Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Algoritmo GRASP para Problemas de Ruteo de Vehículos con Clientes Agrupados

Tesis presentada para optar al título de Licenciado en Ciencias de la Computación

Santiago Iriarte

Directora: Irene Loiseau

Buenos Aires, 2018

Índice general

1. INTRODUCCIÓN

Dentro del conjunto de problemas de optimización combinatoria, uno de los más populares es el problema del viajante de comercio, o TSP [1] por sus siglas en inglés. Este problema fue definido por Hamilton y Kirkman a comienzos del 1800 y responde la siguiente pregunta: dada una lista de ciudades y la distancia entre cada par de ellas, ¿Cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y regresa a la ciudad de origen?
Resolver este problema tiene implicaciones concretas en la vida real, debido a ser el mismo un problema de logística que se puede encontrar en distintas industrias. Por otro lado, TSP pertenece al conjunto de problemas NP-Hard por lo que su estudio es de gran importancia en el área de la teoría de la complejidad computacional.

El problema del viajante de comercio fue y es ampliamente estudiado, y del mismo se desprenden una gran cantidad de problemas que son una generalización del mismo, como el problema del viajante de comercio generalizado [2] (GTSP por sus siglas en inglés) o el problema del viajante de comercio generalizado por familias [3] (FTSP por sus siglas en inglés).

Continuando en la misma línea de problemas que generalizan a TSP, agregando a cada ciudad una demanda a satisfacer por una flota de vehículos con cierta capacidad, estamos ante uno de los problemas más populares, estudiados y aplicados en la industria, el problema de ruteo de vehículos, o VRP [4] por sus siglas en inglés.
Existen muchas variantes del VRP, ya que estos problemas tienen implicaciones en la reducción de costos de logística en variadas industrias.
Dentro de estas variantes algunas de las más conocidas son:

- *Problema de Ruteo del Vehículo con Ventanas de Tiempo*
  VRPTW, por sus siglas en inglés, donde las ubicaciones de entrega tienen ventanas de tiempo, dentro del cual las entregas (o visitas) tiene que ser realizadas.

- *Problema de Ruteo del Vehículo con Capacidad*
  CVRP, por sus siglas en inglés dónde las restricciones son sobre la capacidad de los vehículos, quienes deben entregar o satisfacer cierta demanda por parte de los clientes.

- *Ruteo de Vehículo Abierto*
  OVRP, por sus siglas en inglés, donde las rutas de los vehículos no necesitan terminar en el depósito.

En este trabajo se abordarán dos problemas de esta familia. Uno de ellos es el Problema de Ruteo de Vehículo Agrupados con Restricciones Fuertes sobre los Clusters, Clustered Vehicle Routing Problem with Strong Clusters Constraints en inglés (CluVRP SCC por sus siglas) y el otro, el Problema de Ruteo de Vehículo Agrupados con Restricciones Débiles Sobre los Clusters, Clustered Vehicle Routing Problem with Weak Clusters Constraints en inglés (CluVRP WCC por sus siglas).

CluVRP SCC es una generalización del problema CVRP [5], dónde los clientes están agrupados en clusters, y todos los clientes de un cluster deben ser visitados de forma consecutiva por el mismo vehículo.
CluVRP WCC también es una generalización del problema CVRP, dónde los clientes están agrupados en clusters, y todos los clientes de un cluster deben ser visitados por el mismo vehículo, pero a diferencia de CluVRP SCC, los clientes pueden no ser visitados en forma consecutiva dentro de un mismo cluster, si no que, un vehículo puede salir y reingresar a un cluster múltiples veces visitando a los clientes durante su recorrido.

Dado que todos los problemas mencionados anteriormente pertenecen a la clase NP-Hard, no se conocen algoritmos eficientes que encuentren soluciones óptimas.
Por ese motivo no suele ser común que en la práctica estos problemas se resuelvan de forma exacta, sino que se buscan soluciones de calidad, quizas no óptimas, a un tiempo considerablemente menor. Este objetivo generalmente se logra utilizando técnicas metaheurísticas [6].

En este trabajo implementaremos una solución para los problemas CluVRP SCC y CluVRP WCC aplicando la técnica metaheurística GRASP [7], y estudiaremos su comportamiento, comparando sus resultados con los de otros trabajos realizadas sobre estos dos problemas.

2. ESTADO DEL ARTE

El problema CluVRP fue presentado por primera vez en el trabajo de Sevaux y Sörensen (2008)[8] donde se abordó un problema del mundo real que involucra entregas de paquetes por parte de empresas de mensajería que usan contenedores con productos almacenados.
Los ítems que son entregados a los clientes que pertenecen a una área específica, se guardan en un contenedor. Todos los ítems deben ser entregados, por lo que todos los clientes de una región deben ser visitados antes de que el contenido de otro contenedor sea distribuido.
En la mayoría de los casos se espera que un mismo vehículo visite a todos los clientes de una misma área antes de abandonar la misma, este es el caso del problema CluVRP SCC. En caso que un vehículo pueda visitar clientes de las distintas áreas que se le asignaron (saliendo de una área para visitar a un cliente de otra, y tal vez volviendo a regresar), nos encontramos con el problema CluVRP WCC.

La aplicación del problema CluVRP SCC suele darse en circunstancias donde un conjunto de clientes tiene que ser servido por el mismo vehículo. Un ejemplo de esto es el transporte de personas mayores hacia centros de recreación, donde estas personas (clientes) prefieren ser recogidas junto a sus amigos o vecinos formando grupos (clusters).
En el trabajo de Battarra et. al. (2014) [9] se estudia la aplicación de este problema a la recolección de residuos sólidos en áreas urbanas. En ese caso, los puntos de recolección de cada vecindario están agrupados en clusters que deben ser visitados de forma secuencial para cumplir con una compacidad suficiente en el diseño de la ruta.
Los barrios cerrados (Gates Communities en inglés), o áreas residenciales o industriales cerradas y separadas físicamente por muros, por razones de protección y seguridad, son también un ejemplo donde este problema puede aplicarse. Los clientes que pertenecen a una misma área (cluster) deben ser visitados de forma consecutiva por un mismo vehículo, ya que de otra forma el vehículo gastará tiempo adicional por controles de seguridad en la entrada.

Los problemas resultantes de las distintas generalizaciones del Problema de Ruteo de Vehiculos (VRP) se han convertido en los problemas más estudiados de optimización combinatoria. Esto se debe a la gran variedad de aplicaciones que los mismos tienen en problemas de la vida real y a su importancia económica. Ejemplos de esto son: envío de postales, ruteo de vehículos escolares y distribución de mercadería, entre otros. Este fenómeno se estudia el trabajo de Toth y Vigo (2002) [10] donde se concluye que la aplicación de métodos computarizados en problemas de ruteo, reduce los costos de transporte entre un 5 % y un 20 %.

Respecto a la literatura disponible sobre el problema CluVRP, desde su introducción en el trabajo de Sevaux y Sörensen (2008) [8], año a año han surgido nuevos trabajos, aplicando diferentes técnicas, agregando nuevas instancias de estudio y mejorando en muchos casos los resultados en cada publicación.

En el primer trabajo sobre CluVRP SCC, Sevaux y Sörensen (2008) [8], se estudia el problema clásico de ruteo de vehículos VRP, pero utilizando casos de grandes dimensiones, dividiendo los clientes en zonas y trabajando el ruteo de las mismas en vez de hacerlo individualmente con cada cliente. Esto reduce enormemente el costo computacional, pero deja pendiente la tarea de programar la ruta de clientes a seguir dentro de una zona. Es por esto último que en dicho trabajo se estudia el problema del camino hamiltoniano más corto para armar las rutas de cada zona, y se dan razones a favor de que esa técnica es una buena base para decidir la ruta utilizada para recorrer los clientes de una zona. Para esto se formula una aplicación de programación lineal entera y se estudia su performance. También se estudia como ese método exacto puede embeberse en una metaheurística para resolver el problema. La transformación propuesta al problema VRP por Savaux y Sörensen, termina en la definición formal de CluVRP SCC.

El siguiente trabajo sobre CluVRP SCC fue el de Barthélemy et. al. (2000) [11]. En este trabajo se estudió la aplicación de la técnica metaheurística *Simulated Annealing*, combinada con la técnica que en el trabajo denominaron *big M approach*.
La técnica *big M* determina una gran distancia, llamemos M en la matriz de distancias entre los nodos que no pertenecen a un mismo cluster. De esta forma se obliga al vehículo a visitar a todos los clientes de un mismo cluster antes de abandonar el mismo, ya que lo contrario tiene una gran penalidad respecto a la distancia total recorrida. En este trabajo se realizan pruebas sobre instancias creadas por los autores.

El Problema de Ruteo Generalizado [12] (GVRP por sus siglas en inglés) es una variante de VRP cercana a CluVRP SCC y WCC. En GVRP los clientes están agrupados en clusters, pero para realizar la entrega, solo es necesario visitar un cliente por cluster. En el trabajo de Pop, Kara y Marc (2012) [13], se desarrollan dos métodos exactos para atacar GVRP, extendiendo los mismos para resolver CluVRP SCC. Los resultados para GVRP publicados son contrastados con el trabajo de Ghiani y Improta del año 2012 [14].

En el trabajo de Battarra, Erdogan y Vigo (2014) [9], se realizó un aporte muy significativo al estudio de CluVRP. En dicho trabajo se implementa un algoritmo de programación lineal entera, logrando resultados óptimos en instancias de tamaño medio, y buenos resultados en instancias de tamaño más grande. También se publicaron los resultados obtenidos y los tiempos de procesamiento. Por otro lado se estandarizaron cierto conjunto de instancias de pruebas que fueron utilizadas en trabajos posteriores, logrando así tener puntos de comparación entre los nuevos trabajos, algo que hasta el momento no sucedía.

En Marc at. al. (2015) [15], se estudian distintas derivaciones de VRP, y se propone un algoritmo para resolver CluVRP. En este trabajo se publicaron resultados, pero sin tiempos de ejecución.

El siguiente aporte significativo al problema CluVRP, se realiza en el trabajo de Vidal at. al. (2015) [16]. En este trabajo se proponen dos heurísticas para afrontar el problema. La primera es una adaptación del algoritmo *Iterated Local Search* (ILS) desarrollado por Subramanian [17] para CVRP. En el mismo para evitar la evaluación de gran cantidad de movimientos inútiles, además de las restricciones sobre los clusters, se re-define el concepto de vecindad del problema.

El segundo algoritmo propuesto se basa en la técnica UGHS (Unified Hybrid Genetic Search). En el trabajo se menciona que este método fue aplicado a problemas VRP sin clusters, porque para utilizarlo en CluVRP, pre-calculan todos los caminos dentro de los clusters.
Respecto a los resultados de este trabajo, en términos de calidad fueron muy buenos, pero en términos de tiempo de ejecución muy altos.

En el trabajo de Defryn y Sörensen (2015) [18], se propone descomponer CluVRP SCC en dos niveles de optimización. Por un lado un problema de ruteo de alto nivel para trabajar sobre el ruteo de los clusters y por el otro, uno de bajo nivel para trabajar en el ruteo de clientes dentro de los clusters. Para lograr lo planteado se utiliza la técnica metaheurística *Variable neighborhood search* (VNS) en los dos niveles propuestos. Los resultados publicados se muestran muy cercanos a los del trabajo de Battarra et al. [9].

Dos años más tarde en Expósito-Izquierdo et al. (2016) [19], también se realiza una descomposición del problema en dos niveles, y se propone una solución que combina dos técnicas: el algoritmo *Record-to-Record* a nivel cluster y el algoritmo heurístico *Lin–Kernighan* para determinar los caminos dentro de clientes en un cluster. En este trabajo se estudió el comportamiento del algoritmo propuesto bajo un nuevo set de instancias, obteniendo muy buenos resultados con tiempos bajos.

El último de los trabajos que encontré sobre CluVRP, es el publicado por Defryn et al. [20].
En este trabajo se presenta también una solución de dos niveles, utilizando la técnica heurísta *Variable neighborhood search* en ambos. Los resultados de este trabajo fueron comparados con los trabajos de Battarra [9], Vidal [16] y Expósito-Izquierdo [19]. Los resultados denotan gran calidad, superando o igualando casi todos los obtenidos anteriormente, con excelentes tiempos de cómputo. En este trabajo también se presenta formalmente el problema CluVRP WCC y se evalúa el algoritmo propuesto en dicho problema, comparando los resultados con los obtenidos para CluVRP SCC.

3. DEFINICIÓN DE LOS PROBLEMAS

3.1. CluVRP Strong Cluster Constraints (SCC)

3.1.1. Definición

Imaginemos que trabajamos en una empresa de entrega de paquetes o de mensajería. La logística de entrega en este negocio suele modelarse de la siguiente forma: se clasifican todos los paquetes a entregar en contenedores, donde cada contenedor corresponde a una parte específica y predefinida del área de distribución, llamada zona. Todos los paquetes del contenedor deben entregarse, por lo que todos los clientes de la zona deber ser servidos (su paquete correspondiente entregado) antes que el contenedor de otra zona se distribuya. Para realizar la distribución lo primero que debe resolverse es el planeamiento de la distribución, esto es, asignar los contenedores (zonas) a los vehículos disponibles. Si un vehículo puede cargar más de un contenedor, los contenedores que se le asignen determinará la zonas que el vehículo visitará, por lo tanto el recorrido del mismo. Esto último suponiendo que el vehículo tiene un poder de carga limitado, y cada contenedor tiene su propio valor de carga.

Una vez asignados los contenedores a los vehículos, ya sabemos qué zonas visitará cada vehículo, por lo tanto resta definir el orden en que el vehículo servirá a todos los clientes en cada zona. El orden en que los mismos se visiten tiene un impacto en la distancia total recorrida por un vehículo, por lo tanto estamos ante otro problema de logística a resolver.

Por lo anteriormente dicho, el objetivo del problema CluVRP SCC es el de minimizar la ruta total recorrida por una flota fija de vehículos homogéneos que poseen una cierta capacidad de carga (todos la misma), donde estos deben servir la demanda de distintas áreas (clusters), y dentro de cada área, el vehículo asignado a la misma, debe visitar a todos sus clientes en forma secuencial, retirándose del área una vez que visitó a todos sus clientes. El recorrido de todos los vehículos comienza en un mismo depósito, y luego del realizar las entregas asignadas, finaliza en el mismo depósito desde donde se partió.

3.1.2. Formulación Matemática

Dadas las siguientes definiciones:

- Sea $G = (V, E)$ un grafo no dirigido, donde V es un conjunto de vértices incluyendo el deposito (V_0) y múltiples nodos clientes.
- La distancia d_{ij}, está asociada con la arista $(i, j) \in E$ conectando dos nodos.
- Sea K a un conjunto homogéneo de vehículos con una capacidad máxima Q.
- Todos los vehículos comienzan y terminan su recorrido en el depósito.
- Para cada cliente i la demanda del mismo se denota q_i.
- Un conjunto de clusters definido como R.

- El cluster $r_0 \in R$ solo contiene un nodo, el depósito (V_0).
- Todos los clusters contienen al menos un cliente.
- Sea $C_r = \{i \in V \backslash V_0 : r_i = r\}$, $\forall r \in R$ el conjunto de clientes en un cluster.
- Sea Z cualquier subconjunto propio de V. Definimos $\delta^+(Z)$ el conjunto de aristas $(i,j) \in Z \times V \backslash Z$ y δ^- el conjunto de aristas $(i,j) \in V \backslash Z \times Z$.
- Sea:

$$x_{ijk} = \left\{ \begin{array}{c} \text{1 vehículo k viaja desde el nodo i al nodo j} \\ \text{0 caso contrario} \end{array} \right\}$$

- Sea:

$$y_{ik} = \left\{ \begin{array}{c} \text{1 cliente i es atendido por vehículo k} \\ \text{0 caso contrario} \end{array} \right\}$$

La formulación matemática del problema CluVRP SCC quedó definida en el trabajo de Defryn et al. [20] del siguiente modo:

I. $min \sum_{(i,j)\in E} \sum_{k\in K} d_{ij} x_{ijk}$

Sujeto a:

II. $\sum_{k\in K} y_{ik} = 1 \qquad \forall i \in V \backslash V_0$

III. $\sum_{k\in K} y_{0k} = |K|$

IV. $\sum_{j\in V\backslash V_0} x_{ijk} = \sum_{j\in V\backslash V_0} x_{jik} = y_{ik} \qquad \forall k \in K, \forall i \in V$

V. $\sum_{i\in V} q_i y_{ik} \leqslant Q \qquad \forall k \in K$

VI. $\sum_{i\in Z} \sum_{j\notin Z} x_{ijk} \leqslant y_{hk} \qquad \forall Z \subseteq V \backslash V_0, \forall h \in Z, \forall k \in K$

VII. $\sum_{(i,j)\in\delta^+(C_r)} \sum_{k\in K} x_{ijk} = \sum_{(i,j)\in\delta^-(C_r)} \sum_{k\in K} x_{ijk} = 1 \qquad \forall r \in R$

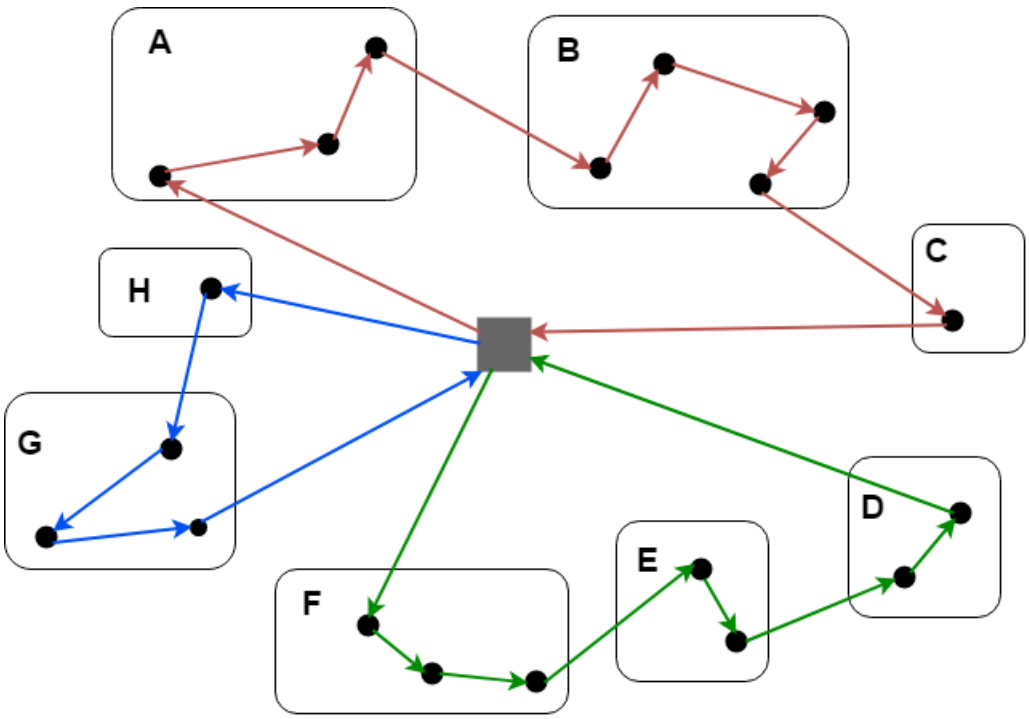
VIII. $x_{ijk} \in \{0,1\} \qquad \forall (i,j) \in E, \forall k \in K$

IX. $y_i \in \{0,1\} \qquad \forall i \in V, \forall k \in K$

Detalles de las restricciones:

- I) función objetivo, minimiza la distancia total recorrida por todos los vehículos.
- II) asegura que cada cliente sea visitado exactamente una vez.
- III) todos los vehículos deben visitar el depósito.
- IV) garantiza que el mismo vehículo que llega a un cliente también abandona ese cliente.
- V) la capacidad del vehículo es respetada.
- VI) si un vehículo vista un nodo entonces este es servido por ese vehículo.
- VII) cada cluster es visitado exactamente una vez por un vehículo
- $VIII$ y IX) definiciones de variables

A continuación un gráfico de ejemplo del problema CluVRP SCC con tres vehículos, ocho clusters y diecinueve clientes.



3.2. CluVRP Weak Cluster Constraints (WCC)

3.2.1. Definición

En el problema CluVRP SCC anteriormente explicado, la asignación de un vehículo a un cluster significaba que este debía visitar a todos los clientes del mismo, de forma consecutiva antes de visitar al primer cliente de otro cluster (o volver al depósito). En este sentido muchas veces, relajar la condición de visitar estrictamente todos los clientes de un cluster antes de salir del mismo, podría ser conveniente para optimizar la ruta de un vehículo. Relajar esa condición, significa que todos los clientes que pertenecen a un mismo cluster deben ser visitados por el mismo vehículo (al cual se le asigno el cluser), pero no necesariamente los clientes tienen que ser visitados de forma consecutiva, ya que el vehículo podría salir de un cluster y visitar clientes de otro cluster que tenga asignado, y luego volver al anterior para seguir sirviendo la totalidad de los clientes.

En algunos problemas de la vida real, como por ejemplo en la entrega de paquetes, los clientes son agrupados en zonas (clusters) para facilitar el proceso de entrega. Esas zonas son asignadas a vehículos disponibles, obteniendo un plan, donde cada vehículo puede servir múltiples zonas en un viaje. Una vez que el vehículo comienza su viaje, no siempre se deben visitar la totalidad de los clientes de una zona de forma consecutiva, ya que durante el viaje puede suceder que dejar la zona en la que se encuentra el vehículo para dirigirse a otra zona para servir clientes y para más tarde volver a la zona inicial, sea algo redituable para el conductor. Lo anterior puede depender de muchos factores, como la ubicación de los clientes, la composición de las zonas, pero también puede depender del estado real del tráfico o de restricciones adicionales como ventanas de tiempo.
Lo anteriormente mencionado define al problema CluVRP WCC, que a diferencia del problema CluVRP SCC, se permite que un vehículo para visitar a los clientes de los clusters asignados, abandone y vuelva a entrar a un cluster múltiples veces en su viaje.

3.2.2. Formulación Matemática

Dadas las siguientes definiciones:

- Sea $G = (V, E)$ un grafo no dirigido, donde V es un conjunto de vértices incluyendo el depósito (V_0) y múltiples nodos clientes.
- La distancia d_{ij}, está asociada con la arista $(i, j) \in E$ conectando dos nodos.
- Sea K a un conjunto homogéneo de vehículos con una máxima capacidad Q.
- Todos los vehículos comienzan y terminan su recorrido en el depósito.
- Para cada cliente i la demanda del mismo se denota q_i.
- Un conjunto de clusters definido como R.
- El cluster $r_0 \in R$ solo contiene un nodo, el depósito (V_0).
- Todos los clusters contienen al menos un cliente.
- Sea $C_r = \{i \in V \backslash V_0 : r_i = r\}$, $\forall r \in R$ el conjunto de clientes en un cluster.

