Inteligencia Artificial

Informe Final: G-VRP

Filip Cornell

30 de junio de 2017

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen

En este artículo, se intenta juntar y sumar lo que ha sido hecho en el campo del problema G-VRP, el Green Vehicle Routing Problem. El G-VRP trata de, primeramente, minimizar la distancia que recorre una flote de vehículos, visitando diferentes clientes con la opción de recargar la energía en la ruta. En varios ejemplos de implementación, se ve también una minimización de la cantidad de vehículos. Se junta lo que se ha hecho anteriormente en un Estado del Arte, y las técnicas y las implementaciones matemáticas. Luego, se intenta implementar una técnica completa, para ver si podría compararse con las técnicas anteriores. Se llega a la conclusión que no es eficiente comparado con las otras técnicas debido a su ineficiencia y la complejidad del problema, aunque mejoró el resultado de una instancia, aunque el algoritmo fue modificado en ese caso.

1. Introducción

Hoy día, con los cambios climáticos, más regulaciones de combustibles están puestas en marcha para reducir los emisiones de gases que provocan los cambios climáticos. [2] Como resultado de esto, compañías de transporte están poniendo sus pensamientos en buscar otras soluciones para entregar sus cargos que con sus vehículos normales. Uno de estas soluciones es empezar usar otros tipos vehículos. Estos tipos de vehículos son eléctricos, y tienen varias ventajas. Primero, tiene un efecto mucho menor al ambiente, ya que no emiten los gases que contribuyen a los cambios climáticos. Segundo, hacen mucho menos ruido que vehículos con motores normales. [2] No obstante, los vehículos eléctricos (EV - electric vehicle), todavía tienen fallas. La mayor falla es, según Schneider (2014), la distancia limitada que pueden recorrer por la capacidad de la batería. [2] Aunque las baterías con el progreso tecnológico están mejorando su tiempo, todavía son muy limitados, y pueden normalmente recorrer 100-150 miles (160-240 km) antes de que necesiten volver a cargarse. [2] Esto lleva la necesidad de planificar bien las rutas recorridas de

los vehículos, para poder minimizar la distancia recorrida, hacer las visitas a clientes en tiempo y minimizar el número de vehículos usados. El problema GVRP intenta hacer esto, considerando unos factores reales, como que los vehículos tienen una cantidad de energía limitada, y que se puede recargar la energía de los vehículos en ciertos lugares.

En este estudio, se busca juntar conocimiento sobre el problema utilizando estudios anteriores. El artículo es organizado en la siguiente manera. En sección 2 se define el problema, y se discute cuáles son los variables, los conjuntos, las restricciones y los parámetros, y se ve si hay problemas relacionados. En sección 3, el Estado Del Arte, se investiga más profundo lo que se ha logrado anteriormente en este problema. Cuales algoritmos se ha usado, qué eficiencia han tenido y cuál sería la más eficiente. En sección 4 se ve unos ejemplos de modelos matemáticos, así como se podría implementar este problema matemáticamente. Sección 5 describe cómo el problema fue representado en la implementación seguido de sección 6 que describe cómo fue implementado el algoritmo Forwardchecking + GBJ. Sección 7 nos cuenta cómo se hizo los experimentos para llegar a los resultados mostrados en sección 8. Finalmente, se llega a las conclusiones en sección 9, donde se discute cuál método sería el mejor, las diferencias entre los modelos diferentes, dónde se debería seguir con las investigaciones, y se discute cómo fue la técnica usada en este árticulo comparado con los anteriores.

2. Definición del Problema

2.1. ¿En qué consiste el problema?

El problema que se enfrenta en este artículo, el Green Vehicle Routing Problem, es, como tal vez se pueda entender del nombre, una variante del problema Vehicle Routing Problem, el VRP. Esto en sí, es un variante del problema clásico TSP (Travelling Salesman Problem). El TSP clásico, consiste en encontrar la ruta más corta para que un vendedor, que viaja entre ciudades, las visite, y luego velva a la ciudad donde empezó. Esto se quiere hacer recorriendo la distancia mínima posible. [5] Con el G-VRP, siendo un variante del VRP y por ende una variación de este problema, se busca siempre minimizar la distancia recorrida de los vehículos involucrados en el problema. También, se implementa el problema en una manera similar. Los clientes y las estaciones se puede considerar como vértices, con rutas entre sí, que se considera como arcos. A estos arcos, hay parámetros relacionados que indican la distancia entre ellos. No obstante, hay varios factores que difiere este problema del TSP clásico.[2]

2.2. El función objetivo

Primeramente, algo que unea todas las variaciones de este problema es que siempre se busca encontrar la solución con el costo práctico menor posible. Dependiendo de cómo se ha definido estos costos, se obtiene funciones objetivos diferentes para el problema. Uno de los factores que difiere el G-VRP del TSP es que no se tiene solamente un viajero; se tiene una flota de vehículos que juntos visitarán todos los clientes en la manera más eficiente posible. Prácticamente es mejor tener la cantidad mínima de vehículos activos, no solamente ya que esto lleva menos conductores, y por ende menos costos para la empresa, pero también porque lleva una mejor solución. Por ejemplo, en "The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations"(Schneider, 2014), se usa la técnica de primero minimizar la cantidad de vehículos en forma de un CSP, ya que una solución con menos vehículos, según M. Schneider (2014), siempre es mejor, para luego implementar el modelo, solamente minimizando la distancia en la función objetivo. [2] En otra variación del problema, se implementa esta parte solamente como una restricción que solamente limita que se puede usar una cantidad máxima de vehículos. [3]

2.3. Variables

Por lo general, se tiene las siguientes variables incluidos el problema.

- Variables de cuáles rutas se usa entre los vértices. Estos vienen siempre en el problema G-VRP, ya que son el fundamento para los variantes del TSP. Suelen ser binarios, y son 1 si se toma una ruta específica, y 0 en otro caso.
- Variables de energía que queda en el vehículo. [2] [1] Este tipo de variable es importante, ya que es fuertemente relacionado a cuando se debe visitar una estación de recarga. Puede ser la cantidad de energía cuando se sale o cuando se entra de una estación. [6]
- Variables de tiempo. En varios variaciones del problema, se considera el tiempo como un variable. Esto, ya que en la realidad clientes solamente pueden recibir el cargo en un par de horas fijas cada día. También hay que tener en cuenta que las visitas a los clientes demoran tiempo, igual que las recargas en las estaciones de lo mismo. [2] Estos pueden ser, por ejemplo, el horario de llegada en una estación o el horario de salida del almacén. [2] [3] [1] [6] Estos son importantes para poder tener en cuenta que clientes en la realidad solo pueden recibir cargas entre horarios específicos, y que el tiempo de recarga no es instante. Esto puede variar, o ser constante, dependiente de cómo se ha definido el problema.

2.4. Restricciones caracterísiticas

Lo que tal vez difiere los más el G-VRP del TSP clásico, es que los vehículos tienen un monto limitado de energía para poder recorrer la zona de clientes. Considerando esto y que cada vehículo tiene que volver al vértice donde empezó (como el TSP clásico), un vehículo tiene dos opciones. Puede tomar una ruta corta para luego volver al almacén, o puede recargar la energía en una de las estaciones de recarga en la ruta para después seguir y no volver al almacén. Muchas veces, las mejores soluciones incluye recargar en una estación, ya que esto resulta en que se puede hacer una ruta mucho más larga, y evitar usar muchos vehículos. En algunos problemas, la distancia a un cliente puede ser tan larga que no hay opción; se tiene que recargar en una estación de recarga en la ruta para alcanzar al cliente.

Las restricciones son varias, y depende mucho de las simplificaciones que se elige hacer. No obstante, se puede decidir unas restricciones que unen todas las variaciones del G-VRP.

