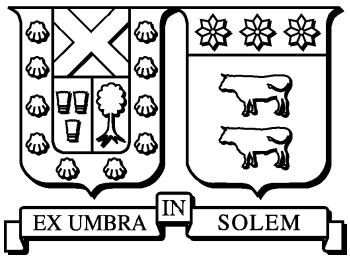
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

SANTIAGO – CHILE



"UN MÉTODO GRASP PARA EL VEHICLE ROUTING PROBLEM CON CROSS-DOCKS"

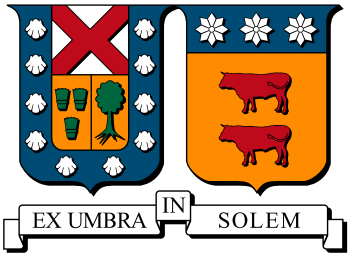
JOSE-MANUEL IGNACIO URTASUN RAMOS

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: ELIZABETH MONTERO

NOVIEMBRE 2018

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE



"UN MÉTODO GRASP PARA EL VEHICLE ROUTING PROBLEM CON CROSS-DOCKS"

JOSE-MANUEL IGNACIO URTASUN RAMOS

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: ELIZABETH MONTERO

PROFESOR CORREFERENTE: CLAUDIO LOBOS

NOVIEMBRE 2018

Agradecimientos

Agradezco a mi familia, en especial a mis padres por estar siempre conmigo, brindarme su apoyo incondicional en lo académico y en la vida en general, y por todo lo que han hecho por mi, ya que todo lo que soy se los debo en gran parte a ustedes. Agradezco a mis abuelos Vicente y Elba por haberme apoyado y ayudado cada vez que lo necesité, y quiero dedicarles en gran parte este trabajo, ya que se lo que significa para ustedes el hecho de que logré ser un buen profesional y estoy muy orgulloso de haber podido cumplir ese sueño.

También agradezco a cada uno de mis amigos, ya que para mi son como mi segunda familia, a Gaston, Sebastian, Felipe que son como mis hermanos y siempre han estado conmigo desde los momentos más difíciles hasta los más alegres. Agradezco también a mi grupo de amigos de la universidad, ya que ellos fueron un pilar fundamental en esta etapa, la cual no fue para nada sencilla, pero que gracias a ellos muchas cosas se volvieron más simples y me dieron la energía para llegar hasta el final.

Finalmente quiero agradecer a la profesora Elizabeth Montero, que me brindó muchísima ayuda y me guió e inspiró a realizar esta memoria. Agradecer su tiempo, dedicación, carisma y alegría, lo cual me impulsaba a seguir probando diferentes opciones y que finalmente nos llevó a crear un gran trabajo.

Resumen

En las operaciones de logística a gran escala, intervienen varios procesos tales como la recolección de recursos o bienes, el traslado a bodegas centrales en donde se pueden guardar un periodo limitado de tiempo, y finalmente el proceso de entrega de dichos bienes a los consumidores. Cada uno de los procesos conlleva un costo, por lo que es de vital importancia realizar cada uno de éstos de manera óptima. Es por esto que el foco central de esta memoria es el Vehicle Routing Problem with Cross-Docks, el cual consiste en un conjunto de clientes que tienen pedidos respecto a distintos proveedores, y se cuenta con una flota de vehículos los cuales son encargados de recolectar los bienes desde éstos últimos, llevarlos a un muelle central y coordinar el reparto con otros vehículos para realizar el despacho de estos bienes a sus respectivos clientes. El objetivo de este problema consiste en encontrar las rutas óptimas para cada vehículo, y su coordinación en las bodegas o muelles centrales, con el fin de minimizar los costos asociados a los traslados, inventario y uso de vehículos.

Para solucionar este problema, se propone un método basado en Greedy Randomized Adaptative Search Procedure (GRASP), con un método adaptivo. Los resultados computacionales obtenidos resultaron ser eficaces para algunas instancias pequeñas, encontrando las mejores soluciones obtenidas hasta la fecha, mientras que para instancias más grandes encuentra soluciones aceptables en un corto período de tiempo.

Índice de Contenidos

Índice de tablas

Índice de figuras

Introducción

Hoy en día, la logística presenta una importancia significativa, posicionándose como un área específica para su tratamiento. Durante las últimas décadas su planteamiento ha ido evolucionando constantemente, desde el concepto de distribución como variable básica del sistema de comercialización de una empresa, hasta el punto de convertirse en una herramienta clave en la economía actual, según el enfoque global de los mercados. La logística comenzó meramente alineada con la consecución del producto concreto, en el sitio justo, en el tiempo oportuno y al menor coste posible. En la actualidad este conjunto de actividades han sido redefinidas y hoy en día son todo un proceso, en ocasiones, de alta complejidad. Bajo este argumento general, queda clara la importancia de la logística en el desarrollo del área comercial, por lo que se debe profundizar en el desarrollo de las fases de análisis del mercado y distribución bajo un perfil de optimización de los costes que incluyen estos procesos.

Según [23] a Chile le falta aun por mejorar sus procesos de logística y deja claro esto bajo la siguiente cita: “En puertos, estamos hablando del T2, de ampliar Valparaíso, de ampliar San Antonio, pero hay empresas argentinas que tienen problemas de transporte y quieren pasar por acá, pero no pueden sacar sus cargas...”.

Es por esto, que es necesario profundizar en estos procesos, optimizando los métodos de distribución y minimizando los costos, por ejemplo, en las empresas de distribución más grandes sería de utilidad optimizar el recorrido de los camiones que transportan los productos, así como su coordinación en las bodegas de almacenamiento, con el fin de minimizar los costos de inventario y disminuir los tiempos de despacho de productos. Debido a la diversidad de los problemas de logística en los escenarios reales, el problema de ruteo de vehículos, o mejor conocido por sus siglas como VRP ha sufrido diversas modificaciones, con el fin de

que pueda ser aplicado para cada situación en particular y apoye en los procesos de logística correspondientes.

En sus comienzo el VRP solo consideraba un depósito central en donde se ubican vehículos homogéneos los cuales deben realizar una ruta de recogida por los proveedores, y luego repartirlos a los clientes según sus demandas correspondientes, como se explica en [3]. Luego al pasar el tiempo comienzan a surgir diversas variaciones con el fin de que el modelo propuesto se adapte a la realidad, dando origen al VRPCD, el cual considera un muelle central en donde los productos pueden ser dejados en inventario o bien ser descargados y cargados hacia otros vehículos para luego ser entregados a los clientes. Posteriormente surgieron trabajos en los cuales se añaden intervalos de tiempo para las entregas, múltiples muelles de intercambio, vehículos heterogéneos, entre otras características para escenarios más reales como por ejemplo [14]. En el siguiente trabajo se abordará el VRPCD, añadiendo una serie de restricciones con el fin de abarcar un escenario realista.

El contenido de este documento es organizado de la siguiente manera. En el capítulo 1, se estudiará el VRPCD en detalle, indicando cada una de sus componentes, características y restricciones a considerar, junto con el modelo matemático correspondiente. Una revisión de la literatura, explicando los acercamientos más relevantes de dicho problema y sus resultados son expuestos en el capítulo 2. Luego, en el capítulo 3 se presenta la descripción detallada de la propuesta de solución al problema, el cual consiste en un algoritmo basado en la metaheurística Greedy Randomized Adaptative Search Procedure (GRASP). Posteriormente, en el capítulo 4 se describen las instancias generadas y las características de los experimentos que se realizaron para validar la propuesta de solución, junto con los resultados obtenidos y los análisis correspondientes. Finalmente se realizan las conclusiones pertinentes de todo el trabajo en el capítulo 5.

Capítulo 1

Definición del Problema

El problema de VRPCD consiste en calcular las rutas óptimas para un conjunto de vehículos, los cuales deben retirar bienes desde uno o varios proveedores (vehículos de carga), luego deben pasar por un muelle de intercambio en donde los bienes son descargados y cargados en los vehículos de reparto, los cuales deben finalmente entregarlos a los distintos clientes de acuerdo a las demandas (conocidas) de cada uno. El objetivo del VRPCD es optimizar el recorrido de los vehículos y coordinar su llegada y salida a los muelles, de manera de minimizar los costos asociados a los viajes y carga (descarga) de los bienes, satisfaciendo todas las demandas de los clientes.

La descripción detallada del VRPCD abordado en esta memoria se explica a continuación. Se cuenta con un conjunto de proveedores, un conjunto de clientes, y al menos un muelle de intercambio. Cada cliente tiene una demanda (pedidos) de bienes definida respecto a un proveedor. Se asumen que los bienes se encuentran organizados en cajas o conjuntos de tamaño uniforme, y que la cantidad de dichos pedidos se encuentran expresados en números enteros respecto a estos conjuntos.

Por otro lado, cada muelle de intercambio se encuentra en una posición determinada (generalmente y para el caso de este trabajo entre los proveedores y clientes), los cuales serán ocupados por los vehículos que descargan y cargan bienes respectivamente. Se considera que los bienes que son descargados desde algún vehículo deben ser inmediatamente cargados en

otro. Por último cada muelle tiene una capacidad ilimitada de vehículos que pueden realizar las consolidaciones en el mismo instante de tiempo.

El VRPCD abordado en esta memoria considera también ventanas (intervalos) de tiempo pre-definidas para los proveedores, muelles y clientes, de tal manera que la recolección de bienes, su traspaso entre vehículos, y la entrega de éstos debe comenzar y terminar dentro de dichos intervalos.

Para el caso de las entregas, se cuenta con una flota de vehículos homogéneos, los cuales poseen una capacidad determinada. Los vehículos consideran un tiempo de carga y descarga de bienes en los muelles, además de un tiempo fijo para la preparación de estas acciones.

También se asume que todos los vehículos parten en un depósito inicial que convenientemente podría ser un muelle. Así mismo todos los vehículos deben finalizar su recorrido en el mismo depósito.

Una ruta es definida como el conjunto de nodos que forman un camino desde el nodo inicial (depósito) en donde se encuentran los vehículos, pasando por un conjunto de clientes y proveedores, hasta el nodo final (el mismo que el inicial). Se asume que cada vehículo puede tener a lo más una ruta. Los vehículos pueden viajar desde un proveedor a otro proveedor, o a un muelle de intercambio. Si un vehículo se encuentra en un muelle, puede viajar a otro muelle, o a un cliente. Por último, si un vehículo se encuentra en un nodo cliente, puede visitar otros clientes o terminar su ruta en el depósito.

Finalmente se cuenta con la restricción de que cada nodo puede ser visitado solo una vez por un solo vehículo.

El problema, formulado de manera formal, consiste en determinar la cantidad de vehículos a utilizar, sus respectivas rutas, y la cantidad de bienes cargados y descargados para cada uno de ellos en cada muelle por el que pasa. El objetivo del problema planteado es minimizar el costo fijo y los costos de transporte asociados a los vehículos utilizados, satisfaciendo todas las demandas de los clientes.

Un ejemplo de una solución a un problema del VRPCD se muestra a continuación en la figura 1.1. El conjunto de proveedores esta compuesto por los nodos 1,2,3,4,5, mientras que el

conjunto de clientes esta compuesto por los nodos -1,-2,-3,-4,-5. El cliente -1 tiene un pedido determinado sobre el proveedor 1, el cliente -2 sobre el proveedor 2, y así sucesivamente. El muelle de intercambio corresponde al nodo con la letra M. En esta solución se tienen 3 rutas correspondientes a 3 vehículos diferentes, en el caso del vehículo 1 se observa que la ruta de recogida que realiza corresponde a los nodos 1 y 2, en el vehículo 2 corresponde a los nodos 3 y 4, y en el vehículo 3 al nodo 5. Como se ilustra en la figura, la ruta de entrega del vehículo 1 corresponde al nodo -1,-2,-3, lo que implica que participó en un proceso de consolidación con el vehículo 2. Este proceso requirió que en el muelle de intercambio el vehículo 2 descargara los bienes correspondientes al nodo 3, para que posteriormente el vehículo 1 los cargara y realizara la entrega de estos. Debido a esto la ruta de entrega del vehículo 2 corresponde solo al nodo -4. Por otro lado el vehículo 3 no participa en ningún proceso de consolidación.

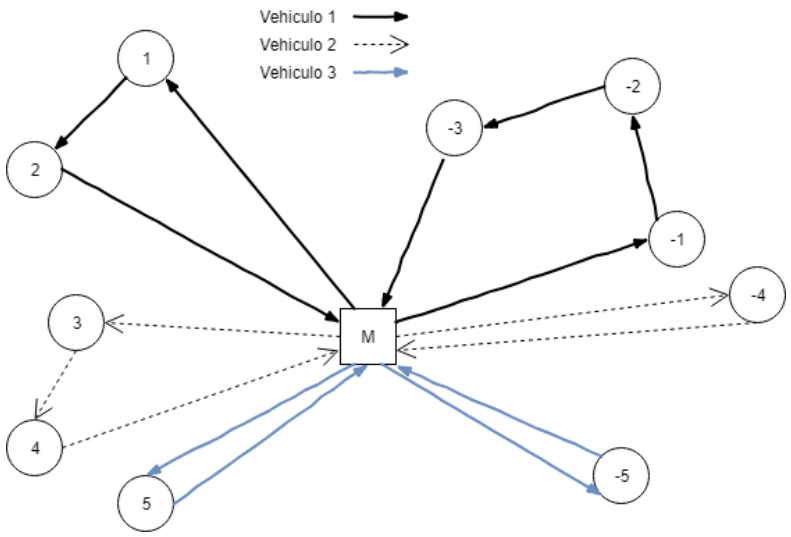


Figura 1.1: Ejemplo de solución de un VRPCD.

1.1. Formulación matemática

Debido a que existen diferentes variantes del VRPCD, en donde se incluyen distintas restricciones y consideraciones, es necesario contar con un modelo matemático que se ajuste a las

características exactas del problema. El modelo a utilizar en la presente memoria corresponde a una formulación de programación lineal entera mixta, el cual se detallara a continuación. Se define un conjunto de nodos de recogida (pickup) como $P = 1, 2, ..., n$ y un conjunto de nodos de entrega (delivery) $D = n + 1, 1 + 2, ..., 2n$. Cada pedido i se encuentra conformado por el par $(i, n + i)$, donde i y $n + i$ corresponde al nodo de recogida y de entrega que participan en el pedido respectivamente. Un muelle de intercambio es representado por el conjunto de cuatro nodos $O = o_1, o_2, o_3, o_4$, en donde los dos primeros representan las posiciones de comienzo y término para las rutas de recogida, mientras que los dos últimos representan el comienzo y término de las rutas de entrega. Además se define el conjunto completo de nodos como $N = P \cup O \cup D$. Con esto, se define un conjunto E de todos los arcos factibles en la red, de tal forma que E contiene los arcos $(i, j) : i, j \in P \cup o_1, o_2, i \neq j$, los cuales corresponden a los de una ruta de recogida, mientras que también contiene los arcos $(i, j) : i, j \in D \cup o_3, o_4, i \neq j$, que corresponden a los de una ruta de entrega. Finalmente se define K como el conjunto de vehículos.

Por otra parte, los parámetros utilizados en el problema se definen a continuación:

c_{ij} = el tiempo de viaje entre el nodo i y el nodo j con $(i, j) \in E$.
$[a_i, b_i]$ = la ventana de tiempo para el nodo i con $i \in N$.
d_i = la cantidad de demanda del pedido i con $i \in P$.
Q = la capacidad de los vehículos.
A = el tiempo fijo de carga y descarga en un muelle de intercambio para cada vehículo.
B = el tiempo de carga y descarga por cada caja de producto.

Las variables del problema son las siguientes:

$$x_{ij}^k = \begin{cases} 1 \text{ si el vehículo k viaja desde el nodo i al nodo j, } (i, j) \in E, k \in K \\ 0 \text{ en otro caso;} \end{cases}$$

$$u_i^k = \begin{cases} 1 \text{ si el vehículo k descarga el pedido i en el muelle } (i \in P, k \in K) \\ 0 \text{ en otro caso;} \end{cases}$$

$$r_i^k = \begin{cases} 1 \text{ si el vehículo k carga el pedido i en el muelle } (i \in P, k \in K) \\ 0 \text{ en otro caso;} \end{cases}$$

$g_k = \begin{cases} 1 \text{ si el vehículo k tiene que descargar bienes en el muelle } (k \in K) \\ 0 \text{ en otro caso;} \end{cases}$

$h_k = \begin{cases} 1 \text{ si el vehículo k tiene que cargar bienes en el muelle } (k \in K) \\ 0 \text{ en otro caso;} \end{cases}$

s_i^k = el tiempo en el que el vehículo k sale del nodo i ($i \in N, k \in K$);

t_k = el tiempo en el que el vehículo k termina de descargar bienes en el muelle ($k \in K$);

w_k = el tiempo en el que el vehículo k comienza a cargar bienes en el muelle ($k \in K$);

v_i = el tiempo en el que el pedido i es descargado en el muelle por el vehículo que lo recogió ($i \in P$).

Adicionalmente, se considera M como una constante arbitrariamente grande.
A continuación, se mostrará la formulación matemática del VRPCD a considerar, explicando brevemente cada una de sus restricciones y consideraciones.

1.2. Función objetivo

$$\text{Minimizar} \sum_{(i,j)\in E} \sum_{k\in K} c_{ij} x_{ij}^k$$

La función objetivo descrita anteriormente consiste en minimizar el tiempo de viaje total, considerando todas las rutas de todos los vehículos. Dentro de cada ruta de recogida, se considera también el costo de ir desde el muelle de intercambio hacia el primer proveedor, y dentro de cada ruta de entrega, se considera el costo del viaje desde el último cliente hacia el muelle de intercambio.

