en lineas generales la idea del algoritmo. Cabe mencionar antes de pasar a la complejidad algunas cosas: cuando verificamos la posición de un centroide de cluster respecto al promedio de las posiciones de sus nodos, en realidad podemos tener un acumulador de estas cosas en el cluster (que es un objeto) y cada vez que se le agrega o saca un nodo actualizarlo, esto nos permite no tener que hacer recorridos innecesarios en partes del programa). Algo parecido sucede cuando decimos que buscamos un vecino más lejano o carga actual del cluster o cantidad de nodos del cluster: cada vez que guardamos o sacamos un nodo actualizamos el valor.

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.

Uno de los primeros problemas que tuvimos fue que encontramos cuál era el costo del refinamiento para una determinada solución (fijando los parámetros de control) pero no podíamos establecer cuánto era el costo de obtener esa solución. Y es razonable que eso pase si miramos lo siguiente: el cluster no está bien o mal, a lo sumo está diferente de como nuestra percepción lo nota, sin embargo en este caso sí hay clusterizaciones correctas o no. Con esto nos referimos a que si la clusterización no asignó a todos los nodos, claramente no sirve para nuestro problema, debemos buscar otra. Pero es un problema complejo y definitamente de complejidad no polinomial asegurarse que hay manera de distribuir en k grupos a los clientes sin sobre-exceder demandas o que queden sin asignar. Básicamente necesitariamos analizar todos los subconjuntos del conjunto de clientes, cosa que para cualquier instancia grande es computacionalmente imposible. Entonces como no podemos saber si para un determinado k 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 que encontramos para establecer la complejidad.

Supongamos que no queremos seguir buscando soluciones con más clusters después de haber encontrado una con menos clusters. Además, una vez encontrada una solución no queremos seguir buscando una más óptima. La cantidad de intentos que le vamos a dar a la construcción de una clusterización va a ser lineal a n porque nos parece razonable pensar que si cada clusterización va a intentar asignar los nodos de tal forma que cada cluster tenga la posición que le corresponde respecto del promedio de posiciones de sus nodos, y cuando esto no pasa 'reorganizamos' y reorganizar podría significar mover nodos, entonces sería bueno que pueda hacerlo algún número de veces que sea lineal a n.

La cantidad de intentos que vamos a darle a clusterizar con una misma cantidad de clusters va a ser un número constante, la realidad es que como usuario se puede utilizar los valores que se quieran y que además esto dará ciertas garantías a la hora de encontrar la solución de mejor calidad. Pero dado que este parametro simplemente indica cuantos tipos de clusterización crear, con la misma cantidad de clusters, para ver si encontramos una solución y como cada clusterización empieza generando posiciones aleatorias y siguiendo los

mismos criterios para asignar los nodos.

Por último: una de las razones por la que nos tomamos la libertad de definir los parámetros de determinada manera para establecer la complejidad, es que al fin y al cabo estamos construyendo nuestra una heurística y podemos tomar ciertas libertades para ajustar determinadas cosas de la manera en que creamos conveniente. En nuestro caso, con un tiempo acotado para resolver el problema con distintos algoritmos, preferimos una configuración que permita correr muchos tipos de tipos de casos de CVRP en un tiempo razonable, con resultados que esperemos sean razonables, analizarlos y compararlos con otros algoritmos.

Dicho esto la complejidad del algoritmo responde a:

$O(TSP) + O(cantIntentosClusterMismoK \cdot cantIntentosParaFormarCluster \cdot costoDeDistribuirNodos)$

Donde:

1. cantIntentosClusterMismoK dijimos que $\in O(cte)$
2. cantIntentosParaFormarCluster $\in O(n)$
3. costoDeDistribuirNodos pensemos que son ciclos que recorren cada par de cluster y un tercer ciclo que recorre los clientes y los envía a uno u otro cluster, esto $\in O(n \cdot k^2)$.
4. $k \in O(n)$, ya que podemos afirmar que en peor caso n clusters seguro funciona, aunque no tendría sentido.
5. TSP $\in O(n^2 \cdot q)$, donde q es cte que explicaremos más a qué se refiere.

Pensemos en la cantidad de repeticiones de todos los ciclos que permiten generar una clusteriación inicial, ver si es válida, si lo es refinarla y realizar TSP y si no construir otra solución. En nuestro caso (habiendo fijado los parámetros como hicimos anteriormente), si consideramos el peor caso de no encontrar ninguna solución para $k < n$ sabemos que cuando $k = n$ ya va a existir solución. Necesitamos ver cuánto cuesta que k llegue a n para poder establecer la complejidad en peor caso. Para toda iteración donde la cantidad de clusters, k, sea menor a n, el costo de construir una clusterización con esa cantidad de clusters es $O(n^4)$, si necesitamos alcanzar los n clusters, es necesario hacerlo $(n-k) \approx n$ veces (como para aumentar el número de cluster hay un tope que es constante, no entra en consideración, cantIntentosClusterMismoK). En total sería $O(n^4 \cdot n \cdot cte) + O(TSP) \approx O(n^5)$. Como no existe nunca una clusterización correcta (excepto para $k = n$), sólo se realiza TSP una vez y como no queremos refinar la solución encontrada (no tiene sentido si hay un cluster por nodo), sólo se realiza una vez y como su complejidad es cuádratica no afecta a n^5.

En total : $O(n^5)$.

Para cualquier caso en donde existan soluciones anteriores, si fijamos el contador de soluciones en algún valor constante w, es decir, que se encuentren w soluciones antes de terminar, la complejidad seguiría siendo la misma, dado que nunca se va a necesitar iterar más que lo relatado anteriormente, ya que no tiene sentido que haya más clusters que nodos, y entonces la única manera de encontrar n soluciones (si no existen termina cuando $k=n$) es encontrando al menos una por cada clusterización con k clusters. El peor caso sería que

encuentre las w soluciones recorriendo todas las clusterizaciones desde el k inicial hasta k = n y ese caso ya lo analizamos arriba.

Podría reducirse la complejidad teórica definiendo a cantIntentosParaFormarCluster como cte y tener $O(n^4)$, pero nos parece que tiene definirlo lineal a n tiene más sentido. Por otro lado tomar el peor caso donde k = n resulta en casos bastante malos, casos que no tendría mucho sentido resolver, pero es la manera que encontramos de establecer su complejidad.

También es importante mencionar que en la práctica los flags que cortan las iteraciones deberían cumplir un rol importante para los tiempos de ejecución y esperamos que se note en las experimentaciones y que por otro lado, sólo para casos muy forzados tendríamos necesidad de que $k \approx n$. De hecho no sería interesante resolver este problema, ya que se traduciría en clientes con demandas tan grandes como la demanda de un camión. Sería, por ejemplo, un problema de distribución de mercaderías para una empresa u organización y en estos casos por lo que analizamos en la introducción teórica suelen aplicarse varios tipos de VRP. En la práctica esperamos que k funcione como una constante y que la complejidad esté más cerca de $O(n^3)$

Retomando un tema nombrado anteriormente en la complejidad de TSP, la constante 'q' representa una constante de refinamiento. Básicamente cuando realizamos TSP, variamos el valor de p (la probabilidad) que inicialmente vale 1 (para conseguir la solución golosa inicial) y después se realiza TSP para p = p-0.1 mientras p >0.5. O sea es un número constante las iteraciones de refinamiento.

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 llamado GRASP. Vamos a agregar a nuestro vecino más cercano una cuota de probabilidad, donde 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.

Iniciamos creando una matriz de distancias entre los nodos (entre los cuales está también el depósito). Introducimos en una cola el depósito, fijamos nodo actual = depósito, agregamos el deposito a una lista de nodos (que representará el camino) y realizamos, mientras exista nodo sin visitar, lo siguiente: marcamos como visitado el nodo actual y buscamos el nodo más cercano al nodo actual y que todavía no haya sido visitado. A medida que recorremos los nodos buscando el más cercano, guardamos también algún nodo que no esté visitado (para utilizar en caso de que salga la probabilidad (1-p)). Si después de esto no hay nodo sin visitar, significa que ya tenemos casi todo terminado y simplemente resta agregarlo al camino y devolverlo. Si en cambio hay nodos que visitar, extraemos una muestra random entre 0 y 1. Si la muestra está entre 0 y p, elegimos el nodo más cercano, en caso contrario elegimos el otro nodo y lo introducimos en el camino y además lo fijamos como nodo actual. Si por casualidad al momento de buscar el nodo más cercano sólo quedaba uno sin visitar, este nodo va a ser tanto el de distancia mínima (trivial) como el que habría que elegir en caso de

probabilidad (1-p), de modo que no hay problema con este caso.