- Sea Z cualquier subconjunto propio de V. Definimos $\delta^+(Z)$ el conjunto de aristas $(i,j) \in Z \times V \backslash Z$ y δ^- el conjunto de aristas $(i,j) \in V \backslash Z \times Z$.
- Sea:

$$x_{ijk} = \left\{ \begin{array}{c} \text{1 vehículo k viaja desde el nodo i al nodo j} \\ \text{0 caso contrario} \end{array} \right\}$$

- Sea:

$$y_{ik} = \left\{ \begin{array}{c} \text{1 cliente i es atendido por vehículo k} \\ \text{0 caso contrario} \end{array} \right\}$$

La formulación matemática del problema CluVRP SCC quedó definida en el trabajo de Defryn et al. [20] del siguiente modo:

I. $min \sum_{(i,j)\in E} \sum_{k\in K} d_{ij} x_{ijk}$

Sujeto a:

II. $\sum_{k\in K} y_{ik} = 1 \qquad \forall i \in V \backslash V_0$

III. $\sum_{k\in K} y_{0k} = |K|$

IV. $\sum_{j\in V\backslash V_0} x_{ijk} = \sum_{j\in V\backslash V_0} x_{jik} = y_{ik} \qquad \forall k \in K, \forall i \in V$

V. $\sum_{i\in V} q_i y_{ik} \leqslant Q \qquad \forall k \in K$

VI. $\sum_{i\in Z} \sum_{j\notin Z} x_{ijk} \leqslant y_{hk} \qquad \forall Z \subseteq V \backslash V_0, \forall h \in Z, \forall k \in K$

VII. $\sum_{(i,j)\in\delta^+(C_r)} \sum_{k\in K} x_{ijk} = \sum_{(i,j)\in\delta^-(C_r)} \sum_{k\in K} x_{ijk} \geq 1 \qquad \forall r \in R$

VIII. $y_{ik} = y_{jk} \quad \forall i,j \in C_r, \forall k \in K$

IX. $x_{ijk} \in \{0,1\} \qquad \forall (i,j) \in E, \forall k \in K$

X. $y_i \in \{0,1\} \qquad \forall i \in V, \forall k \in K$

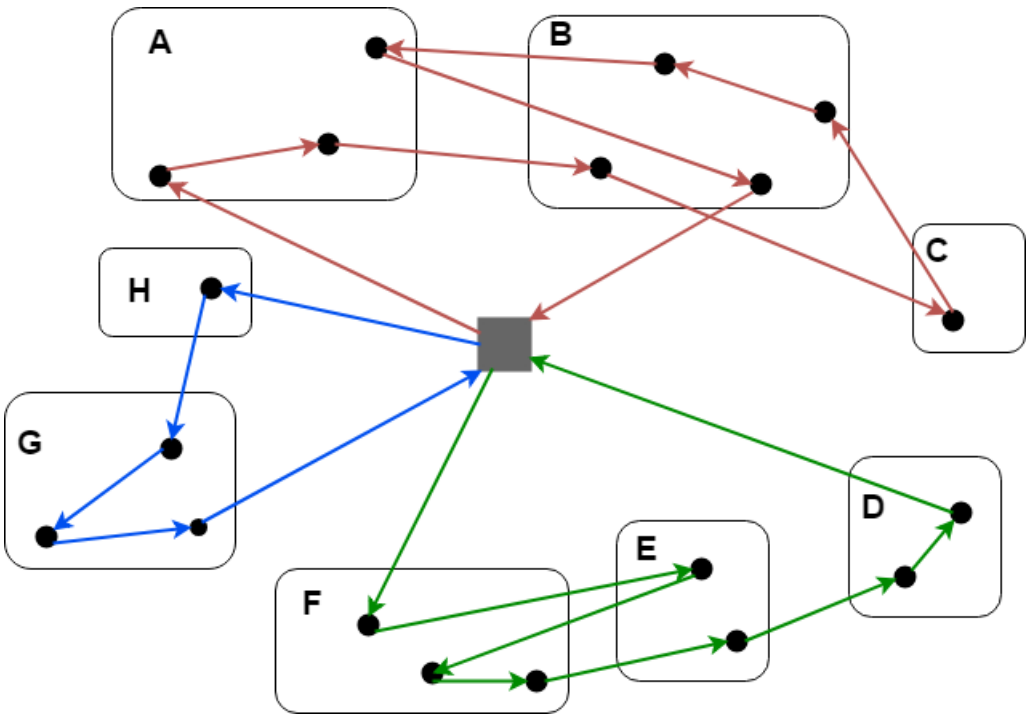
Detalles de las restricciones:

- I) función objetivo, minimiza la distancia total recorrida por todos los vehículos.
- II) asegura que cada cliente sea visitado exactamente una vez.

- III) todos los vehículos deben visitar el depósito.
- IV) garantiza que el mismo vehículo que llega a un cliente también abandona ese cliente.
- V) la capacidad del vehículo es respetada.
- VI) si un vehículo vista un nodo entonces este es servido por ese vehículo.
- VII) los clusters pueden ser visitados múltiples veces
- $VIII$) todos los clientes que pertenecen a un cluster son visitados por el mismo vehículo
- IX y X) definiciones de variables

Cómo se puede ver, respecto a la formulación para CluVRP SCC solo cambió la fórmula 7, que es la que permite relajar la restricción sobre los clusters, y se agrega la fórmula 8, que asegura que los clientes que pertenecen a un mismo cluster sean visitados por el mismo vehículo.

A continuación un gráfico de ejemplo del problema CluVRP WCC con tres vehículos, ocho clusters y diecinueve clientes.



4. METAHEURÍSTICA GRASP

4.1. Introducción

Una heurística es un procedimiento simple, a menudo basado en el sentido común, que supone ofrecer una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido (Zanakis y Evans, 1981 [21]). Según el trabajo de Silver (1980) [22], se pueden clasificar los métodos de resolución mediante heurísticas, de la siguiente forma:

- Métodos constructivos, que se caracterizan por construir una solución definiendo diferentes partes de ella en sucesivos pasos.
- Métodos de descomposición, que dividen el problema en varios más pequeños y la solución se obtiene a partir de la solución de cada uno de estos.
- Métodos de reducción, tratan de identificar alguna característica de la solución que permita simplificar el tratamiento del problema.
- Métodos de manipulación del modelo, que obtienen una solución del problema original a partir de otra de otro problema simplificado (con menos restricciones, linealizandolo, etc.)
- Métodos de búsqueda por entornos, en las que se parte de una solución inicial a la que se realizan modificaciones en sucesivas iteraciones para obtener una solución final. En cada iteración existe un conjunto de soluciones vecinas candidatas a ser nueva solución en el proceso. En este grupo se encuadran las técnicas *metaheurísticas*.

Las técnicas mataheurísticas [23] tienen su origen en los años setenta. A lo largo de los años, nuevas técnicas se fueron sumando en la familia, logrando abarcar más problemas. Esto permitió abrir el abanico de posibilidades a la hora de enfrentar problemas de distinto tipo con esta técnica.
Al igual que los métodos heurísticos, las técnicas metaheurísticas son procedimientos de búsqueda que no garantizan la obtención de la solución óptima del problema y también se basan en la aplicación de reglas simples y sencillas. La diferencia considerable respecto a los métodos heurísiticos, es que las técnicas metaheurísticas tratan de escapar de los óptimos locales re-orientando la búsqueda en cada momento, dependiendo de la evolución del proceso.

Si bien estas técnicas pueden aplicarse a un gran conjunto de problemas, una de las aplicaciones más interesantes y de gran uso, es en los problemas de optimización combinatoria.
Entre las metaheurístcas más populares se encuentran: la búsqueda tabú, la búsqueda "scatter", las colonias de hormigas, los algoritmos genéticos y el recocido simulado, entre otras.
En este trabajo se utilizará la técnica metaheuristica de Procedimiento de búsqueda basado en funciones golosas aleatorias y adaptativas, *Greedy randomized adaptive search*

procedure (en inglés), conocida popularmente como *GRASP*, por sus siglas en inglés.

Según Sadiq y Habib, (1999) [24], la mayoría de las técnicas metaheurísticas poseen todas o algunas de las siguientes características:

- Son ciegas, lo que significa que no saben si llegarán a la solución óptima. Por lo tanto, se les debe indicar cuándo deben detenerse.
- Son algoritmos aproximativos y, por lo tanto, no garantizan la obtención de la solución óptima.
- Aceptan ocasionalmente malos movimientos. Se trata de procesos de búsqueda en los que cada nueva solución no es necesariamente mejor (en términos de la función objetivo) que la inmediatamente anterior. Algunas veces aceptan, incluso, soluciones no factibles como paso intermedio para acceder a nuevas regiones no exploradas.
- Son relativamente sencillas. Todo lo que se necesita es una representación adecuada del espacio de soluciones, una solución inicial (o un conjunto de ellas) y un mecanismo para explorar el campo de soluciones.
- Son generales. Prácticamente se pueden aplicar en la resolución de cualquier problema de optimización de carácter combinatorio. Sin embargo, la definición de la técnica será más o menos eficiente en la medida en que las operaciones tengan relación con el problema considerado.
- La regla de selección depende del instante del proceso y de la historia hasta ese momento. Si en dos iteraciones determinadas, la solución es la misma, la nueva solución de la siguiente iteración no tiene por qué ser necesariamente la misma. En general, no lo será.

Greedy Randomized Adaptive Search Procedure, conocida como GRASP, es una técnica metaheurística de propósito general creada por Feo y Resende en el año 1989 [7] [25].
La técnica GRASP es un algoritmo iterativo en el cual cada iteración consta de dos fases, una primera fase llamada *fase de construcción* y una segunda fase llamada *fase de búsqueda local*.
La primera fase consiste en la aplicación de un algoritmo goloso, randomizado y adaptativo que genera una solución completa del problema a resolver. La segunda fase consiste en aplicar a esta solución ya generada (y completa) distintas técnicas heurísticas de búsqueda local con el fin de encontrar una solución superior a la inicial.

A continuación se presenta su pseudo-código:

Algoritmo 1: GRASP()

```
InstanciaDeEntrada();
while no se satisface condición de parada de GRASP do
    ConstruirSoluciónGolosaRandomizada(solución);
    BúsquedaLocal(solución);
    ActualizarSolución(solución, mejorSolución);
retornar(MejorSolución)
```

En el algoritmo 1, se pueden observar claramente las dos fases que se mencionaron con anterioridad. El ciclo principal del algoritmo está controlado por una condición de parada, y esta condición normalmente es una cantidad fija de iteraciones en la que se busca una mejor solución a la actual, o también una cantidad de iteraciones máxima a la espera de una mejora en la solución.
A la primer fase, *ConstruirSolucionGolosaRamdomizada(solución)*, la llamaremos **fase constructiva**, y a la segunda fase *BúsquedaLocal(solución)*, la llamaremos **fase de búsqueda local**. A continuación daremos más detalles sobre cada una.

4.2. Fase de Construcción

En esta fase se construye una solución completa del problema utilizando un algoritmo goloso-aleatorio [26]. La diferencia entre un algoritmo goloso-aleatorio y el algoritmo goloso clásico, es que en cada paso, en lugar de seleccionar el mejor vecino, se construye una lista de candidatos llamada *Restricted Candidate List* o *RCL* (por sus siglas en inglés) y de esa lista se selecciona un vecino de manera aleatoria.
Esa es la componente *Greedy Randomized* de la técnica GRASP. Esto puede verse en el pseudo-código del algoritmo 2.
La forma en que se crea esta lista varía según la implementación. Más adelante se detallará el criterio utilizado en este trabajo.
El componente aleatorio del algoritmo permite que se obtengan diferentes soluciones para cada iteración del ciclo principal del algoritmo, aumentando el número de soluciones candidatas. El término *adaptativo* que GRASP lleva en su nombre, refiere al hecho de que los beneficios asociados con cada elemento son actualizados en cada iteración de la fase de construcción para reflejar los cambios producidos por las selecciones previas.

Algoritmo 2: ConstruirSoluciónGolosaRamdomizada(solución)

```
solución = ∅;
while no se haya construido Solución do
    ContruirRCL(RCL);
    s = SeleccionarElementoAleatoriamente(RCL);
    solución = solución ∪ {s};
    AdaptarSoluciónGolosa(s);
```

4.3. Fase de Búsqueda Local

Como en el caso de muchos métodos deterministas, la solución generada por una construcción golosa-aleatoria no garantiza la optimalidad local con respecto a la definición simple de *vecindario*. Por esto, es siempre beneficioso aplicar una búsqueda local para mejorar la solución construida en la anterior fase.
Una búsqueda local realiza sucesivos reemplazos a la solución actual en busca de una mejor solución en la vecindad de la actual solución. Este procedimiento en general finaliza o con una cantidad fija de iteraciones, o cuando no es posible hallar una solución mejor.

El algoritmo 3 describe de forma general el procedimiento de búsqueda local. Este se compone de una estructura de vecindad N para un problema P, con relación a una

solución del problema s en el subconjunto de soluciones $N(s)$.
Una solución s es llamada *óptimo local* si no existe mejor solución en N(s). La clave del éxito de este algoritmo consiste en una correcta elección de la estructura para la vecindad y de una eficiente técnica de búsqueda tanto en la vecindad como en la solución inicial.

Algoritmo 3: BusquedaLocal(P, V(P), s)

```
while s no es optimo local do
    Buscar mejor solución t ∈ N(S);
    s = t;
retornar(s como óptimo local para P)
```

4.4. Criterio de Construcción de la RCL

Como se mencionó, la forma en que se construye la lista restringida de candidatos puede variar debido a que no existe una única manera de incluir vecinos en ella. Podría ser un número fijo para cualquier problema, o bien un número en función de la densidad del grafo.
En este trabajo se optó por la solución propuesta en el libro de Resende y Ribeiro (2016) [27], en el que se introduce un parámetro α en la siguiente forma:
Supongamos f la función golosa, α el valor del parámetro para controlar la aleatoriedad del procedimiento, x una solución parcial, C el conjunto de elementos candidato y RCL la lista restringida de candidatos. Entonces se propone el siguiente algoritmo:

Algoritmo 4: ConstruirRCL(RCL)

```
x = ∅;
Inicializar el conjunto de candidatos C;
while x factible do
    a = min{f(t)|t ∈ C};
    b = max{f(t)|t ∈ C};
    RCL = {c ∈ C|f(c) ≤ a + α(b − a)};
    Seleccionar aleatoriamente un elemento c ∈ RCL;
    x ← x ∪ {c};
    Actualizar C;
```

Es importante notar que cuando $\alpha = 0$, entonces $a + \alpha(b - a) = a$, y esta es la distancia al vecino más cercano. Por lo tanto en la RCL solo estará el vecino más cercado, convirtiendo este algoritmo en un algoritmo **goloso puro**. Si por el contrario $\alpha = 1$, entonces $a + \alpha(b - a) = b$, implica que todos los vecinos entrarán en la RCL, por lo que el algoritmo termina teniendo un comportamiento **completamente aleatorio**.

4.5. Algoritmos de Búsqueda Local

Si bien en el esquema anterior se dio una visión general de cómo se aplica la búsqueda local en GRASP, en esta sección se explicaran brevemente algunas de las búsquedas locales que más se utilizan en los problemas de ruteo de vehículos. Para el completo

entendimiento de las búsquedas locales utilizadas en este trabajo, antes se deberá definir la propia implementación de GRASP, para luego explicar el diseño y la utilización de ellas.

Con la idea de ser esta una sección de introducción a la metaheurística GRASP, podemos mencionar que en este trabajo se utilizan algunas de las heurísticas de búsqueda local más utilizadas para la resolución de problemas pertenecientes a la familia de problemas VRP. A continuación y a modo de ejemplo se brindara un subconjunto de las mismas y una breve explicación de su funcionamiento:

- *Reubicación*
  Consisten en cambiar un nodo de lugar, dentro del mismo camino al que pertenece.
- *Intercambio de Nodos*
  Dados dos nodos que pertenecen al mismo camino, se intercambian sus posiciones.
- *Intercambio de Caminos*
  En esta transformación, se toman 2 porciones del camino y se intercambian entre sí.
- *2-opt*
  Consiste en seleccionar un tramo del camino e invertirlo.
- *Insertar*
  Consiste en tratar de insertar un nodo de un camino en otro camino.

5. ALGORITMO IMPLEMENTADO

5.1. Introducción

En este trabajo se implementará un algoritmo de dos fases siguiendo las ideas de Defryn et al. [20] para resolver los problemas CluVRP con restricciones fuertes sobre los clusters (CluVRP SCC) y el problema CluVRP con restricciones débiles sobre los clusters (CluVRP WCC).

La primera fase, la fase de Nivel Cluster, genera una ruta de clusters para un conjunto de vehículos, cumpliendo con la demanda de los mismos. En esta fase no se calculan las rutas de clientes dentro de los clusters.

La segunda fase, la fase de Nivel Cliente, requiere una ruta de cluster como entrada y produce una solución tanto para CluVRP WCC o para CluVRP SCC, según se lo requiera. A continuación una breve introducción de las dos fases implementadas:

Fase de Nivel Cluster

En esta fase se intenta buscar una solución al problema de como minimizar la distancia recorrida por una flota de vehículos fija con cierta capacidad que deben servir a un conjunto de clusters con una cierta demanda, sin visitar a los clientes dentro de los clusters. Planteado de esta forma, el problema se simplifica, convirtiéndose en el conocido problema CVRP [5].
Uno de los problemas de esta fase consiste en determinar cual será la definición de *distancia* entre dos clusters, ya que el problema CluVRP, por definición, solo cuenta con las ubicaciones de los clientes. En el algoritmo implementado para esta fase se utilizarán dos posibles formas de calcular esta distancia:

- La primera forma consiste en utilizar la distancia más corta entre dos clientes pertenecientes a dos clusters distintos. Para esto se calcula la distancia entre todos los clientes de un cluster y el del otro, definiendo la distancia entre ambos clusters como la distancia mínima entre las calculadas.
- La segunda forma consiste en calcular el centroide (centro de masa) de un cluster, utilizando ese punto como ubicación del mismo. Luego la distancia entre dos clusters, es la distancia entre los centroides de cada cluster.

Para resolver el problema de Nivel Cluster se utilizará la técnica GRASP, con los conceptos de distancia propuestos.

Fase de Nivel Cliente

Esta fase tiene como entrada una solución de la fase de Nivel Cluster, y a partir de la misma se busca encontrar una solución al problema completo, esto es, generar un camino de clientes para la flota de vehículos, cumpliendo las restricciones de CluVRP SCC

o CluVRP WCC, dependiendo cual problema se quiera solucionar.
Para enfrentar este problema se aplicará (también como en la fase de Nivel Cluster) la técnica metaheuristica GRASP.

Por último luego de la ejecución de las fases, y generada una solución completa tanto para CluVRP SCC o para CluVP WCC, se aplicarán una serie de técnicas heurísticas en busca de mejorar la solución.
A continuación se plantea el esquema general del algoritmo propuesto:

Algoritmo 5: CluVRP()

```
InstanciaDeEntrada();
iterador = 0;
while iterador < iteracionesTotales do
    solucionNivelCluster = GRASPNivelCluster();
    solución = GRASPNivelCliente(solucionNivelCluster);
    BusquedaLocal(solucion);
    ActualizarSolucion(solución, mejorSolucion);
    iterator = iterador + 1;
retornar(MejorSolucion);
```

Podemos ver en el algoritmo 5 que el esquema principal de la solución implementada es bastante sencillo. Cómo se mencionó, el procedimiento realiza una cantidad fija de iteraciones, creando para cada una, primero una solución del problema a nivel cluster y luego a partir de ella, una solución del problema CluVRP WCC o CluVRP SCC, según se requiera. Una vez que la solución está generada se aplican heurísticas de búsqueda local para intentar mejorar la solución.
Las heurísticas de búsqueda local utilizadas en este proceso *BúsquedaLocal* del algoritmo 5 son las siguientes:

- *IntercambioDeVehículo*
- *InsertarClusterEnVehículo*
- *2-Opt*
- *Relocalizar*
- *Intercambio*

Estas heurísticas son también utilizadas en las dos fases implementadas, por lo que el detalle de las mismas se encontrará en las secciones 5.2.4 y 5.3.5.

5.2. Fase de Nivel Cluster

5.2.1. Introducción

Como se mencionó en la introducción de este capítulo, en esta fase se busca una solución parcial de nivel cluster para el probleam CluVRP (tanto SCC como WCC). Esto significa que dado una flota de vehículos con una cierta capacidad y un conjunto de clientes agrupados en clusters, se buscará minimizar el camino recorrido por la flota de vehículos, pero sin realizar el viaje intra-cluster (sin visitar clientes dentro de un cluster).
El resultado de esta fase es un camino de clusters para cada vehículo, partiendo del cluster *depósito*, visitando los clusters asignados, cumpliendo la demanda, y volviendo al cluster *depósito*.
Este proceso evita el recorrido interno de clientes de un cluster, por lo que solo calcula el camino recorrido visitando cada cluster. El problema aquí es que el *cluster* es un concepto abstracto, ya que una agrupación de clientes forma uno, por lo que no existe la definición de distancia entre dos clusters. Entonces, para armar un camino de clusters es necesario definir el concepto de distancia entre ellos. Como ya se mencionó en la introducción, para esto se utilizarán dos definiciones distintas que se detallarán a continuación.

Distancia mínima entre dos clientes

Consiste en calcular la distancia euclidiana entre todos los clientes de un par de clusters y tomar la mínima. Formalmente, dado dos clusters $C_1 = \{v_1, v_2, ..., v_n\}$ y $C_2 = \{d_1, d_2, ..., d_m\}$, siendo $v_i \in c_1$ y $d_i \in c_2$ sus respectivos clientes, entonces:

$$distancia(C_1, C_2) = min(\forall v \in C_1, \forall v \in C_2 \quad dist(v, d))$$

Distancia entre los centroides de los clusters

Dado un conjuntos de puntos $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ un centroide (o centro de masa) se define de la siguiente forma:

$$\sum_{i=0}^{i=n} \frac{(x_i, y_i)}{n}$$

Por lo que dado un cluster $C = \{v_1, v_2, ..., v_n\}$, los clientes $v_1, v_2, ..., v_n$ que lo conforman, y sea (x_i, y_i) la ubicación del cliente v_i. Entonces el centroide de C se define de la siguiente forma:

$$centroide(C) = \sum_{i=0}^{i=n} \frac{(x_i, y_i)}{n} \quad para \quad (x_i, y_i) = ubicacion(v_i) \quad \forall v_i \in C$$

Las dos formas utilizadas para dar una noción de distancia entre clusters serán testeadas en las pruebas del algoritmo para verificar la eficiencia de una respecto de la otra.

Para abordar el objetivo propuesta de esta fase, se implementó un algoritmo GRASP, donde primero se creará una solución golosa aleatoria y luego se aplicaran distintos algoritmos de búsqueda local, para mejorar la solución.

La solución golosa generada en la primera parte del algoritmo tiene que ser una solución parcial y válida. Parcial porque es un camino de clusters para cada vehículo (y no de clientes) y válida porque la demanda de los clusters tiene que ser atendida. Por lo tanto, en esta solución parcial, un vehículo no puede atender más demanda que la que le proporciona su capacidad.
El problema de satisfacer la demanda suma una complejidad muy importante al proceso, ya que no solo se busca minimizar el camino recorrido por la flota de vehículos, si no que además, puede suceder algunas soluciones sean inválidas debido a que la suma de las demandas de los clusters de algún camino no pueden ser satisfechas por el vehículo asignado. Si la capacidad total (la suma de todas las capacidades de todos los vehículos) es cercana a la demanda total (la suma de la demanda de todos los clusters), puede suceder que la dificultad para generar una solución válida aumente debido a que existan pocas formas de asignar clusters a vehículos.

5.2.2. GRASP de Nivel Cluster

A continuación se presenta el pseudo-código de la versión de GRASP de Nivel Cluster implementada en este trabajo.

Algoritmo 6: GRASPNivelCluster()

```
InstanciaDeEntrada();
iterador = 0;
while iterador < iteracionesTotales do
    algoritmoGoloso = seleccionarAlgoritmoGoloso();
    algoritmoGoloso(solucion);
    if solucion NO es completa then
        RepararSolucion(solucion);
    BusquedaLocal(solucion);
    ActualizarSolución(solución, mejorSolucion);
    iterator = iterador + 1;
retornar(MejorSolucion);
```

Algoritmo 7: RepararSolucion(vehiculos, clustersSinVehiculo)

```
forall clusterAInsertar ∈ clustersSinVehiculo do
    espacioRequerido = clusterAInsertar.demanda();
    forall v1 ∈ vehiculos do
        forall v2 ∈ vehiculos do
            forall c1 ∈ v1.ruta() do
                forall c2 ∈ v2.ruta() do
                    espacioGenerado = espacioGeneradoSwapVehiculo(c1, c2, v1, v2);
                    if espacioGenerado ≤ especioRequerido then
                        cambiarDeVehiculo(c1, c2, v1, v2);
                        insertar(clusterAInsertar, vehiculoConMasEspacio(v1, v2);
```

En el algoritmo 6 puede verse el proceso GRASP implementado para esta fase con algunas diferencias respecto a la estructura clásica. Por un lado en la función *seleccionarAlgoritmoGoloso*, el algoritmo que genera la solución golosa inicial es elegido entre un conjunto de algoritmos distintos, que se describirán más adelante. Por otro lado, se verifica si la solución golosa es completa (todos los clusters tienen un vehículo asignado). Si esto último no pasa, se ejecuta el algoritmo 7, que busca reparar el problema. Este último proceso intenta para cada cluster que quedó sin asignación, generar el espacio (o capacidad) suficiente en algún vehículo, y así lograr atender su demanda. El algoritmo intercambia todos los cluster de los distintos vehículos entre sí, en busca de generar la capacidad suficiente en algún vehículo para lograr satisfacer la demanda del cluster no asignado.

