Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Aprendizaje Automático en Metaheurísticas

Machine Learning applied to Metaheuristics

Viren Sajju Dhanwani Dhanwani

La Laguna, 13 de junio de 2022

Dña. **María Belén Melián Batista**, con N.I.F. 44.311.040-E, Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **Christopher Juan Expósito Izquierdo**, con N.I.F. 78.851.649-J, Profesor Ayudante Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Aprendizaje Automático en Metaheurísticas*"

ha sido realizada bajo su dirección por D. **Viren Sajju Dhanwani Dhanwani**, con N.I.F. 78.857.246-K.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de junio de 2022

Agradecimientos

A la tutora Belén por su gran dedicación y ayuda durante la realización de este Trabajo de Fin de Grado, además de la gran planificación que hizo al comienzo de éste y todos los materiales que puso a mi disposición.

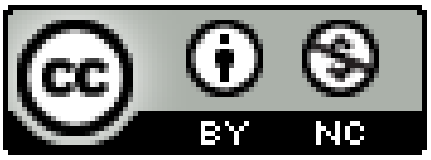
Al cotutor Christopher por ayudarme principalmente en el desarrollo del TFG, añadiendo retroalimentación de calidad al código que realizaba, además de ofrecerme una gran cantidad de conocimientos que me ayudaron a progresar como programador.

Al colaborador Frank por sus perspectivas únicas sobre los algoritmos que desarrollé, además de modificaciones y explicaciones sobre éstos que facilitaron mi trabajo.

A mis padres por estar siempre a mi lado y apoyarme en todas las metas que me propongo, mi hermano por brindarme alegría cuando estoy decaído y el resto de mi familia que, aunque estén lejos, siempre me animan a seguir adelante.

A mis amigos por compartir experiencias inolvidables, tanto dentro como fuera de la Universidad, en esta etapa tan importante de mi vida.

Licencia



Resumen

La logística colaborativa supone una estrategia útil que permite a las empresas ahorrar costes y proporcionar mejor servicio a sus clientes, especialmente a través de centros de cross-docking *donde, en lugar de almacenar mercancía, se consolida y prepara, con un tiempo de permanencia inferior a las 24 horas.*

Con los centros de cross-docking *surgen varios problemas, entre ellos la localización de los centros, la asignación de camiones a las puertas del* cross-dock*, la gestión del flujo interno o del recorrido de los vehículos. En este trabajo de fin de grado, se aborda el problema de optimización de asignación de camiones a puertas de un centro de* cross-docking*, el cual es computacionalmente difícil y resulta de interés en un escenario real. El objetivo es asignar los camiones de los proveedores a puertas de entrada y los de clientes a puertas de salida, minimizando el recorrido interno en el* cross-dock*. En este trabajo se proponen 6 algoritmos metaheurísticos para resolver el problema, donde se aplica aprendizaje automático a 2 de ellos, y se prueban en un conjunto de 50 instancias de la literatura científica. Además, se generan instancias adicionales para ampliar las que encontramos en la literatura, y se visualizan gráficamente las soluciones obtenidas del problema.*

Palabras clave: Metaheurísticas, aprendizaje automático, cross-docking, problema de asignación, GRASP, búsqueda por entornos variables, VND

Abstract

Collaborative logistics strategies are useful because they allow companies to save costs and provide better service to their customers, especially through cross-docking centres where, instead of storing goods, they are consolidated and prepared, where shipments typically spend less than 24 hours in the cross-dock.

Several problems arise with cross-docking centres, including the location of the centres, the assignment of trucks to the cross-dock doors, the internal flow management or the vehicles routing management. In this work, the Cross-Docking Assignment Problem (CDAP) is addressed, which is a challenging optimization problem in supply chain management with important practical applications in the trucking industry. The objective is to assign supplier trucks to inbound doors and customer trucks to outbound doors, while minimizing the internal path in the cross-dock*. In this work, 6 metaheuristic algorithms are proposed to solve the problem, where machine learning is applied to 2 of them, and evaluated on a set of 50 instances obtained from the scientific literature. Furthermore, additional instances are generated to extend those found in the literature, and the obtained solutions are graphically visualized.*

Keywords: Metaheuristics, machine learning, cross-docking, assignment problem, GRASP, variable neighborhood search, VND

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Tradicionalmente, empresas distribuidoras requerían de grandes centros de almacenamiento con el fin de desempeñar su labor (Sparks, 1986). Por ejemplo, una empresa distribuidora podría suministrar artículos a varios hoteles, entre ellos almohadas, sábanas o jabones, provenientes de distintos proveedores. Utilizando centros de almacenamiento, se requiere de cuatro funciones principales: recepción, almacenamiento, preparación de pedidos y envío, siendo el almacenamiento y la preparación de pedidos las más costosas.