5.0.4. Complejidad

Establecer la complejidad en esta parte es bastante claro debido a que está construido sobre un método muy intuitivo y el algoritmo queda muy evidente en términos de costo: sea n la cantidad de nodos, básicamente es $O(n^2)$ crear la matriz de distancias, e iterar sobre los nodos buscando el mínimo también es $O(n^2)$, ya que en peor caso asintóticamente el nodo que queremos 'está' último en el vector y necesitamos buscar el más cercano para cada nodo.

5.1. Experimentación: cluster-first, route-second: con alternativa a sweep algorithm

5.1.1. Parte 1: primeras observaciones de la performance

Esta primera aproximación nos servirá para conocer el funcionamiento del algoritmo frente a instancias de la vida real. Alguna de las preguntas que nos surgen son evidentes: Cuán cerca estará de las soluciones óptimas de la página? Encontraremos casos patólogicos relacionados a la clusterización? Y casos patólogicos que no podamos decidir si es producto de la clusterización? Qué rol en los tiempos de ejecución y calidad de la solución cumplirá el tamaño de la entrada? Nuestros parámeros definidos para refinar una solución realmente le generan un valor agregado?

Dado que explicamos la complejidad fijando ciertos parametros, vamos a experimentar con rangos acotados, para que se mantenga la correlación entre la complejidad teórica y la práctica.

El flag de seguir buscando soluciones con más cluster vamos a tenerla en true. La cantidad de intentos para poder generar una clusterización va a tener como tope una cantidad de iteraciones lineal a n. La cantidad de intentos antes de aumentar la cantidad de cluster para clusterizar lo vamos a fijar en una constante e inicialmente en 500. La cantidad de soluciones que queremos buscar será una constante, inicialmente en 200.

Habiendo planteado algunas interrogantes comenzamos tomando la instancia A-n32-k5.

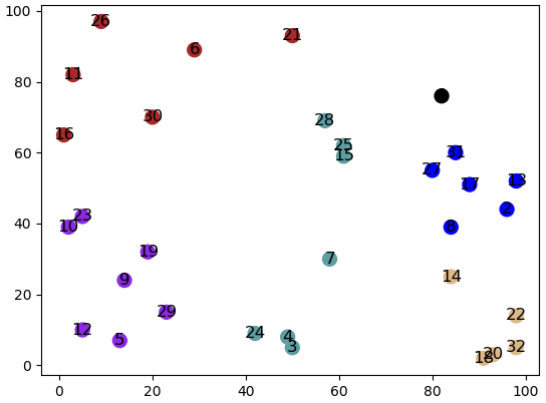


Figura 1: Así queda la clusterización

Sin información de cómo será la solución, parece bastante razonable la forma en que se distribuyeron los recorridos.

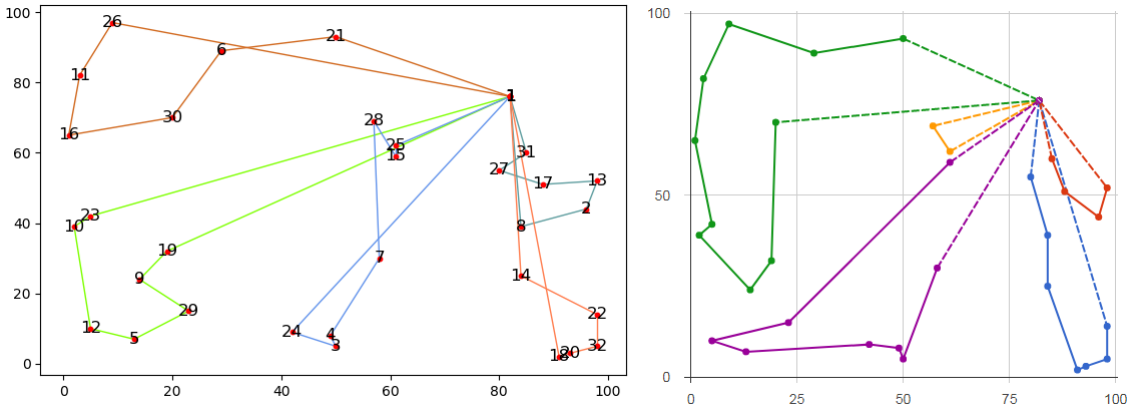


Figura 2: Ruteos: a la izquierda nuestra solución

La solución óptima es de 784 con 5 camiones, mientras que nosotros pudimos conseguir 898 con 5 camiones, computada en 3977ms ≈ 3 segundos. La desviación respecto de la óptima es del 14 % Mirando los caminos vemos que es posible que el camino violeta esté generando mayores distancias, mientras que esa zona en la solución óptima se resolvió distribuyendo unos nodos a caminos ya existentes y se dejo una ruta para sólo dos clientes. Antes de afirmar algo probemos otro test.

Figura 3: Clusterización de instancia A-n55-k9

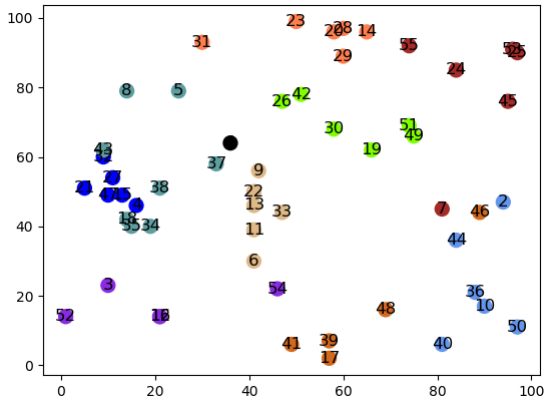
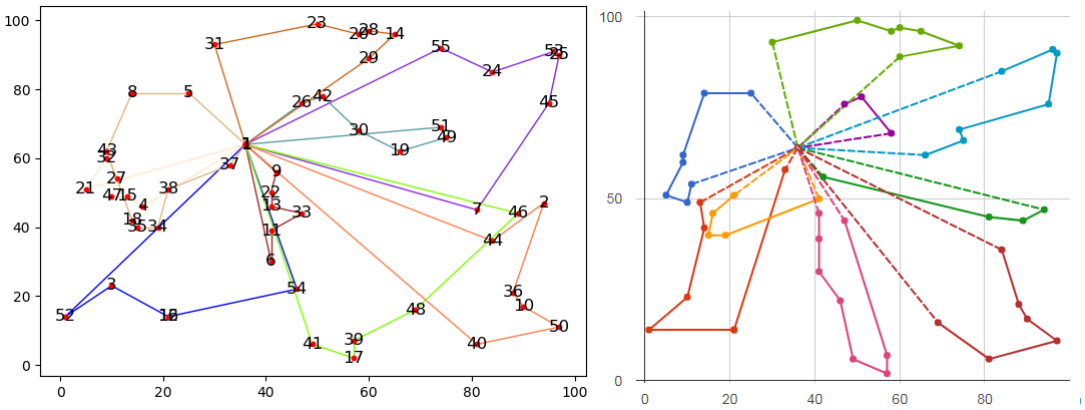
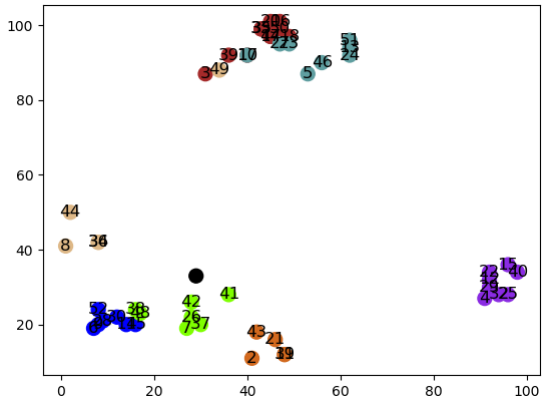


Figura 4: Ruteos: a la izquierda nuestra solución



Nuestra solución consiguió rutear con 9 camiones y un costo de 1185 en 33612 ms $\approx$ 33 segundos, mientras que la óptima era de:1073 con la misma cantidad de camiones. Es una desviación de 10 % de la óptima.