5.2.3. Algoritmos Golosos Aleatorios de Nivel Cluster

Con el fin de aumentar la diversidad de las soluciones golosas iniciales se implementaron cinco algoritmos distintos para dicha tarea. La idea detrás de esto es aumentar la cantidad de soluciones golosas iniciales. Generar una solución válida puede ser complicado en algunas instancias por el problema (ya mencionado) de ajuste entre la demanda de los clusters y la capacidad de los vehículos. Ademas, tener mayor diversidad de soluciones iniciales aumenta la posibilidad de encontrar una solución final de calidad. Los algoritmos implementados para lograr este objetivo fueron los siguientes:

- *SimpleRCL* (Algoritmo 8)
  Consiste completar cada vehículo de a uno por vez con los clusters a visitar. El criterio en la lista RCL es la distancia al último cluster visitado.

- *DobleRCL* (Algoritmo 10)
  Crea una lista RCL de clusters candidatos por vehículo. El criterio de la RCL es la distancia al último cluster que el vehículo visitó. Elige de forma aleatoria un candidato por vehículo. Luego crea otra lista RCL con dichos vehículos también usando la distancia como criterio y elige aleatoriamente el vehículo de esa lista. Ese procedimiento se repite tantas veces como cantidad de clusters.

- *DemandaRCL* (Algoritmo 12)
  Este algoritmo intenta balancear el problema de minimizar la distancia recorrida, con el problema de ajustar la demanda de los clusters a los vehículos. Esto lo hace a través de dos listas RCL, una para cada criterio.

- *MejorAjuste* (Algoritmo 15)
  Este algoritmo es el más eficiente respecto al ajuste entre la demanda de los clusters y la capacidad de los vehículos. El mismo no toma en cuanta la distancia recorrida en el camino de clusters al momento de crear la solución golosa. Fue implementado para resolver instancias donde el problema de la asignación de clusters a vehículos es muy compleja en términos de demanda y capacidad. El algoritmo utilizado para esto es el conocido como *BestFit* [29].

A continuación se presentan los pseudo-cógidos de los algoritmos mencionados.

SimpleRCL

Algoritmo 8: SimpleRCL(vehiculos, clustersPorVisitar, α)

```
for v ∈ Vehiculos do
    for c ∈ Cluster do
        RCL = RCLClusters(v, clustersPorVisitar, α);
        if RCL no es Vacia then
            siguienteCluster = seleccionarDeFormaAleatoriaItem(RCL);
            v.agregarALaRuta(siguienteCluster);
            v.actualizarEspacioDisponible();
            clustersPorVisitar.quitar(siguienteCluster);
```

Algoritmo 9: RCLClusters(vehículo, clustersPorVisitar, *alpha*)

```
RCL = ∅;
ultimoCluster = ultimoClusterVisitado(vehículo);
minDistancia = maximaDistancia(ultimoCluster, clustersPorVisitar);
maxDistancia = minimaDistancia(ultimoCluster, clustersPorVisitar);
condicionRCL = minDistancia + α * (maxDistancia - minDistancia);
for cluster ∈ clustersPorVisitar do
    distancia = distanciaEntreClusters(ultimoCluster, cluster);
    if vehículo.puedeAtenderDemanda(cluster) ∧ distancia ≤ condicionRCL then
        RCL.Agregar(cluster);
return RCL;
```

Como se puede ver en 8, este es un algoritmo bastante simple. Utiliza la distancia entre el último cluster visitado por un vehículo y todos los que quedan por visitar, como criterio para armar la RCL (siempre que el vehículo tenga espacio para la demanda cluster). Luego se selecciona un cluster de forma aleatoria y se agrega a la ruta del vehículo.

DobleRCL

Algoritmo 10: DobleRCL(vehiculos, clustersPorVisitar, α)

```
RCLDeVehiculos = ∅;
siguienteCluster = ∅;
for c ∈ clustersPorVisitar do
    for v ∈ vehiculos do
        RCLDeClusters[v] = RCLClusters(v, clustersPorVisitar, α);
        if RCLDeClusters[v] NO es vacio then
            siguienteCluster[v] = seleccionAleatoria(RCLDeClusters[v]);
    RCLDeVehiculos = RCLVehiculos(vehiculos, siguienteCluster, α);
    siguienteVehiculo = seleccionAleatoria(RCLDeVehiculos);
    clusterAVisitar = siguienteCluster[siguienteVehiculo];
    siguienteVehiculo.agregarALaRuta(clusterAVisitar);
    siguienteVehiculo.actualizarEspacioDisponible();
    clustersPorVisitar.quitar(clusterAVisitar);
```

Algoritmo 11: RCLVehiculos(vehiculos, siguienteCluster, α)

```
minDistancia = +∞;
maxDistancia = −∞;
for v in vehiculos do
    if siguienteCluster[v] es Vacio then
        saltarIteracion;
    ultimoCluster = v.ultimoClusterVisitado();
    clusterAVisitar = siguienteCluster[v];
    distancia = distanciaEntreClusters(ultimoCluster, clusterAVisitar);
    if distancia < minDistancia then
        minDistancia = distancia;
    if maxDistancia < distancia then
        maxDistancia = distancia;
    criterioRCL = minDistancia + α * (maxDistancia - minDistancia);
for v in vehiculos do
    if siguienteCluster[v] es Vacio then
        saltarIteracion;
    ultimoCluster = v.ultimoClusterVisitado();
    clusterAVisitar = siguienteCluster[v];
    distancia = distanciaEntreClusters(ultimoCluster, clusterAVisitar);
    if distancia <= condicionRCL then
        RCL.Agregar(v);
return RCL;
```

En el algoritmo 10 puede verse que primero para todo vehículo se genera una RCL de clusters candidatos a través del algoritmo 9, y luego de esa lista (cada vehículo tiene la propia) se selecciona de forma aleatoria el posible siguiente cluster a visitar para cada vehículo. En el siguiente paso se genera una RCL de vehículos a través del algoritmo 11 de la cual se selecciona un vehículo de forma aleatoria, y es a este vehículo elegido al que

se le agrega el cluster que se selecciono como candidato (en el paso anterior) a su ruta.

DemandaRCL

Algoritmo 12: DemandaRCL(vehiculos, clustersAVisitar, α_1, α_2)

```
for c ∈ clustersAVisitar do
    RCLClustersPorDemanda = RCLPorDemanda(clustersAVisitar, α1);
    siguienteCluster = seleccionAleatoria(RCLClustersPorDemanda);
    RCLVehiculos = RCLVehiculoPorDistancia(vehiculos, siguienteCluster, α2);
    if RCLVehiculos NO es Vacio then
        vehiculo = seleccionAleatoria(RCLVehiculos);
        vehiculo.agregarALaRuta(siguienteCluster);
        vehiculo.actualizarEspacioDisponible();
        clusterAVisitar.quitar(siguienteCluster);
```

Algoritmo 13: RCLPorDemanda(clustersAVisitar, α_1)

```
minDemanda = menorDemandaDeTodosLosClusters(clustersAVisitar)
maxDemanda = mayorDemandaDeTodosLosClusters(clustersAVisitar);
condicionRCL = minDemanda + α1 * (maxDemanda - minDemanda);
for c ∈ clustersPorVisitar do
    if c.demanda() <= condicionRCL then
        RCL.agregar(c);
return RCL;
```

Algoritmo 14: RCLVehiculoPorDistancia(vehiculos, clusterElegido, α_2)

```
minDistancia = minimaDistancia(vehiculos, clusterElegido);
maxDistancia = maximaDistancia(vehiculos, clusterElegido);
condicionRCL = minDistancia + α2 * (maxDistancia - minDistancia);
for v ∈ Vehiculos do
    distancia = distancia(v.ultimoClusterVisitado(), clusterElegido);
    if distancia ≤ condicionRCL ∧ v.soportaDemanda(clusterElegido) then
        RCL.agregar(v);
if RCL es Vacia then
    for v ∈ Vehiculos do
        if v.soportaDemanda(clusterElegido) then
            RCL.agregar(v);
return RCL;
```

En el algoritmo 12 podemos ver que en este caso, primero se arma una RCL de clusters (a través del algoritmo 13) utilizando como criterio la demanda de los mismos. Luego se selecciona un cluster de la RCL de forma aleatoria. Con el cluster ya elegido, el algoritmo 14 genera otra RCL, esta vez de vehículos, utilizando como criterio la distancia a dicho cluster. Finalmente se selecciona de forma aleatoria un vehículo de la RCL y se le agrega

el cluster anteriormente elegido a su ruta.

MejorAjuste

Algoritmo 15: MejorAjuste(vehiculos, clustersAVisitar)

```
vehiculosUsados = 0;
if NO es la primera Ejecución then
    clustersAVisitar = mezclaAleatoria(clustersAVisitar);
else
    clustersAVisitar = ordenarClusterPorDemanda(clustersAVisitar);
for c ∈ clusterAVisitar do
    minimaCapacidad = capacidadDeVehiculos + 1;
    mejorIndice = 0;
    for i ∈ rango(vehiculosUsados) do
        if vehiculos[i].soportaDemanda(c) ∧ vehiculos[i].espacioDisponible() −
         c.demanda() < minimaCapacidad then
            mejorIndice = i;
            minimaCapacidad = vehiculos[i].espacioDisponible() - c.demanda();
    if minimaCapacidad = capacidadDeVehiculos + 1 then
        if vehiculosUsados + 1 < totalVehiculos then
            vehiculosUsados++;
            vehiculos[vehiculosUsados].agregarALaRuta(c);
            vehiculos[vehiculosUsados].actualizarCapacidad();
            clusterAVisitar.quitar(c);
    else
        vehiculos[mejorIndice].agregarALaRuta(c);
        vehiculos[mejorIndice].actualizarCapacidad();
        clusterAVisitar.quitar(c));
```

En el algoritmo 15 podemos ver el clásico algoritmo *BestFit* utilizado en el contexto de este problema. En el mismo se intenta asignar clusters a vehículos de forma tal que la demanda de los clusters se ajuste lo mejor posible a la capacidad de la flota de vehículos. Esto puede ser muy útil en instancias dónde hay pocas combinaciones que resulten válidas respecto al cumplimiento de la demanda. Es importante ver que en esta asignación el problema de satisfacer correctamente la demanda determinará la ruta de clusters, sin importar la distancia. La condición respecto a ser o no la primera ejecución se utiliza para evitar generar siempre el mismo orden de clusters, ya que esto generaría siempre la misma solución inicial, lo que aumentaría la probabilidad de generar soluciones finales repetidas.

5.2.4. Búsquedas Locales de Nivel Cluster

Como describe el algoritmo 6, luego de construir una solución golosa a través de alguno de los algoritmos propuestos, se ejecutan una serie de algoritmos de búsqueda local con el objetivo de mejorar la solución obtenida.
El esquema de ejecución de búsquedas locales de Nivel Cluster es el siguiente:

Algoritmo 16: BusquedasLocales(solución)

```
for iterador ≤ iteracionesBusquedaLocal do
    if Ejecución en orden aleatorio Activada then
        ordenDeEjecución = mezclaAleatoria(ordenDeEjecuccion);
    for busquedaLocal in ordenDeEjecución do
        if busquedaLocal = IntercambioDeVehiculo then
            intercambioDeVehiculo(solución);
        if busquedaLocal = InsertarClusterEnVehiculo then
            insertarClusterEnVehiculo(solución);
        if busquedaLocal = 2-Opt then
            2-Opt(solución);
        if busquedaLocal = Relocalizar then
            relocalizar(solución);
        if busquedaLocal = Intercambiar then
            intercambiar(solución);
        if busquedaLocal = IntercambioAleatorio then
            IntercambioAleatorio(solución);
    iterador++;
```

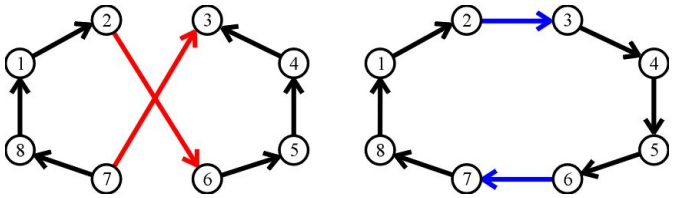
Cómo se puede observar en el algoritmo 16, se itera una cantidad fija de veces ejecutando en cada iteración todas las heurísticas de búsqueda local. El orden en que se ejecutan puede estar previamente configurado, o puede ser aleatorio en cada ejecución.

La mayoría de las búsquedas locales utilizadas en el algoritmo 16 son técnicas bastante sencillas y muy estudiadas por la literatura. En esta implementación todas ellas iteran una cantidad fija de veces en búsqueda de mejorar la solución, y si esto sucede, se comienza a iterar desde cero nuevamente.
A continuación se realizará una breve explicación de cada una:

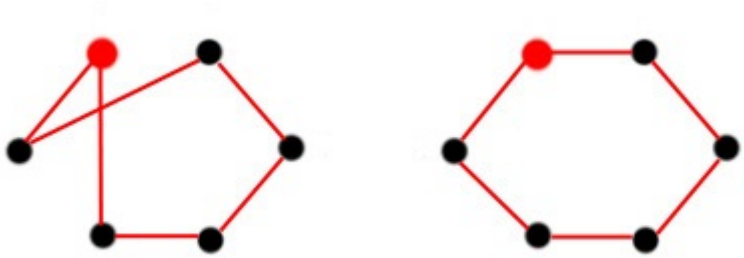
- *IntercambioDeVehículo*
  Esta heurística intenta de intercambiar clusters entre los vehículos, tratando de mejorar las distancias recorridas. Toma un vehículo e intenta intercambiar cada uno de sus clusters con cada uno de los clusters de otro vehículo. Esto se realiza para todos los vehículos. Para cada posible intercambio de clusters se verifica si la distancia mejoró y si el intercambio en términos de capacidad de los vehículos y demanda de los clusters es posible. Si esto es así, se produce el intercambio de los clusters.

- *InsertarClusterEnVehículo*
  Esta heurística toma un vehículo y trata de insertar cada cluster asignado al mismo (de a uno) en todas las posiciones de la ruta de otro vehículo. Esto se hace para todos los vehículos. Para que la inserción sea posible no solo debe mejorar la distancia total, si no que también, el vehículo en el cual se inserta el cluster debe soportar la nueva demanda. Si la inserción es posible, el cluster se elimina de la ruta del vehículo original, y se asigna a la ruta del vehículo destino.

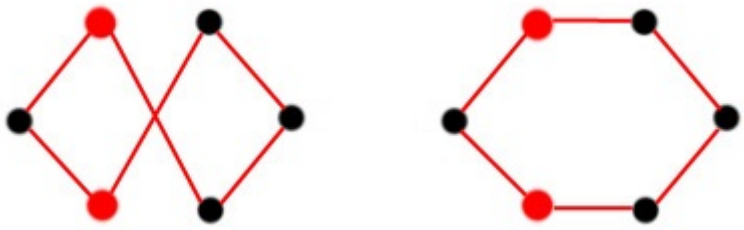
- *2-Opt*
  Es una de las técnicas más utilizadas en los problemas de ruteo de vehículos. La idea de este técnica es bastante simple, consiste en tomar una ruta que cruza sobre sí misma y re ordenarla para que este no suceda.



- *Relocalizar*
  Esta heurística toma un cluster y lo mueve dentro del camino, una a una por todas las posibles posiciones del mismo, buscando mejorar la distancia total.



- *Intercambio*
  Este algoritmo toma dos clusters de un camino e intercambian sus posiciones. Esto lo hace para todo par de clusters, tratando de mejorar la distancia total.



- *IntercambioAleatorio*
  Este algoritmo mantiene la misma lógica que el anterior, pero la diferencia radica en que el orden en que se toma el par de clusters es aleatorio dentro del camino, generando combinaciones que en el anterior (que sigue el orden del primer camino armado) podría no contemplar.

Como resultado del procedimiento GRASP de Nivel Cluster, se genera un camino de clusters para una flota de vehículos, donde la demanda de los clusters es servida. Con este resultado, solo resta para el problema *CluVRP SCC*, determinar el orden en que un vehículo visitará los clientes dentro de un cluster, y para el problema *CluVRP WCC*, determinar la forma en que visitará los clientes de todos sus clusters asignados, sin importar el orden en que los visite. Ambas problemáticas se explicarán en la siguiente sección.

5.3. Fase de Nivel Cliente

5.3.1. Introducción

En está sección abordaremos el trabajo realizado para construir una solución final tanto para el problema CluVRP SCC, cómo para el problema CluVRP WCC a partir de una ruta de clusters resultado de la fase de Nivel Cluster explicada en la sección anterior.

La solución de Nivel Cluster no es más que un conjunto de rutas para una flota de vehículos. Cada vehículo posee una lista ordenada de clusters que debe visitar, partiendo del cluster *Deposito*, y finalizando en el mismo.
Con el recorrido de clusters para cada vehículo ya calculado, en esta nueva fase llamada Fase de Nivel Cliente, se realizará el siguiente procedimiento según el problema a resolver:

- **CluVRP SCC:**
  Para cada cluster que visita un vehículo, se armará un camino de clientes, buscando minimizar la distancia recorrida. Vale remarcar que un punto importante de ese cálculo es el hecho de que los distintos órdenes posibles en que se recorren los clientes en un cluster no solo hacen variar la distancia recorrida dentro del cluster, si no que también afectan a la totalidad del recorrido de una forma particular. Esto se debe a que la entrada y la salida de un cluster, es la conexión entre dos clientes de distintos clusters, y la variación en el orden en que se visitan los clientes de un cluster puede afectar la distancia que el vehículo recorre entre el último cliente del cluster y el primer cliente del siguiente cluster.

- **CluVRP WCC:**
  Para cada vehículo se arma un camino con todos los clientes que pertenecen a los clusters que el vehículo debe servir, dónde no hay restricciones en el orden en que se visitan los clientes del conjunto de clusters. Esto último significa que el vehículo puede visitar un cliente de un cluster, salir del mismo para visitar un cliente de otro cluster, y luego volver al primero para visitar a otro cliente.

En esta fase también se implementó la técnica GRASP para solucionar el problema planteado, existiendo variaciones entre las versiones de CluVRP SCC y de CluVRP WCC.

5.3.2. GRASP de Nivel Cliente

A continuación, el esquema del algoritmo GRASP implementado para la fase de Nivel Cliente para los problemas CluVRP SCC y CluVRP WCC.

Algoritmo 17: GRASPNivelCliente(solucionNivelCluster)

```
iterador = 0;
while iterador < iteracionesTotales do
    solución = generarSolucionGolosa(solucionNivelCluster);
    BusquedaLocal(solución);
    ActualizarSolucion(solución, mejorSolucion);
    iterator = iterador + 1;
retornar(MejorSolucion)
```

Cómo se puede ver en 17, el algoritmo muestra el esquema básico del procedimiento GRASP. El esquema general para CluVRP SCC y CluVRP WCC es el mismo, pero variando en su implementación, ya que la solución golosa que se genera para cada problema es distinta, así como también existen diferencias en las implementaciones de los algoritmos de búsqueda local. Esto se debe a que las restricciones de ambos problemas son diferentes. En la siguiente sección se describirán los detalles de implementación para CluVRP SCC y para CluVRP WCC.

5.3.3. Algoritmo Goloso Aleatorio de Nivel Cliente para CluVRP SCC

A continuación se detallará el algoritmo goloso aleatorio utilizado para generar a partir de una solución de Nivel Cluster, una solución golosa aleatoria del problema CluVRP SCC.

Algoritmo 18: AlgoritmoGolosoCluVRP-WCC(soluciónANivelCluster, α)

```
solucion = ∅;
clienteActual = Deposito;
forall vehiculo ∈ solucionANivelCluster do
    forall cluster ∈ vehiculo.rutaDeClusters do
        clientesAVisitar = clientes(cluster);
        siguienteCluster = siguiente(vehiculo.rutaDeClusters, cluster);
        < clienteFinal, clienteInicialSigCluster > = clientesMasCercanos(cluster,
          siguienteCluster);
        solucion.agregar(clienteActual);
        clientesPorVisitar.Quitar(clienteFinal);
        while clientesPorVisitar NO es Vacia do
            RCL = clientesRCL(clienteActual, clientesPorVisitar, α);
            siguienteCliente = seleccionarDeFormaAleatoriaItem(RCL);
            solucion[vehiculo].agregar(siguienteCliente);
            clientesPorVisitar.quitar(siguienteCliente);
            clienteActual = siguienteCliente;
        solucion[vehiculo].agregar(clienteFinal);
        clienteInicial = clienteInicialSigCluster;
retornar(solucion);
```

Algoritmo 19: clientesRCL(clienteActual, clientesPorVisitar, α)

```
minDistancia = minimaDistancia(clienteActual, clientesPorVisitar);
maxDistancia = maximaDistancia(clienteActual, clientesPorVisitar);
condicionRCL = minDistancia + α * (maxDistancia - minDistancia);
for siguienteCliente ∈ clientesPorVisitar do
    distancia = distanciaEntreClientes(clienteActual, siguienteCliente);
    if distancia <= condicionRCL then
        RCL.agregar(v);
return RCL;
```

El algoritmo 18 es bastante simple, comienza definiendo al depósito como primer cliente ya que el recorrido siempre comienza en ese nodo. Luego determina cual será el cliente de salida del cluster actual y el cliente de entrada al siguiente cluster. Esto lo hace seleccionando el par de clientes que estén a menor distancia entre el cluster actual y el que le sigue. El cliente de salida se quita del conjunto de clientes a visitar. Luego iterando sobre el conjunto de clientes a visitar, va armando un camino utilizando en cada iteración una RCL con un parámetro α a través del algoritmo 19. Al quedar el conjunto de clientes a visitar vacío para el cluster, se inserta al final del camino al cliente de salida del cluster. Luego se redefine como cliente inicial al cliente de entrada del siguiente cluster (ya elegido), y continua con el siguiente cluster para repetir la operación.

5.3.4. Algoritmo Goloso Aleatorio de Nivel Cliente para CluVRP WCC

A continuación se detallará el algoritmo goloso aleatorio utilizado para generar a partir de una solución de Nivel Cluster, una solución golosa aleatoria del problema CluVRP WCC.

Algoritmo 20: AlgoritmoGolosoCluVRP-WCC(soluciónANivelCluster, α)

```
solución = ∅;
forall vehiculo ∈ solucionANivelCluster do
    clientesAVisitar = ∅;
    forall cluster ∈ vehiculo.rutaDeClusters do
        clientesAVisitar.agregar(clientes(cluster));
    solucion[vehículo].agregar(Deposito);
    clienteActual = Deposito;
    while clientesAVisitar No es Vacio do
        RCL = clientesRCL(clientaActual, clientesPorVisitar, α);
        siguienteCliente = seleccionAleatoria(RCL);
        solucion[vehículo].agregar(siguienteCliente);
        clientesPorVisitar.Quitar(siguienteCliente);
        clienteaActual = siguienteCliente;
    solución[vehículo].agregar(Deposito);
retornar(solución);
```

En el algoritmo 20 podemos observar que el procedimiento es similar al del algoritmo

goloso aleatorio de CluVRP SCC. La diferencia radica en que no es una restricción que los clientes de los clusters sean visitados en orden dentro del mismo, por esto, el algoritmo agrupa todos los clientes de todos los clusters asignados a un vehículo y arma un recorrido de los mismos a través de un criterio goloso de selección, donde en cada iteración se crea un RCL, de la cual se selecciona de forma aleatoria el siguiente cliente a visitar. El algoritmo que crea la RCL es el mismo que para CluVRP SCC.

5.3.5. Búsquedas Locales de Nivel Cliente

Los heurísticas de búsqueda local implementadas para el Nivel Cliente tienen como misión mejorar la distancia recorrida en el conjunto caminos de clientes que comienzan en un depósito y terminan en el mismo. Tanto para CluVRP SCC y CluVRP WCC, el problema a tratar por estas heurísticas es prácticamente el mismo. En CluVRP SCC el camino de clientes es dentro de cada cluster, para todos los clusters asignados a un vehículo por separado. En CluVRP SCC el camino de clientes está constituido por los clientes de todos los clusters que el vehículo tiene asignado. Visto con esta perspectiva, la diferencia solo radica en la cantidad de caminos y en el tamaño de los mismos.

Hasta aquí no seria necesario tener diferentes implementaciones entre un problema y el otro. La diferencia surge cuando en un camino de clientes dentro de un cluster de CluVRP SCC se realiza algún movimiento que genera el cambio de posición de un cliente frontera (cliente del cluster actual que se esta procesando con el cliente del siguiente cluster). Cuando esto pasa, la distancia no solo cambia a nivel del camino dentro del cluster, si no que cambia la distancia total recorrida por el vehículo. Esta variación no es válida dentro de los caminos para CluVRP WCC, ya que los mismos están formados por todos los clientes de todos los clusters de un vehículo, sin distinciones.