Por otra parte, la principal alternativa a los centros de almacenamiento son los centros de distribución (Galbreth et al., 2008). Su característica principal es que la mercancía esté el menor tiempo posible y el centro sirva únicamente como intermediario entre proveedores y clientes. Esto permite que la empresa distribuidora se ahorre los costes de almacenamiento, al no requerir de una infraestructura tan compleja, y los costes de preparación de pedidos, reduciendo la mano de obra necesaria. Además, también permite ofrecer un plazo de entrega más corto entre proveedor y cliente. No obstante, los transportistas tendrán que estar sincronizados para ajustarse a las demandas de tiempo y colaborar para no retrasar el envío de los pedidos.

El transporte colaborativo (Okdinawati et al., 2015) supone un cambio de modelo al promover el uso compartido de información, redes, instalaciones o vehículos por parte de los diferentes agentes de transporte. A pesar de las dificultades que plantea el compartir recursos entre empresas privadas, los gobiernos han comenzado a legislar a favor de su uso, entendiendo que los beneficios son importantes para todos los agentes implicados. De este modo, se financian centros de distribución y consolidación de mercancías, que son instalados en lugares económica y socialmente estratégicos. Éste es el caso de los centros de *cross-docking*.

El *cross-docking* es una estrategia de colaboración de la cadena de suministro por la que los envíos de varios proveedores a un determinado cliente pasan por un punto intermedio de transbordo y consolidación de mercancías, denominado *cross-dock* (Agustina et al., 2010), (Boysen and Fliedner, 2010), (Van Belle et al., 2012), (González-La Rotta and Becerra-Fernández, 2017), (Mavi et al., 2020). Los productos entrantes se descargan, clasifican y consolidan según los destinos finales. A continuación, se cargan directamente en los vehículos de salida para el envío a los clientes finales, sin almacenamiento en el muelle. El tiempo de permanencia de los productos en el cross-dock no suele superar las 24 horas, aunque en la mayoría de los casos suele ser inferior a una hora (Bartholdi III and Gue, 2004).

Entre los diversos problemas que surgen en los centros de *cross-docking* y que se mencionarán en el apartado de revisión de la literatura científica, este trabajo de fin de grado aborda exclusivamente el problema de optimización de asignación de camiones a puertas o *Cross-Docking Assignment Problem*, CDAP de aquí en adelante (Nassief et al., 2016), (Guemri et al., 2019), (Gelareh et al., 2020), (Sayed et al., 2020).

Durante el presente capítulo, se describen los objetivos del trabajo y se analiza la literatura asociada a los problemas de optimización que surgen en centros de *cross-docking*. El capítulo 2 define el problema de asignación de camiones a puertas del *cross-dock*. El capítulo 3 describe un generador de instancias del CDAP. El capítulo 4 muestra un visualizador de soluciones del problema. El capítulo 5 plantea diferentes algoritmos constructivos. El capítulo 6 describe distintas búsquedas por entornos. El capítulo 7 introduce las metaheurísticas utilizadas, concretamente los algoritmos GRASP implementados. En el capítulo 8 se muestran los distintos resultados computacionales obtenidos durante el desarrollo del proyecto y un análisis de ellos. El capítulo 9 concluye el trabajo y establece algunas líneas de futuro. El capítulo 10 ofrece un resumen extendido y conclusiones en inglés. Por último, el capítulo 11 muestra el presupuesto asociado al desarrollo del trabajo de fin de grado.

1.1. Objetivos

El objetivo principal de este trabajo de fin de grado es diseñar, implementar y analizar diversos algoritmos para obtener una solución de calidad para el CDAP en un tiempo razonable, puesto que el uso de algoritmos exactos requiere de tiempos de cómputo elevados, como se puede observar en varias propuestas algorítmicas existentes en la literatura científica (Nassief et al., 2016), (Guemri et al., 2019), (Gelareh et al., 2020).

Una alternativa a los algoritmos exactos es el uso de heurísticas que permitan encontrar una solución en un tiempo de cómputo asumible. No obstante, aplicando heurísticas se corre el riesgo de que se diseñen en función de problemas específicos, y cuando se introducen nuevas instancias se obtengan soluciones de baja calidad.

Por lo tanto, en este trabajo se hace uso de metaheurísticas para resolver el CDAP. Una metaheurística es una estrategia de alto nivel para explorar espacios de soluciones utilizando diferentes métodos (Blum and Roli, 2003). Si bien las metaheurísticas no aseguran alcanzar una solución óptima, proporcionan mecanismos para resolver el problema en un tiempo bastante reducido y explorando un espacio de soluciones lo suficientemente amplio para analizar cuál es la mejor del conjunto de soluciones evaluadas.