- Los vehículos tienen una capacidad limitada de recorrer antes que tengan que recargar sus depósitos de energía/combustible. Esta restricción es, como mencionado antes, uno de los factores que más define este tipo de problema y sus características.
- Los tiempos limitados. Como mencionamos antes en la sección de las variables, hay restricciones reales de tiempo que, por ejemplo, los clientes solamente pueden recibir los vehículos en tiempos fijos.
- En unas versiones también, se usa la variables para mantener el nivel de carga de un vehículo. Esto ayuda limitar el viaje del vehículo, y hacer el problema más similar a un problema real.

Todo depende de cuáles factores son los más importantes para la empresa o la organización para la cual se construye el modelo, y cuáles simplificaciones se pueden hacer para más fácil encontrar una

2.5. Campos de aplicación

Es importante notar que este problema también es aplicable en otros campos. No es solamente aplicable a los EV - en muchos casos, empresas de transporte tienen tratos con empresas de gasolina, que lleva que, para ahorrar dinero, se busca planificar las rutas de los vehículos para que solamente usen estaciones de gasolina de la empresa de gasolina específica. [1] También se puede aplicar el problema a rutas de aviones o trenes, cuando se necesita planificar bien cuándo se va a rellenar el vehículo con combustibles. Otra aplicación puede ser para los camiones que colectan y dejan basura. Aquí, el objetivo puede ser evitar a retornar al almacén central para recargar el vehículo. [1] [4] Por ende, no es aplicable solamente en zonas más pequeñas como ciudades, pero también para vehículos viajando rutas con distancias más largas. Estas variaciones del problema llevan, obviamente, diferentes restricciones y factores reales, pero por lo general son muy parecidos, y los modelos serían muy parecidos.

Para sumarlo, generalmente un problema del G-VRP busca minimizar el costo de visitar clientes en una zona limitada, tomando en consideración primeramente la restricción que se puede viajar por una distancia limitada. Este costo puede haber en varias formas, aunque lo más común en las investigaciones hasta ahora, parece que haya sido la distancia recorrida y la cantidad de vehículos usados. No obstante, es importante acordar que se puede cuantificarlo en costos en términos de dinero también.[6] Esto lleva soluciones que quizá tengan el mayor impacto de le economía de la empresa para cual se implementa el modelo, y lo que tal vez les interesa a las empresas lo más.

3. Estado del Arte

Hasta ahora, varios métodos han sido probados para eficientemente resolver el G-VRP. Por lo general, los intentos han sido combinaciones de diferentes algoritmos; uno que explora, otra que explota el espacio de búsqueda. El primer paper sobre el GVRP, fue publicado en 2012 por S. Erdogan y E. Miller-Hooks, ya las técnicas que usaron se puede ver abajo.

3.1. DBCA/MCWS

Primero, S. Erdogan y E. Miller-Hooks intentaron resolver el G-VRP con una combinación del algoritmo Modified Clark and Wright Savings (MCWS) para explotar, y del Density-Based Clustering Algorithm (DBCA) para explorar. [3] Estos son, según Erdogan y Miller-Hooks (2012) diseñados para el problema VRP, de cual el G-VRP es una variante.

El MCWS consiste de, describiéndolo brevemente, primero crear rutas candidatas que empiezan y terminan en el almacén, con solamente un cliente en cada ruta. Se compara estos a las restricciones (primeramente a las restricciones de tiempo y de distancia) y se categoriza cada ruta como válida o no válida. Luego, para cada ruta candidata no válida, se calcula el costo de insertar una estación de recarga entre el almacén y el cliente. Para cada ruta, se inserta la ruta con el menor costo según ecuación (1), donde d es la distancia entre dos vértices dados y c es el costo. Si ahora las dos restricciones (de tiempo y distancia) están cumplidas, se pone la nueva ruta en la lista de las válidas. Si la restricción de distancia está cumplida con ninguna agrega de estación de recarga, se elige botar la ruta para siempre. Luego, se identifica todos los vértices adyacentes al almacén en un tour, para poder crear una nueva lista, un Savings Pair List (SPL) que incluye todos los pares posibles de los vértices mencionados anteriormente. Con la ayuda de esta lista, se calcula los ahorros según ecuación (2) que se puede obtener con cada combinación. Estos se ordena según orden descendente. Mientras todavía hay pares en la lista SPL, se elige el par más arriba en la lista, y se unifica las rutas a 1. Si la nueva ruta es válida, se agrega esta ruta a la lista de rutas válidas. Si no alcanza la restricción de distancia,

se intenta poner una estación de recarga en la ruta para una ruta válida con el costo menor posible, según ecuación (3). Luego, se elimina las estaciones de recarga redundantes y se revisa si se puede eliminar una estación. Si se ha agregado una ruta a las rutas válidas, se vuelve a revisar la lista SPL. Si no, se para. Abajo, se ve un ejemplo de como funciona este algoritmo. [3]

$$c(v_i, v_0) = d(v_i, v_f) + d(v_f, v_0) - d(v_i, v_0)^0 \tag{1}$$

$$s(v_i, vj) = d(v_0, v_i) + d(v_0, v_j) - d(v_i, v_j) \quad donde \quad ((v_i, v_j) \in V \cup F')^0 \tag{2}$$

$$c(v_i, v_j) = d(v_i, v_f) + d(v_f, v_j) - d(v_i, v_0) - d(v_0, v_j) \quad donde \quad ((v_i, v_j) \in V \cup F')^0 \tag{3}$$

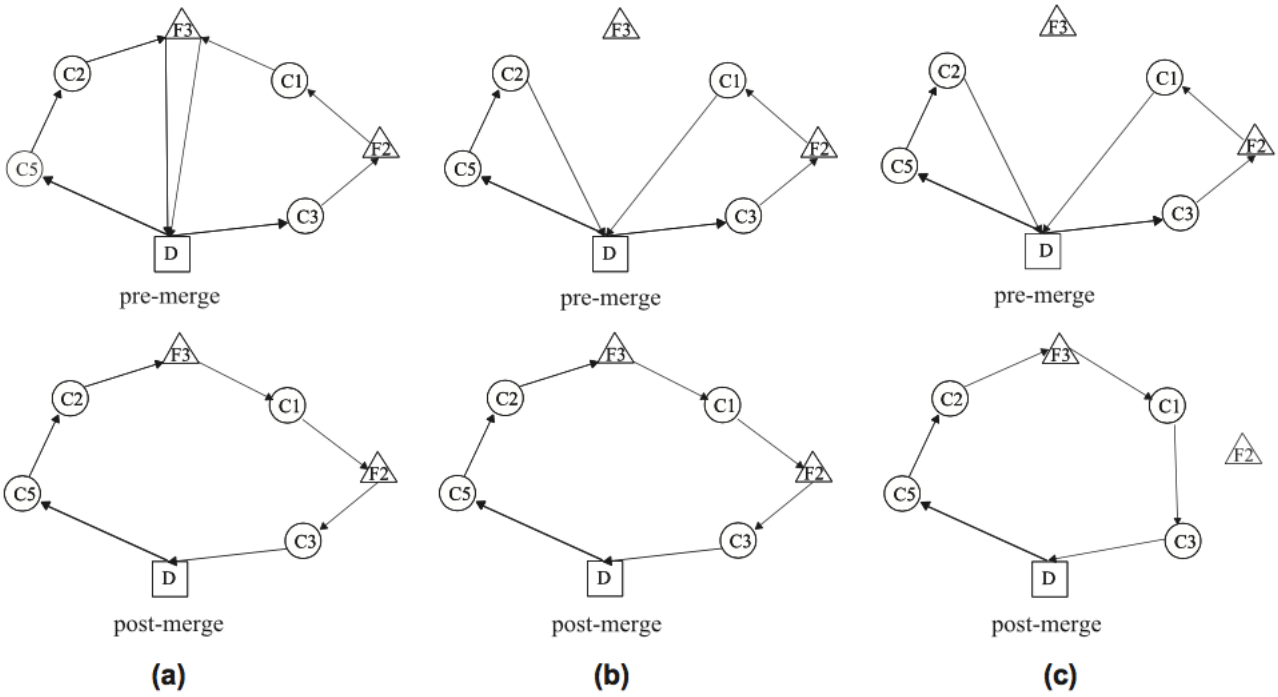

Fig. 1. *Ejemplo del MCWS. Arriba se ve el estado antes, seguido por soluciones posibles. D es el almacén, los C's son los clientes y los F's son las estaciones de recarga. [3]*