Es por ese motivo que existen ligeras diferencias en las implementaciones algorítmicas de las búsquedas locales que no serán reflejadas aquí, ya que no afectan la idea general de cada una de las heurísticas usadas. A continuación se presentan las heurísticas utilizadas en esta fase.

- *2-Opt*
- *Relocalizar*
- *Intercambio*
- *IntercambioAleatorio*

No se explicará el funcionamiento detallado de estas heurísticas ya que la idea general de las mismas fue detallada en la Fase de Nivel Cluster (sección 5.2.4). La diferencia radica que en vez trabajar con caminos de clusters, aquí se trabaja con caminos de clientes.

6. RESULTADOS COMPUTACIONALES

6.1. Introducción

En este capítulo se presentarán los resultados obtenidos aplicando el algoritmo propuesto en diversos grupos de instancias. También se analizará su comportamiento variando los valores de sus parámetros en busca de resultados competitivos.

Los resultados obtenidos serán comparados con los publicados en los distintos trabajos según el set de instancias que se esté evaluando. El problema CluVRP SCC cuenta con resultados para todas las instancias pero CluVRP WCC no, por lo que en los casos en que este dato no esté presente para CluVRP WCC, se realizará una comparación con el mejor resultado conocido de CluVRP SCC para la misma instancia.
A lo largo del capítulo se realizará un análisis del algoritmo variando los valores de sus parámetros de la siguiente forma: cada vez que un parámetro con un cierto valor mejore los resultados, este se fijará, para luego seguir estudiando el resto de los parámetros con ese valor ya fijado. Este método no asegura la parametrización óptima, pero es la forma elegida para enfrentar el problema de encontrar una buena parametrización.

El análisis se realizará sobre un grupo de instancias reducido, las cuales tratarán de ser representativas del set al que pertenecen. Al final del capítulo se presentarán los resultados de la ejecución del algoritmo para todas las instancias de todos los sets disponibles.

6.2. Instancias utilizadas

Las instancias para CluVRP utilizadas en este trabajo alcanzan un total de 245. Las mismas provienen de distintos trabajos, y se separan en tres grandes grupos. A continuación se detallará su origen y sus características:

- **GVRP03**
  Este set está constituido por 79 instancias de tamaño medio, las cuales se separan en cuatro grupos: A, B, G/M y P. Las mismas fueron creadas en el trabajo de Battarra et al. [9]. Las instancias son una adaptación de instancias del problema CVRP utilizadas en Bektas et al. [30]. De aquí en adelante se hará referencia a este set de instancias con el nombre de *GVRP03*.

- **Golden adaptadas por Battarra**
  Este set está conformado por instancias con grandes clusters. Son el conjunto de las 20 instancias con más nodos propuestas en Golden et al. [31] para el problema CVRP. Las mismas van desde 240 a 483 nodos. Fueron creadas también en el trabajo de Battarra et al. [9] con el mismo algoritmo que generó el set GVRP03. Por cada instancia se generaron 11 nuevas, variando la cantidad de clusters y su composición. El resultado es un total de 220 instancias. La cantidad de clusters de las mismas va de 16 a 95, y la cantidad de nodos en los clusters llega a un máximo de 71.
  En este trabajo solo se utilizarán 6 de los 20 conjuntos totales: Golden_1, Golden_5,

Golden_8, Golden_13, Golden_16 y Golden_20. Dado que cada conjunto posee 11 instancias, el total de instancias utilizadas de este set será 66. De aquí en adelante se hará referencia a este set de instancias con el nombre de *Golden-Battarra*.

- **Golden adaptadas por Expósito-Izquierdo**
  Generadas en el trabajo de Expósito-Izquierdo et al. [19], también son adaptaciones de las instancias de CVRP propuestas en Golden et al. [31]. Este set está compuesto por cinco conjuntos de instancias. Cada conjunto es caracterizado por un parámetro ρ representando el rango de *fill* (o llenado) de un vehículo. Cuando $\rho = 100\,\%$, cada vehículo puede servir a lo sumo un cluster, en tanto un bajo porcentaje de llenado indica que un número alto de clusters pueden combinarse en un viaje de un vehículo. Los cinco diferentes conjuntos de instancias se crearon parametrizando $\rho \in \{10, 25, 50, 75, 100\}\,\%$, generado un total de 100 instancias (20 instancias por conjunto). De aquí en adelante se hará referencia a este set de instancias con el nombre de *Golden-Izquierdo*.

Diferencias entre las instancias del problema

Las instancias Golden-Izquierdo no poseen información acerca del número de vehículos a utilizar (a diferencia de GVRP03 y Golden-Battarra), por lo que en la solución se pueden utilizar todos los vehículos que se crean necesarios. En la implementación de este trabajo se decidió definir ese número cómo la cantidad de clusters que la instancia posee. El algoritmo utilizará la cantidad de vehículos que requiera, dejando vacíos (sin clusters) aquellos que no se utilizan.
Por el contrario para los sets de instancias GVRP03 y Golden-Battarra, el número de vehículos es fijo (es dato propio de la instancia) y como condición, ningún vehículo puede quedar vacío. Es por esto que cuando el algoritmo GRASP de Nivel Cluster (algoritmo 6) genera una solución inicial con algún vehículo vacío, se ejecuta un algoritmo que corrige este problema, asignando al vehículo vacío un cluster que minimice el aumento de la distancia total producida por el cambio.

6.3. Parámetros del algoritmo

Como resultado del capítulo 5 se implementó un algoritmo para resolver los problemas CluVRP SCC y CluVRP WCC. El mismo cuenta con un total de 21 parámetros que se detallarán a continuación:

Parámetros Generales

- **Iteraciones Totales**
  Define el número de iteraciones principales que realiza el algoritmo en busca de mejorar la solución del problema.

- **Iteraciones del ciclo de Búsquedas Locales**
  Define el número de veces que se repite el bucle que controla la ejecución de las búsquedas locales tanto de Nivel Cluster como de Nivel Cliente.

- **Iteraciones de heurística Intercambio de Vehículos Fase Final**
  Define el número de iteraciones sin mejora de la búsqueda Intercambio de Vehículos aplicada a la solución final construida.

Parámetros de Nivel Cluster

- **Algoritmo goloso aleatorio**
  Define el algoritmo goloso aleatorio utilizado para construir la solución inicial de Nivel Cluster. Los posibles algoritmos son: SimpleRCL, DobleRCL, DemandaRCL, MejorAjuste y Aleatorio. Este último en cada iteración hace una selección aleatoria de alguno de los anteriores.

- **Método de distancia cluster**
  Define el método para calcular la distancia entre clusters. Los valores posibles pueden ser *Centroide* o *Menor distancia entre clientes*.

- $\alpha - demanda$
  Define el valor del parámetro $\alpha - demanda$ para los algoritmos golosos de Nivel Cluster que lo utilizan. Es un número decimal entre 0 y 1.

- $\alpha - distancia$ **Nivel Cluster**
  Define el valor del parámetro $\alpha - distancia$ para los algoritmos golosos de Nivel Cluster que lo utilizan. Es un número decimal entre 0 y 1.

- **Cantidad de iteraciones de heurística IntercambioDeVehículo Nivel Cluster**
  Define la cantidad de iteraciones sin mejora de la heurística IntercambioDeVehículo de Nivel Cluster.

- **Cantidad de iteraciones de heurística InsertarClusterEnVehículo Nivel Cluster**
  Define la cantidad de iteraciones sin mejora de la heurística InsertarClusterEnVehículo de Nivel Cluster.

- **Cantidad de iteraciones de heurística 2-Opt Nivel Cluster**
  Define la cantidad de iteraciones sin mejora de la heurística 2-Opt de Nivel Cluster.

- **Cantidad de iteraciones de heurística Relocalizar Nivel Cluster**
  Define la cantidad de iteraciones sin mejora de la heurística Relocalizar de Nivel Cluster.

- **Cantidad de iteraciones de heurística Intercambio Nivel Cluster**
  Define la cantidad de iteraciones sin mejora de la heurística Intercambio de Nivel

Cluster.

- **Cantidad de iteraciones de heurística IntercambioAleatorio Nivel Cluster**
  Define la cantidad de iteraciones sin mejora de la heurística IntercambioAleatorio de Nivel Cluster.

- **Orden de ejecución de Búsquedas Locales Nivel Cluster**
  Define el orden en que las heurísticas de búsqueda local antes mencionadas se ejecutarán. Se puede parametrizar como aleatorio.

Parámetros de Nivel Cliente para CluVRP SCC y WCC

- $\alpha - distancia$ **Nivel Cliente**
  Define el valor del parámetro $\alpha - distancia$ para el algoritmo goloso de Nivel Cliente. Es un número decimal entre 0 y 1.

- **Cantidad de iteraciones de heurística 2-Opt Nivel Cliente**
  Define la cantidad de iteraciones sin mejora de la heurística 2-Opt de Nivel Cliente.

- **Cantidad de iteraciones de heurística Relocalizar Nivel Cliente**
  Define la cantidad de iteraciones sin mejora de la heurística Relocalizar de Nivel Cliente.

- **Cantidad de iteraciones de heurística Intercambio Nivel Cliente**
  Define la cantidad de iteraciones sin mejora de la heurística Intercambio de Nivel Cliente.

- **Cantidad de iteraciones de heurística IntercambioAleatorio Nivel Cliente**
  Define la cantidad de iteraciones sin mejora de la heurística IntercambioAleatorio de Nivel Cliente.

- **Orden de ejecución de Búsquedas Locales Nivel Cliente**
  Define el orden en que las heurísticas de búsqueda local antes mencionadas se ejecutarán. Se puede parametrizar como aleatorio.

6.4. Experimentos

En esta sección se realizarán una serie de experimentos para determinar los valores de los parámetros del algoritmo propuesto con el fin de encontrar resultados competitivos para el conjunto de instancias detallado en 6.2.
El problema de encontrar valores para los parámetros del algoritmo en busca de *la mejor parametrización* es un problema complejo. La cantidad de posibles configuraciones es de

orden exponencial respecto la cantidad de parámetros y los valores que los mismos pueden tomar. Es por esto que no puede asegurarse que la parametrización final lograda sea la mejor, pero a través del análisis propuesto se tratará de cotejar que sus resultados sean competitivos respecto a otros trabajos.

El algoritmo se implementó en el lenguaje C# utilizando la versión 4.5 del framework .NET. El equipo utilizado para correr todos los experimentos y pruebas finales fue un Intel Core i5-3337U de 1.80GHz, con 8 Gigas de memoria RAM.

El método propuesto en esta sección consta de estudiar el comportamiento del algoritmo al variar el valor de uno o más parámetros sobre un set acotado de instancias. Dependiendo del resultado se tomará una decisión sobre ese conjunto de parámetros, para luego seguir con el proceso a partir de lo concluido. Las pruebas se correrán tanto para CluVRP SCC, como para CluVRP WCC de forma independiente.

El set de instancias utilizado será acotado y representativo de todos los sets estudiados en este trabajo. Se mantendrá fijo durante todos los experimentos para asegurar que el proceso siempre se realiza en las mismas condiciones.
Los resultados obtenidos en los experimentos serán comparados con los publicados en el trabajo de Defryn et al. [20].

Al final de la experimentación se verificará la performance del algoritmo con los valores hallados para sus parámetros sobre los sets de instancias completos. También se realizarán algunos experimentos, pero esta vez sobre los conjuntos completos de instancias.

6.4.1. Conjunto de instancias seleccionadas para los experimentos

Los experimentos se realizarán en un set acotado de instancias debido a que la ejecución de estas pruebas en las 245 instancias disponibles tomaría mucho tiempo.
Las instancias seleccionadas serán un total de 14, dos de cada subconjunto del set GVRP03 (A, B, G/M, P), dos instancias de los 6 sets de Golden-Battarra elegidos, y cuatro instancias del set Golden-Izquierdo.
Para determinar cuales serán las elegidas, se ejecutará el algoritmo una vez para todas las instancias. Luego se tomará de cada conjunto mencionado una instancia al azar del conjunto formado por aquellas donde se obtuvo la menor distancia a la mejor solución conocida, y una instancia al azar del conjunto formado por aquellas donde se obtuvo la mayor distancia a la mejor solución conocida. Estas comparaciones se realizarán con los resultados publicados en el trabajo de Defryn et. al. [20].

A continuación se muestran los parámetros utilizados en la ejecución del algoritmo y luego las instancias resultantes de la selección según el criterio descrito.

	Parámetro	Valor
Generales	*Iteraciones*	100
	Iteraciones ciclo Búsquedas Locales	1
	Iteraciones IntercambioDeVehículos	50
	Iteraciones IntercambioDeClusters	50
Nivel Cluster	*Algoritmo Goloso Aleatorio*	SimpleRCL
	Método de distancia cluster	Distancia entre clientes
	$\alpha - demanda$	0.5
	$\alpha - distancia$	0.5
	Iteraciones IntercambioDeVehículos	50
	Iteraciones InsertarEnVehículo	50
	Iteraciones 2-Opt	50
	Iteraciones Relocalizar	50
	Iteraciones Intercambio	50
	Iteraciones IntercambioAleatorio	50
	Orden de ejección de Búsquedas Locales	Aleatorio
Nivel Cliente	$\alpha - distancia$	0.5
	Iteraciones 2-Opt	50
	Iteraciones Relocalizar	50
	Iteraciones Intercambio	50
	Iteraciones IntercambioAleatorio	50
	Orden de ejecución de Búsquedas Locales	Aleatorio

Tab. 6.1: Parámetros del algoritmo para determinar instancias a utilizar en los experimentos

	Set	Nodos	Clusters
A-n36-k5-C12-V2	GVRP03	36	12
B-n57-k7-C19-V3	GVRP03	57	19
M-n101-k10-C34-V4	GVRP03	101	34
P-n23-k8-C8-V3	GVRP03	23	8
Golden_13-C51-N253	Golden-Battarra	453	51
Golden_rho10_10	Goden-Izquierdo	324	57
Golden_rho25_11	Goden-Izquierdo	400	64

Tab. 6.2: Instancias seleccionadas con mejores resultados para los experimentos. Parámetros utilizados en tabla 6.1.

	Set	Nodos	Clusters
A-n61-k9-C21-V4	GVRP03	61	21
B-n52-k7-C18-V3	GVRP03	52	18
G-n262-k25-C88-V9	GVRP03	262	88
P-n101-k4-C34-V2	GVRP03	100	34
Golden_20-C85-N421	Golden-Battarra	421	85
Golden_rho10_20	Goden-Izquierdo	421	253
Golden_rho25_18	Goden-Izquierdo	301	129

Tab. 6.3: Instancias seleccionadas con peores resultados para los experimentos. Parámetros utilizados en 6.1.

6.4.2. Experimento I: Método de cálculo de distancia entre clusters

En este experimento se evaluará el comportamiento del algoritmo con la configuración básica propuesta en 6.1 variando el parámetro que define el método de calculo de distancia utilizado por los algoritmos golosos aleatorios de Nivel Cluster.

Parámetro a evaluar:

	Parámetro	Valores a evaluar
Nivel Cluster	*Método de distancia cluster*	{Distancia mínima entre los clientes, Centroide}

	Método Distancia entre Clientes		Método Centroide	
Instancia	**CluVRP SCC**	**CluVRP WCC**	**CluVRP SCC**	**CluVRP WCC**
A-n36-k5-C12-V2.gvrp	0,00 %	-7,31 %	0,00 %	-6,54 %
B-n57-k7-C19-V3.gvrp	0,00 %	-1,89 %	0,00 %	-1,89 %
M-n101-k10-C34-V4.gvrp	-0,65 %	-1,64 %	0,00 %	-2,11 %
P-n23-k8-C8-V3.gvrp	0,00 %	-3,58 %	0,00 %	-2,80 %
Golden_13-C51-N253.gvrp	1,60 %	-4,63 %	2,45 %	-3,11 %
Golden_rho10_10.ccvrp	1,79 %	-3,29 %	2,11 %	-3,56 %
Golden_rho25_11.ccvrp	1,67 %	-3,21 %	1,93 %	-3,20 %
A-n61-k9-C21-V4.gvrp	0,43 %	-0,87 %	0,43 %	1,44 %
B-n52-k7-C18-V3.gvrp	0,23 %	0,23 %	0,23 %	0,89 %
G-n262-k25-C88-V9.gvrp	4,88 %	2,74 %	5,78 %	3,88 %
P-n101-k4-C34-V2.gvrp	1,91 %	-2,35 %	1,97 %	0,55 %
Golden_20-C85-N421.gvrp	5,43 %	-1,90 %	5,43 %	-1,34 %
Golden_rho10_20.ccvrp	13,06 %	12,23 %	16,13 %	14,60 %
Golden_rho25_18.ccvrp	5,98 %	4,13 %	6,45 %	5,88 %
Promedio individual	2,60 %	-0,81 %	3,07 %	0,19 %
Promedio total por método	**2,10 %**		**2,77 %**	

Tab. 6.4: Diferencia promedio a la mejor solución entre los distintos métodos de calculo de distancia de Nivel Cluster.

Cómo se puede ver en en la tabla 6.4 los resultados totales son favorables al método *Menor distancia entre clientes*. También se puede observar que en todos los resultados parciales este método fue superior al de Centroide.

Es importante mencionar que los resultados de CluVRP SCC y CluVRP WCC son comparados con los mejores resultados obtenidos para CluVRP SCC. Esto sucede porque

para el set Golden-Izquierdo no hay publicados resultados para CluVRP WCC, entonces se decidió utilizar (al igual que se hace en el trabajo de Defryn et al. [20]), la comparación de los resultados de CluVRP WCC utilizando como referencia los mejores de CluVRP SCC. A esto se debe que generalmente los resultados de CluVRP WCC tienen distancias menores, ya que al flexibilizar restricciones, se encuentran mejores resultados.
En los siguientes experimentos este parámetro quedará fijado en el método ganador, *Menor distancia entre clientes*.

6.4.3. Experimento II: Algoritmo Goloso Aleatorio de Nivel Cluster I

En este experimento se observará el comportamiento de los distintos algoritmos golosos aleatorios desarrollados para el Nivel Cluster (capítulo 5.2.3). Todos los demás parámetros del algoritmo se mantendrán con el resultado alcanzado hasta este experimento. La forma de evaluar los resultados será contar la cantidad de veces que cada algoritmo goloso aleatorio es utilizado por la mejor solución para cada instancia.

Parámetro a evaluar:

	Parámetro	**Valores a evaluar**
Nivel Cluster	*Algoritmo Goloso Aleatorio*	{SimpleRCL, DemandaRCL, DobleRCL, MejorAjuste}

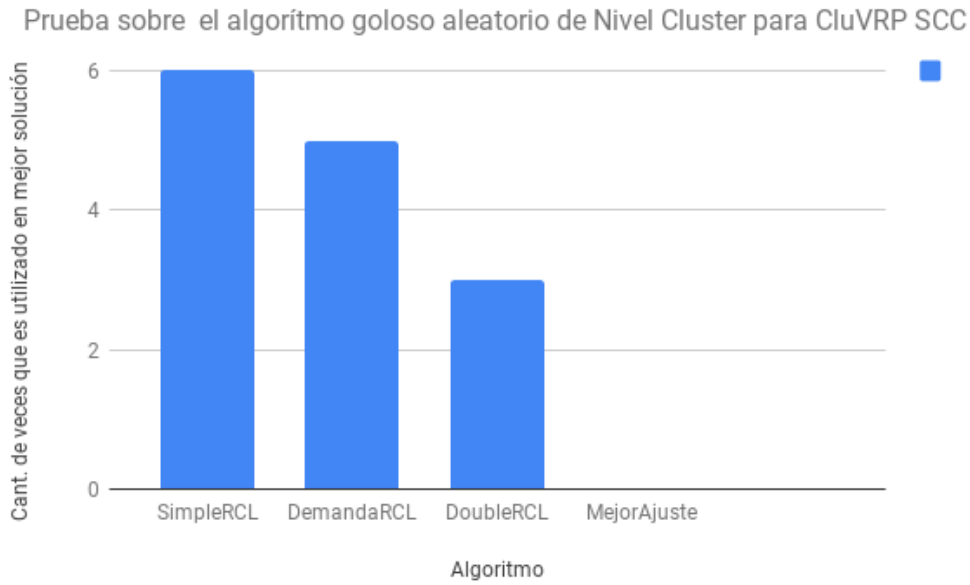


Fig. 6.1: Cantidad de veces que cada algoritmo goloso aleatorio de Nivel Cluster es parte de la mejor solución para CluVRP SCC.

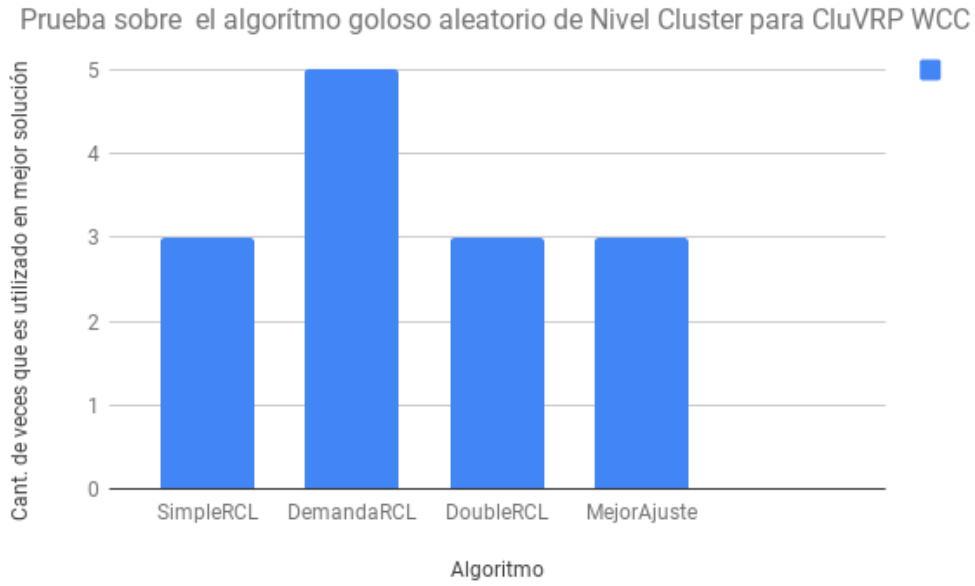


Fig. 6.2: Cantidad de veces que cada algoritmo goloso aleatorio de Nivel Cluster es parte de la mejor solución para CluVRP WCC.

Se puede observar en el gráfico 6.1 que para CluVRP SCC los algoritmos SimpleRCL y DemandaRCL fueron los que alcanzaron mayor cantidad de mejores soluciones. La diferencia entre uno y otro es sólo de 1.
Por otro lado para CluVRP WCC, podemos ver que en el gráfico 6.2 que DemandaRCL aventaja por dos a todos los demás, que alcanzan el resultado de tres mejores soluciones. De aquí en adelante seguiremos experimentando tanto con DemandaRCL como con SimpleRCL ya que ambos lograron buenos resultados para los dos problemas. Tanto MejorAjuste como DoubleRCL no obtuvieron resultados destacables, por lo que no se los seguirá evaluando.

6.4.4. Experimento III: Algoritmo Goloso Aleatorio de Nivel Cluster II

En este experimento se evaluará el comportamiento de los dos algoritmos que obtuvieron mejor performance en el experimento anterior, SimpleRCL y DemandaRCL. Se utilizará el parámetro $\alpha - distancia$ y el parámetro $\alpha - demanda$ (ambos de Nivel Cluster) variando el valor de ambos en [0, 0.5, 1].
La forma de evaluar los resultados será la misma que en el experimento anterior.

Parámetro a evaluar:

	Parámetro	**Valores a evaluar**
Nivel Cluster	*Algoritmo Goloso Aleatorio*	{SimpleRCL, DemandaRCL}
	$\alpha - demanda$	{0,0.1,5}
	$\alpha - distancia$	{0,0.1,5}

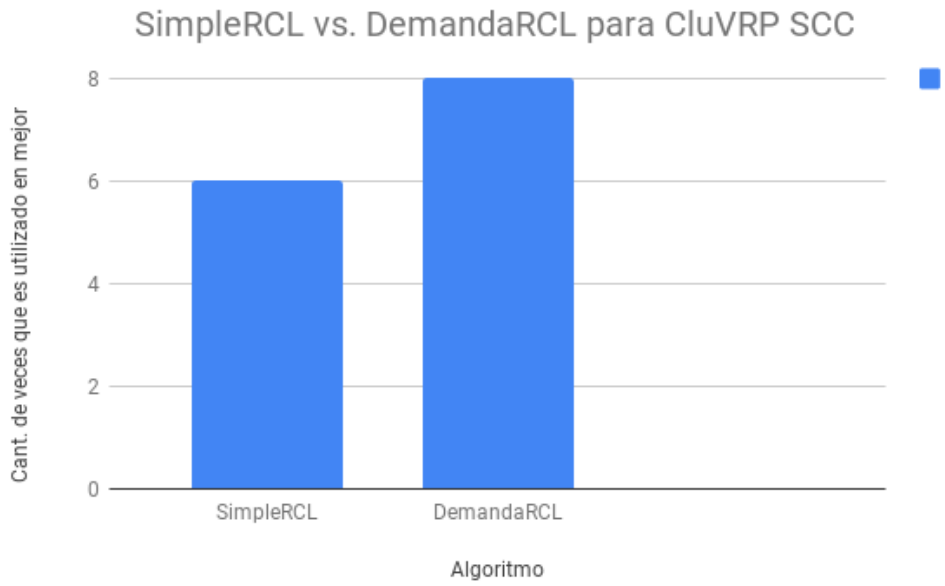


Fig. 6.3: Cantidad de veces que los algoritmos SimpleRCL y DemandaRCL son parte de la mejor solución para CluVRP SCC.