1.3. Restricciones

1.3.1. Restricciones de ruteo de vehículos

1. Cada nodo debe ser visitado una vez por solo un vehículo.

$$\sum_{j:(i,j)\in E} \sum_{k\in K} x_{ij}^k = 1, \qquad \forall i \in P \cup D$$

2. Para cada vehículo, la carga en la ruta de recogida no debe exceder la capacidad del vehículo.

$$\sum_{i\in P} \sum_{j:(i,j)\in E} d_i x_{ij}^k \leq Q, \qquad \forall k \in K$$

3. Para cada vehículo, la carga en la ruta de entrega no debe exceder la capacidad del vehículo.

$$\sum_{i\in D} \sum_{j:(i,j)\in E} d_i x_{ij}^k \leq Q, \qquad \forall k \in K$$

4. Las rutas de recogida y de entrega de cada vehículo deben partir desde los nodos o_1 y o_3 respectivamente.

$$\sum_{j:(h,j)\in E} x_{hj}^k = 1, \qquad \forall h \in \{o_1, o_3\}, k \in K$$

5. Conservación del flujo, es decir, el vehículo que llega a un nodo debe salir de el antes de ir a otro nodo.

$$\sum_{j:(h,j)\in E} x_{ih}^k - \sum_{j:(h,j)\in E} x_{hj}^k = 0, \qquad \forall h \in P \cup D, k \in K$$

6. Cada vehículo debe regresar a o_2 luego de su ruta de recogida, y volver a o_4 luego de su ruta de entrega.

$$\sum_{j:(h,j)\in E} x_{jh}^k = 1, \qquad \forall h \in \{o_2, o_4\}, k \in K$$

7. Tiempo de viaje entre dos nodos si son visitados consecutivamente por el mismo vehículo.

$$s_j^k \geq s_i^k + c_{ij} - M(1 - x_{ij}^k), \qquad \forall (i,j) \in E, k \in K$$

8. Cada nodo debe ser visitado dentro de su ventana de tiempo, y todos los vehículos deben finalizar sus rutas dentro del horizonte de tiempo establecido para el problema.

$$a_i \leq s_i^k \leq b_i, \qquad \forall i \in N, k \in K$$

1.3.2. Restricciones generales de consolidación en el muelle

9. Para las decisiones de consolidación en el muelle, si un vehículo k debe descargar o cargar el pedido i dependerá de sus rutas de recogida y entrega. Esta dependencia se ve reflejada en las siguientes restricciones, en donde se consideran los tres siguientes casos: si el vehículo k recoge el pedido i pero no lo entrega a $i + n$, entonces el vehículo debe descargar el pedido en el muelle; si el vehículo k no recoge el pedido i pero si lo entrega a $i + n$, entonces debe cargar el pedido en el muelle; finalmente si el vehículo no recoge el pedido i y tampoco lo entrega a $i + n$, entonces no carga ni descarga esa orden.
   Para un mejor entendimiento del proceso de consolidación se muestran los tres casos en la Tabla 1.1.

$$u_i^k - r_i^k = \sum_{j \in P \cup \{o_2\}} x_{ij}^k - \sum_{j \in D \cup \{o_4\}} x_{i+n,j}^k, \qquad \forall i \in P, k \in K$$

$$u_i^k + r_i^k \leq 1, \qquad \forall i \in P, k \in K$$

1.3.3. Restricciones del flujo interno de la consolidación en el muelle

10. Asegura que g_k debe ser 1 si el vehículo k necesita descargar un bien en el muelle.

$$\frac{1}{M} \sum_{i \in P} u_i^k \leq g_k \leq \sum_{i \in P} u_i^k, \qquad \forall k \in K$$

11. Calcula el tiempo de descarga para un vehículo k, el cual consiste en el tiempo fijo de preparación para descargar (A) y el tiempo que toma descargar los pedidos necesarios, el cual corresponde al tiempo que toma descargar una caja del bien (B) multiplicado por la cantidad de cajas que corresponden al pedido.

$$t_k = s_{o_2}^k + A g_k + B \sum_{i \in P} d_i u_i^k, \qquad \forall k \in K$$

12. Un vehículo solo puede comenzar a cargar productos después de haber terminado de descargar todos sus bienes correspondientes.

$$w_k \geq t_k, \qquad \forall k \in K$$

13. Un vehículo solo puede comenzar a cargar pedidos hasta que todos estos se encuentren listos para ser cargados.

$$w_k \geq v_i - M(1 - r_i^k), \qquad \forall i \in P, k \in K$$

14. El tiempo para que el pedido i este listo para ser cargado se representa en la siguiente restricción, el cual depende del tiempo en el que el vehículo que recogió el pedido i haya terminado de descargar todos sus respectivos bienes.

$$v_i \geq t_k - M(1 - u_i^k), \qquad \forall i \in P, k \in K$$

15. Asegura que h_k sea 1 si el vehículo k necesita cargar un bien en el muelle.

$$\frac{1}{M} \sum_{i \in P} r_i^k \leq h_k \leq \sum_{i \in P} r_i^k, \qquad \forall k \in K$$

16. Calcula el tiempo de carga para un vehículo k, el cual consiste en el tiempo fijo de preparación para cargar (A) y el tiempo que toma cargar los pedidos necesarios, el cual corresponde al tiempo que toma cargar una caja del bien (B) multiplicado por la cantidad de cajas que corresponden al pedido.

$$s_{o_3}^k = w_k + Ah_k + B \sum_{i \in P} d_i r_i^k, \qquad \forall k \in K$$

17. Asegura los valores binarios para las variables.

$$x_{ij}^k, u_i^k, r_i^k, g_k, h_k \in \{0, 1\}, \qquad \forall i \in P, (i, j) \in E, k \in K$$

18. Asegura los valores positivos para las variables.

$$s_i^k, t_k, w_k \geq 0, \qquad \forall i \in N, k \in K$$

$$v_i \geq 0, \qquad \forall i \in P$$

k recoge i	k entrega $i+n$	k descarga i	k carga i
$\sum_{j \in P \cup \{o_2\}} x_{ij}^k$	$\sum_{j \in D \cup \{o_4\}} x_{i+n,j}^k$	u_i^k	r_i^k
0	0	0	0
1	1	0	0
1	0	1	0
0	1	0	1

Tabla 1.1: Relación entre x_{ij}^k y u_i^k, r_i^k.

Capítulo 2

Estado del Arte

En la literatura existen acercamientos para el Vehicle Routing Problem (VRP) en general y otros para el problema de Cross-Docks en particular. Esta sección se enfocará en la literatura relacionada sobre el Vehicle Routing Problem con Cross-Docks (VRPCD), dando una pequeña introducción al VRP.

El primer trabajo en donde se plantea el VRP es en [6], en donde se estudió y aplicó a un problema de distribución de combustible. Ese mismo año se profundizó y se concluyó que este problema pertenece a la clase NP-Hard en [15], ya que el TSP pertenece a esta clase y éste puede ser tratado como un caso particular del VRP, en donde hay un solo vehículo disponible, sin restricciones de capacidad. Luego en [3] el VRP es descrito formalmente como aquel en el que ”vehículos ubicados en un depósito central son utilizados para visitar clientes localizados geográficamente dispersos para satisfacer las demandas (conocidas) de los clientes”, exigiendo que cada cliente sea visitado solo una vez por uno de los vehículos, respetando las restricciones de capacidad, el tiempo máximo permitido de trabajo, distancia máxima recorrida, etc. En ese mismo trabajo, se propone un modelo de programación lineal entero (ILP), descrito con detalle en [3]. Los mejores resultados obtenidos para el VRP son los algoritmos basados en Tabu Search, los cuales son comparados en [16], siendo la mejor de estas implementaciones la estudiada por Taillard en [27].

Luego comienzan a aparecer diversas variantes del VRP en las cuales se incluyen restricciones adicionales, con el fin de realizar un trabajo más práctico y más cercano a la realidad. Varios estudios fueron realizados tomando en consideración una flota de vehículos heterogéneos y resueltos con diversos métodos, como por ejemplo, en [24] donde se utiliza un algoritmo genético, en [25] donde se utilizan algoritmos meméticos y en [4] donde se utiliza Búsqueda Tabú. Adicionalmente se han realizado trabajos en donde se incorporan ventanas de tiempo en donde pueden ser entregados los pedidos, como es el caso propuesto en [5]. Finalmente existen otros casos en donde se han considerado múltiples depósitos para satisfacer a los clientes, los clientes pueden ser atendidos por varios vehículos (Split-Delivery), entre otros supuestos, siendo el estudio mas completo realizado a la fecha, como es el caso expuesto en [14].

Uno de los primeros acercamientos formales para el VRPCD se encuentra en [29], que estudia el caso de una empresa que opera en Dinamarca cuyos proveedores preparan las ordenes para los supermercados y las envían utilizando muelles de intercambio. La definición del problema que utilizan los autores es, en esencia, un problema de transporte de pedidos de un conjunto de proveedores a sus correspondientes clientes, utilizando la estrategia de muelles de intercambio. En este caso en particular se considera un solo muelle, en donde los pedidos son recogidos por una flota de vehículos y son llevados a éste mismo e inmediatamente entregados a los clientes por el mismo conjunto de vehículos, sin utilizar el almacenamiento intermedio, es decir al momento de ser descargados los pedidos en un muelle, estos deben ser cargados en algún otro vehículo de manera inmediata. Existe una distinción entre ruta de recogida y ruta de entrega, y todas las rutas deben comenzar y terminar en un muelle transversal. Adicionalmente existe una restricción que establece que cada nodo es visitado solo una vez por un vehículo. Cabe destacar que en este artículo, los autores hacen énfasis en la distinción del proceso de carga y descarga de los pedidos, es decir, definen un tiempo fijo de carga y descarga, y además un tiempo de carga y descarga por cada unidad de pedido o producto. Junto a esto crean nuevas restricciones que se hacen cargo de la coordinación en los procesos de carga y descarga entre los vehículos que realizan estas acciones. Considerando lo antes mencionado, los autores proponen una formulación de programación lineal entera mixta, cuyo modelo se muestra con detalle en [29], la función objetivo en este caso es minimizar el tiempo de viaje total de todos los vehículos, satisfaciendo todos los pedidos. Cabe

destacar que el modelo propuesto por los autores permite soluciones infactibles, penalizando la calidad de la solución en estos casos.

Para resolver el problema, los autores proponen un algoritmo Tabu Search inmerso en un procedimiento de memoria adaptativa (AMP), en el cual separan los vecindarios en grandes y pequeños. El movimiento utilizado intercambia un nodo visitado por un vehículo en un determinado tour y lo agrega en otro vehículo distinto. Los pequeños vecindarios son generados por dicho movimiento, el cual opera en un subconjunto de vehículos y un subconjunto de nodos. Por otro lado los vecindarios grandes son generados por este mismo movimiento pero utilizando todos los vehículos y nodos del problema. El algoritmo propuesto primeramente trabaja con solo vecindarios pequeños y si no encuentra una mejora en su solución en una cantidad determinada de iteraciones, pasará a trabajar con los vecindarios grandes. Por otro lado, si se encuentra una mejor solución que la global en un número determinado de iteraciones, éste vuelve a trabajar sobre vecindarios pequeños, en caso contrario el algoritmo se detiene.

Los experimentos fueron realizados utilizando instancias de testing con 20, 30, 50, 100 y 150 pares de nodos generados de manera aleatoria, seleccionando el número correspondiente de nodos proveedores-clientes. Adicionalmente se experimentó utilizando instancias reales de 200 pares de nodos, proveídos de un dataset de Transvision, una consultora de logística danesa. Además realizan una comparación, resolviendo las instancias utilizando los 2 tipos de vecindarios antes nombrados y utilizando solo uno de ellos.

Los resultados obtenidos muestran que al utilizar sólo el vecindario grande, se llega a una solución mejor que al utilizar el vecindario pequeño, sin embargo el tiempo de cómputo al utilizar este primero es mayor en un factor aproximado de 1.7 en relación al tiempo del segundo. A pesar de esto, se muestra que al utilizar la combinación de ambos vecindarios se logra una mejor solución que las encontradas utilizando solo uno de estos, y el tiempo de cómputo utilizado es aproximadamente el promedio de los dos tiempos resultantes al utilizar ambos vecindarios por separado. Por otro lado se observa que el algoritmo se comporta de manera similar para las instancias pequeñas y grandes, ya que en ambas, la solución entregada resulta ser de muy buena calidad (menos de 5 % de diferencia con la óptima), y en un tiempo de cómputo menor a 5 segundos para instancias pequeñas (las de testing) y menor

a 5 minutos para las instancias reales.

Otra propuesta para el VRPCD se encuentra en [20], en donde los autores consideran el problema un poco más relajado, es decir, no poseen intervalos de tiempo para recoger y enviar los pedidos, consideran que la cantidad de pedidos de cada consumidor es menor que las capacidades de los vehículos, también suponen que todos estos poseen la misma capacidad y que se encuentran disponibles en cualquier momento. Además considera que el tiempo de carga-descarga de los pedidos están intrínsecos en los costos de los arcos entre los nodos del problema. Sin embargo, a diferencia del artículo anterior, el problema considera múltiples muelles de intercambio, también permite los viajes directos (parten en los proveedores y terminan en los clientes sin pasar por un muelle) y la función objetivo es minimizar el costo total de transporte de los vehículos, cumpliendo con todas las demandas correspondientes. Para esto, los autores proponen un modelo de programación entera, el cual es detallado en [20].

Para resolver el problema, los autores proponen un algoritmo basado en Ant Colony Optimization (ACO). La regla de transición que se utiliza es la clásica del ACO, y definen una matriz B que representa por cual muelle pasará el vehículo en el trayecto i,j, donde i es un nodo proveedor y j es un nodo cliente. En el caso que sea un viaje directo, aparecerá en la matriz el valor k+1, en donde k es el número de muelles que hay en el problema. Por ejemplo, para un problema con 3 proveedores, 3 clientes y 2 muelles de intercambio, la matriz:

$$B = \begin{pmatrix} 2 & 3 & 1 \\ 3 & 3 & 1 \\ 1 & 2 & 1 \end{pmatrix}$$

Representa que el viaje desde i=1 a j=1 pasa por el muelle k=2; el viaje desde i=1 a j=2, es directo y el viaje desde i=1 a j=3 pasa por el muelle k=1.
Para explicar el flujo del algoritmo se define el parámetro M, que representa la cantidad de hormigas a utilizar y un parámetro N, que representa la cantidad de iteraciones a realizar.
El flujo del algoritmo utilizado se puede resumir en la Figura 2.1:

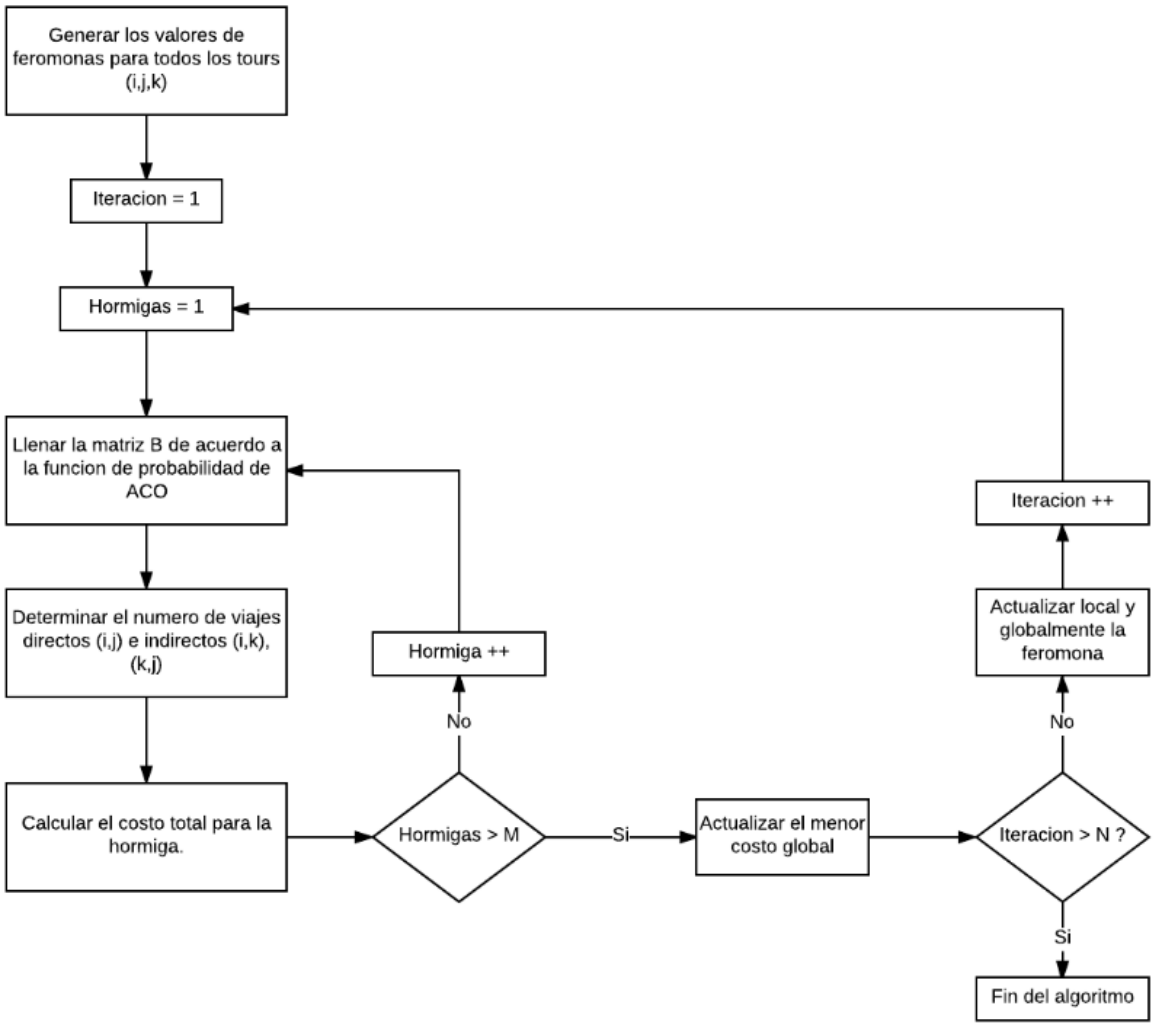


Figura 2.1: Diagrama de Flujo del algoritmo ACO utilizado en [20].

Se utilizaron 4 sets de pruebas, con 10 instancias cada uno para validar el algoritmo propuesto, las características de cada set varían desde 15 a 75 proveedores y clientes y desde 3 a 50 muelles de intercambio, considerando una capacidad de 24 pedidos para todos los vehículos.

Dichos experimentos fueron también resueltos utilizando el solver LINGO, con la técnica de Branch and Bound (B&B), con el fin de comparar los resultados obtenidos al aplicar el ACO propuesto.

Los resultados muestran que para el primer set de datos B&B encuentra la solución óptima

para los 10 experimentos realizados, mientras que ACO encuentra el óptimo para 9 de estos, sin embargo ACO se demora un poco menos en encontrar la solución. Para los sets 2,3 y 4, se tomó un tiempo limite de ejecución de 30 minutos, en este caso ACO muestra ser mucho mas eficiente y eficaz que B&B, ya que sus soluciones obtenidas en dicho tiempo fueron de mejor calidad en todos los experimentos realizados para los sets mencionados. Adicionalmente se aumentó el tiempo límite a 1,2 y 3 horas, mostrando que luego de 1 hora de ejecución B&B mejoraba la calidad de su solución en menos de 3 %. Finalmente se concluye que ACO resulta ser mucho más eficiente y eficaz que B&B en todos los casos, y que la diferencia de las calidades de las soluciones de ambos algoritmos aumenta considerablemente para instancias más grandes.

Otro acercamiento a este problema se presenta en [26], el cual considera lo siguiente: existe solo un muelle de intercambio, los vehículos se encuentran disponibles en todo momento, el costo de carga y descarga de los pedidos están asociados solo con el muelle respectivo y no depende de los vehículos, y no se trabaja con ventanas de tiempo. La función objetivo en este caso es minimizar el costo total de las rutas para todos los vehículos, cumpliendo con todas las demandas correspondientes.

El modelo utilizado por los autores es uno de programación entera, y adicionalmente proponen un Restricted Linear Programming Master (RLPM), el cual lo obtienen relajando el problema original, convirtiéndolo en un problema de Resource Constrained Elementary Shortest Path Problem (RCESPP). Los detalles del modelo propuesto y las especificaciones del RCESPP generado se encuentran en [26].

Para resolver el problema, los autores proponen un algoritmo basado en Branch-and-price (B&P), y lo comparan con el Branch-and-Bound (B&B), analizando los mejores límites inferiores y los mejores límites superiores obtenidos por ambos algoritmos. Las instancias probadas fueron las utilizadas en [29], y se usó el solver ILOG CIPLEX para resolver los modelos propuestos (LP y MIP), considerando un tiempo de 2 horas.

Los resultados obtenidos muestran que el B&P supera al B&B en relación a la calidad de los mejores límites inferiores y superiores obtenidos en el mismo tiempo de cómputo. También se observa que el B&P resuelve de manera óptima 5 instancias, mientras que B&B resuelve

4. Se destaca que la brecha definida como el cociente entre la diferencia entre los límites superior e inferior y el límite superior es menor en B&P (entre 0 % y 15 %) que en B&B (entre 26 % y 50 %). Finalmente se destaca que en los problemas que ambos algoritmos resolvieron de manera óptima, B&P lo realizó en aproximadamente 38 segundos, mientras que B&B lo hizo en 1623 segundos. Por lo tanto, el algoritmo propuesto por los autores muestra una total superioridad en relación con un algoritmo de B&B.