Figura 5: Clusterización test B-n52-k7



En este caso parecería un caso que sea bueno para resolver clusterizando, sin embargo, las demandas de los clientes de la parte superior generan tres cluster en lo que podría pensarse como uno sólo. Esto es producto de las demandas de los clientes y la capacidad del camión. Veamos los caminos:

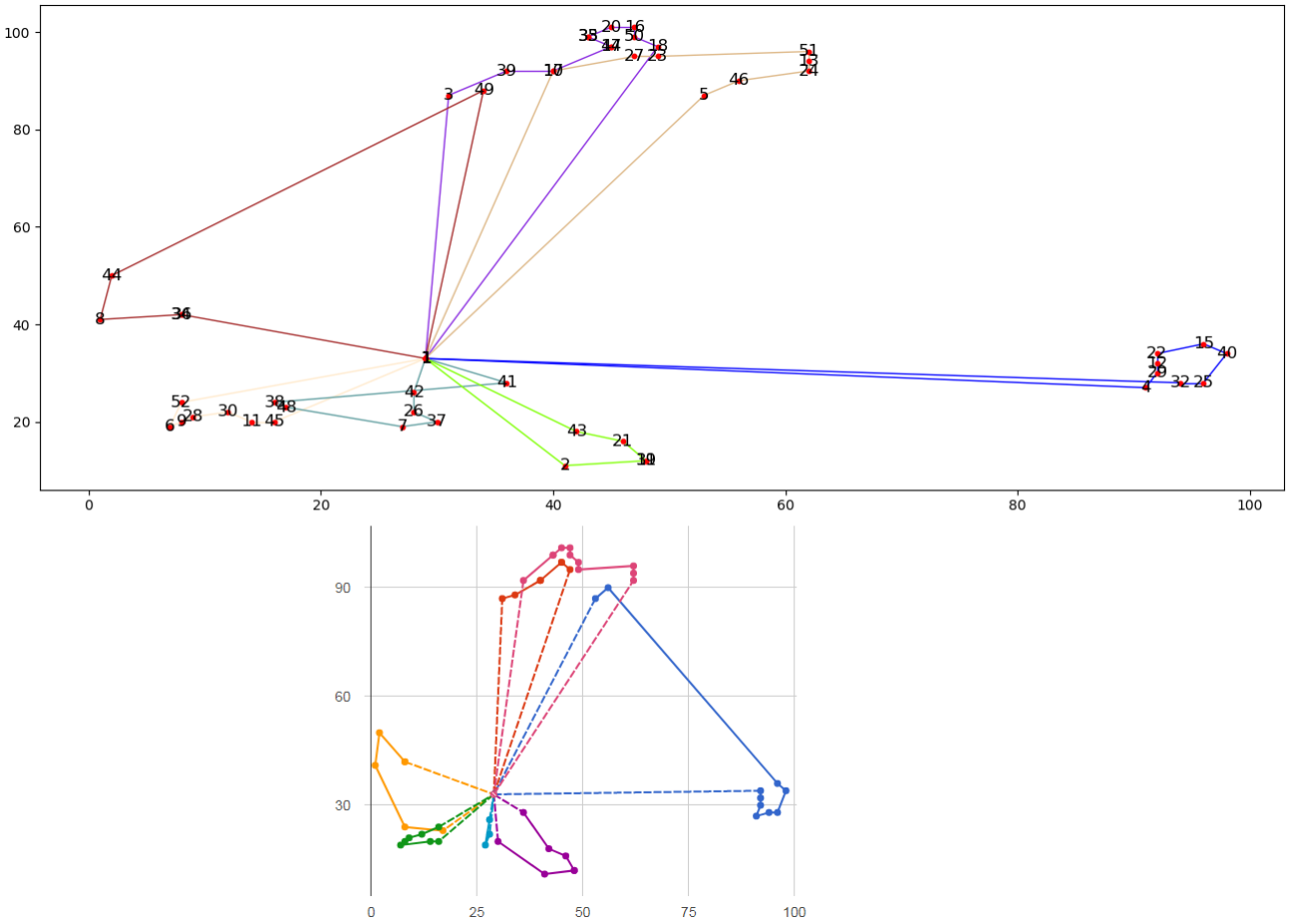
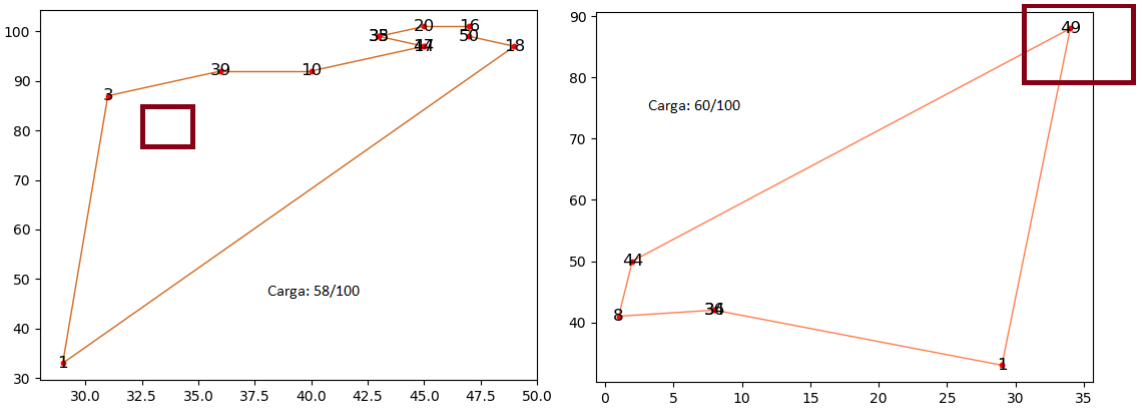


Figura 6: Ruteos: arriba nuestra solución

Nuestra solución utilizó 7 camiones y un costo de 778, mientras que la solución óptima fue de 747 con la misma cantidad de camiones. Una diferencia del 4 %, una muy buena calidad de solución. Respecto a los dos anteriores casos tenemos mejor calidad en la solución. En parte es esperable que suceda para casos en donde sea visible la utilidad de clusterizar: los primeros dos test no tenían a simple vista un patrón que permita agrupar los clientes. Además, mencionamos que en este caso los tres clusteres superiores seguro están perjudicando la solución. La solución óptima también clusteriza así esa parte pero algunos caminos son mejores: los clientes 44,3,36 deberían ser un cluster en sí mismo, a lo sumo junto con otros clientes que estén por debajo, no es conveniente agregar a este cluster cliente de la parte superior porque nos agrega caminos de alto costo.



Como vemos en la imagen anterior, y dado que el cliente 49 tiene de demanda 13, la mejor manera de agrupar esos conjuntos es mandando el 49 al otro cluster, que puede albergarlo.

Probando otro test, el B-n45-k6, nos encontramos con el primer caso en donde no podemos utilizar la cantidad de vehículos de la solución óptima.