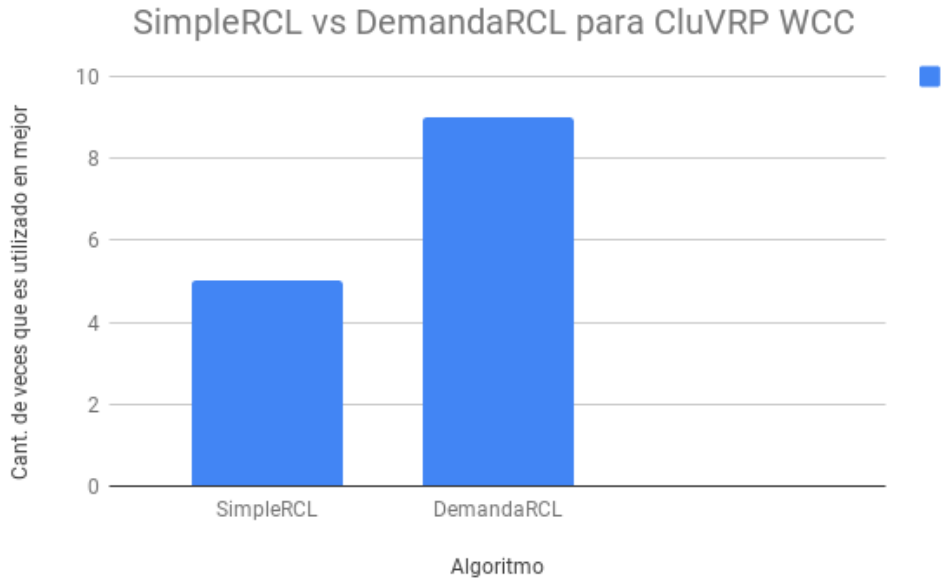


Fig. 6.4: Cantidad de veces que los algoritmos SimpleRCL y DemandaRCL son parte de la mejor solución para CluVRP WCC.

Podemos observar tanto en el gráfico 6.3 y en 6.4 que el algoritmo DemandaRCL obtuvo mejores resultados, siendo parte de mayor cantidad de mejores soluciones para

las instancias del set de prueba. De aquí en adelante este parámetro quedará fijado en *DemandaRCL*.

6.4.5. Experimento IV: Parámetro $\alpha - demanda$

En este experimento observaremos el comportamiento del algoritmo variando el valor del parámetro $\alpha{-}demanda$. Se hará con $\alpha{-}demanda \in \{0, 0.1, 0.2, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$, y con el resto de los parámetros fijados con los valores alcanzados hasta este experimento. Se evaluará la diferencia promedio a los mejores resultados conocidos para el conjunto de instancias de prueba. Esto se realizará para cada uno de los valores del parámetro, tanto para CluVRP SCC cómo para CluVRP WCC.

Parámetro a evaluar:

	Parámetro	**Valores a evaluar**
Nivel Cluster	$\alpha - demanda$	{0, 0.1, 0.2, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1}

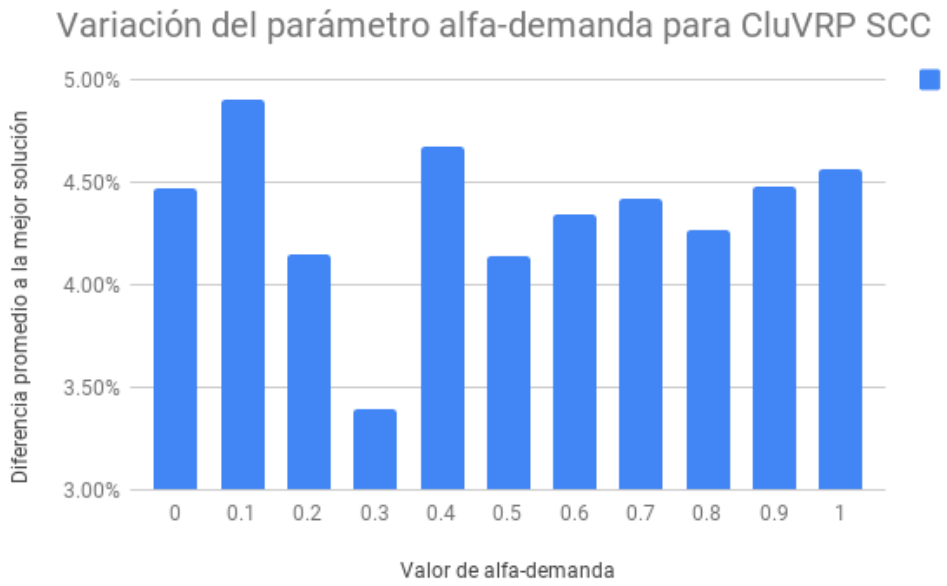


Fig. 6.5: Variación del resultando en función de valores del parámetro $\alpha - demanda$ en CluVRP SCC.

Fig. 6.6: Variación del resultando en función de valores del parámetro $\alpha - demanda$ en CluVRP WCC.

Podemos observar en el gráfico 6.5 que para el problema CluVRP SCC el mejor resultado se encuentra con $\alpha - demanda = 0.3$, con una diferencia promedio de 3.39 %. Por otro lado que para CluVRP WCC en el gráfico 6.6, podemos ver que la menor diferencia promedio se encuentra con $\alpha - demanda = 0.6$, con un valor de 0.89 %.
A partir de este experimento las siguientes pruebas de CluVRP SCC y CluVRP WCC se parametrizarán por separado, ya que en este experimento los valores de $\alpha - demanda$ resultaron ser distintos para el mejor resultado obtenido.

6.4.6. Experimento V: Parámetro $\alpha - distancia$ Nivel Cluster

En este experimento observaremos el comportamiento del algoritmo variando el valor del parámetro $\alpha - distancia$ de Nivel Cluster. Se experimentará con $\alpha - distancia \in \{0, 0.1, 0.2, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$, y con el resto de los parámetros fijados con los valores alcanzados hasta este experimento.
Se evaluará la diferencia promedio a los mejores resultados conocidos para el conjunto de instancias de prueba. Esto se realizará para cada uno de los valores del parámetro, tanto para CluVRP SCC cómo para CluVRP WCC.

Parámetro a evaluar:

	Parámetro	**Valores a evaluar**
Nivel Cluster	$\alpha - distancia$	{0,0.1,0.2,0.3,0.5,0.6,0.7,0.8,0.9,1}

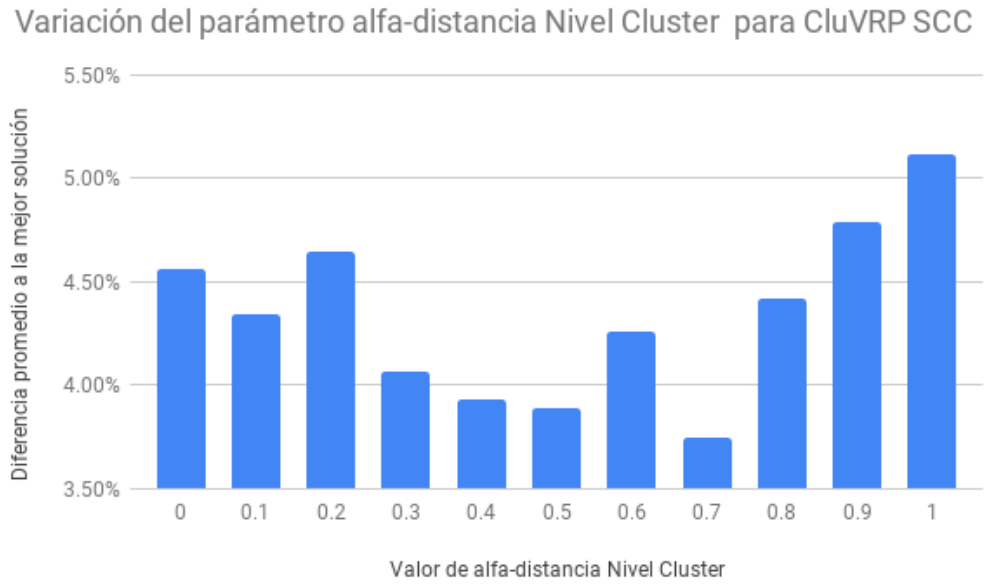


Fig. 6.7: Variación del resultando en función de valores del parámetro $\alpha - distancia$ Nivel Cluster en CluVRP SCC.

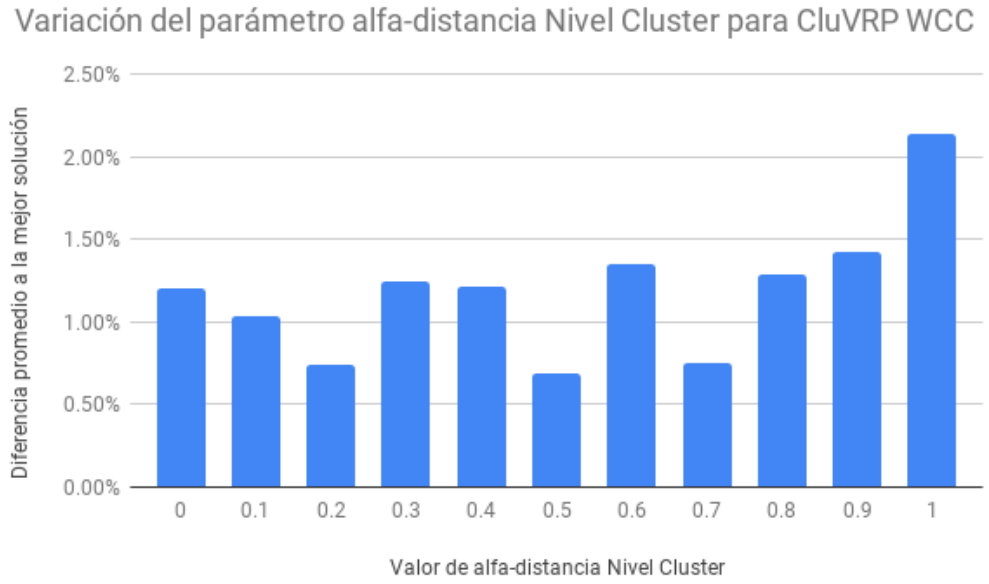


Fig. 6.8: Variación del resultando en función de valores del parámetro $\alpha - distancia$ Nivel Cluster en CluVRP WCC.

Tanto en el gráfico 6.7 como en 6.8 podemos ver que tanto para CluVRP SCC, como para CluVRP WCC, los mejores resultados se logran en los valores de 0.5 y 0.7.

Para CluVRP SCC el mejor resultado se encuentra con $\alpha - distancia = 0.7$, alcanzando a una diferencia promedio de 3.75 %, y para CluVRP WCC el mejor resultado se encuentra con se encuentra con $\alpha - distancia = 0.5$, alcanzando una diferencia promedio de 0.69 %.

6.4.7. Experimento VI: Parámetro $\alpha - distancia$ Nivel Cliente

En este experimento observaremos el comportamiento del algoritmo variando el valor del parámetro $\alpha - distancia$ de Nivel Cliente. Se experimentará con $\alpha - distancia \in \{0, 0.1, 0.2, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$, y con el resto de los parámetros fijados con los valores alcanzados hasta este experimento.
Se evaluará la diferencia promedio a los mejores resultados conocidos para el conjunto de instancias de prueba. Esto se realizará para cada uno de los valores del parámetro, tanto para CluVRP SCC cómo para CluVRP WCC.

Parámetro a evaluar:

	Parámetro	**Valores a evaluar**
Nivel Cluster	$\alpha - distancia$	{0,0.1,0.2,0.3,0.5,0.6,0.7,0.8,0.9,1}

Fig. 6.9: Variación del resultando en función de valores del parámetro $\alpha - distancia$ Nivel Cliente en CluVRP SCC.

Fig. 6.10: Variación del resultando en función de valores del parámetro $\alpha - distancia$ Nivel Cliente en CluVRP WCC.

Podemos observar en 6.9 y en 6.10 que los mejores resultados se encuentran debajo de $\alpha - distancia = 0.5$. En particular la mejor diferencia en CluVRP SCC se encuentra con $\alpha - distancia = 0.1$, llegando a un valor de 3.95 %.
Para CluVRP WCC el mejor resultado se encuentra con $\alpha - distancia = 0$ (completamente goloso), llegando a un valor de diferencia de 0.72 %. Ambos resultados muestran que el comportamiento más goloso respecto al armado de la RCL en los algoritmos golosos aleatorios de Nivel Cliente, produce mejores soluciones.

6.4.8. Experimento VII: Búsquedas Locales de Nivel Cluster

En esta prueba se verificará la performance de las búsquedas locales de Nivel Cluster. La prueba consistente en fijar en 100 iteraciones a todas las heurísticas de este nivel y observar cuál fue la máxima iteración que alcanzó una mejora. Esto se observará para la mejor solución de cada instancia.
Los parámetros restantes del algoritmo permanecerán de acuerdo a los valores alcanzados hasta este experimento.

Parámetros a evaluar:

	Parámetro	Valor
Nivel Cluster	*Iteraciones IntercambioDeVehículos*	100
	Iteraciones InsertarEnVehículo	100
	Iteraciones 2-Opt	100
	Iteraciones Relocalizar	100
	Interaciones Intercambio	100
	Iteraciones IntercambioAleatorio	100

	Máxima iteración con mejora					
Instancia	**TwoOpt**	**Relocalizar**	**Intercambio**	**Inter. Aleatorio**	**Inter. Vehículo**	**Inser.Vehículo**
A-n36-k5-C22-V2	1	1	1	1	1	1
B-n57-k7-C29-V3	1	1	1	2	1	1
M-n212-k21-C32-V2	1	1	1	1	1	1
P-n23-k8-C8-V3	1	1	1	1	1	1
Golden_23-C52-N253	1	1	1	1	1	1
Golden_rho21_21	1	1	1	2	1	1
Golden_rho25_22	1	1	1	1	1	1
A-n62-k9-C22-V2	1	1	1	1	1	1
B-n52-k7-C28-V3	1	1	1	2	1	1
G-n262-k25-C88-V9	1	1	1	2	1	1
P-n212-k2-C32-V2	1	1	1	1	1	1
Golden_21-C85-N222	1	1	1	1	1	1
Golden_rho10_20	1	1	1	2	1	1
Golden_rho25_18	1	1	1	2	1	1
Valor máximo	**1**	**1**	**1**	**2**	**1**	**1**

Tab. 6.5: Máxima iteración con mejora para búsquedas locales de Nivel Cluster en CluVRP SCC.

	Máxima iteración con mejora					
Instancia	**TwoOpt**	**Relocalizar**	**Intercambio**	**Inter. Aleatorio**	**Inter. Vehículo**	**Inser. Vehículo**
A-n36-k5-C22-V2	1	1	1	1	1	2
B-n57-k7-C29-V3	1	2	1	1	1	2
M-n212-k21-C32-V2	1	1	1	1	2	2
P-n23-k8-C8-V3	1	1	1	1	1	1
Golden_23-C52-N253	1	2	1	1	1	2
Golden_rho21_21	1	1	1	1	2	2
Golden_rho25_22	1	1	1	1	2	2
A-n62-k9-C22-V2	1	2	1	1	1	1
B-n52-k7-C28-V3	1	2	1	1	1	2
G-n262-k25-C88-V9	1	2	1	1	1	2
P-n212-k2-C32-V2	1	2	1	1	2	2
Golden_21-C85-N222	1	2	1	1	1	2
Golden_rho10_20	1	2	1	1	2	2
Golden_rho25_18	1	2	1	1	2	2
Valor máximo	**1**	**2**	**1**	**1**	**2**	**2**

Tab. 6.6: Máxima iteración con mejora para búsquedas locales de Nivel Cluster en CluVRP WCC.

Podemos observar en la tabla 6.5 que para CluVRP SCC, el máximo de iteraciones con mejoras es muy bajo respecto a lo configurado, llegando a un máximo de dos en algunas heurísticas para determinadas instancias. Tanto *TwoOpt*, *Intercambio* e *IntercambioAleatorio* no superan una iteración.

Es importante brindar algunas aclaraciones respecto al significado de *máxima iteración con mejora*, para comprender mejor los resultados obtenidos.
Si por ejemplo el resultado es 1 (cómo varios obtenidos), no significa que solo se utiliza una iteración de la heurística. Todas las heurísticas luego de encontrar una mejora, reinician el contador de iteraciones a 0. Luego se comienza nuevamente a buscar mejoras en la solución con el iterador contando desde 0 hasta el número de iteraciones definido en su parámetro. Por lo tanto en estos casos, lo que quiere decir esto (iteración máxima alcanzada = 1) es que la ejecución de una iteración es suficiente para encontrar una mejora, pudiendo repetirse esto muchas veces, ya que cada vez que encuentra una mejora, vuelve a 0 el valor del iterador. Por lo tanto, estos resultados indican que prácticamente no hay iteraciones sin mejora, y que al superar la primera iteración sin mejora, las posibilidades de encontrar mejoras en la siguientes iteraciones es muy baja.

Para CluVRP WCC en la tabla 6.6 podemos ver que hay una variación un poco mayor, pero también al igual que en CluVRP SCC, no se superan las dos iteraciones.

A partir de este experimento se configura en 2 a la cantidad de iteraciones para todas las búsquedas locales de Nivel Cluster, ya que parece ser una buena cota.

6.4.9. Experimento VIII: Búsquedas Locales de Nivel Cliente

En este experimento se verificará la performance de las búsquedas locales de Nivel Cliente. El método de experimentación será el mismo que en el experimento anterior (VII). Estas pruebas se realizarán tanto para CluVRP SCC, como para CluVRP WCC con sus configuraciones alcanzadas respectivamente hasta este experimento.

Parámetros a evaluar:

	Parámetro	**Valor**
Nivel Cliente	*Iteraciones 2-Opt*	100
	Iteraciones Relocalizar	100
	Interaciones Intercambio	100
	Iteraciones IntercambioAleatorio	100

	Máxima iteración con mejora			
Instancia	**TwoOpt**	**Relocalizar**	**Intercambio**	**IntercambioAleatorio**
A-n36-k5-C22-V2	1	1	1	1
B-n57-k7-C29-V3	1	1	1	1
M-n212-k21-C32-V2	1	1	1	1
P-n23-k8-C8-V3	1	1	1	1
Golden_23-C52-N253	1	1	1	1
Golden_rho21_21	1	2	1	1
Golden_rho25_22	1	1	1	1
A-n62-k9-C22-V2	1	1	1	1
B-n52-k7-C28-V3	1	1	1	1
G-n262-k25-C88-V9	1	1	1	1
P-n212-k2-C32-V2	1	1	1	1
Golden_21-C85-N222	1	2	1	1
Golden_rho10_20	1	1	1	1
Golden_rho25_18	1	1	1	2
Valor máximo	**1**	**2**	**1**	**2**

Tab. 6.7: Máxima iteración con mejora para búsquedas locales de Nivel Cliente en CluVRP SCC

	Máxima iteración con mejora			
Instancia	**TwoOpt**	**Relocalizar**	**Intercambio**	**IntercambioAleatorio**
A-n36-k5-C22-V2	2	1	1	1
B-n57-k7-C29-V3	1	1	1	1
M-n212-k21-C32-V2	2	2	1	1
P-n23-k8-C8-V3	2	1	1	1
Golden_23-C52-N253	2	1	1	1
Golden_rho21_21	2	2	2	1
Golden_rho25_22	2	2	1	1
A-n62-k9-C22-V2	1	1	1	1
B-n52-k7-C28-V3	2	1	1	1
G-n262-k25-C88-V9	2	1	2	1
P-n212-k2-C32-V2	2	1	1	1
Golden_21-C85-N222	2	2	1	1
Golden_rho10_20	2	1	1	1
Golden_rho25_18	2	2	2	1
Valor máximo	**2**	**2**	**2**	**1**

Tab. 6.8: Máxima iteración con mejora para búsquedas locales de Nivel Cliente en CluVRP WCC

Podemos observar en la tablas 6.7 y 6.8 que para CluVRP SCC y CluVRP WCC, al igual que en el experimento anterior (de Nivel Cluster), todas las heurísticas registran como máxima iteración con mejora a la primera o la segunda, nunca superando esa cantidad.
Como resultado de la prueba realizada, se configurará en 2 a la cantidad máxima de iteraciones sin mejora para todas las búsquedas locales de Nivel Cliente.

6.4.10. Experimento IX: Iteraciones del ciclo de Búsquedas Locales

En este experimento se estudiará el comportamiento del algoritmo variando la cantidad de iteraciones del parámetro *Iteraciones ciclo Búsquedas Locales*. Vale recordar que este parámetro indica tanto para las búsquedas de Nivel Cluster, como para las de Nivel Cliente, cuantas iteraciones cumple el ciclo que ejecuta al conjunto de ellas.
Hasta aquí en todos los experimentos este parámetro fue configurado con el valor en 1. En este prueba se hará variar este valor en el siguiente rango: [1,2,3,4,5,6,7,8,9,10]. Se observará cuál fue la máxima iteración que alcanzó una mejora para la mejor solución de cada instancia.
Estas pruebas se realizarán tanto para CluVRP SCC, como para CluVRP WCC con sus configuraciones alcanzadas respectivamente hasta este experimento.

Parámetro a evaluar:

	Parámetro	**Valor**
Parámetros Generales	*Iteraciones ciclo Búsquedas Locales*	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

	Máxima iteración con mejora en cliclo búsqueda local Nivel Cluster		**Máxima iteración con mejora en ciclo búsqueda local Nivel Cliente**	
Instancia	**CluVRP SCC**	**CluVRP WCC**	**CluVRP SCC**	**CluVRP WCC**
A-n36-k5-C22-V2	1	2	1	1
B-n57-k7-C29-V3	2	2	1	1
M-n212-k21-C32-V2	2	2	1	1
P-n23-k8-C8-V3	1	1	1	1
Golden_23-C52-N253	3	3	1	1
Golden_rho21_21	2	3	1	1
Golden_rho25_22	2	3	1	1
A-n62-k9-C22-V2	2	3	1	1
B-n52-k7-C28-V3	1	2	1	1
G-n262-k25-C88-V9	4	3	1	1
P-n212-k2-C32-V2	2	5	1	1
Golden_21-C85-N222	5	3	1	1
Golden_rho10_20	3	4	1	1
Golden_rho25_18	4	4	1	1
Valor máximo	5	5	1	1

Tab. 6.9: Ultima iteración con mejora para ciclo de búsquedas locales en CluVRP SCC y CluVRP WCC

Podemos observar en la tabla 6.9 que en el Nivel Cluster los resultados para CluVRP SCC y para CluVRP WCC son muy similares. La cantidad de iteraciones en el ciclo de búsquedas locales de Nivel Cluster alcanza el valor máximo de 5 iteraciones para ambos problemas, promediando para CluVRP SCC el valor en 2, y en 3 para CluVRP WCC.
Para el ciclo de búsquedas locales de Nivel Cliente, no hay variaciones, siendo una iteración suficiente para lograr el mejor resultado, no superando este valor en ningún caso.

Por lo observado en este experimento, se determina en 5 la cantidad de iteraciones del ciclo de búsquedas locales debido a que fue el número máximo alcanzado de iteraciones con mejora. En este sentido, la preferencia de aumentar la probabilidad de encontrar una mejor solución resulto decisiva para esta decisión. Por ese motivo no se eligió el número promedio.