Además, se analiza el uso de aprendizaje automático en metaheurísticas y se compara su desempeño con los otros algoritmos diseñados y con los que se encuentran en la literatura científica, con el fin de saber si esta rama de la inteligencia artificial resulta de interés para la resolución de problemas como el CDAP.

Por otro lado, en este trabajo también se persigue diseñar una aplicación web que permita visualizar soluciones del CDAP de manera gráfica, puesto que es un problema complejo donde se maneja una gran cantidad de datos y resulta difícil ver las asignaciones de los camiones en formato de texto.

Con el fin de cumplir el objetivo principal, es necesario realizar una revisión de la literatura científica del CDAP, investigar las instancias ya existentes para el problema, diseñar los algoritmos metaheurísticos con sus componentes y realizar un análisis computacional que permita estudiar la calidad de los algoritmos implementados.

1.2. Revisión de la literatura científica

En la literatura científica, varios problemas de optimización de *cross-docking* han recibido la atención de los investigadores. Entre ellos, se han considerado ampliamente los problemas relacionados con la localización de centros de *cross-docking* (Sung and Yang, 2008), la planificación de los camiones (Boysen et al., 2012), la asignación de los camiones a las puertas (Guemri et al., 2019), el flujo interno (Chen and Lee, 2009) y el recorrido de los vehículos (Wen et al., 2009). El flujo interno contempla los tiempos de descarga y carga de mercancía de los proveedores y clientes, respectivamente, mientras que el problema del recorrido de los vehículos se centra en los vehículos que transportan las mercancías desde las puertas de entrada hasta las puertas de salida.

Tanto los problemas de planificación de camiones como los de asignación de camiones a las puertas del *cross-docking* han llamado la atención de los investigadores. La gestión de un punto de transferencia intermedio se enfrenta a dos decisiones interrelacionadas: dónde y cuándo deben procesarse los camiones en las puertas de la terminal. Por un lado, la ubicación se obtiene resolviendo el problema de asignación de camiones a puertas y, por otro, el tiempo se obtiene resolviendo el problema de planificación de camiones. Generalmente, el objetivo del problema de asignación es minimizar la distancia total recorrida en el *cross-dock*, mientras que para el problema de planificación se minimiza el tiempo total del horizonte de planificación, o los retrasos en la salida de los camiones del *cross-dock*. Dado que ambos problemas son NP-duros, se han considerado secuencialmente en la literatura; primero se secuencian los camiones y luego se asignan las puertas del cross-dock.

El enfoque secuencial no asegura que haya una puerta disponible que permita asignar un camión de entrada a una puerta cercana a los camiones de salida que llevarán la mercancía a los destinos finales, y no es posible reducir la distancia recorrida en el *cross-dock*. Por tanto, esta solución es peor que la que se obtiene cuando ambas decisiones se toman de forma integrada, dando lugar al problema integrado de secuenciación de camiones y asignación de camiones a puertas, denominado en la literatura como *Cross-Dock Scheduling Problem*, CDSP (Boysen and Fliedner, 2010), (Van Belle et al., 2012), (Ladier and Alpan, 2016), (Rijal et al., 2019). La literatura asociada no es muy extensa y la mayoría de los trabajos resuelven los dos problemas asociados de forma secuencial. Rijal et al. (2019) proponen un algoritmo basado en grandes entornos para resolver el CDSP integrado con un modo mixto de uso de las puertas *cross-dock* (entrada y/o salida). Realizan una revisión de la escasa literatura para este problema, en la que se muestra un cuadro comparativo del enfoque seguido en cada uno de los trabajos referenciados. La mayoría de estos trabajos hacen uso de metaheurísticas para resolver el problema. Sin embargo, ninguno de ellos utiliza el esquema que combina simulación y optimización para dirigir la búsqueda metaheurística hacia soluciones más robustas.

Capítulo 2

Descripción del problema

El problema de asignación de camiones a puertas, *Cross-Dock Assignment Problem* (CDAP), consiste en asignar a cada proveedor y a cada cliente exactamente una puerta de entrada y una puerta de salida, respectivamente, de forma que se satisfagan las restricciones de capacidad de las puertas del centro de distribución y se minimice el coste total de manipulación de mercancías dentro del *cross-dock*.

El coste de manipulación viene dado por la cantidad de *pallets* que se transportan de una puerta de entrada a una puerta de salida en relación a la distancia entre éstas. Esto se podría denominar recorrido interno, y se pretende que la mercancía recorra la menor distancia posible en el centro de *cross-docking*.