[0]Para explicaciones de los variables y conjuntos, revise el modelo en sección 4

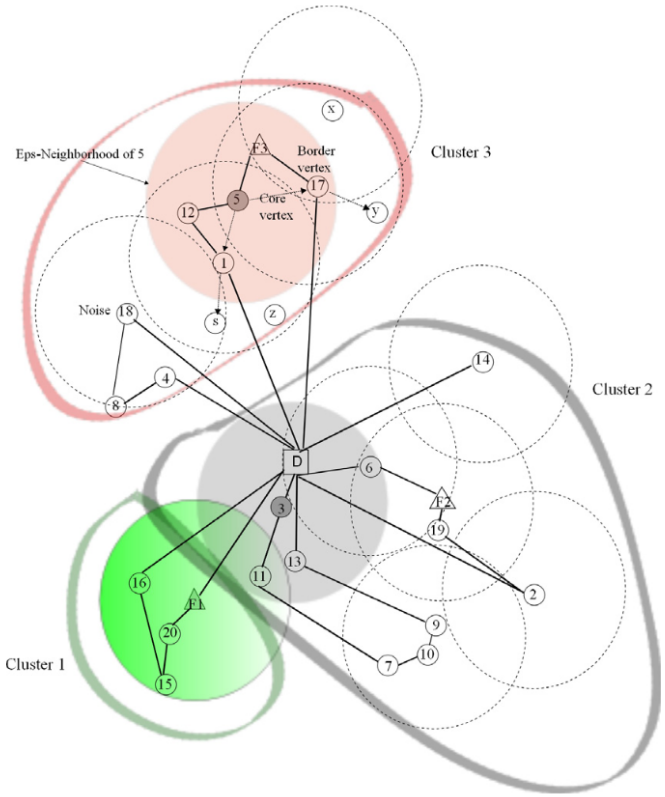

Fig. 2 *Ilustración del DBCA. Se divide las zonas en clusters, por cuales se busca.[3]*

El DBCA, trata de, como mencionado antes, explorar el problema. Este método descompone el problema en dos subproblemas de clustering y elección de ruta. La idea principal es que por cada vértice de un cluster, los alrededores en un radio ϵ dado, hay que haber un número mínimo de vértices, $minPts$. Es decir, se tiene un límite de densidad para moverse a una zona del problema y se puede moverse a un vértice v_i de un vértice v_j si las dos condiciones siguientes son cumplidos. [3]

1. $v_i \in N_\epsilon(v_j)$
2. $|N_\epsilon| \geq minPts$

Donde $N_\epsilon(v_j)$ es el conjunto de vértices en el vecindario en un radio ϵ del vértice v_j.

3.2. VNS/TS

En la investigación de M. Schneider, A. Stenger y D. Goeke, usaron otras técnicas. Ellos aplican Variable Neighborhood Search (VNS) para la exploración y Tabu Search (TS) para la explotación, combinado con un proceso de preparación del problema, en cual se elimina arcos (rutas malas), con cual ya se calcula que no satisfarán las restricciones de tiempo y distancia. [2]

Primeramente, el VNS se usa, como mencionado antes, para diversificación. Por lo general, si se tiene una solución S, el algoritmo de VNS genera al azar una solución cercana S' en el conjunto N_k. Luego, una búsqueda greedy localmente es aplicado a S' para determinar el mínimo local S". Si S" mejora la solución S, el algoritmo la solución y empieza otra vez con N_1,

reemplazando S con S". Si es peor, se sigue a N_{k+1}, otro vecindario y se repite el proceso. Esto se hace hasta que se encuentre una solución mejor. [2]

El algoritmo TS, por el otro lado, empieza de la solución S obtenido por el algoritmo VNS. En cada iteración, se crea los vecinos por las técnicas de 2-opt, Relocate, Exchange y una nueva operador que se llama stationInRe. Cuando han sido generados, se evalúa cuál vecino sería mejor moverse a. Aquí la mejor solución que no sea tabú es elegido.

- *2-opt*. Reemplaza dos vértices con dos nuevos. Un ejemplo de 2-opt, se puede ver abajo. [1]
- *Exchange*. Reemplaza (en este caso) 3 vértices con 3 nuevos en la manera que una sequencia de 3 vértices es reemplazada. El operador [1]
- *Relocate*. Traslada un cliente a una posición diferente en la ruta para ver si eso lleva un menor costo. [1]
- *stationInRe*. Inserta y elimina arcos entre una estación de recarga y cliente. Si no existe un arco entre una estación de recarga y un vértice, se lo inserta. Si hay, se lo elimina. [3]

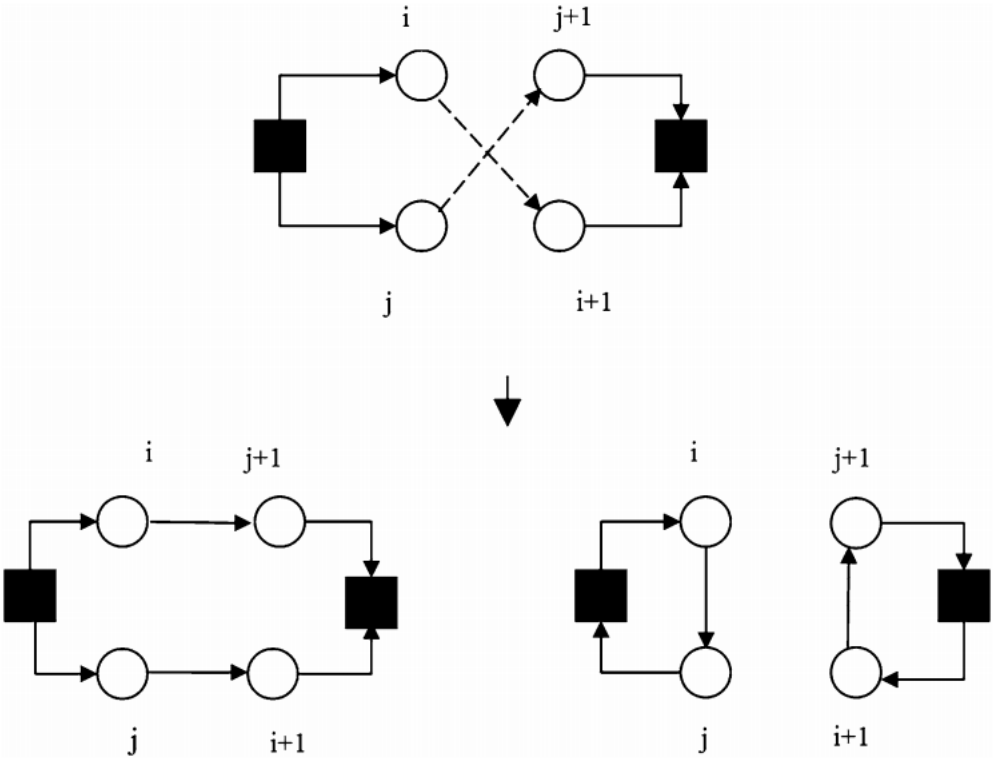

Fig. 3. *Ilustración cómo funciona el 2-opt. [4]*

3.3. AVNS/Búsqueda local con 2-opt, Exchange, Relocate y Replace

En otra obra, se usa, como en el anterior, se usa la técnica Adaptive Variables Neighborhood Search (AVNS), una variación del VNS. El AVNS también busca diversificar por buscar por vecindarios de tamaños diferentes. No obstante, no se genera la nueva solución S' al azar, sino por reglas del problema y de resultados de búsquedas anteriores por estas reglas.[1] Se usa este método ha resultado muy eficiente en otras variantes del VRP, como MDVRP y VRPPC. Por