Respecto a la experimentación con el orden de ejecución tanto de las búsquedas locales de Nivel Cluster, cómo de Nivel Cliente, se ha decido no realizar un experimento sobre ese punto en este trabajo.
En todos los experimentos anteriores dichos parámetros fueron configurados en opción *Aleatoria*, lo que quiere decir que se ejecutaban todas las búsquedas locales de cada nivel, pero en un orden aleatorio en cada ejecución. Esta decisión se tomó para abarcar más posibilidades respecto a los distintos ordenes de ejecución.
Se pudo observar que para las mejores soluciones encontradas los ordenes de ejecución no tenían relación entre las distintas instancias, por lo que no fue posible encontrar relación entre los distintos órdenes y los mejores resultados.

6.4.11. Experimento X: Búsquedas Locales en Solución Final

En este experimento se verificará la performance de las búsquedas locales que se aplican en la solución final. Es importante aclarar que estas heurísticas solo son aplicadas en CluVRP SCC, ya que las mismas actúan sobre los clusters, moviéndolos entre vehículos (*IntercambioDeVehículos*) o de posición dentro del vehículo asignado (*Intercambio*) y es por este motivo que no se aplican en la solución final de CluVRP WCC.

De las dos heurísticas mencionadas, solo *IntercambioDeVehículos* es parametrizable. *Intercambio* no lo es debido a que hace una iteración completa para todos los vehículos, buscando hacer un intercambio en la posición de sus clusters en un único orden, con el fin de mejorar la distancia total recorrida. Si hay cambios vuelve a repetir el ciclo, pero si no hay cambios, finaliza. Esto último sucede porque la iteración se realiza en un único orden, por lo que si en una iteración completa no hay mejora, ya no existirá posibilidad de mejora en la siguiente.

En este experimento se observará la máxima iteración con mejora de *IntercambioDeVehículos* aplicada a la solución final de CluVRP SCC. Para esto se definirá el parámetro con un máximo de 100 iteraciones.
Por otro lado también para verificar la utilidad de la heurística *Intercambio*, se observará cuantas iteraciones sucesivas con mejora realiza.

Parámetros a evaluar:

	Parámetro	**Valor**
Parámetros Generales	*Iteraciones IntercambioDeVehículo*	100

	Máxima iteración con mejora	
Instancia	**Intercambio**	**IntercambioDeVehículos**
A-n36-k5-C22-V2	1	1
B-n57-k7-C29-V3	1	1
M-n212-k21-C32-V2	1	1
P-n23-k8-C8-V3	1	1
Golden_23-C52-N253	1	2
Golden_rho21_21	1	1
Golden_rho25_22	1	1
A-n62-k9-C22-V2	1	1
B-n52-k7-C28-V3	1	1
G-n262-k25-C88-V9	1	1
P-n212-k2-C32-V2	1	2
Golden_21-C85-N222	1	2
Golden_rho10_20	1	1
Golden_rho25_18	1	1
Valor máximo	**1**	**2**

Tab. 6.10: Ultima iteración con mejora para búsquedas locales de solución final para CluVRP SCC

Podemos observar en 6.10, que los resultados son similares a los obtenidos en los anteriores experimentos sobre las búsquedas locales de los diferentes niveles (XIII y IX). La máxima iteración con mejora alcanza es 2.
A partir de este experimento se decide configurar en 2 a la cantidad máxima de iteraciones sin mejoras para la heurística *InsertarEnVehículo* de solución final para CluVRP SCC.

6.4.12. Experimento XI: Iteraciones del ciclo principal

Estudiaremos el comportamiento del algoritmo al variar la cantidad de iteraciones del ciclo principal. Dentro del set de parámetros de la tabla 6.1, este parámetro se encuentra en *Parámetros Generales*, con el nombre de *Iteraciones*.
Hasta aquí este parámetro se mantuvo configurado en 100 para todos los anteriores experimentos, en esta prueba se hará variar al mismo en el siguiente rango de valores: [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000].
Estas pruebas se realizarán tanto para CluVRP SCC, como para CluVRP WCC con sus configuraciones alcanzadas respectivamente hasta este experimento.

Parámetro a evaluar:

	Parámetro	**Valor**
Parámetros Generales	*Iteraciones*	{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000}

	Ultima iteración con mejora	
Instancia	**CluVRP SCC**	**CluVRP WCC**
A-n36-k5-C22-V2	1	29
B-n57-k7-C29-V3	125	163
M-n212-k21-C32-V2	155	504
P-n23-k8-C8-V3	51	628
Golden_23-C52-N253	561	260
Golden_rho21_21	883	658
Golden_rho25_22	63	615
A-n62-k9-C22-V2	187	321
B-n52-k7-C28-V3	588	118
G-n262-k25-C88-V9	175	747
P-n212-k2-C32-V2	105	434
Golden_21-C85-N222	61	830
Golden_rho10_20	517	772
Golden_rho25_18	566	338
Valor máximo	**883**	**830**
Valor promedio	**288**	**458**

Tab. 6.11: Ultima iteración con mejora para ciclo principal de CluVRP WCC y CluVRP WCC.

Podemos observar en la tabla 6.11 que la cantidad máxima de iteraciones para CluVRP SCC y CluVRP WCC es similar, encontrándose entre 800 y 900. No ocurre lo mismo para el promedio, con una diferencia de casi 200.
Observando el promedio de iteraciones, vemos que CluVRP SCC utiliza menos iteraciones para encontrar las mejores soluciones.
A continuación se realizarán dos experimentos más, uno para medir la variación del tiempo en función de la cantidad de iteraciones asignadas y otro observar la variación de los resultados en función también de la cantidad de iteraciones.

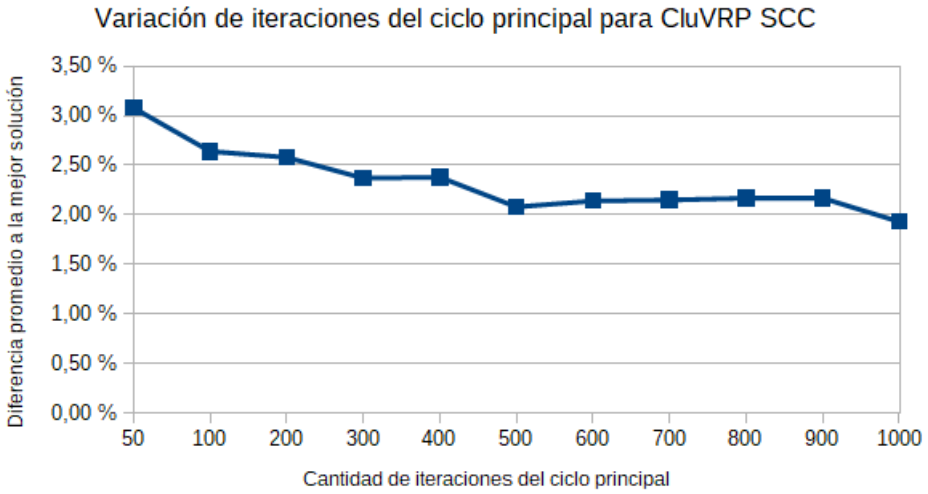


Fig. 6.11: Variación de iteraciones del ciclo principal para CluVRP SCC.

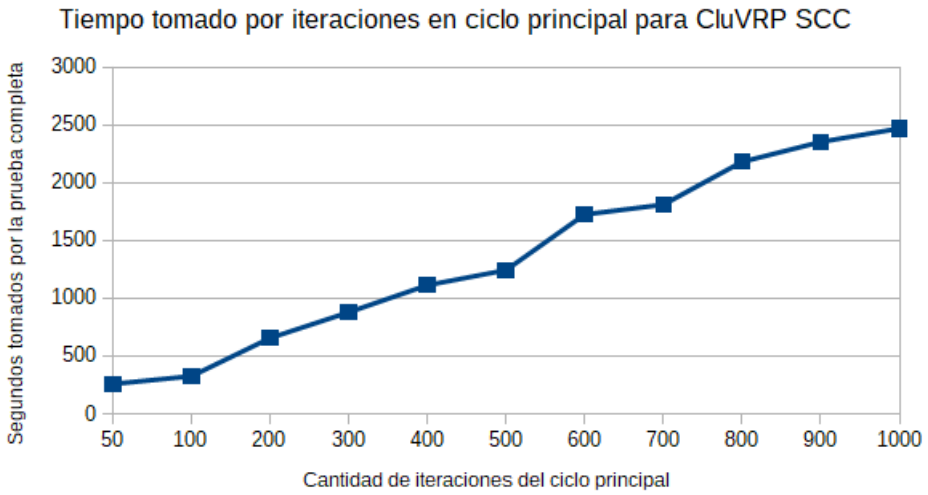


Fig. 6.12: Tiempo en relación con iteraciones del ciclo principal para CluVRP SCC.

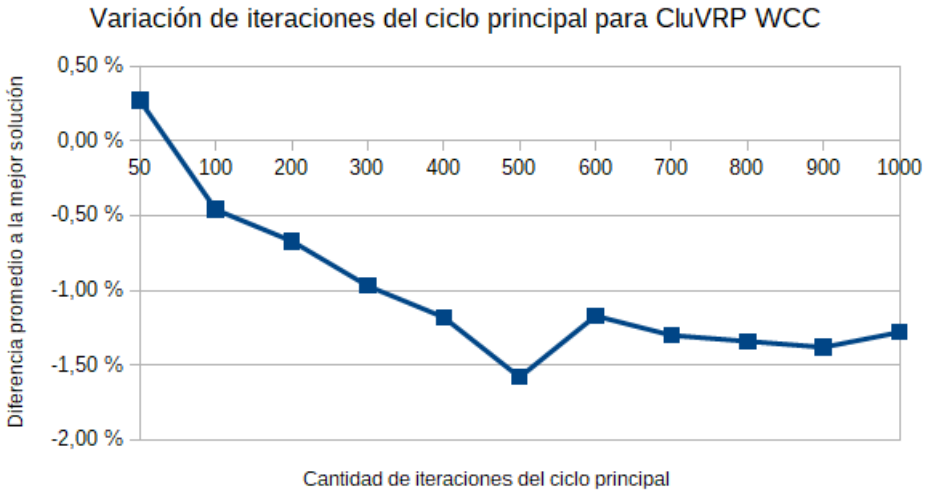


Fig. 6.13: Variación de iteraciones del ciclo principal para CluVRP WCC.

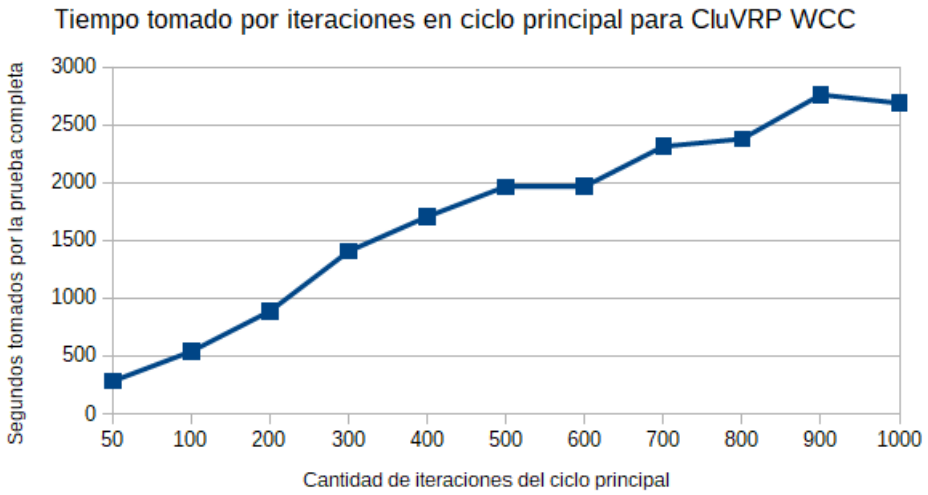


Fig. 6.14: Tiempo en relación con iteraciones del ciclo principal para CluVRP WCC.

En el gráfico 6.11 podemos observar para CluVRP SCC que los resultados (diferencia promedio a los mejores resultados para todas instancia del set) varia solo en un 1 % entre el mejor y el peor resultado obtenido. Comenzando por 2.96 % para 50 iteraciones y terminando con 1.68 % para 1000 iteraciones. También podemos ver en el gráfico 6.12 que la diferencia en el tiempo tomado por las pruebas es considerable, comenzado con 258 segundos para 50 iteraciones, llegando a 2470 segundos para 1000 iteraciones, casi 10 veces más.
En ambos gráficos puede observarse un cierto equilibrio alcanzado en 500 iteraciones, con un resultado de 2,05 % y un tiempo de 1244 segundos.
Como consecuencia de lo anterior, para CluVRP SCC se asignará en 500 a la cantidad de iteraciones del parámetro estudiado.

Para CluVRP WCC, en el gráfico 6.13 podemos observar también que la diferencia entre el mejor y el peor resultado es de aproximadamente 1 %. Comenzando con 50 iteraciones en 0,27 % y finalizando con 1000 iteraciones en -1,28 %.
Lo interesante aquí ocurre en 500 iteraciones, alcanzando el mejor resultado con 1,58 %, un tiempo de 1963 segundos. Luego de las 500 iteraciones los resultados empeoran levemente (menos de 0.30 % en promedio) y no vuelven a mejorar considerablemente. Si bien llama la atención que al aumentar iteraciones el resultado empeore aunque sea levemente, es bastante razonable dada la gran cantidad de decisiones con cierto grado de aleatoriedad que toma el algoritmo.
Dado este resultado se decide parametrizar en 500 (al igual que para CluVRP SCC) a la cantidad de iteraciones del parámetro estudiado para CluVRP WCC.

6.4.13. Experimento XII: Pruebas Finales

Los experimentos hasta aquí realizados trataron de estudiar el comportamiento del algoritmo para comprender como los distintos valores de sus parámetros afectan su performance. Esto no significa que la parametrización obtenida hasta este punto sea la mejor,

ya que por fuera del trabajo realizado quedaron muchos valores por analizar.

Hasta aquí se pudo observar por ejemplo que el estudio del comportamiento de las distintas búsquedas locales no presentó resultados interesantes para distintos valores en sus parámetros. Por otro lado, lo que si resulto interesante fue estudiar el comportamiento variando las iteraciones de los distintos ciclos del algoritmo (Nivel Cluster, Nivel Cliente, Búsquedas Locales).
También quedaron por estudiar muchas combinaciones de los tres parámetros α, dónde también se observaron resultados interesantes en los casos estudiados.

En esta prueba se decidió volver al inicio de la experimentación, para estudiar nuevamente los algoritmos golosos aleatorios de Nivel Cluster y los parámetros α. Esta decisión se basa en que los algoritmos golosos de Nivel Cluster son decisivos en la búsqueda de mejores resultados. Esto se debe a que las distancias entre los clusters suelen ser grandes y afectan de forma fundamental los resultados obtenidos. Por otro lado los distintos parámetros α también producen variaciones importantes en las soluciones, ya que hacen de *selectores* de clusters o clientes candidatos.

En este experimento se realizarán la siguientes pruebas:

- **Prueba 1:** La solución se buscará a partir de tres algoritmos golosos aleatorios de Nivel Cluster, *SimpleRCL*, *DemandaRCL* y *DobleRCL*. Probando en cada iteración de nivel cluster, cual de los tres arroja una mejor solución. Esto se realizará con los demás parámetros configurados con los valores alcanzados hasta este experimento.

- **Prueba 2:** Los parámetros $\alpha - demanda$, $\alpha - distancia$ Nivel Cluster y $\alpha - distancia$ Nivel Cliente, tomarán valores aleatorios en el rango [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1], en cada iteración principal del algoritmo. Esto se realizará con los demás parámetros configurados con los valores alcanzados hasta este experimento.

- **Prueba 3:** La misma configuración de algoritmos golosos aleatorios de Nivel Cluster que en *Prueba 1*, pero los parámetros $\alpha - demanda$, $\alpha - distancia$ Nivel Cluster y $\alpha - distancia$ Nivel Cliente, tomarán valores aleatorios en el rango [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1], en cada iteración principal del algoritmo.

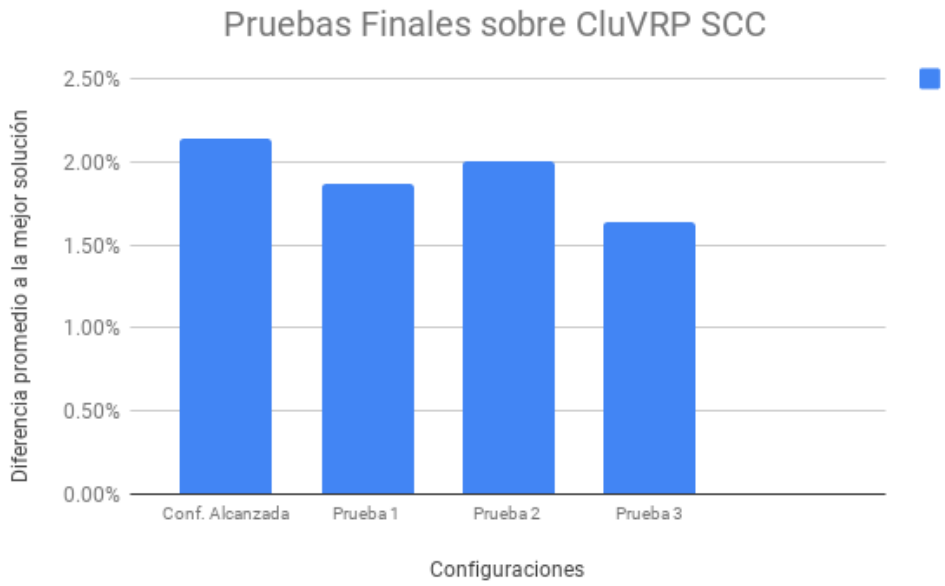


Fig. 6.15: Pruebas finales para CluVRP SCC

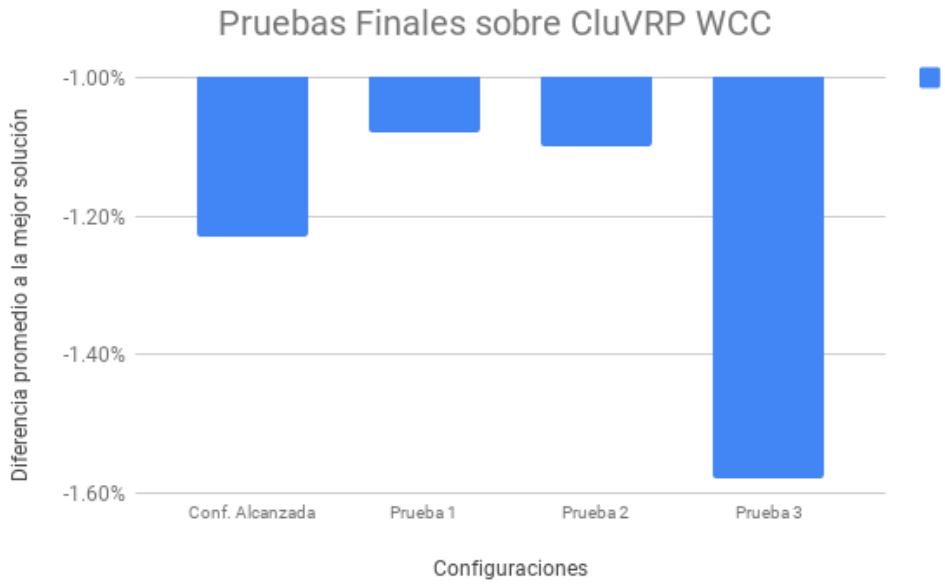


Fig. 6.16: Pruebas finales para CluVRP WCC

Tanto en el gráfico 6.15 para CluVRP SCC, como en el gráfico 6.16 para CluVRP WCC vemos que los resultados de la Prueba 3 son los mejores, pero estando muy cerca del resto de las pruebas. En ambos casos a diferencia entre el mejor y el peor resultado es de solo 0.5 %.

La ligera mejora puede deberse a que que distintas instancias funcionen mejor con distintos algoritmos golosos aleatorios de Nivel Cluster, ya que la forma en que se asignan los clusters en los vehículos tiene gran influencia en la distancia total recorrida. Por otro lado, la parametrización aleatoria de los parámetros α provee mayor diversidad en las soluciones iniciales tanto en Nivel Cluster como en Nivel Cliente.

6.4.14. Parametrización Final

A partir de los trece experimentos realizados, las parametrizaciones finales obtenidas para algoritmo son las siguientes:

	Parámetro	**Valor**
Generales	*Iteraciones*	500
	Iteraciones ciclo Búsquedas Locales	2
	Iteraciones IntercambioDeVehículos	2
	Iteraciones IntercambioDeClusters	1
Nivel Cluster	*Algoritmo Goloso Aleatorio*	DemandaRCL
	Método de distancia cluster	Distancia entre clientes
	$\alpha - demanda$	0.3
	$\alpha - distancia$	0.7
	Iteraciones IntercambioDeVehículos	2
	Iteraciones InsertarEnVehículo	2
	Iteraciones 2-Opt	2
	Iteraciones Relocalizar	2
	Iteraciones Intercambio	2
	Iteraciones IntercambioAleatorio	2
	Orden de ejecución de Búsquedas Locales	Aleatorio
Nivel Cliente	$\alpha - distancia$	0.1
	Iteraciones 2-Opt	2
	Iteraciones Relocalizar	2
	Iteraciones Intercambio	2
	Iteraciones IntercambioAleatorio	2
	Orden de ejecución de Búsquedas Locales	Aleatorio

Tab. 6.12: Parámetros finales algoritmo para CluVRP SCC

	Parámetro	Valor
Generales	*Iteraciones*	500
	Iteraciones ciclo Búsquedas Locales	5
	Iteraciones IntercambioDeVehículos	0
	Iteraciones IntercambioDeClusters	0
Nivel Cluster	*Algoritmo Goloso Aleatorio*	DemandaRCL
	Método de distancia cluster	Distancia entre clientes
	$\alpha - demanda$	0.6
	$\alpha - distancia$	0.5
	Iteraciones IntercambioDeVehículos	2
	Iteraciones InsertarEnVehículo	2
	Iteraciones 2-Opt	2
	Iteraciones Relocalizar	2
	Iteraciones Intercambio	2
	Iteraciones IntercambioAleatorio	2
	Orden de ejección de Búsquedas Locales	Aleatorio
Nivel Cliente	$\alpha - distancia$	0
	Iteraciones 2-Opt	2
	Iteraciones Relocalizar	2
	Iteraciones Intercambio	2
	Iteraciones IntercambioAleatorio	2
	Orden de ejecución de Búsquedas Locales	Aleatorio

Tab. 6.13: Parámetros finales algoritmo para CluVRP WCC

6.4.15. Resultados Finales

En esta sección se presentarán los resultados finales alcanzados por el algoritmo propuesto. Se realizaron tres corridas con tres configuraciones distintas. Por un lado la configuración alcanzada y descrita en la sección anterior (tablas 6.12, 6.13), y por otro utilizando la idea de la *Prueba 1* y de la *Prueba 3* del Experimento XII [6.4.13].
De las tres ejecuciones se tomará para cada instancia el mejor resultado y el tiempo tomado, así como también los resultados promedio.

Algoritmo para CluVRP SCC en instancias GVRP03 comparado con algoritmo exacto de Battarra et al. [9]						
Nombre del conjunto	**#Instancias**	**#Clientes**	**Mejor Diferencia**	**Diferencia Promedio**	**Tiempo Promedio (s)**	**Resultados Óptimos**
A	**27**	**31-79**	0.53 %	0.79 %	2.43	6/27
B	**23**	**31-77**	0.74 %	1.03 %	2.42	2/23
P	**24**	**15-100**	0.58 %	0.60 %	3.30	5/24
M+G	**5**	**100-261**	1.84 %	2.31 %	52.0	1/4
Promedio Total			**0.92 %**	**1.18 %**	**15**	

Tab. 6.14: Resultados del algoritmo para CluVRP SCC en instancias GVRP03

La tabla 6.14 muestra los resultados obtenidos por el algoritmo para el problema CluVRP SCC comparado con los obtenidos por el algoritmo exacto de Battarra et al. [9].

Los mejores resultados promedian debajo del 0.8 %, siendo el grupo de instancias M+G el único que está por encima de dicho valor. Respecto a los resultados óptimos, vale aclarar que para las instancias A, B y P, los resultados del trabajo de Battarra et al. [9] son óptimos, pero para las instancias M+G, no.
Respecto a esto último, se hallaron 14 resultados óptimos, en su mayoría del grupo A y P, y para la instancia *M-n121-k7-C41-V3* el resultado fue mejorada en un 0.66 %.
En los tiempos promedios puede verse algo similar respecto a los resultados, siendo que para A, B y P el tiempo está por debajo de los 4 segundos, y en las instancias G+M se superan los 50 segundos.
Los resultados detallados pueden encontrarse en la tabla 8.1 del apéndice.