En [28] se estudia el mismo problema VRPCD propuesto en [29], pero se trabaja de dos maneras distintas, como una configuración de red cerrada y una abierta. En el caso de la configuración de red cerrada, el muelle de intercambio es el único punto de partida y de término para todos los vehículos, mientras que en la abierta, los nodos de partida y de término pueden ser cualquiera de los proveedores o los clientes. Por otro lado, se consideran dos escenarios distintos, en el primero se asume que se usan diferentes vehículos para el retiro y la entrega de los pedidos, lo que obliga a que éstos sean descargados y recargados en el muelle de intercambio, mientras que en el segundo se asume que los mismos vehículos que retiran los pedidos son los que deben repartirlos utilizando el muelle.

El método de resolución que utilizan los autores consiste en un Tabu Search (TS), embebido en un framework de multi-restart adaptativo. Primero se inicializa un conjunto de soluciones de referencia R, luego comienza el funcionamiento del framework, en el cual se utiliza una heurística que trabaja sobre R para identificar los elementos "prometedores" que sean comunes en estas con el fin de que cada solución inicial, utilizada por el posterior TS, posea información de las soluciones anteriores resultantes y así partir con una solución inicial de buena calidad en cada iteración. Posteriormente se aplica TS con la solución generada anteriormente, y se inicializan sus parámetros con el fin de promover la intensificación. Los movimientos utilizados para generar los vecindarios son los mismos que los explicados en [11]. Luego que el TS termina de iterar, se procede a actualizar el set de soluciones de referencia y comienza una nueva iteración del framework hasta que se cumpla alguna condición de término.

Para evaluar la propuesta de los autores se realizaron experimentos utilizando el set de pruebas propuesto en [29], utilizando un rango entre 40 y 80 iteraciones donde se guardarán los elementos en la lista tabú (largo de la lista), y realizando 3000 iteraciones.

Los resultados muestran que el Adaptative Multi-Restart Tabu Search (AmTS) propuesto es por lejos mas eficiente y eficaz que la propuesta utilizada en [29]. Se observa que en 12 de 19 experimentos el método AmTS supera al de [29], entregando soluciones que mejoran hasta en un 1.5 % su calidad y en un tiempo menor en aproximadamente 2 ordenes de magnitud. Por otro lado, los resultados del escenario en el que se asume que los mismos vehículos que retiran los pedidos son los que luego los repartirán, muestran que los costos son menores a que si se consideran distintos vehículos para recoger y entregar los pedidos en aproximadamente 0.59 %. Finalmente se observa que hay una considerable reducción de costos al utilizar la configuración de red abierta, respecto a la cerrada.

Otra propuesta de resolución se explica en [7], en donde se consideran vehículos homogéneos, existe un solo muelle de intercambio pero con varias puertas o estaciones en donde pueden llegar los vehículos a cargar o descargar los pedidos, asegura la disponibilidad de todos los vehículos, la entrega separada no esta permitida, es decir, cada nodo debe ser visitado solo una vez y por solo un vehículo. El modelo matemático propuesto por los autores es detallado en [7].

Debido a que una gran parte del tiempo de cómputo se encuentra asociado al proceso de asignar a los vehículos las rutas de recogida y entrega, y con el fin de tener una aproximación al VRPCD mas eficiente, los autores proponen un algoritmo de barrido con heurísticas propuesto en [2] para asignar los pedidos a los vehículos y también un modelo modificado para resolver el VRPCD.

El algoritmo de barrido es una heurística que resuelve de manera eficiente los problemas relacionados con VRP. Se asume que hay un solo depósito (en este caso un muelle), y que se dispone de un conjunto de vehículos homogéneos con las mismas capacidades. El depósito funciona como origen de un sistema de coordenadas polares, en el cual cada nodo de proveedor y cliente se encuentra representado en términos de dos coordenadas (el radio y el ángulo) respecto a este. Para el proceso de asignación de vehículos, los nodos de tipo cliente se encuentran ordenados de manera incremental respecto a las coordenadas angulares. Siguiendo dicho orden, los nodos son asignados al vehículo actual que aun disponga de capacidad, en caso contrario se elige uno nuevo, y el proceso continúa hasta que cada nodo haya sido asignado a un vehículo. En [7] se explica de manera matemática el proceso de asignación, así

como el modelo modificado utilizado para ejecutar dicho algoritmo.

Para validar el modelo y el algoritmo propuesto se realizaron diversos experimentos de distintos tamaños y se compararon los resultados obtenidos con los resultantes al utilizar la formulación exacta (modelo antes de ser modificado para usar la técnica de barrido). Se observa que para tamaños de instancia con 10 a 15 pedidos y con 2 a 3 vehículos, las soluciones entregadas por la formulación basada en barrido (SBF) y con la formulación exacta (EF) difieren entre 0.25 % y 2 %, siendo ambas de muy buena calidad. Sin embargo SBF genera las soluciones en un tiempo de aproximadamente 1 orden de magnitud menor que EF para las instancias mas pequeñas y 2 ordenes de magnitud menos que EF para las instancias medianas. Luego se probaron instancias más grandes, con pedidos entre 19 y 50, y con 4 a 10 vehículos, mostrando que el EF no convergía a soluciones luego de 1800 segundos y la diferencia entre sus soluciones aumentaba mientras más grande era la instancia (desde 10.6 % hasta 22.3 %), mientras que SBF encontraba buenas soluciones en aproximadamente 1000 segundos para la mayoría de estas instancias y la diferencia entre éstas era entre 0.1 % a 0.3 %. Finalmente se concluye que SBF es más eficiente que EF para instancias más pequeñas y para instancias más grandes resulta ser mucho mas eficiente y eficaz, entregando soluciones de buena calidad en tiempos razonables menores a 1000 segundos.

Nuevamente en [19] se vuelve a trabajar el problema estudiado en [29] y [28]. En este caso los autores proponen 3 heurísticas para el VRPCD basadas en las metaheurísticas de Iterated Local Search (ILS) propuestas en [13]. Para generar las soluciones iniciales, los autores utilizan una heurística de tipo Greedy llamada 2S-NI. Ésta consiste en comenzar con una solución vacía y seleccionar según la función miope un pedido del conjunto de pedidos, luego se genera una ruta de recogida añadiendo el nodo proveedor correspondiente al pedido seleccionado, y simultáneamente se genera una ruta de entrega con el nodo cliente correspondiente al mismo pedido, todo esto se realiza para un vehículo cualquiera. Cuando no se pueden asignar mas pedidos al mismo vehículo, se escoge otro y se repite el proceso, hasta que no queden mas pedidos por asignar.

La primera heurística que utilizan los autores es S-ILS, la cual consiste en una implementación estándar de ILS. Se comienza con una solución generada por 2S-NI y luego se aplica un procedimiento de búsqueda local basado en Variable Neighborhood Descent (VND), la

cual es una metaheurística que combina un conjunto de búsquedas locales y las conduce de manera eficiente a una solución que es un óptimo local para todas ellas, los detalles de dicha metaheurística se estudian en [22]. Uno de los movimientos utilizados para realizar VND es el *exchange*, el cual evalúa en cada iteración el costo de intercambiar el proveedor y cliente de un pedido perteneciente a un vehículo con el proveedor y clientes de otro pedido perteneciente a otro vehículo distinto, luego se escogerá el primer intercambio que mejore la calidad de la solución actual. El otro movimiento utilizado en VND es el *relocate* el cual evalúa en cada iteración el costo de mover el proveedor y cliente de una ruta de un vehículo hacia otro vehículo distinto, luego se escogerá la mejor reubicación que mejore la calidad de la solución. Posteriormente se comienza un ciclo, en donde en cada iteración se utiliza un movimiento basado en 2-split y k-exchange. 2-split genera una nueva solución dividiendo una ruta de la solución actual en dos rutas mas pequeñas. k-exchange genera la nueva solución intercambiando k proveedores o clientes seleccionados de manera aleatoria de dos vehículos distintos de la solución actual. Luego de realizar uno de estos dos movimientos, escogido de manera aleatoria, se procede a realizar nuevamente VND utilizando los movimientos de *insertion* el cual se explicó anteriormente haciendo referencia a [29], *swap(1,1)* el cual intercambia un proveedor de una ruta de recogida de un vehículo por el proveedor de una ruta de recogida de otro vehículo, y *swap(2,1)* que intercambia 2 proveedores de una ruta de recogida de un vehículo por un proveedor de una ruta de recogida de otro vehículo. Finalmente luego de realizar el VND mencionado se realiza un movimiento de *reinsertion*, el cual evalúa en cada iteración el costo de cambiar de posición un proveedor dentro de una ruta de recogida de un solo vehículo, para luego seleccionar el primer cambio que mejore la solución actual. Finalmente este ciclo se repetirá hasta que se acepte el criterio de aceptación establecido.

La segunda heurística que proponen los autores es X-ILS, la cual extiende a S-ILS. X-ILS mantiene en cada iteración un conjunto de soluciones factibles, en vez de tener sólo una actual. X-ILS tiene una fase de intensificación adicional que se ejecuta cuando el procedimiento se estanca, es decir, se ejecuta luego de una cantidad de iteraciones dada sin mejorar la mejor solución encontrada. El proceso de intensificación consiste en realizar una búsqueda local utilizando el movimiento *drop-route*, el cual evalúa en cada iteración el costo general de mover todos los proveedores de una ruta de recogida de un vehículo hacia una ruta de

recogida de otro vehículo, con el fin de seleccionar la primera solución generada que mejore a la actual escogida del conjunto de soluciones factibles. En caso que ninguna solución generada mejore a la actual se ejecuta nuevamente este procedimiento utilizando otra solución del conjunto de soluciones factibles.

La tercera heurística es SP-ILS, la cual es muy similar a X-ILS, pero difiere en el algoritmo utilizado en la fase de intensificación. En vez de utilizar una búsqueda local con el movimiento *drop-route*, SP-ILS utiliza un procedimiento de intensificación basado en programación lineal entera (ILP) para el problema de Set Partitioning (SP). Este último explora un vecindario exponencialmente grande que consta de todas las soluciones que contienen solo rutas encontradas en alguna solución almacenada en el conjunto de soluciones factibles. Este vecindario puede ser explorado de manera eficiente utilizando técnicas de programación matemática encontradas en el estado del arte de los ILP Solvers, tales como CPLEX y XPRESS.

Para validar las heurísticas propuestas se realizaron pruebas utilizando dos sets de instancias. El primer set corresponde al propuesto en [29]. El segundo set fue propuesto por los autores, el cual se encuentra formado por 16 instancias con 200, 300, 400 y 500 pedidos y 401, 601, 801 y 1001 nodos respectivamente. Los nodos son aleatoriamente distribuidos entre proveedores y clientes. Las capacidades de los vehículos son de 50 unidades.

Los experimentos realizados con el primer set de pruebas se compararon de manera independiente con el Tabu Search (TS) propuesto en [29] y con el TS de [28]. En el primer caso se demuestra que SP-ILS genera soluciones de mejor calidad para todas las instancias excepto dos, la 50a que la lidera el XS-ILS y la instancia 50c que la lidera el TS propuesto en [29]. Se observa que las soluciones entregadas por X-ILS y SP-ILS son mejores que las de S-ILS en todas las instancias, lo que respalda que las mejoras que se realizaron a la heurística estándar de ILS fueron acertadas y que lograron mejorar la calidad de las soluciones de manera correcta y que además supera al TS propuesto en [29]. Los autores mencionan que la superioridad de su heurística se puede deber a que TS acepta soluciones infactibles penalizándolas mediante la función objetivo, mientras que su propuesta solo explora el espacio de soluciones factibles.

Para el segundo caso se utilizó nuevamente el set de pruebas de [29] y se compararon los resultados obtenidos con los expuestos en [28], observando que SP-ILS obtiene mejores soluciones que XL-ILS Y S-ILS para todas las instancias de prueba. Por otro lado el TS propuesto en [28] obtiene mejores resultados en todas las instancias pequeñas (las primeras 10 con 50 a 100 pedidos), mientras que XL-ISL obtiene las mejores soluciones para las 10 instancias mas grandes (entre 100 y 150 pedidos). Estas soluciones resultan ser las mejores obtenidas en la literatura de esas instancias hasta ese tiempo. Los autores explican que la superioridad de su propuesta para instancias grandes se debe al proceso de intensificación y los movimientos utilizados en éste, ya que gracias a esto el espacio de búsqueda explorado por SP-ILS es más amplio que el del TS, además que el proceso de restart del SP-ILS vuelve a una solución probablemente de muy buena calidad mejorando significativamente la solución final obtenida.

Posteriormente los autores proponen un set de pruebas con instancias mucho mas grandes que las existentes en la literatura (de 200 a 500 pedidos) y comparan sus 3 heurísticas propuestas, poniendo como límite 1200 segundos de CPU. Los resultados muestran que SP-ILS obtiene mejores resultados en todas las instancias propuestas, y que X-ILS obtiene mejores resultados que S-ILS en 13 de 16 instancias. La mejora de SP-ILS y X-ILS respecto a S-ILS en relación a las soluciones obtenidas es de un 2.69 % y 2.17 % respectivamente.
Finalmente se concluye que las mejoras realizadas por los autores al método estándar de ILS fueron de gran utilidad, aumentando la calidad de las soluciones en casi todos los casos. Y que las heurísticas propuestas mejoran todos los resultados obtenidos respecto a los de [29]. Por otro lado se concluye que SP-ILS supera al TS propuesto en [28], entregando los mejores resultados obtenidos para el set de instancias de [29] para las instancias más grandes. Todos los pedidos deben ser satisfechos durante un periodo de tiempo fijo (generalmente 24 horas)

Una propuesta diferente a las anteriores se estudia en [1], en donde se trabaja el problema de manera mas restringida. Se asume que cada tipo de producto puede tener distintos precios y ser producido por todos los proveedores. Considera que todos los vehículos son distintos y que éstos tienen distintos costos de operaciones en los muelles de intercambio. El comienzo y el punto de término de las rutas de los vehículos deben ser el mismo. La demanda total de un cliente para un producto en particular, debe ser asignada a solo un proveedor y un muelle,

además debe ser transportado por solo un vehículo de recolección y uno de reparto, implicando que debe existir al menos un vehículo de cada tipo con capacidad superior a un pedido en particular. El modelo propuesto por los autores corresponde a un modelo de programación no lineal entera mixta (MINLP), cuya función objetivo es minimizar los costos de las rutas de los vehículos, cumpliendo con todas las demandas y las restricciones establecidas.

Para resolver el VRPCD planteado, los autores proponen un algoritmo genético híbrido, en el cual, luego de seleccionar los nuevos individuos generados con las respectivas operaciones de cruzamiento y mutación, éstos pasan por un proceso de búsqueda local con el fin de mejorar algunos de ellos.

La representación de las soluciones utilizada propone que cada cromosoma se componga de 4 secciones. La primera, especifica el proveedor para cada producto demandado por los clientes. La segunda sección, indica los productos que deben ser recogidos por cada vehículo de recolección y el muelle de intercambio a utilizar. La tercera sección, indica los productos que deben ser repartidos por cada vehículo de reparto, así se puede determinar en que muelle debe cargar los pedidos. Por último, la cuarta sección determina los tiempos de salida de los vehículos de recolección desde los muelles.

Debido a que con la representación utilizada, los cromosomas pueden representar soluciones infactibles, la función de aptitud es creada a partir de la modificación de la función objetivo propuesta, agregando términos de penalización que reflejan el grado de infactibilidad de las soluciones, principalmente de las restricciones de capacidad y ventanas de tiempo.

El operador de selección de padres propuesto por los autores es la clásica ruleta, con el fin de realizar una selección de manera proporcional a la aptitud de cada individuo.

Para realizar el proceso de cruzamiento, los autores proponen 3 operadores que se aplican simultáneamente a las distintas secciones del cromosoma. A la primera sección, se le aplica un cruzamiento en un punto (one-point). A la segunda y tercera sección se le aplica el cruzamiento de orden (order crossover). Finalmente a la cuarta sección se le aplica el cruzamiento aritmético (whole arithmetic crossover), el cual toma la suma ponderada de los dos valores de los genes padre para cada uno de éstos. Los detalles de los operadores de cruzamientos mencionados se encuentran en [8].

Por otro lado, el proceso de mutación al igual que el de cruzamiento, consiste en 3 operaciones que se aplican a las distintas secciones del cromosoma. A la primera sección, se le aplica la mutación de restablecimiento aleatorio (random resetting mutation) [8] que restablece el valor de cada gen con una pequeña probabilidad a un valor diferente escogido de manera aleatoria de un conjunto de valores permitidos. En la segunda y tercera sección, se aplica la mutación de intercambio (swap), la cual intercambia los valores de dos genes seleccionados de manera aleatoria. En la cuarta sección, se realiza la creep mutation, que funciona aumentando o disminuyendo el valor de cada gen con una pequeña probabilidad.

Finalmente se encuentra el procedimiento de búsqueda local, el cual se aplica secuencialmente a las 3 primeras secciones del cromosoma. Si una sección sufre algún cambio, las otras secciones se dejan intactas, con el fin de evitar modificaciones muy grandes. En la primera sección, el movimiento realizado cambia el valor de un gen escogido aleatoriamente a todos los valores pertenecientes a su dominio, así se consideran todos los proveedores para los productos correspondientes. La mejor solución del vecindario se escoge solo si mejora a la solución actual, y dicho proceso se repite una cantidad fija de iteraciones. Para la segunda y tercera sección, el movimiento utilizado intercambia los valores de dos genes seleccionados aleatoriamente. Se selecciona la mejor solución del vecindario solo si es mejor que la solución actual. El proceso es repetido una cantidad determinada de iteraciones que podría ser distinta a la de la primera sección.

Para comprobar la eficiencia del algoritmo propuesto, los autores generaron un set con dos grupos de problemas. El primero incluye 3 problemas (1-3) con instancias pequeñas desde 2 a 4 proveedores, 2 a 4 muelles, 5 a 15 clientes, 2 a 4 vehículos de recolección y reparto, mientras que el segundo incluye 3 problemas (4-6) con instancias más grandes, 8 a 10 proveedores, 6 a 8 muelles, 30 a 50 clientes, 6 a 8 vehículos de recolección y reparto. Se crearon 10 instancias para cada problema, sumando un total de 60.

Para comparar el rendimiento del algoritmo genético híbrido (HGA), con el modelo MILP, se utilizó el paquete matemático GAMS. Debido a que el modelo MILP no pudo encontrar soluciones factibles dentro de un tiempo de 20 horas, se procedió a comparar el HGA con dos algoritmos alternativos. Uno es llamado GA, que corresponde al algoritmo genético propuesto sin utilizar la búsqueda local. El otro método alternativo es llamado ML, el cual consiste

en un algoritmo generado a partir de las operaciones de mutación definidas anteriormente para realizar la diversificación, en conjunto con el movimiento utilizado en la búsqueda local de HGA.

Los algoritmos se ejecutaron un total de 10 veces para cada instancia de los problemas, en donde se observa que GA es mas efectivo que ML, siendo el mejor de todos HGA. Los resultados obtenidos por HGA son de mejor calidad para todas las instancias, aumentando la brecha con los otros algoritmos a medida que el problema crece. Los resultados computacionales muestran que HGA provee una aproximación eficiente y con un rendimiento razonable.

Luego, en [14] se estudia el VRPCD de similar manera al propuesto en el presente trabajo, con la diferencia que no permite rutas directas desde un proveedor a un cliente sin pasar por un muelle de intercambio. Los autores proponen un modelo MILP para resolver el problema, el cual es detallado en [8].

Para resolver el VRPCD planteado, los autores proponen una metodología que combina una heurística de construcción (CH) con un Simulated Annealing de dos capas (CH-TLSA) o con un Tabu Search de dos capas (CH-TLTS), con el fin de mejorar la solución optimizando la asignación de los vehículos a los muelles y el orden de visita a los clientes.