varias razones, no se queda en óptimas locales con este algoritmo. Primeramente, la complejidad de un tipo del problema VRP posee una complejidad tan alta, que se necesita un algoritmo que diversifica y explora mucho en una manera eficiente. TS se queda, a menudo en óptimas locales, de donde no pueden escapar. Por el otro lado, el AVNS mueve en una manera tan sofisticado que previene que ocurra esto. [1]

En esta investigación, se usa también, como S. Erdogan and E. Miller-Hooks, el MCWS. No obstante, se usa solamente para iniciar soluciones iniciales, y no para explotar el espacio de búsqueda, como en la otra investigación. Para explotar, siguen los siguientes pasos. Primero, violaciones gasolina o cargo en una ruta se maneja por agregar visitas a estaciones intermedios. Si la distancia entre dos recargas es demasiado larga, se inserta la estación de recargar más barata permitida para el problema. Luego, se mejora la solución con, como en la investigación anterior, con los operadores de 2-opt, Exchange, Relocate y Replace. Todos esos fueron explicados en la sección anterior, sino Replace, que tiene el propósito de evaluar para cada visita a una estación si sería más barato visitar otra estación. [1]

3.4. Heurística Constructiva/48A y SA

La última investigación tenía unas diferencias grandes. En todas las investigaciones anteriores, se suponía que cuando se recarga el vehículo, se recarga al nivel máximo. No obstante, esta simplificación lleva que no se obtiene todas las posibles, ya que la recarga demora tiempo también. Por hacerlo posible no recargar a 100 %, se hace más soluciones posibles. [6] También se considera que se usa tecnologías diferentes para recargar los vehículos, algo que no se ha visto en los anteriores. [6] Por ende, fue llamado el GVRP-MTPR, (Green Vehicle Routing Problem with Multiple Technologies and Partial Recharges).

En esta investigación, se usó, para explorar, una heurística constructiva, basada en un método de generación greedy. Eso crea soluciones válidas por empezar con una solución vacía y luego extenderla iterativamente hasta que una solución es creada. [6] Luego, para explotar, se usó una combinación de varios algoritmos; entre ellos se encuentra 2-opt, Reacharge Relocation (RR) y Reinsertion, de cuales se ha visto todos anteriormente sino el último. El último, Reinsertion, se basa en eliminar clientes de la ruta, calcular los ahorros y luego ponerlo donde se hace lo más ahorros. Ya que se puede combinar estos algoritmos en maneras diferentes, con solamente $Reinsertion_b^u$ representando 6 algoritmos en sí, abrió la posibilidad de definir un algoritmo nombrado 48A, que implementa todas las combinaciones diferentes posibles, eligiendo la mejor solución. [6] Sin embargo, para no quedarse en una óptima local, se usa una forma de Simulated Annealing (SA) para escapar de óptimas locales.

3.5. Comparando las técnicas

La combinación de DBCA/MCWS que se vió en 3.1 se mostraron eficientes para resolver el problema, alejándose solamente un promedio de 8 % las soluciones mejores conocidas para las instancias para cuales se implementó estas técnicas. [2] Según los autores, estas técnicas podrían ser usados en el futuro para mayores problemas. [3] No obstante, se ve que el método de explotar de S. Erdogan y E. Miller-Hooks es más larga y requiere más memoria y tiempo que el método de M. Schneider, A. Stenger y D. Goeke, que usaron una heurística más eficiente. Aunque no proveyeron la cantidad de tiempo que demoraron las simulaciones, la heurística VNS/TS resultó mucho mejor, con una distancia promedio de la mejor solución de -0,09 %, que es muy bueno comparado con la combinación MCWS/DBA. No obstante, es importante considerar que no solamente son las técnicas que influyen en los resultados, pero también las diferencias de los modelos, que pueden influir.

En las simulaciones de la investigación en 3.2, se llegó a resultados muy prometidos con esta técnica de híbrida. En el artículo, están concluyendo que aunque sean resultados buenos, hay que simplificar menos en el futuro, incluyendo factores reales como velocidades de los vehículos y otros factores excluidos en su modelo. [2]

Los resultados de la técnica en 3.3 parecieron prometedores. La técnica logró sobrepasar los resultados de la técnica VNS/TS, igual en tiempo resolver el problema como resultado. [1] No obstante, no logró ser mejor que simulaciones de otra investigación, llamado el Hemmelmayr, Doerner, Hartl, and Rath (2013). [1] Ellos mencionan que esta manera de resolver era la mejor, superando todas las otras soluciones. Esta investigación tenía el propósito de colectar residuos, que hemos visto anteriormente que es una aplicación de este problema. Usan una técnica de VNS, como hemos visto antes, combinado con una técnica de programación dinámica para insertar estaciones intermediaros. [4]

Sumando esta parte, podemos concluir que las soluciones mejores anteriores han tenido variantes del VNS, Variable Neighborhood Search. Con estos, se explora el espacio de búsqueda en una manera eficiente.

Se mostró que el algoritmo 48A superó el DBCA que S. Erdogan y E. Miller-Hooks. Obtuvo resultados similares a la técnica VNS/TS, pero en un tiempo menor. Se concluyo que SA era mejor para instancias más grandes, mientras el algoritmo 48A era mejor para instancias de tamaño mediano. [6]

Cuadro 1: Tabla mostrando las diferencias entre los diferentes

Nr	**Autores**
1	S. Erdogan y E. Miller-Hooks
2	M. Schneider, A. Stenger, D. Goeke
3	M. Schneider, A. Stenger, J. Hof
4	A. Felipe, M. Teresa Ortuño, G. Righini, G. Tirado
	Técnica para explorar
1	DBCA
2	VNS
3	AVNS
4	Heurística Constructiva
	Técnica para explotar
1	MCWS
2	Tabu Search con iteraciones creadas con 2-opt, Exchange, Relocate
3	Búsqueda local 2-opt, Exchange, Relocate y Replace
4	48A y SA
	Eficiencia
1	Bien
2	Resultó mejor que (1)
3	Resultó mejor que (1) y (2)
4	Dependió del tamaño

Cuadro 2: Resultados diferentes de las técnicas en instancias que fueron corridos para este artículo

Instancia	f_{obj}	Dif. %	f_{obj}	Dif. %	f_{obj}	Dif. %	f_{obj}	Dif. %	f_{obj}	Tours
20c3sC1	1300,62	10,83	1173,57	0	1173,57	0	1178,97	0,46	1173,57	5
20c3sU2	1614,15	2,50	1574,77	0	1574,78	0	1574,78	0	1574,77	6
20c3sU3	1969,4	15,54	1704,48	0	1704,48	0	1704,48	0	1704,48	7
S1_2i6s	1614,15	2,28	1578,12	0	1578,12	0	1578,12	0	1578,12	6
S1_4i4s	1580,52	5,08	1460,09	0,97	N/A	N/A	1446,08	0	1446,08	6
S1_4i6s	1454,96	4,13	1397,27	0	1397,27	0	1434,14	2,64	1397,27	5