Algoritmo para CluVRP WCC en instancias GVRP03 comparado con el algoritmo *Two-Level* de Defryn et al. [20]						
Nombre del conjunto	**#Instancias**	**#Clientes**	**Mejor Diferencia**	**Diferencia Promedio**	**Tiempo Promedio (s)**	**Resultados Mejorados**
A	**27**	**31-79**	0.70 %	1.12 %	4.19	1/27
B	**23**	**31-77**	0.70 %	0.93 %	3.81	0/23
P	**24**	**15-100**	0.57 %	0.77 %	5.40	2/24
M+G	**5**	**100-261**	2.44 %	3.41 %	67.0	1/5
Promedio Total			**1.10 %**	**1.56 %**	**20**	

Tab. 6.15: Resultados del algoritmo para CluVRP WCC en instancias GVRP03

La tabla 6.15 muestra los resultados obtenidos por el algoritmo propuesto para CluVRP WCC comparado con los obtenidos por el algoritmo *Two-Level* propuesto por Defryn et al. [20] (no existen resultados de algoritmos exactos para CluVRP WCC). Los resultados son muy similares a los CluVRP SCC, estando los conjuntos A, B y P por debajo del 0.8 %, siendo el conjunto G+M el de peores resultados. En cuanto a lo tiempos puede verse un comportamiento similar también a los resultados de CluVRP SCC, pero una con pequeña diferencia mayor.
Por otro lado, se mejoraron un total de 4 resultados.
Los resultados detallados pueden encontrarse en la tabla 8.1 del apéndice.

Algoritmo para CluVRP SCC en instancias Golden-Battarra comparado con algoritmo exacto de Battarra et al. [9]						
Nombre del conjunto	**#Instancias**	**#Clientes**	**Mejor Diferencia**	**Diferencia Promedio**	**Tiempo Promedio (s)**	**Resultados Óptimos**
Golden_1	**11**	**241**	1.81 %	2.08 %	16.84	0/11
Golden_5	**11**	**201**	1.24 %	1.56 %	14.30	0/11
Golden_8	**11**	**441**	3.16 %	3.40 %	51.73	0/11
Golden_13	**11**	**253**	1.03 %	1.37 %	16.20	0/11
Golden_16	**11**	**481**	2.77 %	2.94 %	49.95	0/11
Golden_20	**11**	**421**	2.12 %	2.33 %	47.63	0/11
Promedio Total			**2.02 %**	**2.28 %**	**32.77**	

Tab. 6.16: Resultados del algoritmo para CluVRP SCC en instancias Golden-Battarra

En la tabla 6.16 puede verse el resumen de los resultados del algoritmo propuesto para CluVRP SCC en los conjuntos de instancias 1, 5, 8, 13, 16 y 20 del set Golden-Battarra,

comparado con el algoritmo exacto propuesto por Battarra et al. [9].
Los resultados promedian una diferencia cercana al 2 %, lo cual es considerado un buen resultado siendo el tamaño de las instancias de hasta 481 nodos. Respecto a los tiempos promedios, varían dependiendo del subconjunto de instancias, con un piso de 14 segundos aproximados, llegando a casi 52 segundos, en el tiempo máximo promedio.
Se observa también que no se ha logrado alcanzar resultados óptimos. Esto último parece ser algo bastante difícil, ya que que han sido poco los resultados óptimos reportados por otros trabajos en este set de instancias.
Los resultados detallados pueden encontrarse en la tabla 8.3 del apéndice.

Algoritmo para CluVRP WCC en instancias Golden-Battarra comparado con el algoritmo *Two-Level* de Defryn et al. [20]						
Nombre del conjunto	**#Instancias**	**#Clientes**	**Mejor Diferencia**	**Diferencia Promedio**	**Tiempo Promedio (s)**	**Resultados Mejorados**
Golden_1	**11**	**241**	-0.88 %	-0.37 %	44.80	10/11
Golden_5	**11**	**201**	0.23 %	0.81 %	30.61	3/11
Golden_8	**11**	**441**	-0.26 %	0.52 %	249.64	8/11
Golden_13	**11**	**253**	0.23 %	0.54 %	78.51	3/11
Golden_16	**11**	**481**	1.33 %	1.66 %	238.15	1/11
Golden_20	**11**	**421**	0.26 %	0.90 %	132.65	5/11
Promedio Total			**0.15 %**	**0.68 %**	**129.06**	

Tab. 6.17: Resultados del algoritmo para CluVRP WCC en instancias Golden-Battarra

La tabla 6.17 muestra los resultados obtenidos por el algoritmo propuesto para CluVRP WCC en el set de instancias Golden-Battarra comparándolo con los resultados obtenidos por el algoritmo propuesto por Defryn (2017) [20] (no existen resultados de algoritmos exactos para este problema).
Se pueden observar buenos resultados con diferencias promedio que mejoran al algoritmo de Defryn en el caso de los conjuntos de instancias Golden_1 y Golden_8, llegando casi a un 1 % de diferencia. Por otro lado los conjuntos Golden_5, Golden_13 y Golden_20 promedian por debajo de 0.3 %. Respecto a los tiempos, puede observarse un incremento comparado al tiempo tomado para las mismas instancias en el algoritmo de CluVRP SCC.
Se han logrado mejorar los resultados de 30 de las 66 instancias totales.
Los resultados completos pueden verse en la tabla 8.3 del apéndice.

Algoritmo para CluVRP SCC en instancias Golden-Izquierdo comparado con mejores resultados conocidos						
Nombre del conjunto	**#Instancias**	**#Clientes**	**Mejor Diferencia**	**Diferencia Promedio**	**Tiempo Promedio (s)**	**Resultados Mejorados**
Golden_rho10	**20**	**200-483**	2.89 %	3.24 %	95.38	0/20
Golden_rho25	**20**	**200-483**	1.90 %	2.25 %	34.94	0/20
Golden_rho50	**20**	**200-483**	1.78 %	2.00 %	24	0/20
Golden_rho75	**20**	**200-483**	0.69 %	0.80 %	26	0/20
Golden_rho100	**20**	**200-483**	0.84 %	1.00 %	43	0/20
Promedio Total			**1.62 %**	**1.86 %**	**44.66**	

Tab. 6.18: Resultados del algoritmo para CluVRP SCC en instancias Golden-Izquierdo

En la tabla 6.18 puede verse el resumen de los resultados del algoritmo propuesto para CluVRP SCC en los conjuntos de instancias Golden-Izquierdo comparado con los obtenidos por los trabajos de Expósito-Izquierdo et al. [19] y Defryn et al. [20], ambos algoritmos no exactos.
Los mejores resultados promedio obtenidos están a una diferencia de 1.62 % variando según el conjunto de instancias. Para Golden_rho10 se obtiene el peor resultado con 2.89 % y máximo tiempo promedio con 95.38 segundos, y en Golden_rho75 se obtiene un resultado promedio de 0.69 % con un tiempo de casi un tercio respecto al anterior.
Los resultados completos pueden verse en la tabla 8.5 del apéndice.

Algoritmo para CluVRP WCC en instancias Golden-Izquierdo comparado con resultados obtenidos en CluVRP SCC						
Nombre del conjunto	**#Instancias**	**#Clientes**	**Mejor Diferencia**	**Diferencia Promedio**	**Tiempo Promedio (s)**	**Resultados Mejorados**
Golden_rho10	**20**	**200-483**	-0.21 %	-0.01 %	38.34	15/20
Golden_rho25	**20**	**200-483**	-0.42 %	-0.30 %	24.45	18/20
Golden_rho50	**20**	**200-483**	-4.92 %	-4.64 %	33.92	20/20
Golden_rho75	**20**	**200-483**	-5.91 %	-5.53 %	47.25	20/20
Golden_rho100	**20**	**200-483**	-2.68 %	-2.13	107.71	15/20
Promedio Total			**-2.83 %**	**-2.52 %**	**50.33**	

Tab. 6.19: Resultados del algoritmo para CluVRP WCC en instancias Golden-Izquierdo

En la tabla 6.19 podemos ver los resultados obtenidos por el algoritmo propuesto para CluVRP WCC en las de instancia de Golden-Izquierdo comparado con los obtenidos por el algoritmo propuesto en este trabajo para CluVRP SCC.
La comparación se realizó de esta forma debido a que no existen resultados publicados para CluVRP WCC sobre este grupo de instancias. La idea es mostrar cuanto puede mejorar un resultado flexibilizando las restricciones sobre los clusters.
En cuanto a los resultados, puede verse que en los promedios totales que las distancias se reducen en un 2.38 %, llegando a un máximo de 5.91 %. Las instancias que mejoraron su resultado respecto a CluVRP SCC llegan a 88 de las 100 del conjunto.
Respecto a los tiempos, si bien el promedio es muy cercano al de CluVRP SCC, los mismos varían según el grupo de instancias, llegando al máximo en Golden_rho100 con 107 segundos, donde para CluVRP WCC era de solo 43 segundos.
Los resultados completos pueden verse en la tabla 8.5 del apéndice.

7. CONCLUSIONES Y TRABAJOS FUTUROS

En el presente trabajo se implementó un algoritmo de dos niveles para solucionar dos problemas la misma familia VRP: CluVRP con restricciones fuertes sobre los clusters (SCC) y CluVRP con restricciones débiles sobre los clusters (WCC). La implementación propuesta separó el problema en dos niveles, el Nivel Cluster y el Nivel Cliente, aplicando en ambos la técnica metaheurística GRASP, proponiendo distintos algoritmos golosos aleatorios y variadas técnicas de búsqueda local.
Se realizó un profundo estudio del comportamiento del algoritmo analizando los distintos resultandos al variar su parametrización con diversos criterios.

Los resultados obtenidos para el algoritmo implementado de CluVRP SCC comparado con el algoritmo exacto de Battarra et al. [9] fueron en promedio de menos del 1.5 % de diferencia para las instancias GVRP03 y Golden-Battarra, con buenos tiempos de ejecución, logrando resultados óptimos en algunas instancias de GVRP03, mejorando el resultado de una instancia.
Respecto a las instancias Golden-Izquierdo, el algoritmo de CluVRP SCC logró una diferencia promedio de 1.6 % comparado a los mejores resultados conocidos hasta el momento publicados en los trabajos Expósito-Izquierdo et al. [19] y Defryn et al. [18], pero sin lograr mejorarlos.

El algoritmo propuesto para CluVRP WCC logró muy buenos resultados. El mismo fue comparado para las instancias GVRP03 y Golden-Battarra con el algoritmo *Two Level* del trabajo de Defryn et al. [20], siendo este el único trabajo que encontré con resultados publicados.
Para las instancias GVRP03 se logró una diferencia promedio de 1.10 % mejorando los resultados de 4 instancias. Para las instancias Golden-Battarra se alcanzó una diferencia promedio de 0.15 %, pero mejorando los resultados de un total de 30 de las 66 instancias.

Para las instancias Golden-Izquierdo, al no haber encontrado trabajos publicados de CluVRP WCC para este set, se lo comparó con los resultados obtenidos por el propio algoritmo para CluVRP SCC. Se obtuvieron mejoras promedio de 2.83 %, logrando mejorar el resultado 88 de las 100 instancias del conjunto, demostrando que la flexibilización de las restricciones sobre los clusters logra mejorar la distancia total recorrida en este set de instancias.

Los posibles trabajos futuros relacionados a este trabajo y a los problemas estudiados son los siguientes:

- Implementar mejoras como *Path Relinking* o *Reactive GRASP* en el algoritmo GRASP implementado, tanto en el Nivel Cluster como en el Nivel Cliente, para agregar un concepto de *memoria* al algoritmo.

- Desarrollar nuevos algoritmos golosos aleatorios en el Nivel Cliente, ya que a diferencia del Nivel Cluster donde se desarrollaron cuatro, en el Nivel Cliente solo se

desarrolló uno.

- Analizar la relación entre definir la cantidad de vehículos de forma fija (instancias GVRP y Golden-Battarra) y libre (instancias Golden-Izquierdo) en relación a los posibles resultados.

- Analizar la relación entre la asignación de los clusters a los vehículos, la capacidad disponible y la variación de los resultados a Nivel Cluster.

- Desarrollar técnicas de búsquedas locales más especificas para el problema CluVRP WCC en el Nivel Cliente.

- Completar las pruebas para las instancias Golden-Battarra no estudiadas en este trabajo.

- Implementar un algoritmo exacto para CluVRP WCC en busca de registrar soluciones óptimas para el problema, ya que actualmente solo hay disponible trabajos con técnicas heurísticas.

- Implementar un algoritmo exacto para CluVRP SCC en busca de registrar soluciones óptimas en las instancias Golden-Izquierdo.

8. APENDICE

	CluVRP SCC				CluVRP WCC			
	Algoritmo Exacto Battarra	Mejor Resultado Alcanzado	Diferencia	Tiempo (s)	Metaheuristica TwoLevel Defryn	Mejor Resultado Alcanzado	Diferencia	Tiempo (s)
A-n32-k5-C11-V2	522	523	0,19 %	1,31	515	517	0,39 %	1,95
A-n33-k5-C11-V2	472	473	0,21 %	1,40	**461**	**461**	**0,00 %**	**2,05**
A-n33-k6-C11-V2	**562**	**562**	**0,00 %**	**1,40**	554	555	0,18 %	2,03
A-n34-k5-C12-V2	**547**	**547**	**0,00 %**	**1,75**	538	539	0,19 %	2,31
A-n36-k5-C12-V2	**588**	**588**	**0,00 %**	**1,62**	543	545	0,37 %	2,54
A-n37-k5-C13-V2	569	571	0,35 %	1,76	**555**	**551**	**-0,72 %**	**2,76**
A-n37-k6-C13-V2	615	621	0,98 %	1,36	605	618	2,15 %	2,55
A-n38-k5-C13-V2	507	510	0,59 %	1,80	507	510	0,59 %	2,79
A-n39-k5-C13-V2	610	618	1,31 %	1,75	588	594	1,02 %	2,78
A-n39-k6-C13-V2	613	614	0,16 %	1,74	603	604	0,17 %	3,01
A-n44-k6-C15-V2	714	724	1,40 %	0,43	691	696	0,72 %	0,56
A-n45-k6-C15-V3	712	715	0,42 %	2,66	652	655	0,46 %	3,66
A-n45-k7-C15-V3	**664**	**664**	**0,00 %**	**2,07**	**661**	**661**	**0,00 %**	**3,21**
A-n46-k7-C16-V3	**664**	**664**	**0,00 %**	**2,83**	642	644	0,31 %	3,62
A-n48-k7-C16-V3	**683**	**683**	**0,00 %**	**2,45**	**680**	**680**	**0,00 %**	**3,48**
A-n53-k7-C18-V3	651	653	0,31 %	3,21	627	629	0,32 %	4,47
A-n54-k7-C18-V3	724	728	0,55 %	2,82	699	704	0,72 %	4,17
A-n55-k9-C19-V3	653	655	0,31 %	2,60	645	647	0,31 %	3,98
A-n60-k9-C20-V3	787	797	1,27 %	3,03	762	767	0,66 %	4,99
A-n61-k9-C21-V4	682	685	0,44 %	3,53	671	681	1,49 %	4,35
A-n62-k8-C21-V3	778	782	0,51 %	4,00	771	776	0,65 %	4,36
A-n63-k10-C21-V4	801	805	0,50 %	3,65	779	783	0,51 %	4,28
A-n63-k9-C21-V3	865	874	1,04 %	1,02	837	888	3,00 %	0,64
A-n64-k9-C22-V3	773	774	0,13 %	4,48	767	768	0,13 %	5,58
A-n65-k9-C22-V3	725	730	0,69 %	3,60	693	698	0,72 %	4,36
A-n69-k9-C23-V3	814	832	2,21 %	1,76	794	829	4,41 %	1,14
A-n80-k10-C27-V4	972	980	0,31 %	5,66	944	946	0,21 %	31,51
B-n31-k5-C11-V2	375	377	0,53 %	1,38	375	377	0,53 %	2,18
B-n34-k5-C12-V2	**416**	**416**	**0,00 %**	**1,51**	415	416	0,24 %	2,57
B-n35-k5-C12-V2	562	563	0,18 %	1,53	**557**	**557**	**0,00 %**	**2,45**
B-n38-k6-C13-V2	431	432	0,23 %	1,30	427	428	0,23 %	2,72
B-n39-k5-C13-V2	321	322	0,31 %	1,65	**317**	**317**	**0,00 %**	**3,07**
B-n41-k6-C14-V2	476	481	1,05 %	1,69	469	471	0,43 %	3,19
B-n43-k6-C15-V2	415	420	1,20 %	1,86	405	410	1,23 %	3,74
B-n44-k7-C15-V3	447	452	1,12 %	1,88	443	449	1,35 %	2,94
B-n45-k5-C15-V2	506	512	1,19 %	2,16	489	492	0,61 %	4,16
B-n45-k6-C15-V2	391	393	0,51 %	1,45	386	389	0,78 %	2,77
B-n50-k7-C17-V3	467	472	1,07 %	2,46	464	468	0,86 %	4,30
B-n50-k8-C17-V3	666	673	1,05 %	2,06	661	665	0,61 %	3,61
B-n51-k7-C17-V3	585	586	0,17 %	2,42	578	580	0,35 %	3,81
B-n52-k7-C18-V3	427	428	0,23 %	2,87	427	428	0,23 %	4,44
B-n56-k7-C19-V3	433	439	1,39 %	3,26	420	423	0,72 %	5,25
B-n57-k7-C19-V3	634	634	0,00 %	3,29	622	622	0,00 %	5,24
B-n57-k9-C19-V3	753	754	0,13 %	2,82	746	749	0,40 %	3,63
B-n63-k10-C21-V3	685	692	1,02 %	1,50	685	690	0,73 %	2,77
B-n64-k9-C22-V4	526	529	0,57 %	3,72	524	528	0,76 %	4,31
B-n66-k9-C22-V3	687	694	1,02 %	3,33	683	689	0,88 %	5,22
B-n67-k10-C23-V4	626	631	0,80 %	3,91	619	626	1,13 %	4,55
B-n68-k9-C23-V3	588	603	2,55 %	2,63	582	601	3,26 %	4,66
B-n78-k10-C26-V4	721	726	0,69 %	5,13	704	709	0,71 %	6,26

Tab. 8.1: Resultados del algoritmo propuesto para CluVRP SCC en instancias GVRP03 comparado con los resultados del algoritmo exacto de Battarra et al. [9] y resultados del algoritmo propuesto para CluVRP WCC en las mismas instancias comparado con los resultados del algoritmo propuesto por Defryn et al. [20].(*) Resultado de algoritmo de Defryn et al. [20] ya que el algoritmo exacto no reporto resultados para dicha instancia. En negrita los resultados que igualaron o mejoraron al resultado comparado. Continua en la siguiente página.

	CluVRP SCC				CluVRP WCC			
	Algoritmo Exacto Battarra	Mejor Resultado Alcanzado	Diferencia	Tiempo (s)	Metaheuristica TwoLevel Defryn	Mejor Resultado Alcanzado	Diferencia	Tiempo (s)
G-n262-k25-C88-V9	3310 (*)	3436	3,81 %	110,83	3.196	3343	4,60 %	141,08
M-n101-k10-C34-V4	**607**	**603**	**-0,66 %**	**21,35**	**598**	**595**	**-0,50 %**	**28,42**
M-n121-k7-C41-V3	691	702	1,59 %	17,86	681	695	2,06 %	58,92
M-n151-k12-C51-V4	804	818	1,74 %	33,28	759	784	3,29 %	51,39
M-n200-k16-C67-V6	914	939	2,74 %	76,73	874	898	2,75 %	54,69
P-n101-k4-C34-V2	679	688	1,33 %	14,26	649	665	2,47 %	31,93
P-n16-k8-C6-V4	**253**	**253**	**0,00 %**	**0,37**	**251**	**251**	**0,00 %**	**0,48**
P-n19-k2-C7-V1	**186**	**186**	**0,00 %**	**0,62**	170	171	0,59 %	3,06
P-n20-k2-C7-V1	200	201	0,50 %	0,61	177	178	0,56 %	3,45
P-n21-k2-C7-V1	190	191	0,53 %	0,65	179	180	0,56 %	1,46
P-n22-k2-C8-V1	202	204	0,99 %	0,83	183	184	0,55 %	1,91
P-n22-k8-C8-V4	**365**	**365**	**0,00 %**	**0,64**	**365**	**362**	**-0,82 %**	**2,97**
P-n23-k8-C8-V3	**279**	**279**	**0,00 %**	**0,62**	**270**	**269**	**-0,37 %**	**1,01**
P-n40-k5-C14-V2	396	399	0,76 %	1,77	381	382	0,26 %	4,27
P-n45-k5-C15-V2	440	442	0,45 %	2,09	422	423	0,24 %	4,10
P-n50-k10-C17-V4	491	492	0,20 %	2,26	471	471	0,00 %	3,12
P-n50-k7-C17-V3	447	448	0,22 %	2,49	**430**	**431**	**0,23 %**	**9,16**
P-n50-k8-C17-V3	460	462	0,43 %	1,92	**441**	**441**	**0,00 %**	**3,36**
P-n51-k10-C17-V4	537	540	0,56 %	2,26	493	494	0,20 %	3,25
P-n55-k10-C19-V4	500	502	0,40 %	2,95	**481**	**481**	**0,00 %**	**4,19**
P-n55-k15-C19-V6	595	595	0,00 %	6,85	572	573	0,17 %	6,66
P-n55-k7-C19-V3	462	465	0,65 %	3,72	456	458	0,44 %	5,11
P-n55-k8-C19-V3	471	474	0,64 %	3,16	456	458	0,44 %	4,03
P-n60-k10-C20-V4	552	556	0,72 %	2,99	535	539	0,75 %	3,72
P-n60-k15-C20-V5	611	630	3,11 %	0,77	591	603	2,03 %	0,39
P-n65-k10-C22-V4	619	622	0,48 %	3,56	**582**	**582**	**0,00 %**	**4,38**
P-n70-k10-C24-V4	643	647	0,62 %	9,38	601	606	0,83 %	5,06
P-n76-k4-C26-V2	581	585	0,69 %	7,27	557	571	2,51 %	12,01
P-n76-k5-C26-V2	581	585	0,69 %	7,08	556	567	1,98 %	10,69