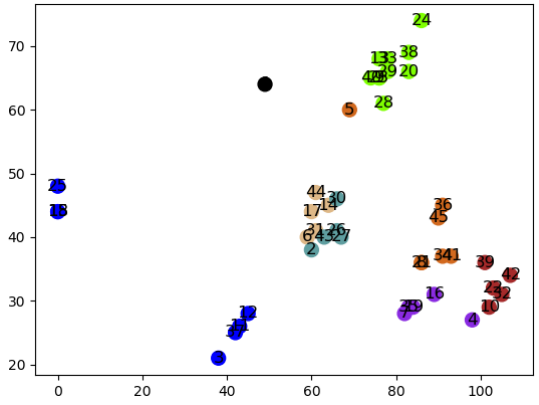
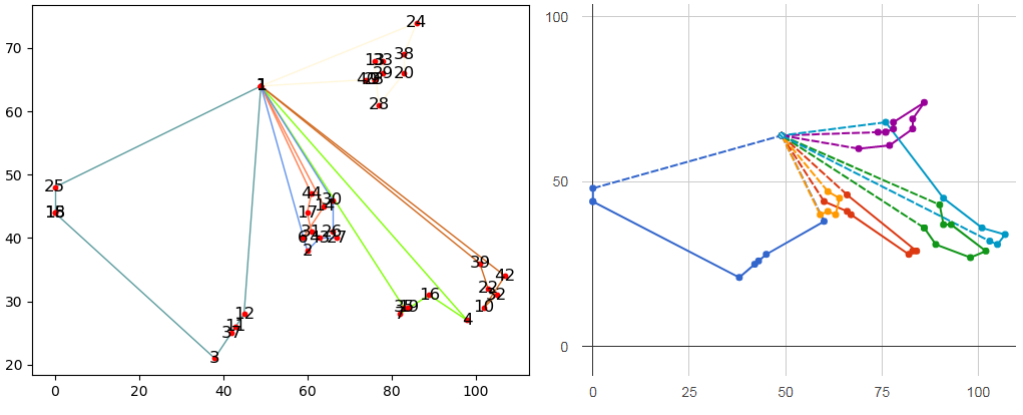
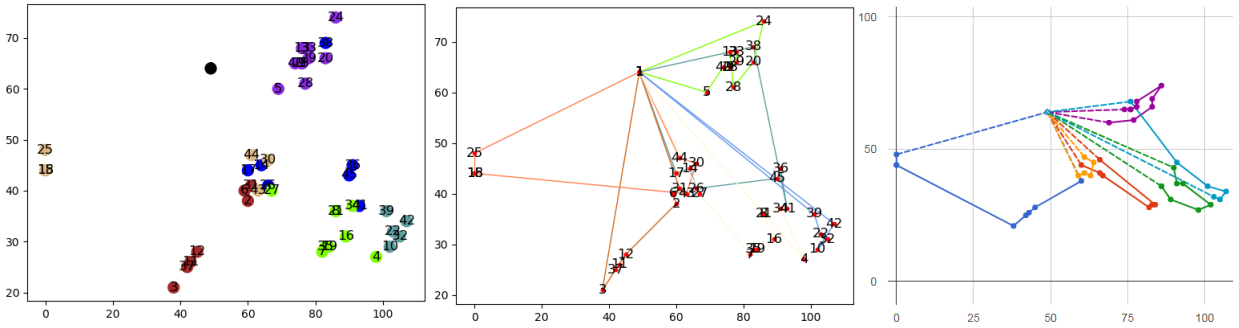


Figura 7: Clusterización



Nuestra solución necesitó de 7 camiones, mientras que la solución óptima utiliza 6. Respecto de los costos obtuvimos 752 contra 678, un 10 % de diferencia. El tiempo de ejecución fue de 25 segundos. Sin embargo, vemos la posibilidad de testear la funcionalidad de uno de nuestros control flags. Dado que buscamos exprimir al máximo

las posibilidades de conseguir la solución con 6 clusters aumentamos a 5000 la cantidad de intentos de ese flag. Los resultados son los siguientes:



Por un lado conseguimos obtener 6 rutas. El costo fue de 768 vs los 678 de la óptima, un

Figura 8: Ruteos: a la izquierda nuestra solución

13 % de diferencia. Otro punto a resaltar fueron los 126514 ms ≈ poco más de 2 minutos de ejecución contra los 25 segundos anteriores. Y acá es donde podemos aprovechar otra funcionalidad de los flags: nosotros teníamos 'activo' la posibilidad de buscar soluciones con más cluster: habiendo encontrado una con menos y siempre y cuando los acumuladores de los otros flags no lleguen a su tope. Durante la experimentación tenemos este flag activo por si encontramos algo interesante que remarcar. Al desactivarlo esperamos ver una reducción de los tiempos. Sin sorpresas respecto de la solución, ahora sí mejoramos el tiempo de ejecución: 65083ms ≈ 1 minuto.
También ya que estamos podríamos aumentar el acumulador de optimización de solución en busca de un costo más cercano al óptimo. Aumentamos este acumulador a 5000. Sin embargo en este caso seguimos encontrando el mismo costo. Vamos a ver más adelante donde volver a probarlo.

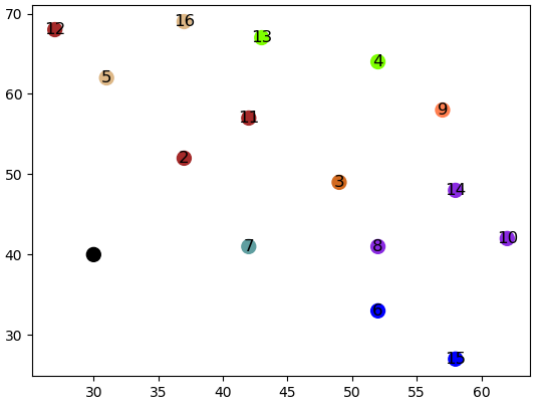


Figura 9: Clusterización test P-n16-k8

En este caso, las demandas van de 0 a 31 y la capacidad es de 35. Es altamente probable que pase lo que mencionamos anteriormente.
La solución óptima utiliza 8 camiones y tiene un costo de 450, mientras que la nuestra utiliza los mismos camiones y tiene un costo de: 462. Tiempos de ejecución:124658ms ≈ 2 minutos . Una diferencia de 2.5 %, evidentemente es una muy buena aproximación a la

solución óptima. Los caminos a simple vista parecen muy coherentes y similares. Algunas decisiones de camino incluso son idénticas, el único camino realmente diferente es el que visita los clientes 12 y 11. En la solución óptima los agrupa separadamente y nos atrevemos a decir que es esta la única decisión del algoritmo que nos separa de la solución óptima.

La observación importante que podemos hacer con este caso es que hay cierta relación entre los tiempos de ejecución y el número de clusters que se usan. En el test anterior teníamos casi el triple de clientes y con la misma consideración de flags el tiempo fue de 25 segundos. Sólo cuando aumentamos los topes el tiempo fue de 2 minutos, pero en esos casos en particular estamos casi obligando a iterar muchas más veces. En este caso el problema sin exigirle grandes refinamientos probablemente necesita iterar muchas veces, probablemente debido a dificultades para armar la clusterización (al haber una distribución tan poco 'agrupable', cualquier cambio en la posición del centro de un cluster genera movimientos de clientes a lo largo de toda la clusterización. Si además las demandas están tan cercas de la capacidad, cualquier movimiento de clientes a otro cluster requiere eliminar un cliente de ese cluster para no exceder la demanda. Estas son algunas de las cosas que deben estar pasando en este caso relacionado al tiempo de ejecución. Pero la calidad de la solución al fin y al cabo es muy buena.

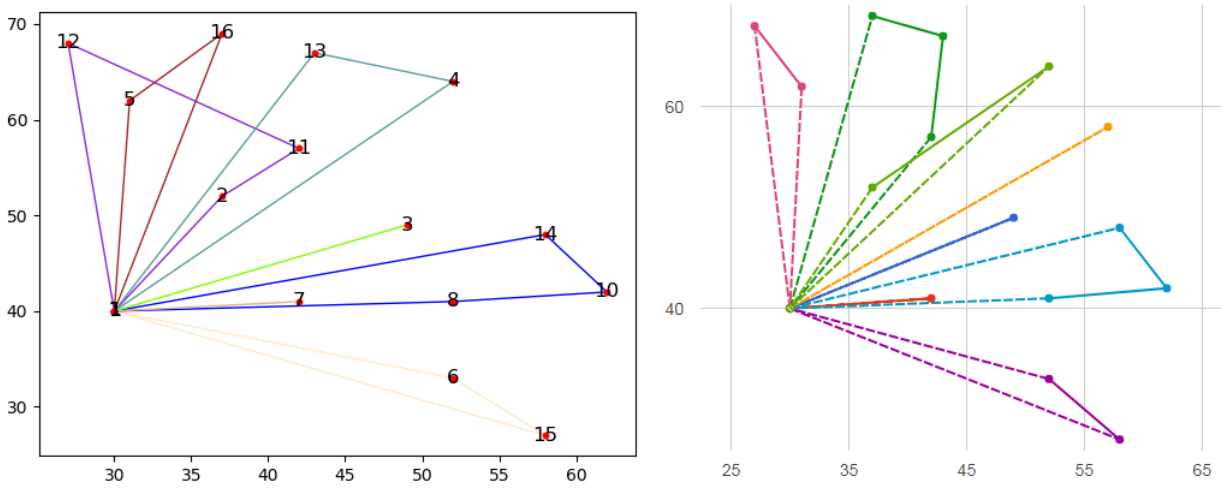


Figura 10: Ruteos: a la izquierda nuestra solución

Para compararlos, tomamos el test P-n45-k5. Las demandas van de 0 a 41 y la capacidad del camion es 150. Buscamos ver que en este caso los tiempos de ejecución sean menores sobre todo en proporción al caso anterior.