4. Modelo Matemático

4.1. Explicación de conjuntos, variables y parámetros

Cuadro 3: Definiciones de conjuntos, variables y parámetros para modelo 4.2

Conjuntos	**Descripción**
0	Instancia del almacén
I	Conjunto de clientes I = {1,...,N}
Φ	Vértices dummy. Importantes para poder visitar estaciones de recarga varias veces.
F'	Conjunto de estaciones de recarga y vértices dummy. $F' = F \cup \Phi$
F'_0	Conjunto de visitas de recarga, incluyendo la instancia 0
V'	Conjunto de vértices de clientes, incluyendo visitas a estaciones de recarga. $V' = I \cup F'$
Parámetro	**Descripción**
d_{ij}	Distancia entre vértice i y j
t_{ij}	Tiempo de viajar entre nodo i y j
p_i	Tiempo de servicio en estación o cliente i
r	Tasa de consumo de energía
Q	Capacidad de la batería
Variable	Descripción
τ_i	Variable manejando el tiempo de llegada al vértice i.
y_j	Variable que especifica la cantidad de energía que queda en llegada al vértice i
x_{ij}	Variable binaria que es 1 si se viaja entre arco i y j

4.2. El modelo

El siguiente modelo que se abajo es del artículo de S. Erdogan de 2012. [3]

$$min \sum_{i,j \in V'} d_{ij} x_{ij} \tag{1}$$

$$\sum_{j \in V', i \neq j} x_{ij} = 1 \quad \forall i \in I \tag{2}$$

$$\sum_{j \in V', i \neq j} x_{ij} \leq 1 \quad i \in F_0 \tag{3}$$

$$\sum_{j \in V', i \neq j} x_{ji} - \sum_{i \in V', i \neq j} x_{ij} = 0 \quad \forall j \in V' \tag{4}$$

$$\sum_{j \in V_{\setminus\{0\}}} x_{0j} \leq m \tag{5}$$

$$\sum_{j \in V_{\setminus\{0\}}} x_{j0} \leq m \tag{6}$$

$$\tau_j \geq \tau_i + (t_{ij} - p_j)x_{ij} - T_{max}(1 - x_{ij}), \quad i \in V', \quad \forall j \in V' \setminus \{0\} \quad and \quad i \neq j \tag{7}$$

$$0 \leq \tau_0 \leq T_{max} \tag{8}$$

$$t_{0j} \leq \tau_j \leq T_{max} - (t_{j0} + p_j) \tag{9}$$

$$y_j \leq y_i - r \cdot d_{ij}x_{ij}Q(1 - xij), \quad \forall j \in I \quad and \quad i \in V', \quad i \neq j \tag{10}$$

$$y_j = Q \quad \forall j \in F_0' \tag{11}$$

$$y_j \geq min\{r \cdot d_{j0}, r \cdot (d_{jl} + d_{l0})\}, \quad \forall j \in I, \quad \forall l \in F \tag{12}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j \tag{13}$$

4.3. Explicaciónes de las restricciones el función objetivo

1. El función objetivo busca, como siempre, minimizar la distancia.
2. Restricción que fuerza que cada cliente sea visitado una vez y solamente una vez. Es decir, para cada cliente j, un arco que viene al cliente debe ser usado. Es decir, cada estación de recarga puede estar conectado a máximo un cliente. Esto que solamente se puede visitar una estación de recarga como máximo una vez con este modelo.
3. Restricción que asegura que cada estación de recarga tendrá máximo un vértice como sucesor.
4. Restricción que asegura el flujo de los vértices: que siempre se salga de un arco, y se entre por otro.
5. Esta restricción asegura que no más que m salidas de la estación de recarga puede ocurrir.
6. Esta restricción asegura que no más que m entradas a la estación de recarga puede ocurrir.
7. Esta restricción fuerza que cada vértice sea visitado en el tiempo permitido.
8. Esta restricción especifica un tiempo de salida del depot, que tiene que ocurrir dentro de un tiempo permitido.
9. Garantiza, como los dos anteriores, que ningún vehículo retorna al depot después del tiempo especificado, T_{max}.
10. Esta restricción controla, junto con (11), el nivel de la batería, y calcula el nivel de la batería en cada parada.
11. Asegura que el nivel de combustión será Q cuando se llega a una estación de recarga.
12. Esta restricción garantiza que hay suficiente de combustión para volver al depot inmediatamente, o por una estación de recarga.
13. Esta restricción lleva que las variables x_{ij} sean binarias.

4.4. Espacio de búsqueda

En este modelo, se tiene el siguiente espacio de búsqueda.

$$EB = 2^{|V'|^2} \cdot T_{maximo}^{|V' \setminus \{0\}|} \cdot C^{|V|} \cdot (C+1)^{|F|+1} \cdot (Q+1)^{|V'|}$$

Aquí T_{maximo} es el tiempo máximo posible (no especificado en el problema) que puede tener el variable τ. Primeramente, tenemos $2^{|V'|^2}$ variables binarios que indican si se usa el arco entre vértice i y j o no. Luego, el variable τ puede tener un valor entre 1 y T_{maximo} para cada todos los vértices, sino el primer vértice, $\{0\}$, ya que nunca se llega a esa estación. Se puede suponer que no se puede llegar inmediatamente a otro vértice, y por eso se puede excluir el 0 del dominio del variable τ. El variable u_i puede tener valores entre 1 y C, así C valores diferentes, para todos los vértices de los clientes. Se puede suponer que no tiene el valor 0, porque esto resultaría que no se tiene ningún paquete a entregar al cliente, que resultaría que se visitaría el cliente sin su paquete y el modelo sería inadecuado. Por el otro lado, se puede tener valores entre 0 y C para los vértices de las estaciones de recarga y el almacén a la llegada. Esto porque hay una posibilidad que se tiene que visitar una estación de recarga a la vuelta del último cliente, cuando no se tiene ningún paquete más de entregar a un cliente. Aquí, el óptimo es, de hecho, tener 0 paquetes cuando se llega al almacén. Finalmente, suponiendo que el variable y_j no es discreto, se puede tener un nivel de energía en una estación entre los valores 0 y Q, que lleva el último factor en el espacio de búsqueda.

4.5. Comparando los modelos diferentes

Es importante notar las diferencias de los modelos. Aunque los modelos y las técnicas correspondientes fueron aplicados a las mismas instancias muchas veces, debido a las estructuras fundamentalmente similares, tienen diferentes consideraciones, que claramente afectan a los resultados.

Primero, se ve diferencias en cómo se considera el tiempo y como se recarga el vehículo con energía. La primera técnica que se vió resuelve un modelo (4.1) más simple, en cual no se considera el nivel de carga del vehículo, y se considera el tiempo de recarga constante, dependiente de la estación, igual que el segundo modelo. En la tercera investigación, que usó el algoritmo visto en 3.3, se consideró el tiempo como una función lineal de cuánto de energía que se tiene que rellenar el vehículo. En la cuarta, se consideró el tiempo de recarga como una función lineal dependiente del nivel de energía del vehículo. No obstante, se abrió la posibilidad de no recargar el vehículo completamente. Esto creó la posibilidad de soluciones para instancias que no habrían sido posibles si los vehículos solamente pudiesen recargar hasta que estén llenos.

También, se ve diferencias en como se minimiza la cantidad de vehículos. En la primera investigación, se tenía solamente un límite de la cantidad de vehículos que se puede usar, mientras en la segunda, se minimizó la cantidad de vehículos antes de comenzar minimizar la distancia recorrida. En el tercer modelo, se implementó esto en el función objetivo como un costo fijo para cada vehículo que se agregó, combinado con una restricción que hace que no se usa más que m vehículos.