La CH consiste en 4 pasos. En primer lugar, los clientes y proveedores son asignados a los muelles de intercambio. Todos los bienes que son requeridos por un cliente perteneciente a un muelle en particular, deben ser transferidos a través de este mismo muelle. Una vez que el tipo, origen, destino y la cantidad de pedidos transferidos a través de un muelle son fijados, cada muelle y sus correspondientes proveedores y clientes pueden ser considerados como un sistema de muelles independiente. En segundo lugar, un algoritmo Greedy asigna los vehículos a cada sistema de muelles mencionado anteriormente. En tercer lugar, las rutas de los vehículos en cada sistema de muelles se identifican determinando el orden en que se visitan los proveedores y los clientes. Por último, los tiempos de llegada de los vehículos (los horarios de los vehículos) son calculados de acuerdo con las rutas y restricciones de las ventanas de tiempo.

El TLSA consiste en un SA interno y un SA externo. El SA externo es utilizado para optimizar la asignación de los vehículos a los muelles de intercambio, realizando un swap entre

los vehículos de diferentes muelles. Por otro lado el SA interno es empleado para optimizar el orden de los vehículos (el cual fue asignado en el CH), realizando un swap entre ellos.

La representación de la solución utilizada por los autores consiste en la asignación de los vehículos a los muelles y el orden de visita de los vehículos a éstos últimos. Para crear los vecindarios correspondientes en el SA externo, se selecciona de manera aleatoria dos muelles e intercambian uno por uno cada vehículo perteneciente a éstos, utilizando dos operaciones de tipo swap. Si uno de los muelles seleccionados corresponde al muelle terminal (donde deben ir todos los vehículos luego de terminar su ruta), se realiza el primer swap, en donde se intercambia un vehículo de dicho muelle con uno o varios vehículos del otro muelle seleccionado. En caso contrario, se adopta el segundo tipo de swap, en donde dos vehículos son escogidos de manera aleatoria de los dos muelles seleccionados respectivamente y se intercambian entre ellos. En el caso del SA interno, el movimiento utilizando en el SA interno consiste en realizar un swap del orden de dos vehículos escogidos aleatoriamente en un sistema de muelles.

Por otro lado, se tiene el TLTS el cual posee la misma estructura de dos capas que el TLSA, es decir, hay un TS interno y un TS externo. Al igual que en TLSA, el TS externo es utilizado para optimizar la asignación de los vehículos a los muelles, mientras que el TS interno optimiza el orden de los vehículos en cada muelle. En cada una de las capas del TLTS, se tiene una lista tabú en donde se guardan los pares de vehículos más recientemente intercambiados, de manera que cuando la lista se encuentre llena, se remueva el par más antiguo de ésta. El TLTS utiliza la misma representación de la solución que TLSA, y la solución inicial es generada igualmente por CH. Los movimientos utilizados por cada capa del TLTS son los mimos respectivamente que los utilizados por TLSA.

Para validar los métodos propuestos por los autores, tanto el CH-TLSA como el CH-TLTS son probados de manera independiente en un conjunto de instancias diferentes y sus resultados son comparados con el MILP solver CPLEX. Los problemas a resolver fueron generados de manera aleatoria, clasificándolos en pequeños (2 a 4 proveedores, 2 a 3 muelles, 3 a 8 clientes y 6 a 12 vehículos), medianos (10 proveedores, 5 a 6 muelles, 20 a 30 clientes y 40 a 50 vehículos) y grandes (20 a 30 proveedores, 20 a 15 muelles, 60 a 70 clientes y 180 200 vehículos). Para cada tamaño de problema se crearon 3 combinaciones de numero de

proveedores, muelles, clientes y vehículos y para cada combinación se crearon 3 instancias, formando un total de 27.

Los resultados obtenidos muestran que para instancias pequeñas, tanto CH-TLSA como CH-TLTS encuentran la solución óptima al igual que CPLEX para 3 de 9 experimentos, sin embargo los algoritmos propuestos resultan ser mucho más rápido (menor a 0.001 segundos) en comparación con CPLEX (promedio de 2100 segundos). Para las otras instancias CPLEX no encuentra resultados, mientras que las propuestas encuentran soluciones de buena calidad en la misma cantidad de tiempo mencionada. Cabe destacar que para todas las instancias pequeñas la solucion inicial resulta ser la mísma que la final, lo que indica que el CH es muy eficaz, provocando que TLTS Y TLSA no puedan mejorar dicha solucion.

Para instancias medianas y grandes CPLEX no encontró soluciones de buena calidad, por lo que no se considera en las siguientes comparaciones. Para las instancias medianas se observa que CH-TLTS obtiene soluciones de mejor calidad que CH-TLSA en tiempos similares (ambos inferior a 1 minuto), mientras que para las instancias más grandes CH-TLSA supera a CH-TLTS encontrando soluciones de mejor calidad y en un tiempo menor (10-30 minutos para CH-TKSA y 15-75 para CH-TLTS).

Finalmente, en [11] se vuelve a estudiar el problema tratado en [29], [19] y [28]. En esta ocasión, los autores proponen una metaheurística basada en Large Neighborhood Search (LNS) para crear un conjunto de rutas, luego dicho conjunto es usado en un problema de Set Partitioning and matching (SPM), para finalmente ser resuelto utilizando branch-and-check, el cual es un método híbrido que se basa en un solver de programación entera mixta (MIP) y uno de programación de restricciones (CP).

El componente principal del algoritmo propuesto por los autores es el LNS, el cual se encuentra reforzado por una solución obtenida por el SMP. Cuando el LNS encuentra una nueva solución, tanto la ruta de recogida, como la de entrega son agregadas a un conjunto de rutas que actúa como una memoria. El SMP se encuentra basado en el conocido Set Partitioning Problem (SPP), en donde el conjunto a particionar se encuentra compuesto por los proveedores y los clientes, y las particiones se encuentran en el conjunto de rutas mencionado anteriormente.

El LNS planteado modifica la solución actual utilizando diversos métodos llamados destructores y reparadores. Los primeros, consisten en remover pedidos de la solución actual, mientras que los segundos consisten en insertar pedidos en dicha solución. Se utilizan una variedad de métodos diferentes, los cuales son seleccionados al azar en cada iteración.

Entre los métodos destructores se encuentra el llamado *Random removal*, el cual remueve un pedido de manera aleatoria, el *Worst removal*, en el cual para cada pedido se calcula la diferencia entre el costo de la solución con y sin el pedido, y luego se ordenan de mayor a menor, seleccionando un pedido aleatorio utilizando un proceso similar a la ruleta de los algoritmos genéticos. El *Historical node-pair removal* ordena los pedidos de acuerdo al coste de sus arcos (trayectos en las rutas) involucrados, luego se elimina el pedido que involucre los arcos con el mayor coste asociado, utilizando un proceso de selección similar al de *Worst removal*.

Por otro lado, entre los métodos reparadores se encuentra el *Best insertion*, en donde se inserta a la solución actual el pedido no asignado que resulte en el menor costo de inserción, considerando tanto la ruta de recogida como la de entrega. El método *Best insertion* calcula para cada pedido sin satisfacer y para cada par de vehículos (recogida y entrega, que podrían ser el mismo) el costo de la inserción con menos costo, luego con dichos costos calculados se calcula un indicador que es la diferencia entre el costo de dicha inserción y la segunda con menor inserción. Finalmente se inserta el pedido cuyo indicador sea el mayor. Los detalles acerca de los métodos utilizados y la estructura del algoritmo propuesto se encuentran en [11].

Para validar el método propuesto, los autores realizan una serie de experimentos utilizando los set de pruebas de [29] y [19], y establecen un criterio de término de 20.000 iteraciones, ejecutando cada instancia 10 veces. Los experimentos son realizados utilizando *CPLEX y CP Optimizer de IBM ILOG Cplex Studio* como el MIP y CP solver respectivamente.

Los experimentos muestran que el método propuesto por los autores encuentra mejores soluciones que los métodos existentes para las instancias de [29] y tiene mejores resultados promedios para todas excepto 4 de las instancias propuestas, también supera las mejores soluciones para 19 de las 35 instancias con un promedio de 0.57 %. Se menciona que los

tiempos de ejecución son generalmente mas altos que los que se encuentran en la literatura, sin embargo se muestra que la convergencia de las soluciones es rápida, y éstas son mejores en varias pruebas respecto a las existentes. Respecto a los resultados obtenidos en [28], se observa que el LNS+SPM propuesto supera en todas las instancias excepto las más pequeñas, por el hecho de que encuentra soluciones de una calidad muy similar en un tiempo significativamente más corto. Finalmente se concluye que LNS+SPM es una muy buena alternativa para las instancias medianas (100 a 200 nodos), mientras que en las pequeñas (50 nodos) y grandes (300 a 500 nodos) no tiene muy buenos resultados.

2.1. Resumen

Debido a la gran cantidad y variedad de acercamientos al problema del VRPCD realizados hasta la fecha, es posible realizar un cuadro comparativo de los estudios mencionados, mostrando las características principales que se abordan en cada uno de ellos. La Tabla 2.1 muestra un resumen de la literatura comentada anteriormente sobre el VRPCD junto a sus restricciones. La segunda columna *Tipo prod* describe cuatro clasificaciones: igual (todos los proveedores proveen el mismo producto), proveedores (las demandas de los clientes son independientes por cada proveedor), múltiples (hay una variedad de productos, los cuales puede ser ofrecidos por múltiples proveedores), y 1:1 (cada cliente tiene demanda de solo un proveedor y éste puede proveer a sólo un cliente). La tercera columna *Muelles* indica la cantidad de muelles de intercambio que considera el problema, estos valores pueden ser 1 o *Múltiples*. La cuarta columna indica si los vehículos considerados son heterogéneos (cada uno con características distintas a los demás) o no. La quinta columna *Viaje Directo* señala si es permitido hacer un viaje desde un proveedor hacia un cliente de manera directa, sin pasar por un muelle de intercambio. La sexta columna *P-P C-C* indica si los trayectos de proveedor a proveedor y cliente a cliente se encuentran permitidas. La séptima columna *E.D* indica si se permiten las entregas divididas, es decir, cuando varios vehículos entregan bienes al mismo cliente en el mismo o diferente tiempo, esto permite que los nodos puedan ser visitados mas de una vez por diferentes vehículos. La octava columna *V.T.* indica si el problema considera ventanas de tiempo. Finalmente la última columna *Invent.* señala si los muelles de

intercambio permiten almacenar bienes por un periodo de tiempo establecido, como modo inventario bajo un cierto costo.

Artículo	Tipo Prod.	Muelles	Vehículos Heteros	Viaje Directo	P-P C-C	E.D	V.T	Invent.
[29]	1:1	1	-	-	✓	-	✓	-
[11]	1:1	1	-	-	✓	-	✓	-
[20]	Prov.	Multi.	-	✓	-	-	-	-
[26]	Igual	1	-	-	✓	-	-	-
[28]	1:1	1	-	-	✓	-	✓	-
[7]	Multi.	1	-	-	✓	-	✓	-
[19]	1:1	1	-	-	✓	-	✓	✓
[1]	Multi.	Multi.	✓	-	✓	-	-	✓
[14]	Prov.	Multi.	✓	-	✓	✓	✓	✓
Memoria	1:1	Muti.	-	-	✓	-	✓	-

Tabla 2.1: Comparación de la literatura existente del VRPCD.

Capítulo 3

Propuesta

En la presente sección, se propone un método basado en GRASP para resolver el VRPCD descrito en el capítulo 1, primero se describe y explica de manera general el método propuesto, y posteriormente, se presenta cada una de sus componentes de manera detallada, explicando las representaciones correspondientes y brindando una explicación detallada del algoritmo a utilizar.

3.1. GRASP

La palabra GRASP proviene de las siglas de Greedy Randomized Adaptive Search Procedures lo cual se puede traducir como Procedimientos de Búsqueda Voraces Aleatorios y Adaptativos. Esta metaheurística fue propuesta inicialmente por Feo y Resende en 1989 [9], y luego fue definida formalmente por los mismos autores en 1994 en [10], en donde explican los componentes de la metaheurística de manera detallada e introducen una especie de pauta para desarrollar algoritmos de este tipo. Esta técnica se enfoca en su mayor parte en construir soluciones de alta calidad que posteriormente son procesadas para obtener otras soluciones aún mejores. Los algoritmos basados en GRASP son iterativos, en donde cada iteración se compone principalmente de dos fases llamadas fase de construcción y fase de post-procesamiento. En la fase de construcción se construye una solución inicial utilizando

un algoritmo voraz (Greedy) aleatorizado, es decir, para construir la solución, en cada paso se cuenta con una lista de candidatos llamada RCL (Restricted Candidate List), en la que se incluyen las mejores opciones en orden decreciente respecto a su calidad, escogidas por la función miope. Luego se escoge de manera aleatoria un candidato de la lista para agregarlo a la solución actual, de manera que en cada iteración se obtienen diferentes soluciones iniciales de buena calidad. Por otro lado, una vez finalizada la fase de construcción, se inicia la fase de post-procesamiento, en la cual se aplica un procedimiento de búsqueda local a la solución generada, con el objetivo de mejorarla y así encontrar una solución óptima local. En ocasiones se puede agregar una fase adicional, llamada fase de pre-procesamiento, en donde se pueden identificar subestructuras que formen parte de la solución óptima, de manera de reducir el tiempo en la fase de construcción. Esta premisa se basa en que dichas subestructuras pueden servir como distintos puntos de partida para comenzar la búsqueda, sin embargo, para que ésta fase sea de utilidad se debe tener un conocimiento previo sobre la solución óptima, para poder identificar de manera correcta las subestructuras correspondientes y aplicarlas de manera correcta en la fase de construcción. Para el caso particular del VRPCD estudiado en esta memoria, no se implementará la fase de pre-procesamiento.

La estructura general del GRASP se muestra a continuación en el Algoritmo 1.

Algoritmo 1 Pseudocódigo de un GRASP general

```
Inicializar()
while no se cumpla el criterio de término do
    Preprocesamiento()
    SolucionInicial ← AlgoritmoConstructivo()
    SolucionActual ← AlgoritmoBusquedaLocal(SolucionInicial)
    if SolucionActual tiene mejor calidad que MejorSolucion then
        MejorSolucion ← SolucionActual
    end if
end while
Return MejorSolucion
```

3.2. Propuesta de GRASP para el VRPCD

En esta sección se describe en detalle el algoritmo GRASP propuesto, explicando la representación de la solución considerada junto con los distintos elementos que conforman las fases de construcción y post-procesamiento.

3.2.1. Representación

El objetivo del VRPCD es conocer las rutas de recogida y de entrega de bienes para cada vehículo utilizado en el proceso, con el fin de minimizar el costo total que conlleva realizar todo el proceso de repartición de bienes. A partir de esta necesidad es que se utiliza una representación utilizando vectores, la cual es explicada en detalle a continuación.

3.2.2. Ruta de un vehículo

Se define como la ruta de un vehículo al conjunto de los nodos visitados por este, partiendo y terminando en un muelle de intercambio. Por conveniencia, las rutas asociadas a cada vehículo son divididas en rutas de recogida, en donde los vehículos visitan los nodos para recoger productos de los respectivos proveedores, rutas de consolidación, en donde los vehículos viajan entre los distintos muelles disponibles para realizar los procesos de carga y descarga de los productos, y rutas de entrega, en donde los nodos son visitados para entregar los productos a los clientes correspondientes. Las rutas de un vehículo son representadas utilizando un vector de números enteros, los cuales con signo positivo representan los nodos de los proveedores, mientras que los negativos representan los nodos de los clientes que tienen demanda sobre el respectivo proveedor del mismo número. Por ejemplo, el nodo -1 tiene una cierta demanda sobre el proveedor 1, el -2 sobre el 2, etc. El muelle de intercambio es representado por el número 0. La representación de las rutas correspondientes a un vehículo

se muestra a continuación.

$$Ruta\ Recogida = [1, 3, 4, 7]$$
$$Ruta\ Consolidación = [0]$$
$$Ruta\ Entrega = [-1, -3, -4, -7]$$

En este ejemplo, se muestra que el vehículo parte en el muelle (siempre debe ser el punto de partida) y comienza la ruta de recogida visitando los proveedores 1, 3, 4 y finalmente el 7, luego el vehículo vuelve hacia el muelle (denotado por 0), para posteriormente realizar la ruta de entrega hacia los clientes -1, el cual tiene una demanda determinada sobre el proveedor 1, análogamente visita los clientes -3, -4 y -7, para finalmente volver al muelle 0, el cual debe ser también el nodo final.

3.2.3. Solución del VRPCD

Como el problema busca encontrar las rutas óptimas para todos los vehículos que se utilicen para satisfacer todas las demandas correspondientes, es necesario una representación que permita mostrar dicho conocimiento. Una solución del VRPCD es representada utilizando un vector de vectores de rutas, en donde la posición i de la solución corresponde al vector que contiene las 3 rutas correspondientes del vehículo i. La representación utilizada se muestra a continuación.

$$\begin{aligned} Solución = [&[RutaRecogida1, RutaConsolidacion1, RutaEntrega1], \\ &[RutaRecogida2, RutaConsolidacion2, RutaEntrega2], \\ &\vdots \\ &[RutaRecogidaN, RutaConsolidacionN, RutaEntregaN]] \end{aligned}$$

Para un mejor entendimiento de la representación utilizada, se muestra el siguiente ejemplo.

$$\begin{aligned} Solución = &[[[1, 3, 4], [0], [-1, -2, -3]], \\ &[[2, 5], [0], [-4, -5]], \\ &[[6, 7], [0], [-6, -7]]] \end{aligned}$$

En este caso, el vehículo 1 parte su ruta de recogida por los proveedores 1, 3 y 4, mientras que el vehículo 2 visita los proveedores 2 y 5. Luego ambos vehículos van hacia el muelle 0 y realizan el proceso de consolidación, en donde el vehículo 1 descarga las bienes correspondientes al proveedor 4 y el vehículo 2 descarga los bienes del proveedor 2. Acto seguido dichos bienes son intercambiados, es decir, el vehículo 1 carga los bienes del proveedor 2 y el vehículo 2 carga los bienes del proveedor 4. Finalmente ambos comienzan la ruta de entrega visitando los clientes -1, -2, -3 y -4, -5 respectivamente. Por otro lado, el vehículo 3 realiza la ruta de recogida visitando los proveedores 6 y 7, luego se dirige al muelle 0 para comenzar la ruta de entrega por los clientes -6 y -7.

3.2.4. Inicialización

La fase de inicialización es la primera de todo el proceso de encontrar una solución para el VRPCD. Dicha fase se realiza solo una vez al momento de leer la instancia, y es la encargada de obtener la mayor cantidad de información posible sobre el problema y la instancia a resolver. Al momento de realizar la lectura de los datos, se obtiene la información de todos los pedidos que tienen los clientes sobre sus respectivos proveedores. La información de cada pedido esta compuesta por la cantidad de bienes que solicita un determinado cliente respecto a un proveedor. Adicionalmente se tienen las coordenadas físicas de ambos y las ventanas de tiempo correspondientes para cada uno de estos. Con esta información es posible crear las estructuras correspondientes a los pedidos y a los nodos totales (clientes, proveedores y muelles). Por otro lado al leer los datos es posible rescatar la información correspondiente a las capacidades de los vehículos, el tiempo fijo de cada uno cuando se preparen en un muelle, y el tiempo de carga y descarga por unidad de bienes, por lo que es posible inicializar las estructuras de los vehículos con dicha información.