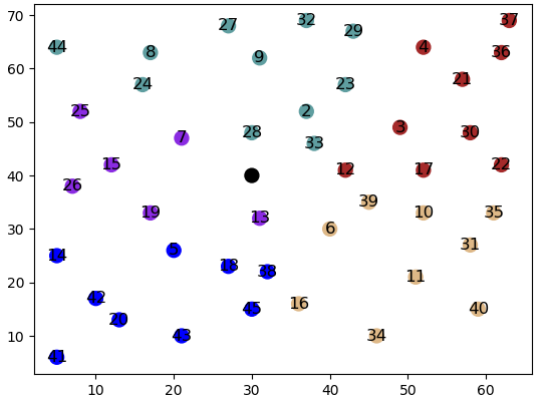


Figura 11: Clusterización test P-n45-k5

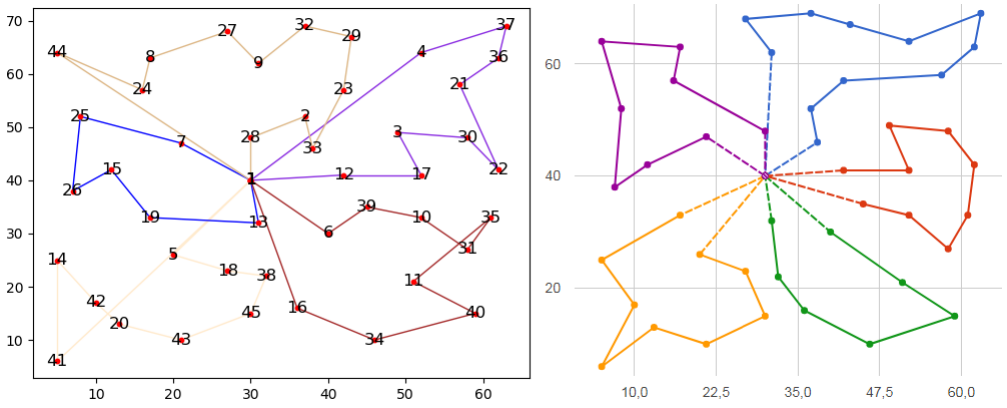


Figura 12: Ruteos: a la izquierda nuestra solución

Solución optima: 5 camiones y costo : 510. Nuestra solución: 5 camiones y costo:582, diferencia: 14 %. Viendo los caminos no hay mucho más para analizar, anteriormente mencionamos la necesidad de cierto refinamiento adicional para poder conseguir los mejores caminos, haciendo algo parecido a una permutación de clientes entre clusters y ver si mejora. Nosotros eso no lo tenemos y por eso consideramos que existen estas diferencias, sin embargo, consideramos que son soluciones de bastante buena calidad. Lo que nos interesaba discutir de este caso es los tiempos de ejecución. El tiempo de ejecución fue de 9699 ms $\approx$ 9 segundos.

Pasemos al caso de test P-n60-k15.

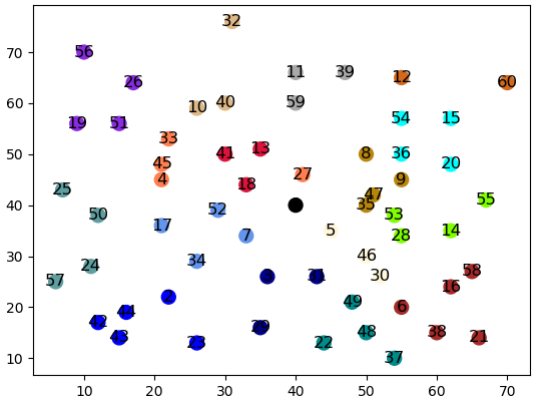


Figura 13: Clusterización test P-n60-k15

La primera observación que podemos hacer es que la clusterización tiene muchisimos más colores (evidente por tener más cluster). Otra no tan evidente a simple vista es que no pudimos encontrar un ruteo con 15 vehículos: necesitamos usar 16 con costo 1045. El tiempo de ejecución fue 793727 ms $\approx$ 13 minutos. La solución óptima acusó 15 camiones y costo de:968.

Hay varias cosas para resaltar: es un ejemplo práctico de lo que analizamos anteriormente, necesitó muchas iteraciones para ir aumentando de cluster y dado que no encontró solución con menos clusters, significa que consumió todos los contadores de control, particularmente el que controla las iteraciones de refinamiento de una clusterización, que va en el orden de n. Si para una clusterización no encuentra ninguna solución, va a necesitar muchas iteraciones para poder avanzar de cluster, hasta llegar a uno que sí permita encontrar solución.

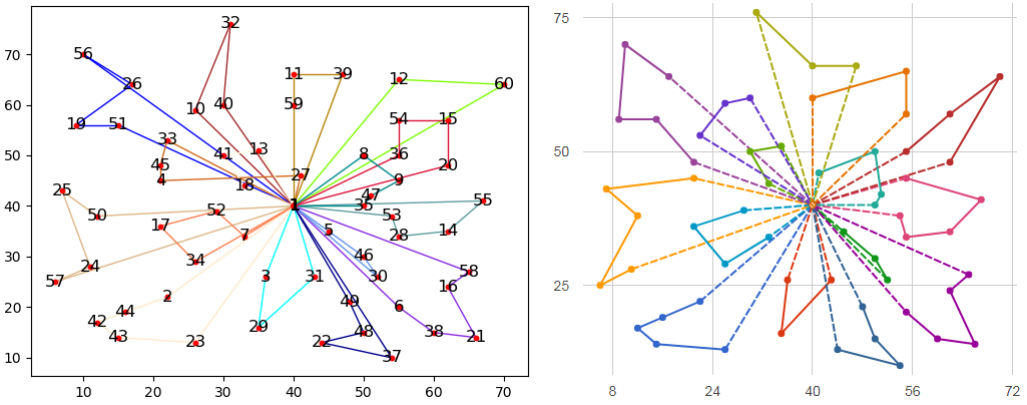
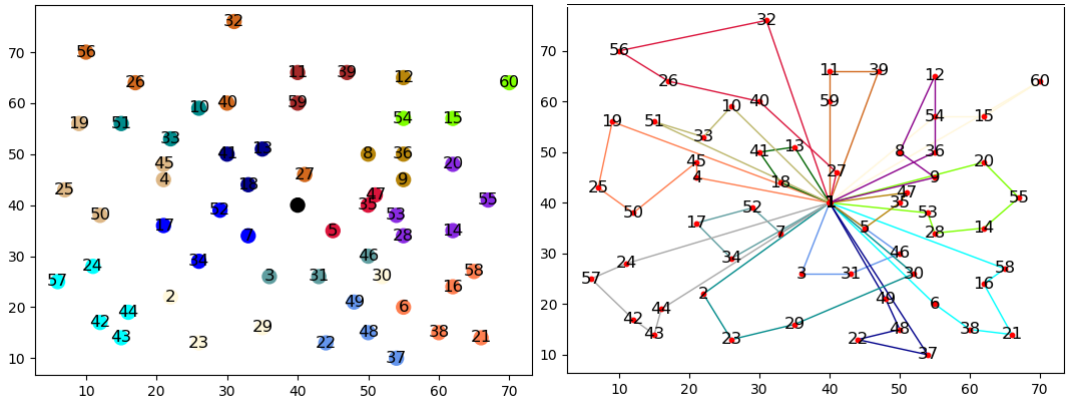


Figura 14: Ruteos: a la izquierda nuestra solución

Este tipo de casos podemos empezar a pensar que son considerablemente malos para nuestro método. Primero porque la distribución de los puntos genera que cualquier movimiento en la posición del centro de un cluster puede mover muchisimos nodos de cluster. Por otro lado, hay muchos clusters debido a que cada cliente tiene demandas muy cercanas a la total del camión y además internamente cada vez que podamos mover un nodo a otro cluster es altamente probable que necesitemos sacar algún nodo de ese clsuter (para que no exceda a la demanda del camión) y entonces cada iteración no necesariamente se 'libera' de un nodo, al contrario, muchas iteraciones siguen teniendo los

mismos nodos por asignar.
En línea con lo que intentamos anteriormente, vamos a desactivar el flag de seguir buscando solución con más clusters, vamos a aumentar los intentos de optimización de solución y intentos para aumentar cluster a 5000. Veamos si conseguimos algo mejor. Los resultados son:



Como anteriormente paso, logramos rutear en 15 vehículos e incluso reduciendo el costo:1023, ahora la diferencia es de 5 % pero con la misma cantidad de camiones. Obviamente que todo tiene un costo y en este caso fue pasar de 13 minutos a 18 minutos.