También se ve diferencias en como se modela el tiempo. En el primer, tercer y cuarto modelo, se considera solamente un T_{max}, y que cada vehículo entra y sale dentro de este tiempo. En el segundo es similar, pero se introduce horarios cuando están abiertos las estaciones de servicio.

En todos los modelos sino el primer, se considera la carga del vehículo, que los hacen más similares a la realidad, limitando un vehículo a su capacidad máxima de cargo.

Es discutible cuales simplificaciones y diferencias son mejores. Mientras unas simplificaciones del problema simplifica el problema principal y reduce el espacio de búsqueda, otras pueden complicar y excluir posibles soluciones. Por ejemplo, el cuarto modelo, donde era posible recargar solo a un cierto nivel y no completamente en una estación encontró soluciones en instancias en cuales no habría sido posible sin ese factor, así para los otros modelos. [6] También tenían la posibilidad de usar diferentes tecnologías, algo que los otros modelos no tenían. Resultó que ninguna tecnología era mucho mejor que todos los otros, que muestra que la elección de tecnología para recargar depende mucho de la estructura del problema. [6]

5. Representación

Para resolver este problema, se eligió usar varias estructuras combinadas. Para resumirlo, el enfoque ha sido utilizar vectores de structs para simular la realidad, pero también matrices para representar las variables y los parámetros del modelo. Lo que se ha utilizado ha sido lo siguiente.

- Structs. Estos se han mostrado útiles para lograr la información requerida en ciertas situaciones, y para simular varios objetos y circunstancias reales. Los siguientes structs se ha utilizado.

  1. Struct **location**. Este struct almacena toda la información sobre un lugar definido en el problema. Puede ser el depot, un cliente o una estación de recarga.
  2. Struct **stop**. Estas paradas, se puede considerar como la variable de decisión; ¿cuál parada es la próxima que se visitará? Esto se mostró ser una manera eficiente de representar el modelo, y facilmente implementar el algoritmo. Estas paradas tienen unos parámetros diferentes.

     a) El lugar visitado en la parada. Indica cual lugar que se visita en el parado especificado.

     b) Cuanto de energía que queda en el vehículo.

     c) El tiempo utilizado durante la ruta.

     También tiene dos vectores, conteniendo paradas nuevas; un vector para las paradas a los clientes, un vector para las paradas a las estaciones de recarga. Estos juntos, se considera el dominio de la parada.
  3. Struct **solution**. Esto se tiene para poder lograr una solución, para luego poder comparar con otras soluciones. Primeramente, se planificó lograr todas las soluciones encontrados; algo que luego se mostró imposible ya que habían demasiadas. En vez, solamente se decidió lograr siempre la mejor solución encontrada.

- Vectores del librería estándar de C++. Estos han sido útiles, primeramente para lograr y manejar los structs en las maneras que se quiere. La facilidad de almacenar, cambiar tamaño y acceder los objetos logrados en el vector se mostró útil. Se ha usado los vectores resolviendo este problema en las siguientes maneras.

  - Para almacenar los lugares. Se ha tenido 3 vectores; 1 para almacenar todos (no necesario realmente, solamente en el comienzo), uno para almacenar los clientes y uno para las estaciones de recarga (incluyendo el depot). Sería posible tener todos los lugares en el mismo vector, pero debido a las priorizaciones y cómo se quiere iterar por las soluciones posibles, esto era lo más fácil hacer.
  - Para almacenar las paradas (struct stop). Se ha tenido 2 tipos de vectores diferentes para las paradas en la representación.

1. Los vectores contiendo las paradas nuevas posibles, es decir, los dominios de los próximos pasos (vea arriba).
2. El vector conteniendo la solución posible (con el nombre road). Este vector logra las iteraciones, es decir, las paradas que se ha hecho. Se llama road por la razón simple que se puede considerarlo como la ruta en total que recorren todos los vehículos, y que es toda la solución.

- Se ha usado, también, matrices de punteros. Estos logran las variables y los parámetros necesarios para resolver el problema. Estos contribuyen a que se puede acceder rápidamente los valores necesarios para ciertos cálculos, obteniéndolos por índice. Estos resultaron innecesarios por fin, y sería posible implementar el algoritmo sin estos. No obstante, se quedaron después del primer intento de inventar el algoritmo.

El primer intento fue más similar a la representación matemática. No obstante, se resultó más fácil implementar una solución con orientación a objetos, debido a que eso era menos abstracto. Esto resultó en una representación muy similar a la realidad; es decir, con structs que representan lugares de visitar y posibles paradas en estos, indicando niveles de combustibles, tiempo etc. Esto resultó ser una buena representación, que probablemente es utilizable para aplicar otros algoritmos. No obstante, es importante tener en cuenta que esto posiblemente puede causar que el programa corre más lento.

6. Descripción del algoritmo

Para este problema, se decidió implementar una técnica completa, Graph Back Jumping (GBJ) y Forward Checking. Estos se usa cuando se quiere encontrar todas las soluciones para un problema, o al menos todas en un área lo mayor posible del espacio de búsqueda, cuando se tiene un tiempo limitado.

Forwardchecking se basa en backtracking. Es decir, se va por todas las combinaciones posibles de los variables por las cuales se itera. Se puede hacer las iteraciones ambos iterativamente como recursivamente, aunque conviene hacerlo recursivamente a un extenso. No obstante, forwardchecking tiene una ventaja antes que Backtracking. Antes de instanciar el próximo nivel del árbol, se filtra el dominio de los próximos variables.

Siendo difícil encontrar maneras de filtrar los dominios de los variables de este problema, se encontró una manera de hacerlo. Cuando un variable entre un nodo de un cliente y cualquier otro nodo (incluyendo nodos de clientes) se conecta, se filtra inmediatamente la posibilidad de utilizar este nodo del cliente otra vez, ya que cada nodo solo puede ser utilizado una vez. Esto lleva una rapidez más alta cuando se instancia el dominio para el próximo variable. Cuando se instancia el dominio de la variable, se controla también que las posibilidades en el dominio cumplen las restricciones de tiempo y de energía, que también es una filtración.

El GBJ se basa en que, cuando se crea el árbol de soluciones por iterar con Forwardchecking, Backtracking o un algoritmo similar, volver al nodo en árbol más cercana conectada al nodo en cual se tenía un conflicto. Este método no se mostró eficiente en este problema, ya que todos los nodos, es decir, los lugares definidos en el problema, son conectados, ya que se puede ir de cualquier lugar a cualquier otro. Por ende, el algoritmo no se mostró eficiente resolver este problema. Esto es debido a la representación también. Ya que solamente se va entre un lugar posiblemente conectado a otro, ya que si son conectados, esto significa que se puede visitar el uno después del otro, esto lleva que se para en el anterior, que lleva que el GBJ se convierte a Backtracking.

Resumido, se sigue los siguientes pasos en el algoritmo.

1. Logre todos los lugares en dos vectores; uno para los clientes, otro para el depot y las estaciones de recarga. Cree otro vector de paradas, y genera la primera parada; la primera salida del depot.

2. Genera las posibles nuevas paradas; es decir, las nuevas visitas para la última parada en la vector de las paradas. Esto es, cree el dominio de posibles lugares de visitar. Creando este dominio, controle que las nuevas soluciones cumplan las restricciones y las reglas del forwardchecking. Si una nueva parada cumple todos los criterios para ser una nueva posible parada, se agrega a un vector. Hay dos vectores; uno para las paradas a los clientes; uno para las paradas de las estaciones de recarga.