3.2.5. Fase constructiva

Como se menciono anteriormente, la fase constructiva del algoritmo propuesto corresponde a un Greedy con una componente aleatoria, con el fin de generar soluciones iniciales de

buena calidad y diferentes en cada iteración. Como el objetivo del VRPCD es minimizar los costos asociados a las rutas de los vehículos utilizados, se cuenta con una lista de pedidos insatisfechos y una función miope que selecciona el pedido que minimice el costo total de la ruta del vehículo que la va a satisfacer y que cumpla con las restricciones de las ventanas de tiempo y la capacidad del vehículo, es decir, se comienza con un vehículo con sus rutas vacías, luego en cada iteración se evalúan los costos de cada pedido cuya demanda sea menor o igual a la capacidad actual del vehículo. La evaluación de cada pedido consiste en calcular el costo del viaje desde el último nodo visitado de su ruta de recogida (en caso de ser la primera iteración, el nodo inicial es el muelle), hacia el proveedor de dicho pedido. Luego a este costo se le suma el viaje de dicho proveedor hacia el muelle de intercambio más cercano, y posteriormente se calcula el costo de ir desde el último cliente visitado en la ruta de entrega (en el caso de ser la primera iteración, se considera el muelle como punto de partida), hacia el cliente del respectivo pedido. Finalmente, se calcula el costo de ir desde dicho cliente hasta el depósito de vehículos, y con esto se obtiene el costo total que incurre utilizar dicho vehículo para realizar la ruta y satisfacer el pedido evaluado. Finalmente se selecciona el pedido que tenga el costo más bajo y que cumpla con las restricciones de las ventanas de tiempo establecidas para cada nodo, y se añade el proveedor y el cliente del pedido a la ruta de recogida y entrega del vehículo respectivamente, luego se disminuye la capacidad actual del vehículo en la misma cantidad que la demanda del pedido seleccionado, posteriormente el pedido se elimina de la lista de pedidos insatisfechos. Cuando en una iteración ningún pedido pueda ser satisfecho por el vehículo actual, se crea un nuevo vehículo y se repite el proceso hasta que todos los pedidos hayan sido asignados a un vehículo.

Como la fase constructiva del GRASP posee una componente aleatoria, se utiliza la función miope mencionada anteriormente y se genera una lista ordenada de manera creciente de los pedidos menos costosos, la cual corresponde a la lista restringida de candidatos (LRC) que forma parte de todo algoritmo GRASP. Luego en cada iteración se selecciona un pedido de la LRC de manera aleatoria y se añaden los nodos respectivos al vehículo como se mencionó anteriormente. De esta manera, se tendrá en cada iteración una solución de buena calidad y diferente a las anteriores. El algoritmo 2 muestra el pseudocódigo de la fase constructiva recientemente explicada.

Algoritmo 2 Pseudocodigo de la Fase Constructiva del GRASP para el VRPCD

```
Inicializar lista pedidosPendientes
Inicializar Solucion vacia
N ← Largo LRC
while pedidosPendientes != ∅ do
    vehiculoActual ← Crear nuevo vehículo
    while exista al menos un pedido en pedidosPendientes que cumpla con las restricciones
    para el vehiculoActual do
        LRC = Obtener los N pedidos menos costosos
        Pedido ← Seleccionar un pedido aleatorio de LRC
        vehiculoActual ← Asignar proveedor y cliente de Pedido a la ruta de recogida y
        entrega respectivamente
        Eliminar Pedido de pedidosPendientes
    end while
    Agregar vehiculoActual a Solucion
end while
Return Solucion
```

3.2.6. Fase de post-procesamiento

El objetivo de la fase de post-procesamiento es mejorar las soluciones obtenidas en la fase de construcción, utilizando un procedimiento de búsqueda local. Esta fase se compone de diferentes movimientos, los cuales modifican las rutas de los distintos vehículos de diferentes formas. Los movimientos utilizados, así como su descripción se presentan en esta sección.

2-OPT en rutas de recogida y entrega

Como la representación de las rutas definidas para realizar el algoritmo consiste en un vector de números enteros, los cuales representan los distintos nodos por los que debe pasar el vehículo, resulta factible realizar la operación *2-OPT exchange* en ambas rutas, con el fin de explorar distintas combinaciones de nodos dentro de la misma ruta. Dicho movimiento

consiste en seleccionar un vehículo al azar de la solución inicial y escoger de manera estocástica si se modificará la ruta de recogida o la de entrega. Luego se selecciona al azar un nodo de la ruta seleccionada y se intercambia de lugar con su nodo adyacente posterior. En el caso que el nodo seleccionado sea el último de la ruta, se intercambia por el nodo adyacente anterior. Este movimiento se realiza de dicha forma para favorecer la factibilidad respecto a las ventanas de tiempo, ya que es más probable que la solución obtenida sea factible debido a que al intercambiar nodos adyacentes no transcurre demasiado tiempo de viaje entre estos, aumentando la probabilidad de que se cumplan ambas ventanas de tiempo. Para un mejor entendimiento del movimiento descrito se presenta el siguiente ejemplo, en donde el nodo seleccionado sera el de la posición 3, por lo tanto, este se intercambiará por el de la posición 4.

$$Ruta\ original = [1, 3, 7, 4, 6, 5, 9, 8, 2]$$
$$Ruta\ después\ de\ 2-OPT = [1, 3, 4, 7, 6, 5, 9, 8, 2]$$

Intercambio de nodos entre vehículos

Por las mismas razones mencionada en el primer movimiento descrito, resulta provechoso realizar un intercambio de nodos del mismo tipo de rutas (ambas de recogida o de entrega) entre dos vehículos distintos, con el fin de explorar distintas combinaciones de nodos para cada vehículo. De esta manera se busca diversificar el espacio de búsqueda. Este movimiento tiene como consecuencia generar el proceso de consolidación en los muelles de intercambio, ya que intercambiar un nodo de cualquier ruta por otro que no haya sido del mismo pedido, conlleva a que unos vehículos deban descargar bienes en un muelle central, y otros deban cargarlos para luego ser entregados. El movimiento consiste en seleccionar un vehículo de manera aleatoria y se intercambia cada nodo de una ruta de dicho vehículo con cada nodo del mismo tipo de ruta de todos los otros vehículos de la solución, hasta que se encuentre una mejora respecto a la solución actual. Cuando se encuentre dicha mejora, el procedimiento para y se retorna la nueva solución. En el caso de que no se encuentre una mejora se retorna la solución a la cual se le aplicó el movimiento. Para un mejor entendimiento el algoritmo 3 muestra el pseudocódigo del movimiento.

Algoritmo 3 Pseudocódigo movimiento Intercambio de nodos

```
Inicializar solucionActual
nuevaSolucion = solucionActual
vehiculoSeleccionado ← random(vehiculos de solucionActual)
while queden vehiculos en solucionActual sin probar do
  vehiculoPrueba ← vehiculo de la solucionActual sin probar
  for cada nodo i ∈ ruta de vehiculoSeleccionado do
    for cada nodo j ∈ ruta de vehiculoPrueba do
      intercambiar nodo i con j
      nuevaSolucion ← actualizar los vehiculos con las nuevas rutas
      if funcionEvaluacion(nuevaSolucion) < funcionEvaluacion(solucionActual) then
        return nuevaSolucion
      else
        nuevaSolucion = solucionActual
      end if
    end for
  end for
end while
return solucionActual
```

Cambio de nodo entre vehículos

Con el fin de diversificar la forma de las soluciones y así explorar de manera mas amplia el espacio de búsqueda, resulta conveniente realizar un movimiento de cambio de nodos entre 2 vehículos, ya que existe la posibilidad de que disminuyendo la carga de algunas rutas, disminuya también el costo de la solución. Este movimiento consiste en seleccionar un vehículo de manera aleatoria, al igual que el tipo de ruta a modificar. Luego se comienza a iterar sobre los demás vehículos de la solución, quitando cada nodo de la ruta escogida del vehículo seleccionado e insertándolo en cada posición posible de la ruta correspondiente del vehículo de la iteración. En el caso de encontrar una mejora en la solución, esta se retorna inmediatamente dando fin al movimiento. Si no encuentra una mejora se retorna la solución a

la cual se le aplicó el movimiento. Para un mejor entendimiento, se muestra el pseudocódigo del movimiento en el algoritmo 4.

Algoritmo 4 Pseudocódigo movimiento Cambio de nodos

```
Inicializar solucionActual
nuevaSolucion = solucionActual
vehiculoSeleccionado ← random(vehiculos de solucionActual)
while queden vehiculos en solucionActual sin probar do
  vehiculoPrueba ← vehiculo de la solucionActual sin probar
  for cada nodo i ∈ ruta de vehiculoSeleccionado do
    quitar nodo i de vehiculoSeleccionado
    for cada nodo j ∈ ruta de vehiculoPrueba do
      insertar nodo i en la posicion j de vehiculoPrueba
      nuevaSolucion ← actualizar los vehiculos con las nuevas rutas
      if funcionEvaluacion(nuevaSolucion) < funcionEvaluacion(solucionActual) then
        return nuevaSolucion
      else
        nuevaSolucion = solucionActual
      end if
    end for
  end for
end while
return solucionActual
```

Selección del movimiento.

Los movimientos pueden priorizar la exploración o la explotación según corresponda, por ejemplo en el caso del *2-OPT* se prioriza la exploración, mientras que en el *cambio de nodos* se prioriza la explotación. Es por esto que resulta conveniente poder controlar que movimiento se seleccionará en que etapa de la ejecución con el fin de poder aprovechar su funcionalidad, por ejemplo puede resultar conveniente dar prioridad a la exploración al comienzo de la ejecución y luego intensificar, o detectar cuando el algoritmo quede atascado

en un óptimo local y priorizar la exploración. Por otro lado, las instancias en las que se prueba el algoritmo son muy variadas en cuanto a tamaño y características, por lo que resulta interesante poder controlar que operador utilizar dependiendo de estas variables.

Para seleccionar el movimiento a utilizar en cada iteración se emplea un método de selección adaptivo, el cual determina el rendimiento de cada movimiento a medida que avanzan las iteraciones con el fin de utilizar el más efectivo en cada una de ellas. Este método de selección esta basado en el *fitness-rate-rank-based multiarmed bandit* o conocido por sus siglas como FRRMAB propuesto en [17]. El método está compuesto principalmente de dos fases, una de asignación de puntaje y otra de selección de movimiento.

- **Fase 1: Asignación de puntaje.**
  La primera fase se encarga de medir el impacto que tiene un movimiento en el proceso de búsqueda y de cómo asignarle un puntaje adecuado basado en dicho impacto. Para esto, se utilizan directamente los valores brutos de las mejoras de la función de evaluación, causado por los usos recientes del movimiento a evaluar. Sin embargo, es normal que debido a la convergencia de los métodos de búsqueda, la mejora bruta de la función de evaluación sea mayor en las primeras iteraciones y disminuya considerablemente a medida que éstas avanzan, por lo que se utiliza una tasa de mejora llamada *fitness improvement rate* (FIR). El FIR obtenido por el iterador i en el tiempo (iteración) t está dado por:

$$FIR_{it} = \frac{pf_{it} - cf_{it}}{pf_{it}}$$

  en donde pf_{it} es la calidad de la solución padre (antes de aplicar el movimiento) y cf_{it} es la calidad de la solución obtenida luego de aplicar el movimiento. Para analizar el comportamiento de los operadores a medida que avanzan las iteraciones, se define una ventana deslizante W de tamaño fijo, la cual se encarga de almacenar los valores del FIR para los operadores utilizados recientemente y cuyo método de actualización sigue el criterio FIFO (first-in first-out). Cada lugar de W almacena tanto el operador utilizado como el respectivo FIR obtenido, de esta manera se logra guardar el comportamiento de los operadores en un intervalo definido de iteraciones, controlando el

problema de la convergencia mencionado anteriormente. Para la asignación de un puntaje adecuado a cada operador, se calcula la recompensa de cada uno de ellos como la suma de todos sus respectivos FIR almacenados en W. Luego, se realiza un ranking descendente de los operadores según dicha recompensa, y para dar mas oportunidad a los mejores operadores (intensificar), se introduce un factor de decaimiento $D \in [0, 1]$ como:

$$Decaimiento_i = D^{Rank_i} \times Recompensa_i$$

Posteriormente se asigna el puntaje final a cada operador llamado *fitness-rate-rank* definido como:

$$FRR_{it} = \frac{Decaimiento_i}{\sum_{j=1}^{K} Decaimiento_j}$$

Para controlar la intensificación en el proceso se varia el valor de D. Un valor pequeño de D favorece la influencia del mejor operador. Para un mejor entendimiento de esta fase, en el algoritmo 5 se muestra el pseudocódigo del procedimiento descrito, en donde n_i corresponde a la cantidad de veces que el movimiento i se encuentre en W y K es la cantidad de movimientos.

Algoritmo 5 Pseudocódigo de asignación de puntaje

Inicializar cada recompensa $recompensa_i = 0$

Inicializar $n_i = 0$

for i ← 1 hasta W.size() **do**

mov = W.getMov(i)

FIR = W.getFIR(i)

$recompensa_{mov}$ += FIR

n_{mov}++

end for

Rankear los movimientos de acuerdo al FIR en orden descendiente.

$Rank_i$ ← ranking del movimiento i

for mov ← 1 hasta K **do**

$decaimiento_mov = D^{Rank_{mov}} \times recompensa_{mov}$

end for

decSum = $\sum_{mov=1}^{K} decaimiento_{mov}$

for mov ← 1 hasta K **do**

$FRR_{mov} = \frac{decaimiento_{mov}}{decSum}$

end for

- **Fase 2: Selección de movimiento.**

  Basado en los FRR calculados en la Fase 1, el método seleccionara el operador adecuado en cada iteración para generar una nueva solución. En el caso de que algunos movimientos tengan muy buenos resultados en algunas iteraciones, estos serán mas propensos a ser seleccionados nuevamente, provocando que aquellos que no han dado buenos resultados en las últimas iteraciones no puedan volver a ser escogidos. Es por esto que se aplica el factor C que controlará que lo mencionado anteriormente no suceda, y que permitirá que el algoritmo pueda explorar de mejor forma el espacio de la solución.

  Cabe destacar que al comienzo de la búsqueda, ningún movimiento se ha aplicado aun, por lo que se le da a todos los movimientos la misma probabilidad de ser seleccionados al principio, hasta que todos estos hayan sido aplicados una vez. Posteriormente, la selección del movimiento a utilizar se realizara como se explica en el algoritmo 6.

Algoritmo 6 Pseudocódigo selección de movimiento

if hay movimientos que no han sido utilizados **then**

mov = random(movimientos sin utilizar)

else

mov = $argmax_{i=[1...K]}(FRR_{it} + C\sqrt{\frac{2x\log\sum_{j=1}^{K} n_{jt}}{n_{it}}})$

end if

Cabe destacar que el sumando de la derecha de la fórmula mostrada en el algoritmo 6, permite regular la exploración del algoritmo, provocando que los movimientos que menos hayan sido seleccionados tengan más posibilidad de ejecutarse en la siguiente iteración.

Capítulo 4

Implementación

4.1. Instancias de Prueba

Para ejecutar el algoritmo propuesto, se consideraron dos sets de instancias que han sido importantes a lo largo de la literatura del VRPCD. El primer set corresponde al propuesto en [29], mientras que el segundo corresponde al propuesto en [19].

Primer set: Instancias de [29]

Los datos utilizados para estas instancias se generaron a partir de datos reales pertenecientes a Transvision, una consultora de logística danesa. Los datos de prueba consisten en cinco conjuntos indicados por 200a, 200b, 200c, 200d y 200e, respectivamente, donde 200 representa la cantidad de pares proveedor-cliente. Cada conjunto de datos entrega información sobre la posición de cada proveedor y cliente en el formato (x, y). También se entregan las ventanas de tiempo para cada cliente y proveedor, siendo cada una de estas de 2 horas. Dichas ventanas están pensadas para que la operación se realice en un horizonte de tiempo entre las 6:00 am y 22:00 pm. También se entrega la demanda en unidades que tiene cada cliente con su respectivo proveedor. Adicionalmente se entrega información sobre la capacidad de los

vehículos y su velocidad, la cual es de 33 y 60 km/h respectivamente para todas las instancias. Por otro lado, se entrega la información del tiempo fijo que necesita un vehículo para cargar-descargar en un muelle y el tiempo que demora en cargar-descargar una unidad de un producto, los cuales son 10 y 1 minutos respectivamente. Adicionalmente en [29] se generaron instancias con 20, 30, 50, 100 y 150 pares de proveedor-cliente, utilizando subconjuntos de datos aleatorios obtenidos de las instancias de tamaño 200.

Segundo set: Instancias de [19]

Las instancias propuestas en este artículo fueron generadas utilizando las mismas coordenadas de los nodos de las instancias propuestas en [12], las cuales tienen características de escenarios realistas para el VRPCD. Los nodos fueron divididos de manera aleatoria entre proveedores y clientes. La capacidad de los vehículos es igual a 50 unidades y las ventanas de tiempo fueron generadas como se sugiere en [18]. El tiempo fijo para el proceso de carga y descarga de un vehículo es de 5 minutos y el tiempo de carga y descarga de unidad de producto es de 1 minuto. La ubicación de los nodos se muestra en coordenadas cartesianas, y el costo de viajar entre un nodo y otro es igual a la distancia euclidiana entre estos.

Este conjunto de pruebas posee instancias de tamaño 200, 300, 400 y 500, en donde para cada uno de estos hay 5 instancias diferentes.
A diferencia de las instancias de [29], los autores no mencionan la velocidad de los vehículos, por lo que se asume que ésta es de 1 km/h.

4.2. Experimentos

Los experimentos realizados consistieron en ejecutar todas las instancias descritas en la sección anterior. Para esto se realizó un proceso de sintonización, en donde se encontraron los mejores valores de parámetros para ejecutar dichas instancias. El proceso de sintonización se describe en la siguiente sección.

4.2.1. Sintonización

El proceso de sintonización consiste principalmente en encontrar los mejores valores posibles de parámetros para realizar la posterior ejecución del algoritmo. El método de sintonización utilizado en esta memoria fue *ParamILS*, el cual se encuentra basado en el proceso de sintonización manual. ParamILS trabaja como un algoritmo de búsqueda local iterativa, en el cual se comienza con una configuración inicial por defecto y de manera iterativa intenta mejorar el desempeño del algoritmo cambiando los valores (uno a la vez) de los parámetros. ParamILS recibe como input un conjunto finito de valores para cada parámetro a sintonizar y retorna la mejor configuración de parámetros encontrada.

Para realizar el proceso de sintonización del GRASP propuesto se escogieron los siguientes valores para los parámetros a sintonizar.

1	Tamaño Lista	1, 2, 3, 4, 5, 7, 10
2	Iteraciones GRASP	10, 100, 500, 1000
3	Slide Window	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
4	Decay Factor	0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
5	Explore Factor	0.5, 1.0, 2.0, 5.0

Tabla 4.1: Tabla que muestra los valores posibles para los distintos parámetros en la sintonización del GRASP adaptivo propuesto.

El primer parámetro indica el tamaño de la LRC perteneciente al GRASP, es decir, la cantidad de soluciones candidatas que se considerarán al momento de construir la solución inicial. El segundo, indica la cantidad de veces que se ejecutara el algoritmo GRASP completo. El tercer parámetro indica el tamaño de la ventana deslizante como porcentaje respecto a la cantidad de iteraciones del algoritmo, en la fase 1 del proceso de selección de movimiento del GRASP, es decir, la cantidad de iteraciones que se guardaran los valores del FIR junto a su respectivo operador. El cuarto parámetro indica el factor de decaimiento, el cual controla la intensificación del algoritmo, utilizado en el mismo proceso que el parámetro 3. Finalmente el último parámetro indica el factor de exploración de la fase 2 de selección de movimiento, el cual controla la diversificación del algoritmo. Por otro lado, se consideró una cantidad

fija para las evaluaciones totales de 10000000, por lo que para calcular las iteraciones de la fase de post-procesamiento del GRASP se divide dicho valor en el valor del parámetro *Iteraciones GRASP*. Adicionalmente, de modo de evaluar la efectividad del mecanismo de control adaptivo, se implementó un GRASP estándar, el cual no posee la parte de selección de movimiento adaptiva, es decir cada movimiento tiene una probabilidad fija de ser ejecutado en cada iteración, en donde dichas probabilidades son dadas como parámetros del algoritmo. Luego, el GRASP estándar se sintonizó utilizando *ParamILS* y se ejecutaron las mismas pruebas que el GRASP adaptivo, con el fin de comparar el desempeño del mecanismo de control de movimientos y observar el comportamiento de éste en las distintas instancias.