Probemos con otro tipo de test respecto de los que veníamos probando: B-n63-k10.

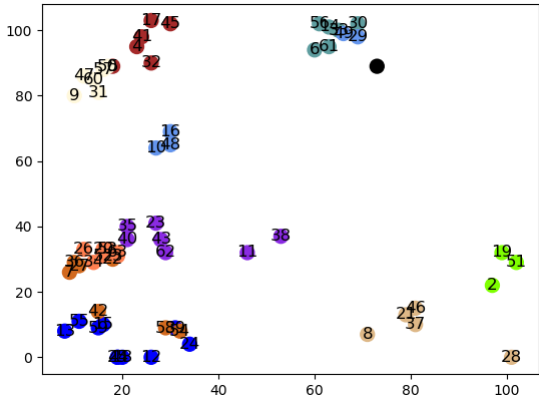


Figura 15: Clusterización test B-n63-k10

Primera observación: en este caso sí pudimos encontrar solución con 10 camiones y de costo 1601. La solución óptima es de 10 camiones y costo: 1496, un 7 % desviado de la óptima. El tiempo de ejecución fue de 59550ms ≈ 1 minuto. La observación más importante que podemos hacer es relacionada con el caso anterior: tenemos el mismo tamaño de clientes y sin embargo ambos test tardan tiempos muy diferentes. Esto nos empieza a indicar que si bien el tamaño de la entrada nos afecta, enntran en juego tambien cómo o cuántos clusters se necesitarán para rutear. Vimos que instancias donde necesitamos aumentar los flags para mejorar la solución el tiempo aumentaba mucho más cuándo los clientes estaban dsitribuidos de forma uniforme.

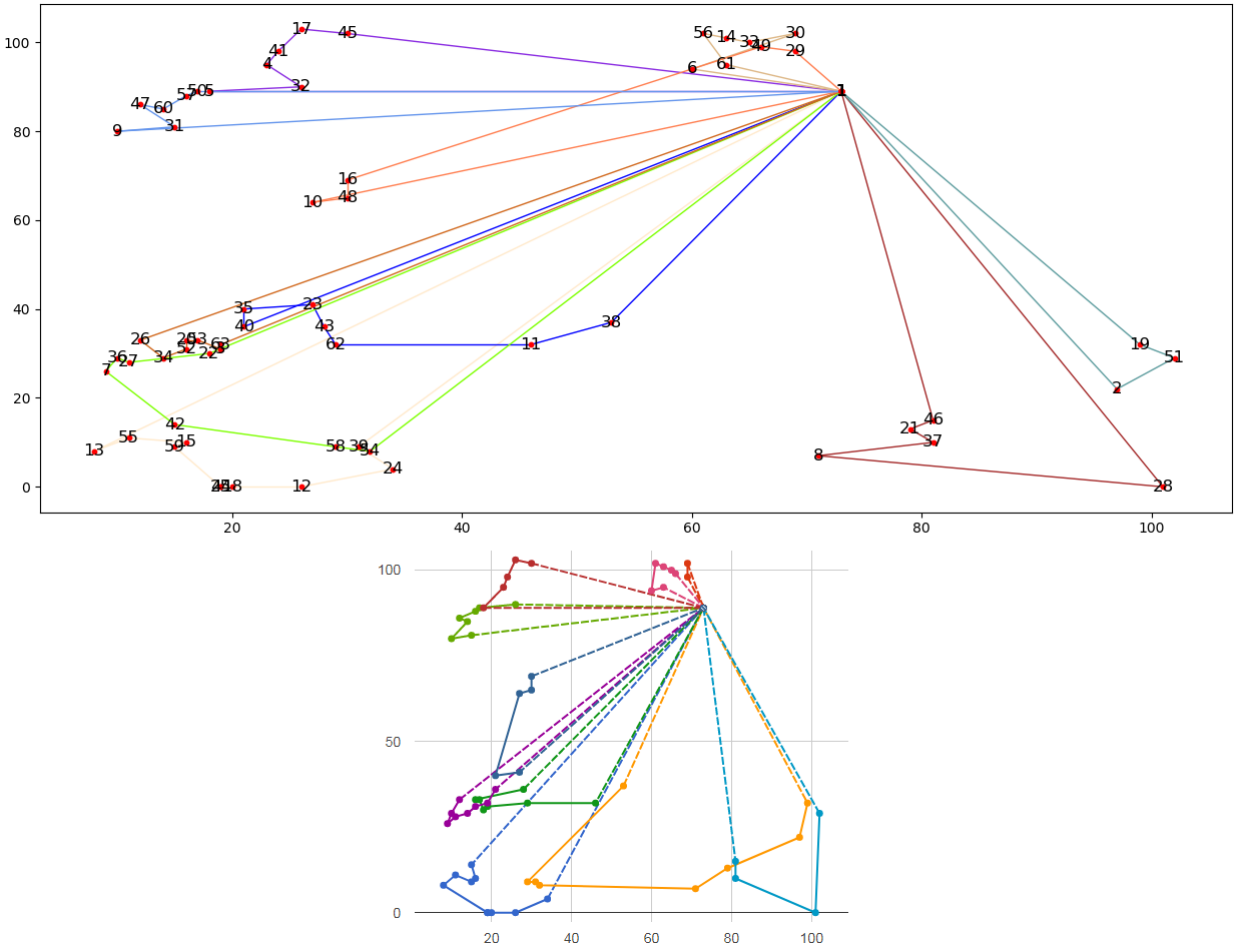


Figura 16: Ruteos: arriba nuestra solución

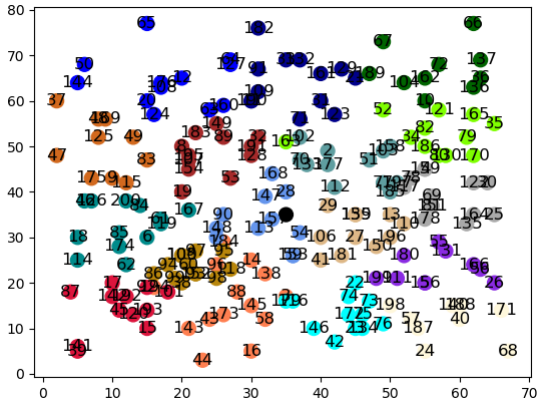
Antes de pasar a la experimentación final nos interesaría comprobar una cosa. Nosotros pudimos mejorar soluciones (sonbre todo reducir los vehículos) aumentando algunos flags de control. Esto nos generaba naturalmente mayores tiempos de ejecución por la naturaleza de su funcionamiento: al aumentarlos exigimos que determina parte del programa siga intentando más veces a ver si encuentra algo mejor. Pudimos tener éxito haicendo esto tanto en casos con 10 clientes como con 150, en casos en donde la distribución uniforme de los clientes no acusaba, a simple vista, una clusterización y también en casos en donde la clusterización era más evidente. Lo que todavía no terminamos de ver es la siguiente relación: si bien es verdad que los tiempos de ejecución aumentan al aumentar los flags, creemos que lo hacen de forma menos agresiva cuando la distribución de los puntos no es uniforme, sino que hay concentraciones en determinados puntos.

5.1.2. Parte 2: configuración según el caso para la mejor performance

Para esta parte de la experimentación vamos a buscar casos de test bien representantivos de lo que buscamos mostrar, explicarllo y comentar qué esperamos ver y posteriormente correr el algoritmo.