3. Si se usa el ordenamiento normal; es decir, la de forwardchecking describido en el siguiente punto, genere el dominio del dominio de las paradas. Esto, para luego poder ordenar las paradas en la manera que requiere el FC. Si se usa el otro sorting, no se hace este paso.

4. Haga un sort de las listas de las paradas en la próxima iteración según las siguientes restricciones:

   a) Cuántas paradas se puede visitar. Es decir, el dominio de posibles visitas. en el próximo paso.

   b) Cuánto de tiempo que ha transcurrido. Lo más tiempo que ha pasado, lo peor.

   c) Cuánto de gas que queda en el tanque. Para tener rutas tan largas posibles, se prefiere tener tanto de gas posible.

   Es importante notar que para muchas de las pruebas, se hizo otro sorting, ya que se pensó que iba a ser mejor. Aunque no siendo el principio de forwardchecking, se mostró más eficiente, que se puede ver en los resultados. En este sorting, no se tenía el primer factor, es decir, cuántas paradas se podía visitar en la próxima iteración, sino solamente las últimas dos factores.

5. Agrega una parada al vector que contiene la posible solución. Esto es lo que se considera una iteración; agregar una nueva parada en la ruta. Controle si se ha encontrado una solución. Si sí se ha encontrado una solución, y es mejor que el anterior, entrega la solución. En todo caso, siga iterando por recursivamente llamar al función que corresponde a una iteración, volviendo a paso 2.

Resultó lo más fácil implementar el código recursivamente. Primero, se preocupó por el uso de memoria, pensando que se utilizaría mucha memoría. Esto no es el caso, no obstante, ya que cada llamada a una función por recursión corresponde a una iteración con una parada en la secuencia. Esto lleva que la cantidad de funciones llamadas al mismo tiempo nunca será más que la larga de la ruta, que se intenta minimizar, y siempre tiene, en todas las instancias, un número pequeño, considerando la cantidad de lugares.

En la próxima página se ve un pseudo-código del algoritmo.

Algorithm 1 La función iteration

```
1: procedure ITERATION(NEWSTOP)
2:     initialize allClients[]
3:     initialize allGasStation[];
4:     initialize currentRoute[];
5:     add NewStop to currentRoute
6:     if currentRoute is a solution and currentRoute is better than previously best solution
   then
7:         bestSolution ← currentRoute
8:         print bestSolution
9:     else
10:        for client in allClients do
11:            create stopAtClient(client)
12:            if stopAtClient is valid then
13:                add to clientDomain of last element in currentRoute
14:                generateDomains for client
15:        for gasStation in allGasStation do
16:            create stopAtStation(gasStation)
17:            if stopAtStation is valid then
18:                add to gasStationDomain of last element in currentRoute
19:                generateDomains for gasStation
20:        sort(clientDomain)
21:        sort(gasStationDomain)
22:        for client in clientDomain do
23:            iteration(client)
24:        for gasStation in gasStationDomain do
25:            Iteration(gasStation)
26:        iteration(depot)
```

7. Experimentos

7.1. Cómo se definió parámetros

Para la técnica completa de FC+GBJ, no hay (necesariamente) parámetros. Por ende, no había nada de definir. No obstante, para la implementación se puso varias veces el parámetro m, la cantidad de tours, a 10, para limitar la cantidad de rutas y solo buscar parte del espacio de búsqueda, sabiendo que las mejores soluciones tendría que tener pocas rutas. No obstante, esto no fue tan necesario, debido a la estructura que priorizaba agregar clientes, luego estaciones de recarga, y últimamente el depot.

Algo que no se considera como un parámetro, pero que también importa, es, como se ha mencionado antes, como se eligió ordenar los dominios para el forwardchecking. Para la mayoría de los tests, se hizo un corrido con el primer ordenamiento, es decir, el estándar para FC, y un corrido con el otro ordenamiento.

7.2. Cuales instancias se eligió correr

Se eligió primeramente intentar con las menores instancias, sabiendo que sería difícil obtener soluciones para los mayores, teniendo un algoritmo completo.

Primeramente, se usó el otro sorting. Con este sorting, se intentó encontrar soluciones con rutas tan largas posibles; es decir, se ordenó las paradas, así los dominios, en orden de más combustible a menor combustible para encontrar una solución adecuada, y segundamente cuánto tiempo que quedaba.

Se intentó con una de las instancias más grandes. Esta, no obstante, no convergió. Se dejó corriendo por la noche, pero todavía no había entregado una solución en la mañana. Luego de eso, se decidió no gastar capacidad computacional en las otras dos instancias grandes.

7.3. Cómo se trataron las instancias

Se hizo varios experimentos durante un periodo de unas semanas. Esto se hizo, por conectarse por ssh a los computadores de las salas de LabComp en la Universidad Técnica Federico Santa María, y luego correr las instancias en esos computadores. En los tests, se podía observar el tiempo, cantidad de iteraciones y si se encontraba una solución mejor que el anterior. Es importante mencionar que en el comienzo, no se tenía en cuenta lograr cuánto tiempo se demoraron.

8. Resultados

En los experimentos, se logró lo siguiente.

Cuadro 4: Resultados

Instancia	**# iter.**	f_{obj}	**#tours**	**# iter. solución**	**Tiempo**	**Orden**
20c3sC1	26 000 000	2499	8	3 873 989	N/A	A
20c3sC1	153 000 000	1585	6	18 462 943	N/A	B
20c3sU2	27 000 000	2714	8	530 621	N/A	A
20c3sU2	82 000 000	1625	5	32 511	N/A	B
20c3sU3	8 000 000	2545	8	7053427	3:45:32	A
20c3sU3	153 000 000	1558	5	78 318 166	5:50:14	B
S1_2i6s	56 000 000	2604	8	772 256	0:45:41	A
S1_2i6s	52 000 000	1707	6	281 464	0:01:14	B
S1_4i4s	77 000 000	1596	9	588 669	0:26:37	A
S1_4i6s	46 000 000	2944	9	355 604	0:25:51	A

Cuadro 5: Los resultados comparados con BKS (best known solution)

Instancia	Orden	Diferencia de BKS
20c3sC1	A	112,94 %
20c3sC1	B	35,06 %
20c3sU2	A	72,34 %
20c3sU2	B	3,19 %
20c3sU3	B	-8,59 %
S1_2i6s	A	65,01 %
S1_2i6s	B	8,17 %
S1_4i4s	A	10,37 %
S1_4i6s	A	110,70 %

8.1. Explicaciones

En los resultados que se ve arriba, hay varios factores a considerar. Primero, columna 2, # iter., indica la cantidad de iteraciones que hizo la instancia, indicando por cuántas iteraciones se corrió el programa. La diferencia de las iteraciones no es conocido, pero es probable que los computadores se apaguen por falta de capacidad después de un tiempo. La tercera columna indica el valor del función objetivo del problema; es decir, la distancia recorrida por los vehículos. Columna 4 indica cuántas veces se va y vuelve al depot, es decir, cuántas tours se hace en la mejor solución encontrada. Columna 5 indica en cual iteración se encontró la mejor solución. Columna 7 indica, si se midió, el tiempo demorado para encontrar la solución. Finalmente, columna 8 indica cuál de los dos ordenamientos se usó en el problema.