Tab. 8.2: Continuación de tabla 8.1

	CluVRP SCC				CluVRP WCC			
	Algoritmo Exacto Battarra	**Mejor Resultado Alcanzado**	**Diferencia**	**Tiempo (s)**	**Metaheuristica TwoLevel Defryn**	**Mejor Resultado Alcanzado**	**Diferencia**	**Tiempo (s)**
Golden_1-C17-N241	4831	4952	2,50 %	17,53	**4667**	**4664**	**-0,06 %**	**43,36**
Golden_1-C18-N241	4847	4972	2,58 %	13,11	**4711**	**4667**	**-0,93 %**	**42,94**
Golden_1-C19-N241	4872	4978	2,18 %	13,20	**4730**	**4674**	**-1,18 %**	**43,80**
Golden_1-C21-N241	4889	4981	1,88 %	13,03	**4752**	**4672**	**-1,68 %**	**39,41**
Golden_1-C22-N241	4908	5021	2,30 %	13,26	**4694**	**4650**	**-0,94 %**	**41,41**
Golden_1-C25-N241	4899	4984	1,74 %	14,36	**4717**	**4672**	**-0,95 %**	**39,95**
Golden_1-C27-N241	4934	4996	1,26 %	14,47	**4697**	**4667**	**-0,64 %**	**43,54**
Golden_1-C31-N241	5050	5093	0,85 %	15,84	**4763**	**4661**	**-2,14 %**	**43,85**
Golden_1-C35-N241	5102	5164	1,22 %	16,67	**4707**	**4635**	**-1,53 %**	**43,93**
Golden_1-C41-N241	5097	5182	1,67 %	23,38	**4695**	**4690**	**-0,11 %**	**49,99**
Golden_1-C49-N241	5000	5086	1,72 %	30,36	463	4659	0,54 %	60,64
Golden_5-C14-N201	7622	7768	1,92 %	15,71	**7082**	**7060**	**-0,31 %**	**32,95**
Golden_5-C15-N201	7424	7582	2,13 %	13,96	**6794**	6833	0,57 %	8,58
Golden_5-C16-N201	7491	7568	1,03 %	15,05	6823	6910	1,28 %	13,42
Golden_5-C17-N201	7434	7599	2,22 %	15,30	6868	6880	0,17 %	6,68
Golden_5-C19-N201	7576	7673	1,28 %	11,70	7028	7054	0,37 %	38,98
Golden_5-C21-N201	7596	7660	0,84 %	11,04	**6912**	**6856**	**-0,81 %**	**39,62**
Golden_5-C23-N201	7643	7742	1,30 %	11,47	6777	6783	0,09 %	39,38
Golden_5-C26-N201	7560	7641	1,07 %	13,05	6719	6751	0,48 %	37,81
Golden_5-C29-N201	7410	7449	0,53 %	14,14	6744	6751	0,10 %	35,87
Golden_5-C34-N201	7429	7469	0,54 %	16,21	6711	6756	0,67 %	38,75
Golden_5-C41-N201	7241	7329	1,22 %	19,69	**6575**	**6573**	**-0,03 %**	**44,64**
Golden_8-C30-N441	10866	11170	2,80 %	43,87	10519	10577	0,55 %	249,14
Golden_8-C32-N441	10831	11187	3,29 %	42,82	**10531**	**10501**	**-0,28 %**	**236,14**
Golden_8-C34-N441	10847	11151	2,80 %	40,06	**10569**	**10480**	**-0,84 %**	**227,49**
Golden_8-C37-N441	10859	11202	3,16 %	37,01	**10514**	**10487**	**-0,26 %**	**215,49**
Golden_8-C41-N441	10934	11232	2,73 %	38,79	10544	10571	0,26 %	234,10
Golden_8-C45-N441	10960	11307	3,17 %	39,79	**10529**	**10513**	**-0,15 %**	**259,89**
Golden_8-C49-N441	11042	11396	3,21 %	35,51	**10603**	**10491**	**-1,06 %**	**265,13**
Golden_8-C56-N441	11194	11551	3,19 %	40,81	10578	10616	0,36 %	262,72
Golden_8-C63-N441	11252	11600	3,09 %	53,67	**10596**	**10542**	**-0,51 %**	**241,63**
Golden_8-C74-N441	11321	11712	3,45 %	78,16	**10622**	**10619**	**-0,03 %**	**255,52**
Golden_8-C89-N441	11209	11640	3,85 %	118,51	**10526**	**10429**	**-0,92 %**	**298,80**
Golden_13-C17-N253	552	557	0,91 %	14,06	**537**	**535**	**-0,37 %**	**81,41**
Golden_13-C19-N253	549	554	0,91 %	13,23	**527**	**527**	**0,00 %**	**83,62**
Golden_13-C20-N253	548	553	0,91 %	13,27	**530**	**528**	**-0,38 %**	**74,56**
Golden_13-C22-N253	548	550	0,36 %	13,06	**529**	**528**	**-0,19 %**	**68,22**
Golden_13-C23-N253	548	553	0,91 %	12,54	526	528	0,38 %	66,19
Golden_13-C26-N253	542	549	1,29 %	12,58	527	529	0,38 %	74,75
Golden_13-C29-N253	540	546	1,11 %	13,67	526	528	0,38 %	80,88
Golden_13-C32-N253	543	548	0,92 %	14,53	**527**	**527**	**0,00 %**	**79,62**
Golden_13-C37-N253	545	552	1,28 %	17,79	526	530	0,76 %	81,15
Golden_13-C43-N253	553	560	1,27 %	22,90	526	532	1,14 %	87,25
Golden_13-C51-N253	560	568	1,43 %	30,59	527	529	0,38 %	85,94
Golden_16-C33-N481	1030	1046	1,55 %	32,37	**1030**	**1010**	**-1,94 %**	**267,25**
Golden_16-C35-N481	1028	1045	1,65 %	28,92	1001	1010	0,90 %	256,01
Golden_16-C37-N481	1028	1048	1,95 %	28,21	998	1008	1,00 %	218,13
Golden_16-C41-N481	1032	1052	1,94 %	30,59	1006	1011	0,50 %	217,91
Golden_16-C44-N481	1028	1053	2,43 %	31,93	1004	1014	1,00 %	210,34
Golden_16-C49-N481	1031	1057	2,52 %	35,97	1001	1016	1,50 %	209,76
Golden_16-C54-N481	1022	1050	2,74 %	38,85	996	1011	1,51 %	231,68
Golden_16-C61-N481	1013	1050	3,65 %	47,58	991	1014	2,32 %	232,14
Golden_16-C69-N481	1012	1050	3,75 %	57,15	991	1012	2,12 %	241,92
Golden_16-C81-N481	1018	1056	3,73 %	86,36	993	1015	2,22 %	245,17

Tab. 8.3: Resultados del algoritmo propuesto para CluVRP SCC en instancias Golden-Battarra comparado con los resultados del algoritmo exacto de Battarra et al. [9] y resultados del algoritmo propuesto para CluVRP WCC en las mismas instancias comparado con los resultados del algoritmo propuesto por Defryn et al. [20]. En negrita los resultados que igualaron o mejoraron al resultado comparado. Continua en la siguiente página.

	CluVRP SCC				CluVRP WCC			
	Algoritmo Exacto Battarra	**Mejor Resultado Alcanzado**	**Diferencia**	**Tiempo (s)**	**Metaheuristica TwoLevel Defryn**	**Mejor Resultado Alcanzado**	**Diferencia**	**Tiempo (s)**
Golden_16-C97-N481	1018	1064	4,52 %	131,52	993	1028	3,52 %	289,38
Golden_20-C29-N421	1220	1232	0,98 %	42,54	**1198**	**1182**	**-1,34 %**	**55,52**
Golden_20-C31-N421	1232	1242	0,81 %	35,89	**1213**	**1197**	**-1,32 %**	**58,70**
Golden_20-C33-N421	1208	1223	1,24 %	34,53	**1191**	**1188**	**-0,25 %**	**65,52**
Golden_20-C36-N421	1059	1082	2,17 %	36,42	1019	1022	0,29 %	154,32
Golden_20-C39-N421	1052	1062	0,95 %	30,68	**1014**	**1010**	**-0,39 %**	**176,02**
Golden_20-C43-N421	1052	1077	2,38 %	30,63	1007	1015	0,79 %	152,60
Golden_20-C47-N421	1053	1077	2,28 %	33,64	1012	1017	0,49 %	152,76
Golden_20-C53-N421	1058	1086	2,65 %	41,72	1010	1013	0,30 %	153,11
Golden_20-C61-N421	1058	1087	2,74 %	54,73	1008	1020	1,19 %	150,17
Golden_20-C71-N421	1059	1094	3,31 %	76,54	1009	1016	0,69 %	159,01
Golden_20-C85-N421	1049	1089	3,81 %	106,62	1006	1030	2,39 %	181,39

Tab. 8.4: Continuación de tabla 8.3

	CluVRP SCC				CluVRP WCC		
	Mejor Resultado Conocido	**Mejor Resultado Alcanzado**	**Diferencia**	**Tiempo (s)**	**Mejor Resultado Alcanzado**	**Diferencia con resultados de CluVRP SCC**	**Tiempo (s)**
Golden_rho10_01	5759,25	5973,07	3,71 %	115,54	5957,76	3,45 %	127,38
Golden_rho10_02	9247,92	9538,81	3,15 %	78,64	9164,54	-0,90 %	96,12
Golden_rho10_03	12904,6	13378,22	3,67 %	66,02	12340,55	-4,37 %	96,48
Golden_rho10_04	17810,4	18303,83	2,77 %	73,27	16656,26	-6,48 %	114,95
Golden_rho10_05	8960,31	9241,22	3,14 %	18,68	7551,08	-15,73 %	29,47
Golden_rho10_06	10976,5	11306,33	3,00 %	30,37	9796,71	-10,75 %	46,91
Golden_rho10_07	12485,8	12968,6	3,87 %	50,16	11857,19	-5,03 %	71,47
Golden_rho10_08	13331,2	13908,19	4,33 %	61,73	13061,38	-2,02 %	97,21
Golden_rho10_09	710,64	721,91	1,59 %	9,07	686,36	-3,42 %	13,87
Golden_rho10_10	908,89	918,24	1,03 %	11,66	873,99	-3,84 %	19,57
Golden_rho10_11	1139,51	1154,55	1,32 %	15,35	1090,33	-4,32 %	26,19
Golden_rho10_12	1384,29	1401,47	1,24 %	19,84	1339,39	-3,24 %	36,09
Golden_rho10_13	1030,42	1035,68	0,51 %	26,56	1006,87	-2,29 %	26,38
Golden_rho10_14	1324,96	1344,14	1,45 %	37,03	1291,34	-2,54 %	35,03
Golden_rho10_15	1668,39	1692,87	1,47 %	43,52	1623,86	-2,67 %	38,67
Golden_rho10_16	2053,47	2071,68	0,89 %	57,17	1986,01	-3,29 %	57,12
Golden_rho10_17	840,53	871,42	3,68 %	94,42	848,21	0,91 %	93,83
Golden_rho10_18	1097,51	1147,51	4,56 %	194,43	1133,65	3,29 %	197,06
Golden_rho10_19	1522,83	1600,9	5,13 %	328,60	1576,85	3,55 %	317,02
Golden_rho10_20	2019,55	2167,93	7,35 %	575,63	2140,46	5,99 %	613,47
Golden_rho25_01	6051,04	6122,36	1,18 %	9,76	5853,32	-4,39 %	17,62
Golden_rho25_02	9725,9	9938,45	2,19 %	11,76	9229,08	-7,14 %	30,55
Golden_rho25_03	13692,6	13915,42	1,63 %	17,10	12856,02	-7,61 %	52,78
Golden_rho25_04	16977,9	17345,45	2,16 %	23,18	15749,61	-9,20 %	75,64
Golden_rho25_05	9340,7	9538,64	2,12 %	7,11	7989,38	-16,24 %	23,74
Golden_rho25_06	10840,7	11053,45	1,96 %	10,61	10014,56	-9,40 %	35,29
Golden_rho25_07	12348,1	12559,01	1,71 %	15,10	11400,91	-9,22 %	45,88
Golden_rho25_08	14100,8	14473,03	2,64 %	20,08	13186,94	-8,89 %	57,01
Golden_rho25_09	717,63	724,06	0,90 %	11,68	687,64	-5,03 %	17,78
Golden_rho25_10	908,26	920,45	1,34 %	14,12	868,87	-5,60 %	24,19
Golden_rho25_11	1131,84	1145,9	1,24 %	19,34	1093,9	-4,54 %	32,84
Golden_rho25_12	1387,67	1402,85	1,09 %	24,88	1344,94	-4,13 %	42,81
Golden_rho25_13	1034,3	1043,02	0,84 %	32,00	1006,55	-3,50 %	30,69
Golden_rho25_14	1317,05	1326,55	0,72 %	43,94	1267,62	-4,44 %	42,87
Golden_rho25_15	1667,08	1691,52	1,47 %	54,22	1601,78	-5,31 %	50,18
Golden_rho25_16	2048,08	2068,26	0,99 %	68,11	1982,03	-4,17 %	62,17
Golden_rho25_17	795,33	820,07	3,11 %	40,99	807,08	-1,58 %	39,31
Golden_rho25_18	1122,73	1164,56	3,73 %	65,51	1126,62	-3,26 %	62,79
Golden_rho25_19	1538,2	1593,86	3,62 %	95,82	1546,25	-2,99 %	90,61
Golden_rho25_20	2036,19	2103,45	3,30 %	113,63	2068,91	-1,64 %	110,26
Golden_rho50_01	6551,04	6732,55	2,77 %	11,27	6329,64	-5,98 %	18,80
Golden_rho50_02	9787,09	9996,49	2,14 %	17,02	9414,96	-5,82 %	40,26
Golden_rho50_03	13287	13611,11	2,44 %	34,42	12826,26	-5,77 %	56,42
Golden_rho50_04	17569,9	17952,77	2,18 %	60,01	16877,96	-5,99 %	88,79
Golden_rho50_05	8597,31	8656,9	0,69 %	15,27	8164,75	-5,69 %	28,06
Golden_rho50_06	10550,8	10752,05	1,91 %	20,38	10250,86	-4,66 %	40,01
Golden_rho50_07	12673,7	12898,62	1,77 %	27,42	12067,07	-6,45 %	56,65
Golden_rho50_08	13766,3	13970,91	1,49 %	42,68	13136,64	-5,97 %	69,61
Golden_rho50_09	698,04	709,02	1,57 %	10,57	670,11	-5,49 %	16,58
Golden_rho50_10	890,87	905,67	1,66 %	16,97	864,22	-4,58 %	23,29
Golden_rho50_11	1107,88	1121,38	1,22 %	22,99	1068,46	-4,72 %	31,90
Golden_rho50_12	1317,47	1329,67	0,93 %	27,31	1275	-4,11 %	45,20
Golden_rho50_13	1053,47	1060,47	0,66 %	15,39	1016,26	-4,17 %	13,00
Golden_rho50_14	1342,7	1358,77	1,20 %	17,75	1291,12	-4,98 %	16,97
Golden_rho50_15	1657,22	1682,02	1,50 %	33,32	1601,7	-4,78 %	24,54
Golden_rho50_16	2003,1	2033,37	1,51 %	33,19	1919,66	-5,59 %	29,18
Golden_rho50_17	881,66	895,98	1,62 %	9,74	864,02	-3,57 %	11,30
Golden_rho50_18	1200,98	1223,3	1,86 %	13,94	1185,41	-3,10 %	15,98
Golden_rho50_19	1612,33	1657,74	2,82 %	19,13	1613,26	-2,68 %	21,37

Tab. 8.5: Resultados del algoritmo propuesto para CluVRP SCC en instancias Golden-Izquierdo comparado con los resultados obtenidos por los trabajos de Expósito-Izquierdo et al. [19] y Defryn et al. [20] (ambos algoritmos no exactos) y resultados del algoritmo propuesto para CluVRP WCC comparados con los resultados obtenidos por el algoritmo propuesto para CluVRP SCC (no existen trabajos para CluVRP WCC en este grupo de instancias). Continua en la siguiente página.

	CluVRP SCC				CluVRP WCC		
Instancia	**Mejor Resultado Conocido**	**Mejor Resultado Alcanzado**	**Diferencia**	**Tiempo (s)**	**Mejor Resultado Alcanzado**	**Diferencia con resultados de CluVRP SCC**	**Tiempo (s)**
Golden_rho50_20	2278,64	2360,87	3,61 %	31,63	2256,63	-4,42 %	30,53
Golden_rho75_01	6736,15	6758,77	0,34 %	16,28	6758,09	-0,01 %	16,23
Golden_rho75_02	10204,3	10320,23	1,14 %	31,87	10314,87	-0,05 %	23,83
Golden_rho75_03	13575,7	13818,26	1,79 %	46,80	13780,95	-0,27 %	39,63
Golden_rho75_04	17077,59	17294,67	1,27 %	69,96	17285,53	-0,05 %	66,64
Golden_rho75_05	8664,94	8801,3	1,57 %	23,46	8699,68	-1,15 %	21,28
Golden_rho75_06	11452,01	11595,96	1,26 %	41,16	11591,55	-0,04 %	33,15
Golden_rho75_07	12901,41	13079,63	1,38 %	43,62	13030,13	-0,38 %	42,54
Golden_rho75_08	13926,4	14039,55	0,81 %	56,17	13973,36	-0,47 %	57,32
Golden_rho75_09	773,39	776,93	0,46 %	13,46	771,11	-0,75 %	15,16
Golden_rho75_10	1000,51	1006,65	0,61 %	18,06	1006,68	0,00 %	18,07
Golden_rho75_11	1223,66	1231,11	0,61 %	26,83	1229,17	-0,16 %	25,06
Golden_rho75_12	1475,68	1486,4	0,73 %	33,91	1484,43	-0,13 %	34,29
Golden_rho75_13	1183,12	1186,82	0,31 %	8,72	1183,55	-0,28 %	8,11
Golden_rho75_14	1520,55	1524,51	0,26 %	12,48	1479,4	-2,96 %	12,20
Golden_rho75_15	1825,29	1835,05	0,53 %	16,31	1831,28	-0,21 %	16,19
Golden_rho75_16	2265,54	2274,97	0,42 %	19,53	2274,28	-0,03 %	18,62
Golden_rho75_17	1001,02	1001,02	0,00 %	6,31	1001,32	0,03 %	6,62
Golden_rho75_18	1392,15	1394,17	0,15 %	8,41	1393,15	-0,07 %	8,65
Golden_rho75_19	1951,77	1952,14	0,02 %	13,66	1929,73	-1,15 %	11,48
Golden_rho75_20	2540,22	2545,73	0,22 %	13,19	2539,45	-0,25 %	14,09
Golden_rho100_01	6293,04	6325,97	0,52 %	40,90	6314,62	-0,18 %	42,63
Golden_rho100_02	9879,59	9982,24	1,04 %	47,23	9980,63	-0,02 %	57,33
Golden_rho100_03	12361,09	12549,04	1,52 %	99,45	12506,68	-0,34 %	80,35
Golden_rho100_04	16130,39	16429,09	1,85 %	111,19	16357,81	-0,43 %	102,26
Golden_rho100_05	8394,11	8442,01	0,57 %	35,43	8440,09	-0,02 %	29,22
Golden_rho100_06	10777,33	10867,74	0,84 %	48,44	10820,1	-0,44 %	41,72
Golden_rho100_07	11346,11	11603,01	2,26 %	82,14	11516,64	-0,74 %	65,01
Golden_rho100_08	13188,94	13365,43	1,34 %	93,54	13383,22	0,13 %	70,42
Golden_rho100_09	705,19	708,88	0,52 %	17,20	704,62	-0,60 %	41,34
Golden_rho100_10	837,52	845,17	0,91 %	24,51	843,13	-0,24 %	22,09
Golden_rho100_11	1054,13	1064,4	0,97 %	48,76	1064,81	0,04 %	30,55
Golden_rho100_12	1297,31	1308,2	0,84 %	60,63	1309,71	0,12 %	44,45
Golden_rho100_13	996,36	999,79	0,34 %	9,98	996,81	-0,30 %	22,70
Golden_rho100_14	1223,09	1227,1	0,33 %	27,34	1229,09	0,16 %	13,34
Golden_rho100_15	1531,29	1545,04	0,90 %	18,40	1536,63	-0,54 %	25,31
Golden_rho100_16	1874,69	1889,68	0,80 %	39,18	1881,54	-0,43 %	20,85
Golden_rho100_17	844,27	846,38	0,25 %	17,37	846,44	0,01 %	17,57
Golden_rho100_18	1212,97	1219,58	0,54 %	10,71	1216,81	-0,23 %	10,77
Golden_rho100_19	1667,45	1670,91	0,21 %	12,95	1669,44	-0,09 %	12,90
Golden_rho100_20	2128,6	2134,67	0,29 %	15,78	2134,09	-0,03 %	15,88

Tab. 8.6: Continuación de tabla 8.5

Bibliografía

[1] D. L. Applegate, R. E. Bixby, V. Chvátal, W. Cook, **The Traveling Salesman Problem: A Computational Study, Princeton Series in Applied**, Mathematics, Princeton University Press, 2006.

[2] I. Kara, H. Guden, O. N. Koc, **New Formulations for the Generalized Traveling Salesman Problem**, 5th EURO/INFORMS Joint International Meeting, Istanbul, Turkey, 2003.

[3] A. Soifer, **Algoritmos de Colonia de Hormigas para el Problema del Viajante de Comercio por Familias y para el Problema de Ruteo de Vehículos por Familias**, Tesis de Licenciatura, Departamento de Computación, FCEyN, Universidad de Buenos Aires, 2016.

[4] P. Toth, D.Vigo, **The Vehicle Routing Problem**, Society for Industrial and Applied Mathematics, Philadelphia, United States, 2002.

[5] R. Fukasawa, H. Longo, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, R. Werneck, **Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem**, Mathematical Programming, Ser. A 106, 491–511, 2006.

[6] M. Gendreau, J. Y. Potvin, **Handbook of Metaheuristics**, International Series in Operations Research & Management Science, vol. 146, Springer, 2010.

[7] T.A. Feo and M.G.C. Resende **A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters**, 1989.

[8] M. Sevaux, K. Sörensen, **Hamiltonian paths in large clustered routing problems**, Proceedings of the EU/MEeting 2008 workshop on Metaheuristics for Logistics and Vehicle Routing, EU/ME, vol. 8, p 411–417, 2008.

[9] M. Batarra, G. Erdogan, D. Vigo **Exact algorithms for the clustered vehicle routing problem**, Operations Research, vol. 62, p 58–71 2014.

[10] P. Toth, D.Vigo, **The Vehicle Routing Problem**, Society for Industrial and Applied Mathematics, Philadelphia, United States, 2002.

[11] T. Barthelemy, A. Rossi, M. Sevaux, K. Sörensen, **Metaheuristic approach for the clustered VRP**, EU/MEeting: 10th Anniversary of the Metaheuristics, Community-Université de Bretagne Sud, France, 2010.

[12] M. H. Ha, N. Bostel, A. Langevin, L. M. Rousseau, **An Exact Algorithm and a Metaheuristic for the Generalized Vehicle Routing Problem**, Computers and Operations Research, vol. 43, Elsevier Publishing, p 9-19, 2014.

[13] P.C. Pop, I. Kara, A.H. Marc, A.H., 2012, **New mathematical models of the generalized vehicle routing problem and extensions**, Applied Mathematical Modelling, vol. 36, p 97–107, 2012.

[14] G. Ghiani, G. Improta, **An efficient transformation of the generalized vehicle routing problem**, European Journal of Operational Research, ol. 122, p 11–17, 2010.

[15] A.H. Marc, L. Fuksz, P.C. Pop, D. Danciulescu, **A novel hybrid algorithm for solving the clustered vehicle routing problem**, Hybrid Artificial Intelligent Systems, Springer, p 679–689, 2015.

[16] T. Vidal, M. Battarra, A. Subramanian, G. Erdogan, **Hybrid metaheuristics for the clustered vehicle routing problem** Computers And Operations Research, vol. 58, p 87–99, 2015.

[17] A. Subramanian, **Heuristic, Exact and Hybrid Approaches for Vehicle Routing Problems**, University Federal Fluminense, Niterois (Brazil), Ph.D. thesis, 2012.

[18] C. Defryn, K. Sörensen, **A Two-Level Variable Neighbourhood Search for the Euclidean Clustered Vehicle Routing Problem** Research paper, University of Antwerp, Faculty of Applied Economics, 2015.

[19] C. Expósito-Izquierdo, A. Rossi, M. Sevaux, **A two-level solution approach to solve the clustered capacitated vehicle routing problem**, Computers & Industrial Engineering, vol. 91, p 274–289, 2016.

[20] C. Defryn, K. Sörensen, **A fast two-level variable neighborhood search for the clustered vehicle routing problem**, Computers & Operations Research, vol. 83 p 78–94, 2017.

[21] S. H. Zanakis, J. R. Evans, **Heuristic "Optimization": Why, When, and How to Use It**, Interfaces, vol. 11, issue 5, p 84-91, 1981.

[22] E. A. Silver, **A tutorial on Heuristic Methods**, European Journal of Operational Reseach, Vol. 5, 1980.

[23] A. G. Sánchez, **Las técnicas metaheuristicas**, Escuela Politécnica Superior de Ingenieros Industriales, Universidad Politécnica de Madrid, 2013.

[24] S.M, Sadiq, Y Habib, **Iterative Computer Algorithms with Applications in Engineering**, Solving Combinatorial Optimization Problems, Wiley, 1999.

[25] T.A. Feo, M.G.C. Resende, **Greedy randomized adaptive search procedures. Journal of Global Optimization**, 1995.

[26] L.F. Morán-Mirabal, J.L. Gonzalez-Velarde, M.G.C. Resende, **Randomized heuristics for the family traveling salesperson problem**, International Transactions in Operational Research, 2013.

[27] M. Resende, C. Ribeiro, **Optimization by GRASP**, Springer-Verlag New York, 2016.

[28] M.R. Garey, D.S. Johnson, **"Strong" NP-Completeness results: motivation, examples, and implications**, Journal of the ACM (JACM), vol. 25 (3), p 499–508, 1978.

[29] G. Dósa1, J. Sgall, **Optimal analysis of Best Fit bin packing**, Department of Mathematics, University of Pannonia, Veszprém, Hungary, 2014

[30] T. Bektas, G. Laporte, **The Pollution-Routing Problem** Transportation Research, Part B, Methodological, vol. 45(8) p 1232-1250, September , 2011.

[31] B.L. Golden, E.A. Wasil, J. P. Kelly, I. Chao, **Metaheuristics in vehicle routing**, Fleet Management and Logistics, p 33–56, Kluwer, Boston, 1998.