Como breve resumen sabemos que instancias de distribución uniforme de puntos en el plano no son buenas para los tiempos de ejecución o para encontrar una solución cercana a la óptima o para tratar de mejorar aún más la solución. Cuando se trata de distribución con ciertos grupos de clientes claramente separados (por lo menos para nuestra percepción) los tiempos de ejecución son mucho más asumibles y en caso de buscar óptimizar aún más la solución esto se suele mantener. Otro de las cosas que nos generan problemas es cierta disparidad en las demandas de los clientes y particularmente suele afectar más a la clusterización. Cuando anteriormente tuvimos una distribución homogénea de puntos a lo largo de todo el plano y no conseguimos una solución óptima (en función de la cantidad de camiones) volvimos a correr el algoritmo mejorando algunos flags y finalmente encontramos una mejor solución. Esto es algo que cuando los puntos se distribuyen en zonas suficientemente alejadas y encuentra este tipo de problemas, no suele poder evitarlos y termina usando más camiones. Creemos que esto pasa porque cuando hay muchos puntos juntos por todo el plano, sacar a uno de un lado y enviarlo a otro no afecta mucho a la solución final dado que las diferencias quizá son mínimas y porque la cercanía de los clientes suele permitir que existe siempre alguno más o menos cercano al que se puede unir. Cuando no hay una distribución así, sacar algo de un cluster y enviarlo a otro puede generar o rutas muy costosas (por la distancia entre los clusters) y/o necesitar más clusters, debido a que el cluster al que se quiere mandar un cliente ya esta lleno. Si bien nuestro algoritmo trata de evitar esto realizando muchos tipos de clusterización con el mismo numero de cluster y buscando la mejor, realizar esto de forma precisa y exacta es exponencial y nuestra heuristica para resolver el problema apunta a una solución aproximada de la mejor calidad posible en un tiempo razonable. Hechos los comentarios pertinentes, comenzamos.

Figura 17: Clusterización test M-n200-k17



Elegimos comenzar por un caso de los que venimos mencionando: clientes distribuidos homogéneamente aunque demandas considerablemente bajas en comparación a la capacidad del camión. Dado que en comparación a n la cantidad de clusters no parece ser tan grande, creemos que seteando los topes en 500 y el flag de buscar solución para más clusters en false (para obtener mejores tiempos) podemos conseguir rutear en 17 rutas. Si como en casos anteriores habría más 'colores' es posible que aumentaríamos los topes. Los tiempos de ejecución seguramente serán alto incluso así y es posible que no encuentre la solución óptima en función de la cantidad de rutas o del costo total . Veamos los resultados:

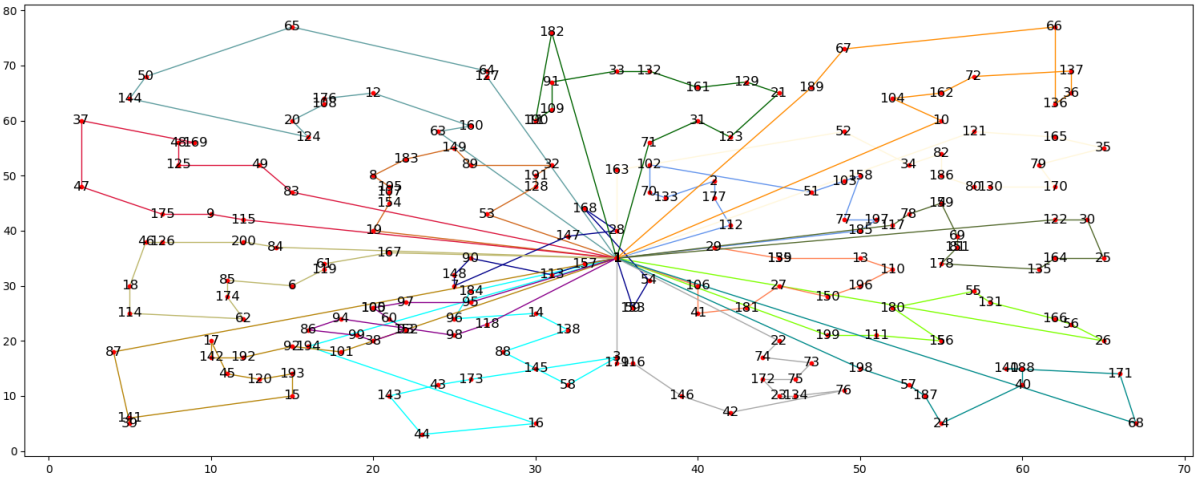
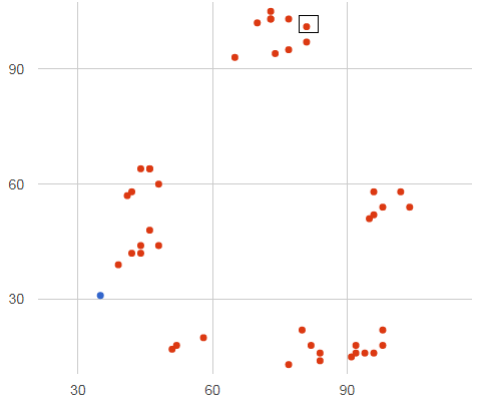


Figura 18: Clusterización test M-n200-k17

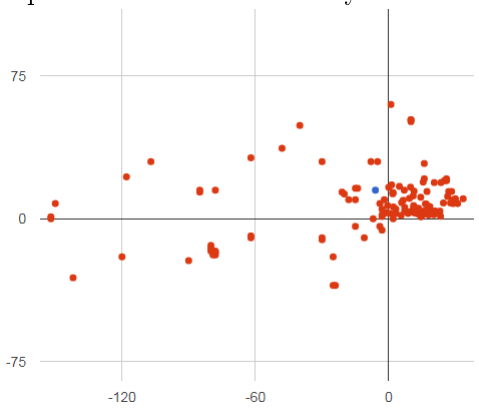
Los caminos los ponemos a modo ilustrativo, no sirve mucho para compararlos con la solución óptima porque se convertiría en una discusión aparte acerca de las decisiones tomadas por los algoritmos. Lo interesante es que el algoritmo logro rutear en 17 rutas, tal como la solución óptima. La diferencia en calidad en este caso radica en los costos: 1536 vs 1275 respectivamente (un 20 % más). Si seguimos el hilo de los anteriores test, encontramos que para los estándares que veníamos manejando es una relativamente mala solución. Además hay que comentar los tiempos de ejecución: alrededor de 30 minutos.

Seguimos con un caso que pensamos sacará ciertas ventajas en la clusterización. El test es el B-n44-k7. La capacidad total es 100 y las demandas están entre 0 y 69.



En la imagen identificamos un punto problemático: tiene demanda 69. Como vimos, esto puede generar un problema pero dado que la distribución de los puntos nos conceda ciertas ventajas. Además vamos a establecer en 200 los topes y en false el flag de seguir buscando solución con más clusters. Los mayores problemas suponemos que estarán cuando tengamos que juntar clientes de forma 'inteligente' con el cliente que tiene una demanda de 69. En caso de no poder hacerlo en las iteraciones que le damos, nos agregará más cluster a la solución, sin embargo, la distancia de los cluster y la cantidad de puntos problemáticos (sólo uno) nos da confianza en poder resolverlo sin grandes topes.

Seguimos con un caso que no termina de ser muy evidente en que grupo estaría.



Por un lado no hay una distribución muy homogénea a lo largo del plano, tampoco hay grupos de clientes evidentes, la capacidad es de 2210 y las demandas van entre 0 y 1126. Sin embargo analizando los clientes, no encontramos el que tiene demanda 1126, de hecho la mayoría ronda en 0 a 100 de demanda, de modo que suponemos que el que tiene esa demanda esta dentro de la zona cercana al depósito de mayor densidad. El hecho de que haya pocos clusters en relación a los clientes nos da cierto indicio de que puede ser un caso de los 'buenos'. El hecho de que no haya muchos puntos problemáticos (no marcamos ninguno porque no lo encontramos) también sería un indicador de menos problemas. La configuración que elegimos va a tener en cuenta todo esto y si bien confiamos en poder resolverlo cerca de la solución óptima, dado que tiene cierto parecido a una distribución uniforme y que hay 135 clientes, podría ser un limitante a la hora de computar una solución. Seteamos los topes en 400 y el flag de seguir buscando solución en false.