8.2. Anotaciones

Llegando al ordenamiento, es importante saber que A es el ordenamiento normal de FC, mientras B es el sorting que se usó para maximizar la larga de las rutas. Se ve evidentemente que usando el sorting B, se obtiene mejores resultados; de hecho casi comparables a las técnicas mencionadas en el Estado del Arte. No obstante, los resultados son un poco difíciles de interpretar, debido a la gran variación en la diferencia entre la mejor solución conocida y el resultado

de los experimentos. Una teoría de la gran variación puede ser las circunstancias en cuales se ha corrido las instancias. Mientras se ha corrido los programas, no se ha tomado en cuenta que puede ser otras personas usando el mismo computador al mismo tiempo; es decir, puede haber habido otros procesos corriendo en el computador. Esto es soportado por la gran variación de cantidad de iteraciones total que corre cada instancia; mientras unas corrieron por más de 100 000 000 iteraciones, otras fueron parados del computador después de solamente 26 000 000. Esto los hacen difíciles comparar. Sin embargo, es muy importante mencionar que la diferencia entre los dos ordenamientos fue mostrado en otra manera. Fue mostrado por el hecho que las primeras soluciones encontradas con el primer ordenamiento, fueron significativamente peores que las primes encontradas por ordenamiento B. Esto indica que se comienza la búsqueda en una parte del espacio de búsqueda mejor con B que con el ordenamiento A.

Otra razón por qué hay tantas diferencias es una diferencia de la implementación. Para hacer ordenamiento A, se necesita instanciar los dominios de las paradas en los dominios de la parada actual. Esto requiere un tiempo exponential más para el computador de hacer, y lleva que el programa corre mucho más lento para FC+GBJ normal, comparado con usando ordenamiento B. Por esto, también es difícil comparar, ya que FC+GBJ no tiene tanto de tiempo de explorar el EB antes que se apague el computador. Queda claro, no obstante, que una combinación de FC y GBJ no es cerca de ser tan eficiente como las heuristicas en el Estado del Arte.

Como mencionado anteriormente, se intentó con una instancia más larga con ordenamiento B. Esta instancia nunca convergió, y no se decidió no gastar más tiempo computacional en las instancias más largas.

9. Conclusiones

Para concluir todo, es difícil decidir exactamente cuál es la mejor técnica. Todos los problemas son variantes del mismo problema con diferentes simplificaciones. Por ende, hay tantos factores, que es difícil decidir exactamente cuál factor mejoró las soluciones frente a otros resultados. Aunque signifique un incremento del espacio de búsqueda al introducir nuevos factores de la realidad, que en sí prolonga el tiempo de resolver el problema, también abre la posibilidad de nuevas, mejores soluciones. Algo para el futuro sería investigar más cerca cuales simplificaciones simplifica el problema lo más posible, excluyendo la cantidad mínima posible de soluciones para el problema real. También sería interesante ver de qué depende los mejores resultados de las investigaciones que vienen luego; si es por los modelos, o más por las técnicas usadas.

Las técnicas de los diferentes problemas eran bastante similares. En particular en el sentido que todas tienen una parte para explorar, y una para explotar, que muestra que grave es dividir las técnicas en estas dos partes para obtener el mejor resultado. Todas las técnicas son mejores del uno o del otro, y se necesita las dos partes para encontrar la óptima con una heurística.

Se puede decir que probablemente el ADNS/Búsqueda local con 2-opt, Exchange, Relocate y Replace fue mejor que VNS/TS debido a sus adaptaciones específicas al problema. Se ve que el enfoque principal para explorar el problema es un variante del VNS, y una combinación de los operadores 2-opt, Exchange, Relocate y Replace para explotar. Debido al tamaño de este problema, una buena técnica de exploración es grave tener, y tiene que ser bien adaptada al problema. Para explotación, vimos que las operaciones 2-opt, Exchange, Relocate y Replace se mostraron eficientes, y probablemente sean operadores muy utilizables para este problema.

Algo que se puede concluir sobre las técnicas de los autores de las investigaciones, es que es grave combinar diferentes métodos para obtener el mejor resultado posible. Esto porque todos

los métodos tienen sus ventajas y sus fallas, pero combinadas correctamente pueden resultar en encontrar el óptimo en una manera eficiente. También se tiene que variar entre métodos de explotación y explorar con una ponderación correcta, que en este problema es más ponderado a exploración. Esto, porque el problema es tan grande, con muchas optimas locales donde se puede quedar con un método inadecuado.

Es fácil, no obstante, concluir que GBJ y FC no resultaron bien para resolver este problema. GBJ no fue posible implementar, debido a que todos las variables que se itera sobre son conectadas, y FC con su ordenamiento normal no iteró eficientemente sobre el EB. Los mejores resultados dependió mucho de la manera que se ordenaba los variables, que indica que una heurística constructiva hubiese servido mejor que una técnica completa, para encontrar soluciones rápidamente. En la mayoría de los casos, no se encontró las mejores soluciones antes que se apagó el computador. Es decir, el espacio de búsqueda es demasiado grande para encontrar todas las soluciones, y cuando se ha encontrado la mejor solución en un área, es demasiado lejos a irse a otra área con una solución mejor. Esto indica el hecho que comenzar varias veces en diferentes áreas podría facilitar encontrar la mejor solución. Esto en sí indica que se debería usar una heurística para resolver el problema para obtener soluciones más rápido, y se puede ver que las técnicas completas no son especialmente eficientes para problemas de este nivel de complejidad.

Una observación interesante, es que el ordenamiento normal de forwardchecking, es decir, ordenar las variables de decisión según dominio, no encontró soluciones mejores que el otro ordenamiento. Aquí se refiere al dominio como cuánto de combustible y tiempo que queda en un tour, ya que se ignoró la restricción de la cantidad de rutas se podía tomar. A lo contrario; el otro ordenamiento encontró una solución mejor que los anteriores encontrados del CPLEX, S. Erdogan y E. Miller-Hooks, y tenía resultados similares en los otros. Con esto en cuenta, se puede decir que una heurística interesante para encontrar las soluciones mejores posibles, sería maximizar la utilización del tiempo en cada recorrido por intentar siempre tener tanto de estaciones posibles en un tour. Es decir, una maximización de utilización de la ruta en una minimización de la cantidad de tours. Por ende, se puede concluir que una heurística, utilizando ambos exploración y explotación en su algoritmo, probablemente sería más eficiente para encontrar una solución adecuada para este problema. La combinación FC-GBJ solamente se queda en un área del espacio de búsqueda por demasiado tiempo, y no converge antes que se apague un computador normal. Como se vio, el ordenamiento normal se resultó más ineficiente que el otro, que buscó maximizar la utilización de tiempo y combustible en un tour; algo que indica que FC no es una buena técnica para resolver este problema.

Todavía queda mucho de investigar en este área. Hay pocas investigaciones sobre este problema, y lo que queda investigar sería cómo se podría adaptar los métodos más al problema, y cuales técnicas se debería usar, dependiente de cuales factores reales son los más importantes. Esto porque la estructura del problema cambia dependiente de los detalles, aunque todas las variantes todavía tengan más o menos la misma estructura fundamental. No obstante, lo que queda claro es que un algoritmo de FC y GBJ no sirve para este problema, debido a su gran espacio de búsqueda y estructura especial (que todos los nodos en el grafo son conectadas).

10. Bibliografía

Referencias

[1] Michael Schneider Andreas Stenger Julian Hof. An adaptive vns algorithm for vehicle routing problems with intermediate stops. *OR Spectrum*, 37(353), 2015.

[2] Dominik Goeke Michael Schneider, Andreas Stenger. The electric vehicle-routing problem with time windows and recharging stations. *Articles in Advance*, Published online 6/3 - 2014, 2014.

[3] S. Erdogan E. Miller-Hooks. A green vehicle routing problem. *Transportation Research*, Part E:100–114, 2012.

[4] Hemmelmayr V. Doerner K. Hartl R.F. A heuristic solution method for node routing based solid waste collection problems. *J Heuristics*, 19(129), 2013.

[5] Travelling Sales Man Problem description. Accessed: 2017-05-09.

[6] Ángel Felipe M. Teresa Ortuño Giovanni Righini Gregorio Tirad. A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges. *Transportation Research*, Part E:111–128, 2014.