1	Tamaño Lista	1, 2, 3, 4, 5, 7, 10
2	Iteraciones GRASP	10, 100, 500, 1000
3	Prob. Mov. 2-OPT	[0,1]
4	Prob. Mov. Change Node	[0,1]
5	Prob. Mov. Swap Pick	[0,1]
5	Prob. Mov. Swap Delivery	[0,1]

Tabla 4.2: Tabla que muestra los valores posibles para los distintos parámetros en la sintonización del GRASP estándar.

Para ambos procesos de sintonización se utilizó un tiempo límite de 300 segundos. Los resultados de la sintonización del GRASP adaptivo se muestran en la tabla 4.3.

1	Tamaño Lista	3
2	Iteraciones GRASP	1000
3	Slide Window	0.6
4	Decay Factor	0.7
5	Explore Factor	2

Tabla 4.3: Tabla que muestra los resultados de la sintonización del GRASP adaptivo.

Respecto a los resultados de la sintonización de la tabla 4.3, se observa que el resultado del tamaño de la LRC es de 3, lo que significa que para generar la solución inicial, se dispondrá

de los 3 mejores candidatos en cada fase de la construcción de la solución. Dicha cantidad es aceptable para las instancias pequeñas (20 a 50 nodos), ya que la cantidad de nodos es pequeña y el espacio de búsqueda no es tan grande, permitiendo obtener una solución inicial de buena calidad, sin embargo para las instancias más grandes (100 a 500 nodos) dicha cantidad es muy pequeña, considerando la cantidad de nodos que pueden ser candidatos para su selección, lo que puede provocar dificultades para que la solución inicial comience en un punto lejano al anterior en el espacio de búsqueda. Por otro lado, el valor para la *slide window* es de 0.6, lo que indica que su tamaño será del 60 % de las iteraciones que realice la fase de post-procesamiento del GRASP. Mientras más grande sea el tamaño de la *slide window*, los puntajes obtenidos por cada movimiento ejecutado durarán más iteraciones, lo que permitirá que los movimientos que en algún momento obtuvieron buenos resultados puedan volver a ejecutarse en las próximas iteraciones. En el caso del *decay factor*, se obtuvo un valor de 0.7 en un rango de 0 a 1, lo cual es bastante alto, evitando que el proceso de control se sesgue siempre hacia los mejores operadores. Finalmente el valor para el parámetro *explore factor* es de 2, lo que permite que los movimientos que hayan sido utilizados pocas veces tengan más posibilidades de ser seleccionados en las próximas iteraciones.

Por otro lado, los resultados de la sintonización del GRASP estándar se muestran en la tabla 4.4.

1	Tamaño Lista	1
2	Iteraciones GRASP	10
3	Prob. Mov. 2-OPT	0.64
4	Prob. Mov. Change Node	0.07
5	Prob. Mov. Swap Pick	0.29
5	Prob. Mov. Swap Delivery	0

Tabla 4.4: Tabla que muestra los resultados de la sintonización del GRASP estándar.

Como se puede observar en la tabla4.4, el valor del tamaño de la LRC es de 1, lo que indica que solo se seleccionará el mejor candidato en cada iteración al construir la solución inicial la cual será igual en todas las fases constructivas del GRASP, dejando de lado el componente aleatorio, lo que corresponde al procedimiento que realiza el algoritmo voraz (greedy). En

el caso de la probabilidad del movimiento *2-OPT*, se observa que es de un 64 %, lo que favorece en gran parte a la exploración del algoritmo, siendo esto completamente necesario para las instancias más grandes. Para la probabilidad de uso del movimiento *change node*, se tiene un valor del 7 %, lo cual es pequeño considerando que es el único movimiento que puede realizar cambios en la estructura de las soluciones, creando rutas más pequeñas y otras más largas, afectando tanto la exploración como la explotación. En el caso del movimiento *swap pick*, se tiene una probabilidad del 29 %, lo cual es razonable debido a que las rutas de pickup son las que más afectan a los tiempos de consolidación, favoreciendo en gran manera a la intensificación del algoritmo. Finalmente se observa que el movimiento *swap delivery* posee una probabilidad de 0 %, provocando que nunca se ejecute, esto se debe a que dicho movimiento no genera grandes mejoras en los tiempos, ya que sólo permite cambiar tiempos en la ruta de entrega, y no en el proceso de consolidación, lo cual también puede ser realizado por los otros movimientos.

4.2.2. Pruebas

Luego de realizar el proceso de sintonización y obtener los mejores valores para los distintos parámetros, se procedieron a realizar las pruebas correspondientes para evaluar el rendimiento del GRASP propuesto. Dichas pruebas consistieron en ejecutar las instancias descritas anteriormente con los parámetros resultantes de la sintonización y comparar los resultados obtenidos por ambos acercamientos con el fin de comprobar la eficacia y eficiencia de la implementación del control de operadores en un GRASP. También se compararon los resultados obtenidos por ambos acercamientos con los resultados existentes en la literatura, con la intención de comprobar el rendimiento de los algoritmos propuestos respecto a los acercamientos de otros autores. Dado la naturaleza estocástica de los algoritmos propuestos, cada instancia se ejecutó un total de 20 veces utilizando distintas semillas aleatorias, para obtener varias soluciones para una misma instancia, con la intención de poder obtener un comportamiento general y libre de aleatoriedad de los algoritmos y así poder compararlos de mejor manera.

4.3. Análisis y Resultados

4.3.1. GRASP adaptivo V/S GRASP estándar

En esta sección se muestran los resultados obtenidos al comparar el algoritmo GRASP adaptivo (selección de movimiento adaptivo) con el GRASP estándar (selección de movimiento aleatorio con probabilidades fijas). Los resultados se obtuvieron utilizando los parámetros resultantes de la sintonización de ambos algoritmos. En primer lugar se compara tanto la calidad de las soluciones obtenidas para todas las instancias descritas anteriormente, como la dispersión de las soluciones obtenidas por ambos algoritmos. Para obtener una mejor visualización y comparación de los resultados, se realizaron distintos boxplots con las calidades de las soluciones obtenidas para cada instancia. Dichos boxplots fueron agrupados en conjuntos de tamaño 5, representando cada grupo a todas las instancias de un tamaño determinado.

Un boxplot es una forma estandarizada de mostrar la distribución de los datos basados en 5 valores distintos, los cuales se muestran en la figura 4.1. Los valores atípicos u outliers son representados por pequeños círculos, y corresponden a aquellos valores que son muy inusuales respecto a los demás, es decir o son extremadamente grandes o extremadamente pequeños, y en algunas situaciones pueden perjudicar el análisis de los datos. Los valores L_i y L_s, corresponden a los valores mínimos y máximos que no se consideran atípicos, es decir los valores límites que pueden tener los datos para que no se consideren fuera de lo común. El valor Q_1 corresponde al primer cuartil, lo que significa que desde el L_i hasta ese valor se encuentra el primer 25 % de los datos de la muestra. El valor Q_2, es el segundo cuartil y corresponde a la mediana, lo que implica que hasta dicho valor se encuentran la mitad de los datos de la muestra. Así mismo. el valor de Q_3 representa el tercer cuartil, indicando que hasta dicho valor se encuentran agrupados el 75 % de los datos.

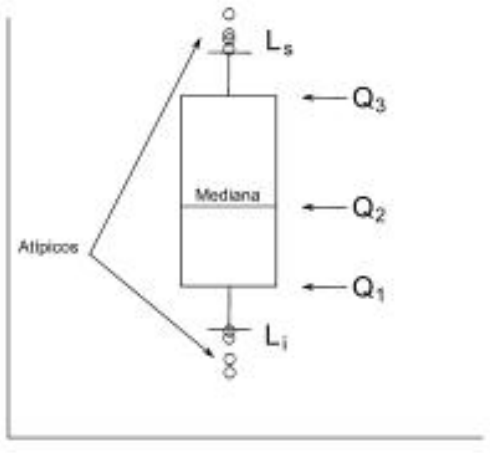


Figura 4.1: Boxplot de ejemplo.

Resultados instancias del set [29]

Para ejecutar las pruebas correspondientes al conjunto de [29], se utilizó un tiempo límite de ejecución de 300 segundos, con el fin de poder establecer una comparación directa de los algoritmos propuestos con el acercamiento de los autores.

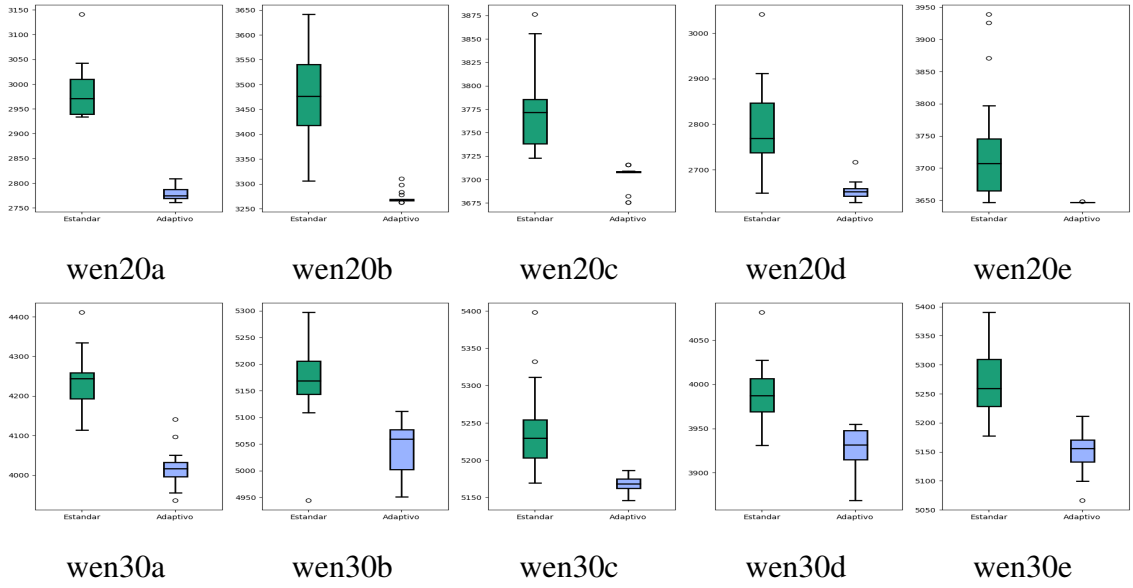


Figura 4.2: Boxplots de los resultados obtenidos para las instancias de [29] de tamaño 20 y 30

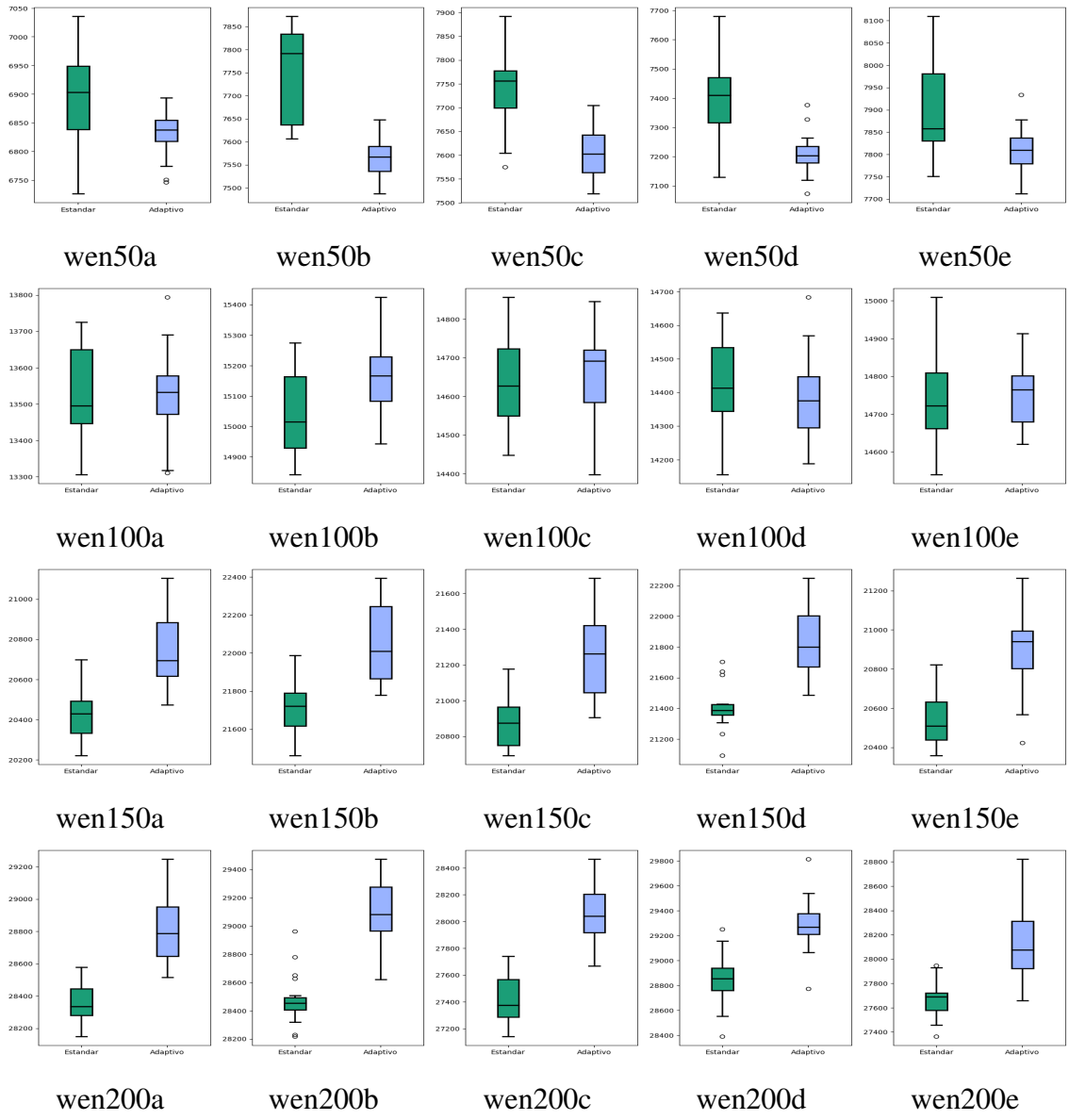


Figura 4.3: Boxplots de los resultados obtenidos para las instancias de [29] de tamaño 50 hasta 200

Al analizar los distintos boxplots expuestos en la figura 4.2, se puede observar que aproximadamente en el 60 % de las instancias (18 de 30) el método GRASP adaptivo posee una menor dispersión de los datos que el método estándar, esto se ve reflejado netamente en el tamaño de los boxplots. También se observa que en general, el método adaptivo posee una mayor cantidad de outliers que el método estándar (25 contra 19), mostrando una mayor

existencia de valores atípicos en sus resultados. Esto tiene sentido, ya que al tener una menor dispersión, los datos se encuentran muy agrupados entre si y en torno a su media, lo que hace que sea mas sencillo clasificar un valor muy alto o bajo como un outlier.
Respecto a la calidad de las soluciones, se muestra que para las instancias más pequeñas (tamaño 20 y 30), el método adaptivo es mejor que el estándar, ya que para el 80 % de dichas instancias(8 de 10) el primero obtiene mejores resultados en cuanto a la mejor solución encontrada, mientras que en la última instancia de tamaño 20 ambos métodos llegan a la misma solución mínima, también el primero posee una mediana menor, y en todos los casos el tercer rango intercuartílico (RIQ, parte superior de la caja) del método adaptivo, se encuentra bajo el primer RIQ (parte inferior de la caja) del método estándar, lo que muestra que el primer 75 % de las soluciones encontradas por el primer método son generalmente de mejor calidad que el último 75 % de las soluciones encontradas por el segundo método.

Por otro lado, a medida que el tamaño de las instancias va aumentando, la brecha entre ambos métodos va disminuyendo, como se observa en la figura 4.3, en el caso de las instancias de tamaño 50 se observa que lo mencionado anteriormente respecto a los RIQ no se cumple para todas las instancias, si no que sólo para 3 de ellas, sin embargo la mediana del método adaptivo sigue siendo menor para todas las instancias respecto del método estándar, demostrando nuevamente que sigue siendo mejor para este tamaño. Dicha tendencia cambia a partir de las instancias de tamaño 100, en donde se muestra que el método adaptivo tiene una mediana mas baja que el estándar en una instancia y que posee en general una menor dispersión de los datos, sin embargo en el resto de ellas el método estándar tiene una mediana mas baja. Por otro lado, a pesar de lo anterior mencionado, el método estándar encuentra una solución mejor que el método adaptivo en 4 de las 5 instancias de este tamaño, siendo mas eficaz en este tipo de instancias.

La misma tendencia anterior se muestra en las instancias de tamaño 150 en donde se observa que el método estándar posee una mediana más baja que el adaptivo para todas las instancias, y lo supera en todas oportunidades obteniendo el valor mínimo de las soluciones. Cabe mencionar que para estos casos se consideran en el análisis los outliers, ya que a pesar de ser valores atípicos en el set de datos, se considera igual una solución del problema, por lo que si aparece un outlier inferior sigue siendo considerado una mejor solución que las otras del

conjunto. También se observa que sucede el mismo fenómeno descrito para las instancias de tamaño 20 y 30, ya que para todos los casos el tercer RIQ del método estándar se encuentra por debajo del primer RIQ del método adaptivo, mostrando nuevamente una superioridad en este tamaño de instancias.

Finalmente para las instancias de tamaño 200, se muestra que nuevamente el método estándar posee menor mediana que el estándar para todas las instancias y además tiene una menor dispersión de los datos respecto a su mediana en todas las oportunidades. También el método estándar supera al adaptivo en todas las instancias encontrando una mejor solución.

Resultados instancias del set [19]

Para ejecutar las pruebas correspondientes al conjunto de [29], se utilizó un tiempo límite de ejecución de 3000 segundos, con el fin de poder establecer una comparación directa de los algoritmos propuestos con el acercamiento de los autores.

Al analizar los distintos boxplots de la figura 4.4, las cuales representan unas instancias mas grandes que las anteriores, se observa que en el caso de las de tamaño 200 el método adaptivo posee una menor mediana en todas las instancias y una menor dispersión de los datos que el método estándar en 4 de 5 de ellas, también se destaca que el primero supera al segundo encontrando soluciones de mejor calidad en 3 de las 5 instancias.

En el caso de las instancias de tamaño 300, el método estándar posee una menor mediana que el adaptivo en todas las instancias, sin embargo éste último posee una menor dispersión de datos en 4 de 5 de ellas. Respecto a la mejor solución encontrada, se muestra que el método estándar supera al adaptivo en todas las instancias, en donde en un caso dicha solución es considerada un outlier.

Para las instancias de tamaño 400 se observa que el método estándar supera al adaptivo ya que en todas las instancias posee una menor mediana. También se destaca que en todas las ocasiones el método estándar encuentra la mejor solución. Por otro lado, se observa que en 3 de 5 instancias el tercer RIQ del método estándar se encuentra por debajo del primer RIQ del adaptivo.