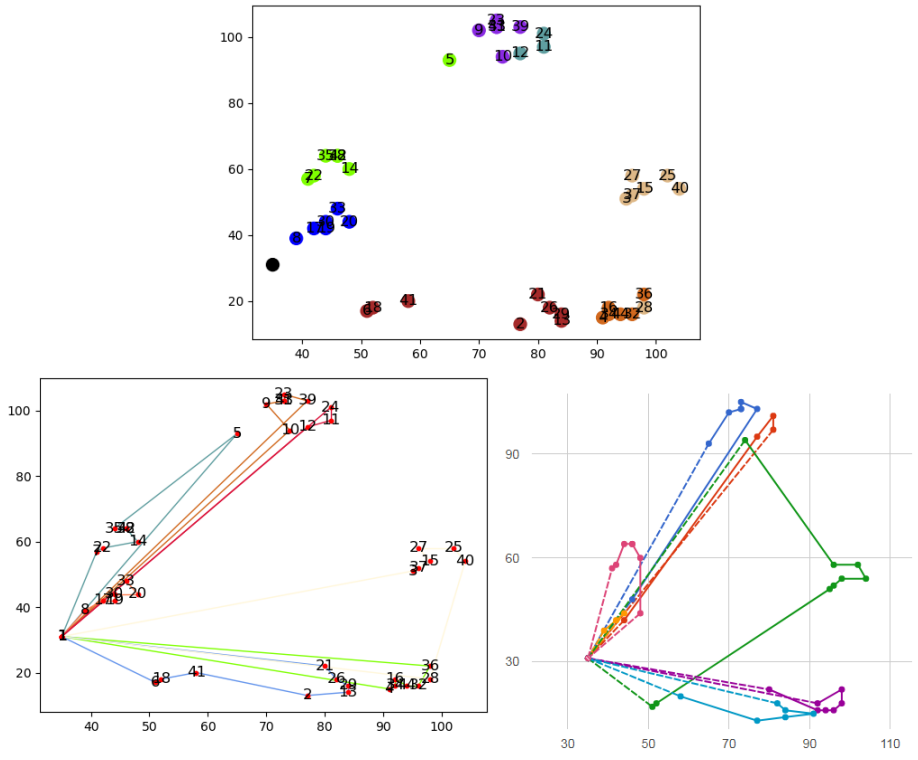


Figura 19: Clusterización test B-n44-k7

Figura 20: A la izquierda nuestra solución

Los resultados son: la solución óptima usa 7 vehículos y costo de rutas de: 909 mientras que nuestra solución es de 7 vehículos y 967 de costo en 37 segundos. Una muy buena solución, del tipo de lo que venímos encontrando en este tipo de casos con desviaciones de menos del 10 % de la óptima.
Inicialmente lo que obtenemos es un ruteo con 4 rutas y costo: 546, contra los 534 de la óptima, en 35 segundos de ejecución. Aumentando los flags que venimos utilizando a 5000 seguimos obteniendo un ruteo con 4 clusters y 546 de costo en 95234 ms $\approx$ 1 minuto y medio. En este caso es difícil generalizar la situación. El problema radica en que el cliente 22 tiene una demanda de 1500, por lo que agregarlo al cluster de la izquierda no es una opción. Lo que queda es agregarlo a los cluster de la derecha pero ahí surge un problema: los centroides de esos clusters están lo suficientemente alejados de este cliente como para que matcheen en algún momento. Por esa razón, el algoritmo termina decidiendo, despues de agotar las iteraciones, que hay que aumentar de 3 a 4 clusters y ahí si encuentra solución.
Respecto de los tiempos, nos sirve para ver que junto con los casos anteriores, cuando los puntos no están tan uniformemente distribuídos los incrementos en tiempos de ejecución son mucho más leves.

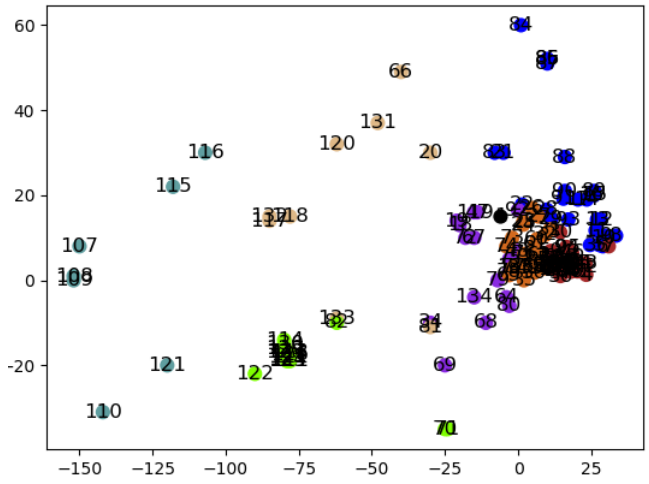


Figura 21: Clusterización test F-n135-k7

En este caso, no vemos beneficio en mostrar los caminos, sino simplemente la clusterización y la solución en sí. La clusterización se ve bastante parecido a lo que discutimos: al no tener tantos puntos problemáticos puede pensarse este dataset casi como un caso bueno para este algoritmo. Por otro lado, el resultado fue: 7 rutas de costo total de 1356 y ejecución en 1 minuto y la solución óptima es de 1162. La desviación es de 16 % lo cual consideramos una buena calidad de solución.

Por último como caso curioso presentamos el E-n30-k3. Lo raro de este caso es que no pudimos clusterizar en 3 rutas sino en 4 con 546 de costo, y la solución óptima es de 534 de costo con 3 vehículos. Es curioso porque la cantidad de cluster es relativamente chica, la cantidad de clientes también y sin embargo incluso probando con aumentar los topes seguimos sin poder mejorarla. Antes de explicar las teorías dejamos unos gráficos:

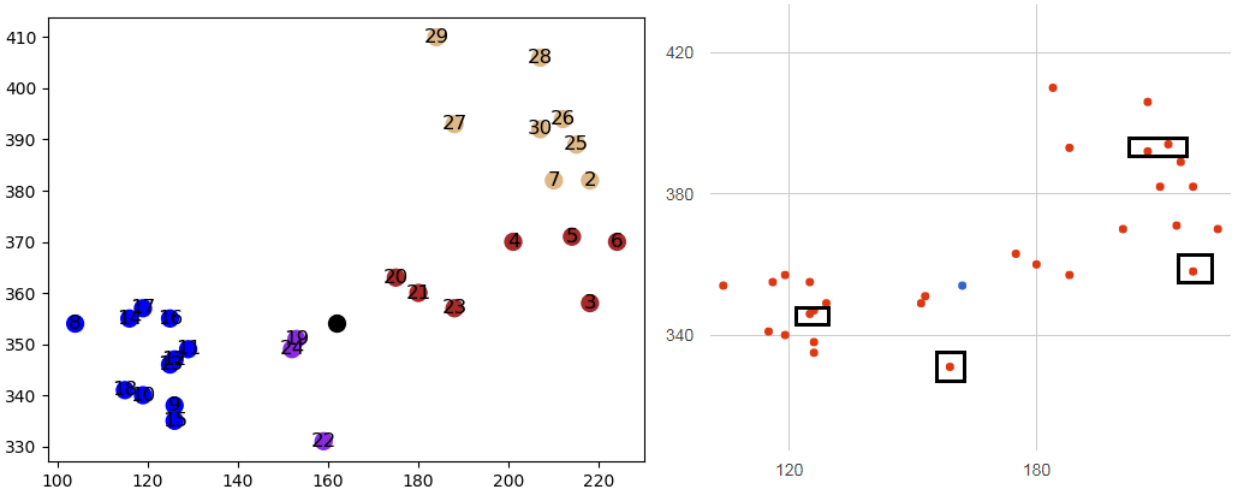


Figura 22: Clusterización test E-n30-k3

El motivo tiene que ver con los clientes señalados: habíamos visto en anteriores instancias que había puntos problemáticos, que era posible que generaran muchas más iteraciones por las demandas que tienen y particularmente que podría afectar más a distribuciones heterogéneas debido a la 'falta' o 'lejanía' de vecinos para matchear en ciertos casos. En este caso sucede algo de este tipo, la capacidad del vehículo es 3100 y las demandas van de 0 a 3100, donde los clientes señalados tienen mucha más demanda que sus vecinos y por lo tanto al clusterizar generan problemas.