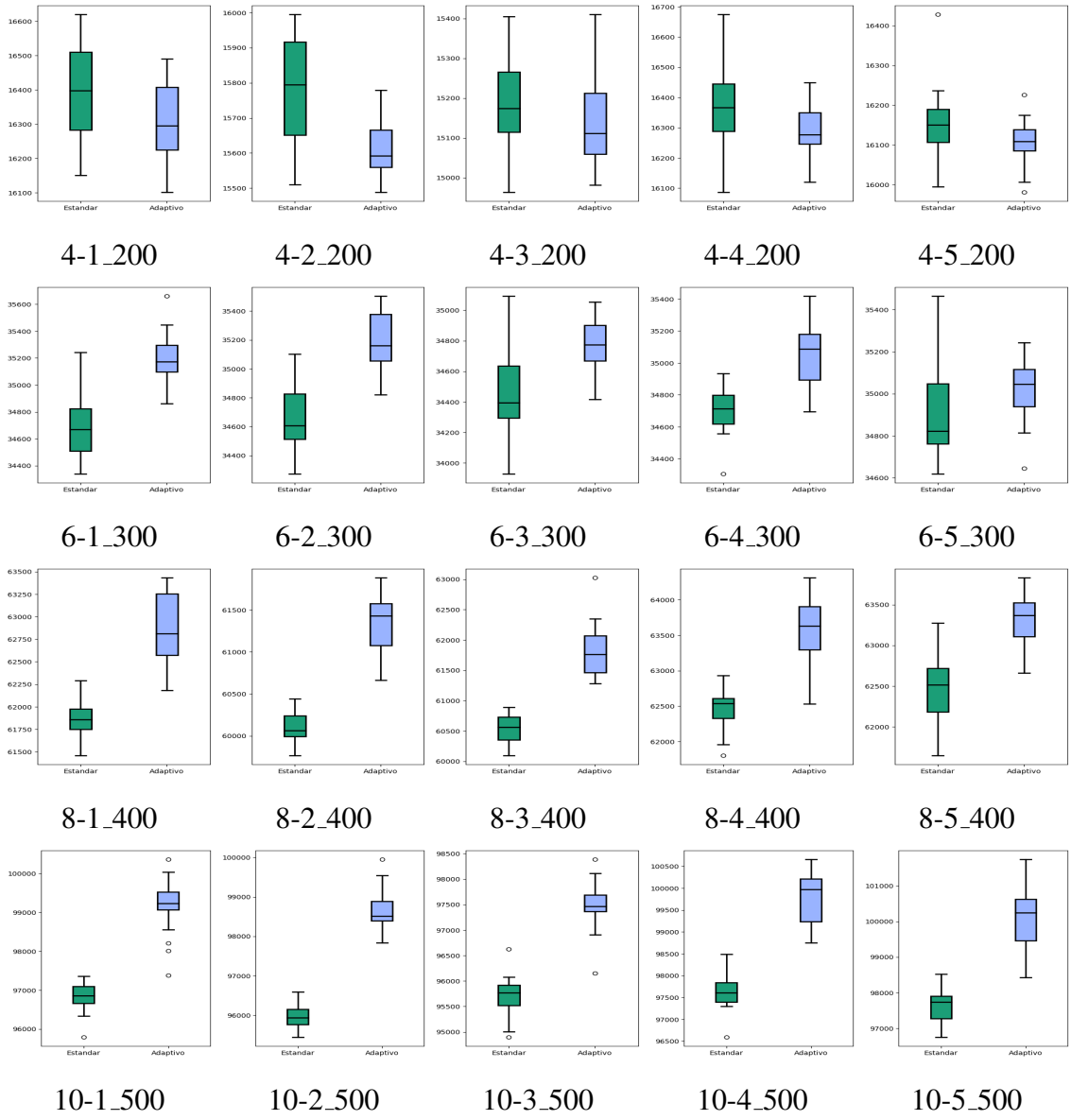


Figura 4.4: Boxplots de los resultados obtenidos para las instancias de [19]

Finalmente, para las instancias de tamaño 500 el método estándar supera al adaptivo, ya que para todas ellas éste posee una menor mediana y una mejor solución. Por otro lado cabe mencionar que la dispersión de los datos en ambos métodos es bastante similar, por lo que ninguno sobresale en éste ámbito. Por último se destaca que para todas las instancias el tercer RIQ del método estándar se encuentra por debajo del primer RIQ del adaptivo, por lo que el primer método supera con creces al segundo.

4.3.2. Comparación con el estado del arte

En esta sección se presentan las mejores soluciones obtenidas por la literatura para las instancias experimentadas, en donde se consideran los resultados de los artículos [29], [28], [19], [21] y [11]. Adicionalmente se agregan los resultados obtenidos por el GRASP adaptivo y por el GRASP estándar, con el fin de comparar los resultados de los métodos propuestos con los existentes en el estado del arte.

Para realizar las pruebas, los autores de [29] realizaron 25 ejecuciones para cada instancia, y muestran las mejores soluciones encontradas dentro de 300 segundos. [28] realizó 3 ejecuciones para cada instancia con un tiempo límite de 3000 segundos. [19] realizó 40 ejecuciones de su algoritmo para cada instancia con un tiempo límite de 3000 segundos y [21] realizó 10 ejecuciones para cada instancia, sin embargo estos último no poseían límite de tiempo, ocupando desde 70 hasta 2000 segundos para sus pruebas. Es por esto que no se puede realizar una comparación directa y en igualdad de condiciones entre todos los algoritmos.

Instancias de [29]

En la tabla 4.5 se muestran los mejores resultados obtenidos para las instancias de tamaño 20 y 30, mientras que en la tabla 4.6 se muestran los mejores resultados obtenidos para las instancias de tamaño 50 a 200.

Instancia	[29]	G.A.	G.E.	Best	GAP G.A %	GAP G.E %
20a	**2668.8**	2760.9	2933.9	2668.8	**3.45**	9.93
20b	**3260.9**	3262.3	3306.3	3230.9	**0.97**	2.33
20c	-	**3675.5**	3722.7	3675.9	**0.00**	1.29
20d	2632.0	**2628.1**	2648.8	2618.2	**0.38**	1.17
20e	**3646.5**	**3646.5**	3646.5	3646.5	**0.00**	**0.00**
30a	**3908.2**	3935.9	4113.8	3908.2	**0.71**	5.26
30b	**4855.6**	4951.0	4944.0	4855.6	1.97	**1.82**
30c	5121.2	**5109.9**	5169.4	5109.9	**0.70**	1.16
30d	**3865.0**	3868.7	3931.0	3865.0	**0.10**	1.71
30e	5041.4	**5039.3**	5177.6	5039.3	**0.53**	2.75
Promedio	-	-	-	-	**0.88**	2.74

Tabla 4.5: Tabla de los mejores resultados obtenidos en la literatura y los métodos propuestos para las instancias de [29] de tamaño 20 y 30.

Respecto a los resultados obtenidos por los GRASP propuestos, estos se pueden comparar de manera directa con los obtenidos en [29], ya que se utilizaron los mismos tiempos límites de ejecución. Con el fin de poder analizar y comparar de mejor manera los resultados se define la medida GAP que brinda información sobre la diferencia porcentual entre la mejor solución y la obtenida por un algoritmo. La fórmula utilizada para calcular el GAP es la siguiente.

$$GAP = \frac{MejorSolución - SoluciónAlgoritmo}{MejorSolución} \times 100$$

Al analizar los datos obtenidos en la tabla 4.5 se observa que el método propuesto por [29] supera a los GRASP propuestos en esta memoria en el 60 % de las instancias pequeñas, sin embargo el GAP promedio respecto al GRASP adaptivo y al GRASP estándar es de 0.88 % y 2.74 % respectivamente, por lo que la diferencia entre las soluciones es muy pequeña. Por otro lado, se observa que el GRASP adaptivo supera al método propuesto en [29] en el 40 % de dichas instancias, mientras que el GRASP estándar no logra liderar en ningún caso.

Al comparar el GRASP adaptivo con el estándar se observa que el primero posee un GAP promedio menor que el segundo, lo que implica que sus soluciones son en general de mejor calidad para este tipo de instancias. Esto puede ser debido a que como las instancias son pequeñas y como el resultado de la sintonización del método estándar fueron tantas iteraciones que sólo se alcanza a ejecutar una iteración del GRASP, existe mucha probabilidad que en la ejecución el algoritmo se quede estancado en un óptimo local y no pueda explorar lo suficiente para encontrar una mejor solución. Por otro lado, el método adaptivo gracias a su selección inteligente de operadores y al parámetro del factor de exploración es capaz de diversificar de mejor manera el espacio de solución en estos casos, obteniendo mejores resultados para dichas instancias.

Respecto a los datos de la tabla 4.6 se puede observar que para el caso de [28], [19] y [11], el acercamiento que mejores resultados obtuvo en general fue el propuesto en [11], obteniendo las mejores soluciones para el 75 % de las instancias (15 de 20). En segundo lugar viene el método propuesto en [28] y [21], obteniendo la mejor solución en 2 instancias y finalmente [19] resaltando solo en una instancia. Por otro lado se observa que el GAP promedio entre las mejores soluciones y las obtenidas por [29] es de 1.9 %, lo cual es muy

bajo considerando que el tiempo utilizado en las otras instancias fue de un orden de magnitud mayor, dejando en evidencia que a medida que el tiempo aumenta la convergencia de los algoritmos se vuelve mucho mas lenta, concluyendo que no es necesario para estas instancias el otorgar mayor tiempo de ejecución para obtener una mejor solución, ya que el tiempo empleado podría ser mucho mayor y la diferencia entre las soluciones obtenidas es muy pequeña.

En el ámbito de los GRASP propuestos, se puede observar que ninguno logra superar a las soluciones encontradas en la literatura para estas instancias, sin embargo el GAP promedio respecto a la mejor solución para el GRASP adaptivo y estándar es de menos de 5 % para ambos, lo que indica que los resultados obtenidos son bastante aceptables ya que presentan poca diferencia con los mejores resultados de la literatura. También se puede observar que respecto al GAP, el método estándar supera al adaptivo en el 75 % de estas instancias (15 de 20). y además posee un GAP promedio menor de 4.13 % respecto al 4.7 % del adaptivo. A pesar que la diferencia entre los resultados de ambos algoritmos es muy poca, el estándar tiene un mejor desempeño en estas instancias, lo que puede deberse a lo mencionado anteriormente sobre los movimientos de cada algoritmo. Para las instancias más grandes el espacio de búsqueda aumenta considerablemente, por lo que tener un buen mecanismo de diversificación es fundamental al momento de encontrar buenas soluciones. Como los movimientos del método adaptivo se encuentran muy enfocados en la intensificación, es probable que en las ejecuciones el algoritmo se haya atascado en óptimos locales y como el método de selección de parámetros va guardando un historial de los movimientos utilizados junto con un puntaje, es probable que los movimientos especializados en intensificar predominen sobre los de diversificación, provocando que al momento de estancarse en un óptimo local, le cueste más tiempo e iteraciones en salir de ahí antes de comenzar a explotar otra zona diferente del espacio de búsqueda. Por otra parte, el método estándar al tener una selección de operadores completamente estocástica, le resulta mucho mas sencillo realizar una exploración, y en el caso de un posible estancamiento en un óptimo local, sólo necesitará de una selección aleatoria de un movimiento que diversifique la zona y así poder explorar otra parte del espacio de búsqueda.

Instancia	[29]	[28]	[19]	[21]	[11]	G.A.	G.E.	Best	GAP A	GAP E
50a	6497.30	6450.28	6453.08	**6450.28**	6455.77	6784.72	6725.98	6450.28	4.58	**4.27**
50b	7466.30	7428.54	7434.90	7428.54	**7320.77**	7497.73	7606.77	7320.77	**2.27**	3.91
50c	7350.50	7311.77	7317.35	7311.77	**7311.77**	7528.62	7575.09	7311.77	**2.83**	3.60
50d	7074.00	**7021.39**	7035.50	7028.22	7028.69	7123.74	7130.30	7021.39	**0.74**	1.55
50e	7571.50	7451.42	7482.01	**7451.42**	7452.83	7666.75	7750.92	7451.42	**3.49**	4.02
100a	12878.00	-	**12765.16**	-	-	13334.10	13305.80	12765.16	4.27	**4.24**
100b	14646.80	14405.52	14441.01	14398.17	**14349.60**	14838.80	14842.00	14349.60	4.14	**3.43**
100c	14054.40	13889.22	13932.78	13869.80	**13784.70**	14494.10	14447.70	13784.70	**4.44**	4.81
100d	13844.40	**13564.23**	13708.81	13603.03	13577.20	14053.00	14155.70	13564.23	4.60	**4.36**
100e	14300.40	14059.62	14122.32	14063.29	**13943.10**	14499.50	14539.80	13943.10	4.86	**4.28**
150a	19784.00	19638.04	19532.28	19391.16	**19358.90**	20334.10	20221.00	19358.90	5.76	**4.45**
150b	21098.10	20922.27	20823.40	20764.50	**20581.50**	21429.40	21461.00	20581.50	5.82	**4.27**
150c	20166.20	20019.50	19964.59	19864.86	**19726.80**	20558.30	20692.30	19726.80	5.97	**4.89**
150d	20747.20	20600.33	20509.97	20355.27	**20318.80**	21255.70	21091.60	20318.80	5.75	**3.80**
150e	19888.50	19782.00	19716.87	19634.47	**19449.50**	20359.30	20357.90	19449.50	5.0	**4.67**
200a	27537.40	27397.31	27112.48	27073.57	**26816.50**	28034.70	28148.40	26816.50	6.33	**4.97**
200b	27851.70	27582.87	27509.08	27337.49	**27215.10**	28250.00	28216.00	27215.10	5.16	**3.68**
200c	26472.50	26425.29	26320.39	26181.73	**25926.00**	27066.20	27141.30	25926.00	6.72	**4.69**
200d	27935.30	29818.77	27686.75	27439.50	**27328.70**	28383.30	28390.30	27328.70	5.29	**3.88**
200e	26703.40	26704.81	26443.29	26305.30	**26063.50**	27205.60	27362.30	26063.50	6.11	**4.98**
GAP Promedio %	1.9	-	-	-	-	-	-	-	4.70	**4.14**

Tabla 4.6: Tabla de los mejores resultados obtenidos en la literatura y los métodos propuestos para las instancias de [29] de tamaño 50 a 200.

Instancias de [19]

Al analizar los datos de la tabla 4.7, se observa que para estas instancias los mejores resultados los posee el algoritmo propuesto en [21], obteniendo las mejores soluciones en 68.75 % de las instancias (11 de 16), sin embargo la comparación de dicha propuesta con las otras no puede ser realizada de forma completamente directa, ya que en [19] se ejecuta una cantidad de 10 veces cada instancia sin tener un tiempo límite, y en el caso de estas instancias que son de mayor tamaño. el tiempo ocupado por los autores para las ejecuciones fue desde 973 el mínimo hasta 6670 segundos el máximo, lo que duplica el tiempo límite establecido por los demás autores y el propuesto en esta memoria. Por otro lado, el GAP promedio entre las soluciones en las que lidera [21] y los resultados de [19] es de tan solo 2.1 %, lo que demuestra el mismo comportamiento de rendimiento respecto al tiempo visto para las instancias de [29]. Respecto a la propuesta de [11], se observa que obtiene las mejores soluciones para 5 instancias y que el GAP respecto a los resultados de [19] es de un 2.3 %, obteniendo un mejor rendimiento especialmente para las instancias de tamaño 300, en donde supera en 3 de 5 ocasiones a los otros métodos propuestos.

Respecto a los GRASP propuestos, se puede observar que ninguno logró superar a la literatura existente, sin embargo se observa que los GAPS de ambos son menor a 5.5 % y son calculados respecto a la mejor solución encontrada. Como la propuesta de [21] obtuvo la mejor solución en la mayoría de las instancias, en cierta forma se realiza la comparación con sus soluciones, las cuales fueron obtenidas en un tiempo de ejecución mucho mayor al utilizado en esta memoria, por lo que no se puede hacer una comparación completamente directa.

Por otro lado se observa que el GAP del método estándar es menor que el del adaptivo en 17 instancias equivalente al 85 % de ellas, también el GAP promedio es menor, lo que significa que obtiene mejores soluciones en general para dichas instancias. Esto posiblemente se debe a que el método adaptivo se focaliza más en intensificar el espacio de búsqueda y no tanto a diversificar, por lo que si llega un punto en el que los movimientos de explotación llegan a dar buenos resultados, en la ventana deslizante se guardará el historial de éstos, y el método de diversificación (2-OPT) no tendrá oportunidad de ejecutarse, provocando un estancamiento local del algoritmo. Por otro lado, el método estándar permite la selección del movimiento

de diversificación mediante la selección aleatoria. Este efecto es el mismo que se mencionó para las instancias de la tabla 4.6, sin embargo, como las instancias de la tabla 4.7 son mucho más grandes, es mucho más importante el proceso de exploración, y el estancamiento en un óptimo local es mucho más probable, por lo que tiene mayor efecto para estos casos. Esto queda demostrado al analizar la diferencia de GAP entre ambos algoritmos, en el caso de la tabla 4.6, la diferencia es de 0.56 %, mientras que para el de la tabla 4.7 es de 1.17 %.

Instancia	[19]	[21]	[11]	G.A.	G.E.	Best	GAP A	GAP E
4-1_200	15445.28	15208.84	**15170.00**	16101.00	16151.10	15170.00	**6.14**	6.47
4-2_200	14850.75	**14614.02**	14626.90	15488.00	15510.10	14614.02	**5.98**	6.13
4-3_200	14332.27	**14101.73**	14146.60	14982.20	14963.30	14101.73	6.24	**6.11**
4-4_200	15521.49	**15282.02**	15293.30	16119.90	16086.10	15282.02	5.48	**5.26**
4-5_200	-	-	-	**15980.50**	15994.80	15980.50	**0.00**	0.09
6-1_300	33511.04	32696.90	**32598.10**	34860.00	34338.10	32598.10	6.94	**5.34**
6-2_300	33540.56	**32623.17**	32628.70	34820.60	34272.20	32623.17	6.74	**5.05**
6-3_300	33282.54	32624.50	**32571.50**	34417.50	33928.10	32571.50	5.67	**4.16**
6-4_300	33468.72	32748.70	**32746.30**	34695.90	34304.90	32746.30	5.95	**4.76**
6-5_300	-	-	-	34644.30	**34617.60**	34617.60	0.08	**0.00**
8-1_400	60300.22	58961.82	**58831.10**	62180.40	61457.00	58831.10	5.69	**4.46**
8-2_400	58113.85	**56894.76**	56956.40	60663.00	59765.30	56894.76	6.62	**5.05**
8-3_400	58558.94	**57154.27**	57421.70	61279.40	60092.50	57154.27	7.27	**5.20**
8-4_400	60502.26	**59169.87**	59295.30	62531.90	61799.60	59169.87	5.68	**4.44**
8-5_400	-	-	-	62662.50	**61645.30**	61645.30	1.65	**0.00**
10-1_500	94080.68	**91657.18**	91949.20	97375.80	95787.30	91657.18	6.24	**4.51**
10-2_500	92792.34	**91001.23**	91005.80	97840.10	95439.70	91001.23	7.52	**4.88**
10-3_500	93222.85	**91016.40**	91317.00	96450.20	94894.00	91016.40	5.64	**4.26**
10-4_500	94372.82	**92112.35**	92341.10	98752.20	96588.40	92112.35	7.21	**4.86**
10-5_500	-	-	-	98423.30	**96744.50**	96744.50	1.74	**0.00**
GAP Promedio %	-	-	-	-	-	-	5.22	4.05

Tabla 4.7: Tabla de los mejores resultados obtenidos en la literatura y los métodos propuestos para las instancias de [19] de tamaño 200 a 500.

4.3.3. Comparación de tiempos GRASP adaptivo V/S GRASP estándar

Con el fin de comparar desde otro punto de vista ambos métodos propuestos en esta memoria, resulta interesante analizar el tiempo promedio que demora cada algoritmo en encontrar la mejor solución para cada tamaño de instancia.

En la tabla 4.8 se muestran los resultados de los tiempos promedios que demoró cada algoritmo en encontrar su mejor solución para las instancias de [29]. Se puede observar que el GRASP estándar supera con creces al adaptivo para 4 de los 6 tamaños, y con una diferencia mayor a 94 % para 3 de ellos, mientras que el adaptivo solo supera en 2 tamaños al estándar por una diferencia menor a 8 %. En el caso de las instancias mas pequeñas (20 a 50), se observa que la diferencia de tiempo entre ambos algoritmos es muy grande, lo que puede deberse a las características de las instancias, al ser relativamente simples y con pocos nodos resulta mucho mas eficiente seleccionar un operador de manera estocástica, que calcular el mejor operador a ser seleccionado en cada iteración, sobretodo porque el método adaptivo no depende de la cantidad de nodos para elegir el operador, por lo que este proceso no depende del tamaño de la instancia y termina ocupando una cantidad importante de tiempo de cómputo en comparación con elegir el movimiento utilizando un número aleatorio. Por otro lado al ser la instancia mas pequeña y al estar ambos algoritmos sintonizados, se favorece en mayor parte el método estándar ya que es ejecutado con los porcentajes de uso óptimos para cada movimiento, mientras que el método adaptivo se ejecuta con los parámetros óptimos como la ventana deslizante, el factor de exploración y explotación, los cuales no favorecen de manera directa a un operador en específico. Otro punto a considerar es que el objetivo del método adaptivo es que a medida que avancen las iteraciones el método "aprenda" a escoger de manera sabia el operador a utilizar, por lo que requiere algunas iteraciones de "prueba" para así analizar el escenario y escoger el operador óptimo, lo que implica que al comienzo de la ejecución los movimientos seleccionados no generen los resultados esperados, mientras que el método estándar realiza este proceso de manera estocástica y directa, por lo que no requiere de un período de "entrenamiento" para seleccionar el operador a utilizar.

En el caso de las instancias más grandes (150 y 200) el método adaptivo encontró la solución en menos tiempo, lo que puede justificarse con lo mencionado anteriormente del proceso

de selección de operadores. Al ser las instancias más grandes, requieren de mayor tiempo e iteraciones para ir mejorando las soluciones, por lo que el método adaptivo a medida que estas transcurren va eligiendo el operador óptimo para cada instante de tiempo, logrando la convergencia a la solución de manera más rápida que el método estándar, ya que este último depende netamente de la naturaleza estocástica para seleccionar los movimientos.

Tamaño Instancia	G.A.	G.E.	Diferencia	GAP %
20	146.19	**0.61**	145.57	99.58
30	147.60	**1.80**	145.80	98.77
50	175.10	**10.35**	164.76	94.09
100	186.37	**70.72**	115.65	62.05
150	**220.41**	236.04	15.63	7.07
200	**277.92**	287.44	9.52	3.43

Tabla 4.8: Tabla en donde se muestra el tiempo que demora cada algoritmo en encontrar su mejor solución para las instancias de [29]

Para las instancias de [19], los resultados se muestran en la tabla 4.9.

Como se puede observar en la tabla 4.9, tanto el GRASP adaptivo como el estándar logran obtener el menor tiempo en 2 instancias. A pesar que las instancias de [19] son más grandes que las de [29], se observa el mismo comportamiento que el de la tabla 4.8, en donde el GRASP estándar supera al adaptivo en las instancias más pequeñas (200 y 300 nodos para este caso), mientras que sucede lo contrario para las instancias más grandes (400 y 500 nodos). Esto se debe principalmente a las características propias de cada instancia, ya que en el caso de la tabla 4.8 se observa que para el tamaño 150 y 200 el método adaptivo encuentra la solución en un menor tiempo, lo cual no sucede para las instancias de [19]. Sin embargo el patrón se vuelve a repetir, dominando en las instancias más grandes el método adaptivo, debido a su capacidad de escoger de manera inteligente el parámetro a utilizar en cada iteración, logra encontrar la mejor solución en menos tiempo que el estándar, el cual depende netamente de un operador seleccionado de manera aleatoria, lo que hace más difícil intensificar el espacio de búsqueda de manera más rápida.

Tamaño Instancia	G.A.	G.E.	Diferencia	GAP %
200	1483.28	**478.04**	1005.24	67.77
300	1945.17	**1752.84**	192.33	9.88
400	**2250.69**	2931.05	680.36	30.22
500	**2708.80**	2991.18	282.38	10.42

Tabla 4.9: Tabla en donde se muestra el tiempo que demora cada algoritmo en encontrar su mejor solución para las instancias de [19]

4.3.4. Análisis de selección de movimientos

Como el GRASP adaptivo va seleccionando de manera inteligente el mejor operador en cada iteración, resulta interesante analizar dicha selección para ver el comportamiento de cada movimiento a medida que avanzan las iteraciones. Como el método adaptivo sólo se basa en puntajes y recompensas que otorga cada movimiento en sus ejecuciones, no hay nada estocástico en su lógica, por lo que no se puede definir una probabilidad real de uso para cada operador, sin embargo, como se selecciona el método que posea mejor "puntaje" hasta el momento, se puede utilizar el FRR de cada operador y normalizarlos de manera tal que se pueda interpretar como una "probabilidad de ser seleccionado" en cada iteración.
La figura 4.5, muestra un gráfico que representa el FRR normalizado de cada operador en función de la cantidad de iteraciones. Los movimientos *Cambio de nodos*, *2-OPT*, *Intercambio de nodo recogida* e *Intercambio de nodo entrega*, son representados por los colores azul, verde, rojo y amarillo respectivamente.

Al analizar dicho gráfico, se observa que el movimiento *Intercambio de nodo entrega* en la mayoría de las iteraciones del GRASP tiene un *peak* de ejecución, en donde da muy buenos resultados, y luego decae drásticamente, lo que indica que la ruta de entrega de cada vehículo posee una gran influencia en los costos finales. También se observa que el movimiento *Cambio de nodos* posee una alta probabilidad de ejecución a lo largo de las iteraciones, alcanzando el *peak* de ejecución en algunas ocasiones, lo que indica que éste movimiento es

fundamental para la mejora de las soluciones. Esto se debe principalmente a que dicho movimiento es el único que hace cambios en la estructura de las rutas de los vehículos, es decir, es el único movimiento que permite acortar y alargar las rutas, generando una estructura de solución diferente a la que se tenía antes de su ejecución.

En el caso del movimiento *Intercambio de nodos recogida*, se observa que al comienzo de cada iteración del GRASP parte con una probabilidad de uso muy baja (menor a 0.25 en la mayoría de los casos), pero en algunas ocasiones tiene una fuerte alza, alcanzando un *peak* de ejecución en varias iteraciones al final. Este comportamiento posiblemente se crea cuando los demás operadores no pueden conseguir alguna mejora en la solución, el factor de exploración permite la ejecución de este movimiento y logra mejorarla, aumentando así su puntaje y permitiendo que su probabilidad de uso aumente considerablemente.

Finalmente, se observa que el movimiento *2-OPT*, es el operador con menos probabilidad de uso en general, ya que hay varias iteraciones en donde este no se aprecia. Cabe destacar que este movimiento es el que más diversifica el espacio de búsqueda, y es el único movimiento que permite cambiar la solución actual por una que tiene menor calidad, por lo tanto al ser un operador que privilegia la exploración, tiene momentos en la que la probabilidad de uso aumenta considerablemente y otros en el que no se utiliza, coincidiendo con lo que se explicó en los análisis de los datos, de que el método adaptivo se enfocaba mucho más en la explotación que en la exploración, haciéndolo más vulnerable a atascarse en óptimos locales.

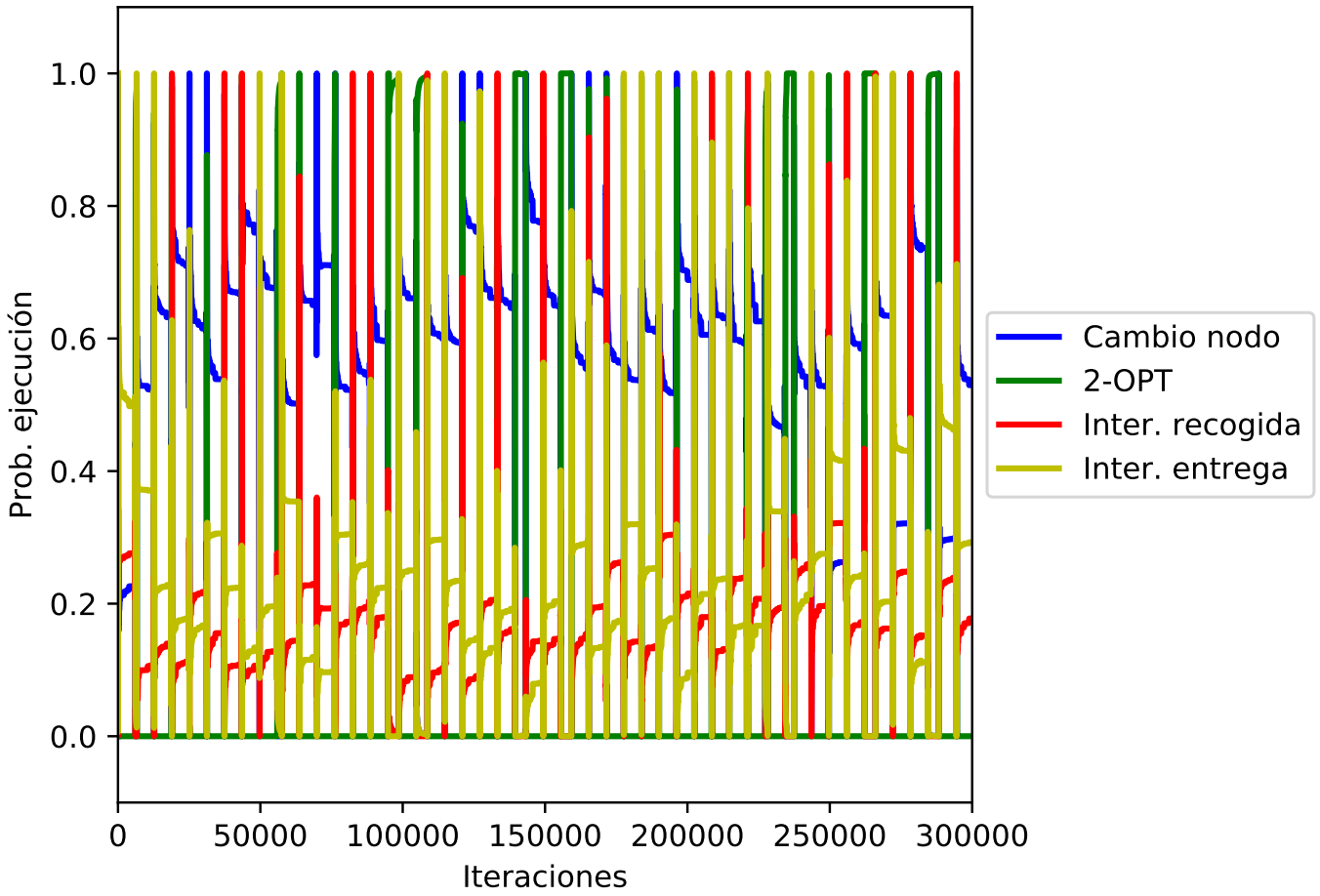


Figura 4.5: Gráfico de la probabilidad de uso de los operadores, en función de la cantidad de iteraciones.

Conclusiones

En el presente trabajo, se investigaron y presentaron los acercamientos mas importantes del VRPCD, los cuales fueron cambiando constantemente para ser aplicados a situaciones reales, incluyendo una gran cantidad de restricciones y modificando alguno de sus objetivos, pero siempre con la premisa de minimizar el costo total de satisfacer todos los pedidos de los clientes.

Posteriormente, se explicó el objetivo y las características que poseen los algoritmos basados en GRASP, para luego, realizar una propuesta de dicho método para resolver el VRPCD planteado. El GRASP propuesto considera una fase de construcción enfocada en la minimización de tiempos de viaje, ya que para crear la solución inicial selecciona en cada iteración el proveedor y cliente que resulte en el menor costo de transporte, lo que es de utilidad para crear soluciones iniciales de buena calidad. También el GRASP considera una fase de post-procesamiento, en donde se implementaron cuatro operadores que permiten modificar y obtener soluciones de mejor calidad. El operador *2-OPT* se enfoca principalmente en la diversificación del algoritmo, siendo el único de ellos que puede retornar una solución de peor calidad que la recibida. El operador *cambio de nodos* se enfoca principalmente en la intensificación del algoritmo, pero igual permite realizar exploración, ya que es el único que tiene la facultad de cambiar la estructura de una solución, generando rutas más cortas y más largas. En el caso del operador *intercambio de nodos*, realiza un *swap* entre nodos de distintos vehículos, permitiendo cambiar los tiempos de consolidación y el tiempo de las rutas de los vehículos, este operador también prioriza la intensificación.

Adicionalmente, el GRASP propuesto posee una selección de operadores adaptiva, en donde se escogerá en cada iteración el movimiento más adecuado para ser ejecutado, teniendo control sobre los operadores que se utilizan a lo largo de la búsqueda de soluciones.

Posteriormente se realizó un proceso de sintonización del GRASP adaptivo y el estándar, utilizando *ParamILS* para obtener los mejores valores de parámetros de cada algoritmo. Luego se realizaron distintos experimentos utilizando dos conjuntos de datos conocidos en la literatura del VRPCD. Dichos conjuntos son los propuestos en [29] y en [19], los cuales poseen instancias de distintos tamaños, variando desde 20 hasta 500 pares de nodos.

Respecto a los resultados obtenidos, se observó que el GRASP adaptivo resultó ser eficaz en las instancias más pequeñas y que logró obtener mejores resultados que la literatura existente para las instancias más chicas del conjunto [29], mientras que el método estándar no logró superar en ningún caso a las otras propuestas.
Por otro lado, se puede concluir que el método adaptivo posee en general una menor dispersión de los datos que el método estándar y que el primero resultó ser más eficaz para las instancias pequeñas debido a que es capaz de intensificar de mejor manera el vecindario que el método estándar gracias a su selección inteligente de operadores, sin embargo el método adaptivo en general demora más en encontrar la mejor solución que el estándar, ya que necesita del transcurso de las iteraciones para aplicar su método de selección de operadores de manera eficaz.
En el caso de las instancias más grandes como el set de [19], se observa que el método estándar supera al adaptivo, teniendo un GAP menor respecto a la mejor solución encontrada por la literatura, lo cual se debe principalmente a que este método es capaz de explorar de mejor manera que el adaptivo, lo cual es fundamental para este tipo de instancias, en donde el espacio de búsqueda es más grande.

Finalmente, se puede concluir que el algoritmo propuesto en esta memoria, a pesar de haber superado en pocas instancias a las propuestas existentes en la literatura, resultó ser eficaz para las instancias más pequeñas, encontrando muy buenas soluciones en poco tiempo, mientras que para las instancias más grandes encontró soluciones aceptables con un GAP menor al 5 %, en un tiempo mucho menor al utilizado por las propuestas que tienen la mejor solución.

Para un trabajo futuro, puede resultar interesante mezclar distintas meta-heurísticas de búsqueda local en la fase de post-procesamiento del GRASP, enfocándolas en distintos objetivos, es decir, se puede utilizar un *simulated annealing* enfocado en la exploración, para poder ubicar distintas zonas del espacio de búsqueda, y luego utilizar una búsqueda tabú para realizar el

proceso de intensificación. De esta misma manera se puede modificar la selección adaptiva, con el fin de que escoja la meta-heurística se utilizará dependiendo del tiempo y del estado en el que se encuentre el algoritmo, así podría utilizar para enfocarse en la explotación, y si llegase a estancarse en algún óptimo local, hacer la selección de la otra heurística para favorecer la exploración y así cambiar de localidad en el espacio de búsqueda.

Bibliografía

[1] Fardin Ahmadizar, Mehdi Zeynivand, and Jamal Arkat. Two-level vehicle routing with cross-docking in a three-echeleon supply chain: A genetic algorithm approach. *Elsevier*, 2015.

[2] Guillet B. and Miller L. A heuristic algorithm for the vehicle dispatch problem. *Operations Research*, 22:340–349, 1974.

[3] L. Bodin, B. Golden, A. Assad, and M. Bal. The state of art in the routing and scheduling of vehicles and crews. *Computers Operations Research*, 10:63–212, 1983.

[4] J. Brandäo. A deterministic tabu search algorithm for the fleet size and mix vehicle routing problem. *European Journal of Operational Research*, 195:716–728, 2009.

[5] R. Cordone and R.W. Calvo. A heuristic for the vehicle routing problem with time windows. *Journal of Heuristics*, 7:107–129, 2001.

[6] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[7] Rodolfo Dondo and Jaime Cerdá. A sweep-heuristic based formulation for the vehicle routing problem with cross-docking. *Elsevier*, 48:293–311, 2013.

[8] A.E. Eiben and J.E. Smith. Introduction to evolutionary computing. *Springer*, 2003.

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

[10] Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 1994.

[11] Philippe Grangier, Michel Gendreau, Fabien Lehuédé, and Louis-Martin Rousseau. A matheuristic based on large neighborhood search for the vehicle routing problem with cross-docking. *Elsevier*, pages 116–126, 2017.

[12] Gehring H. and Homberger J. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. *EUROGEN*, 2, 1999.

[13] Lourenco H.R., Martin O.C., and Stuzle T. Iterated local search: Framework and applications. in handbook of metaheuristics. *International series in operations research and management science*, 57:363–398, 2010.

[14] Wang J., Ranganathan Jagannathan, A.K. Zuo X, and Murray C.C. Two-layer simulated annealing and tabu search heuristics for a vehicle routing problem with cross docks and split deliveries. *Computer Industrial Engineering*, 2017.

[15] G. Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics*, 54:811–819, 2007.

[16] Gilbert Laporte, Michel Gendreau, Jean-Yves Potvin, and Frédéric Semet. Classical and modern heuristics for the vehicle routing problem. *Elsevier*, 7:285–300, 2000.

[17] Ke Li, Álvaro Fialho, Sam Kwong, and Qingfu Zhang. Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. Technical report, 2014.

[18] Solomon M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35, 1987.

[19] Vinicius W.C. Morais, Geraldo R. Mateus, and Thiago F. Noronha. Iterated local search heuristics for the vehicle routing problem with cross-docking. *Elsevier*, 2014.

[20] Rami Musa, Jean-Paul Arnaut, and Hosang Jung. Ant colony optimization algorithm to solve for the transportation problem of cross-docking network. *Elsevier*, 59:85–92, March 2010.

[21] Amalia I. Nikolopoulou, Panagiotis P. Repoussis, Christos D. Tarantilis, and Emmanouil E. Zachariadis. Moving products between location pairs: Cross-docking versus direct-shipping. *European Journal of Operational Research*, 2016.

[22] Hansen P. and Mladenovíc N. Variable neighborhood search: Principles and applications. *European journal of Operational Research*, 130:449–467, 2001.

[23] Portal Portuario. Gianni contenla: "la logística en chile tiene desafíos enormes". urlhttp://sindominio.net/ash, 2016.

[24] C. Prins. Efficient heuristics for the heterogeneous fleet multitrip vrp with application to a large-scale real case. *Journal of Mathematical Modeling and Algorithms*, 1:135–150, 2002.

[25] C. Prins. Two memetic algorithms for heterogeneous fleet vehicle routing problems. *Engineering Applications of Artificial Intelligence*, 22:916–928, 2009.

[26] Fernando Afonso Santos, Geraldo Robson Mateus, and Alexandre Salles da Cunha. A branch-and-price algorithm for a vehicle routing problem with cross-docking. *Elsevier*, 37:249–254, 2011.

[27] É.D Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1992.

[28] Christos D. Tarantilis. Adaptative multi-restart tabu search algorithm for the vehicle routing problem with cross-docking. *Springer*, 2012.

[29] M. Wen, J. Larsen, J. Clausen, J-F. Cordeau, and G. Laporte. Vehicle routing with cross-docking. *Journal of the Operational Research Society*, 60:1708–1718, 2009.