A continuación, se describen los modelos matemáticos propuestos para la resolución del problema abordado en este trabajo. En primer lugar, se presenta una formulación cuadrática estándar y, en segundo lugar, se propone una formulación lineal entera mixta basada en flujo, que permitirá obtener soluciones óptimas para un subconjunto de las instancias de la literatura consideradas.

2.1. Parámetros

- M: Conjunto de proveedores (camiones entrantes).
- N: Conjunto de clientes (camiones salientes).
- I: Conjunto de puertas de entrada.
- J: Conjunto de puertas de salida.
- f_{kl}: Número de *pallets* que tiene que enviar el proveedor $k \in M$ al cliente $l \in N$.
- $F = (f_{kl})$: Matriz de flujo entre cada proveedor $k \in M$ y cada cliente $l \in N$.
- a_i: (capacidad de la puerta de entrada i) Máximo número de *pallets* que puede gestionar la puerta de entrada i.
- b_j: (capacidad de la puerta de salida j) Máximo número de *pallets* que puede gestionar la puerta de salida j.
- d_{ij}: Distancia entre la puerta de entrada *i* y la puerta de salida *j*.

- q: Número de *pallets* que se pueden mover en cada viaje desde la puerta de entrada *i* a la puerta de salida *j*.
- $s_k = \sum_{\forall l \in N} f_{kl}$: Número total de *pallets* que envía el proveedor $k \in M$.
- $r_l = \sum_{\forall k \in M} f_{kl}$: Número total de *pallets* que recibe el cliente $l \in N$.

2.2. Suposiciones

- El número de proveedores es mayor o igual al número de puertas de entrada. Esto es, $|M| \geq |I|$.
- El número de clientes es mayor o igual al número de puertas de salida. Esto es, $|N| \geq |J|$.

2.3. Formulación cuadrática estándar

Variables de decisión:

$$x_{ki} = \begin{cases} 1 & \text{si el proveedor } k \text{ está asignado a la puerta de entrada } i \\ 0 & \text{en otro caso} \end{cases}$$

$$y_{lj} = \begin{cases} 1 & \text{si el cliente } l \text{ está asignado a la puerta de salida } j \\ 0 & \text{en otro caso} \end{cases}$$

$$minz = \sum_{\forall k \in M} \sum_{\forall i \in I} \sum_{\forall j \in J} \sum_{\forall l \in N} d_{ij} \frac{f_{kl}}{q} x_{ki} y_{lj} \tag{2.1}$$

$$\sum_{\forall i \in I} x_{ki} = 1 \qquad (\forall k \in M) \tag{2.2}$$

$$\sum_{\forall j \in J} y_{lj} = 1 \qquad (\forall l \in N) \tag{2.3}$$

$$\sum_{\forall k \in M} s_k x_{ki} \leq a_i \qquad (\forall I \in I) \tag{2.4}$$

$$\sum_{\forall l \in N} r_l y_{lj} \leq b_j \qquad (\forall j \in J) \tag{2.5}$$

$$x_{ki} \in \{0,1\} \qquad (\forall k \in M, \forall i \in I)$$

$$y_{lj} \in \{0,1\} \qquad (\forall l \in N, \forall j \in J) \tag{2.6}$$

Si el proveedor *k* está asignado a la puerta de entrada *i* y el cliente *l* está asignado a la puerta de salida *j*, la expresión $d_{ij} \frac{f_{kl}}{q}$ en la función objetivo (2.1) representa la distancia total a recorrer para mover todos los *pallets* desde el proveedor *k* al cliente *l*.

El conjunto de restricciones (2.2) asegura que cada proveedor esté asignado a una única puerta de entrada, mientras que las restricciones (2.3) aseguran que cada cliente esté asignado a una única puerta de salida. Las restricciones (2.4) y (2.5) aseguran que las capacidades de las puertas, de entrada y de salida respectivamente, no se exceden.

2.4. Formulación lineal entera mixta basada en el flujo

Utilizando las variables de asignación anteriores x_{ki} y y_{lj}, y definiendo las variables de flujo adicionales z_{kij}, que representan el número de *pallets* que el proveedor *k* mueve desde la puerta de entrada *i* hasta la puerta de salida *j*, es posible formular el siguiente modelo lineal entero mixto.

$$minz = \frac{1}{q} \sum_{\forall k \in M} \sum_{\forall i \in I} \sum_{\forall i \in J} d_{ij} z_{kij} \tag{2.7}$$

$$\sum_{\forall i \in I} x_{ki} = 1 \qquad (\forall k \in M) \tag{2.8}$$

$$\sum_{\forall j \in J} y_{lj} = 1 \qquad (\forall l \in N) \tag{2.9}$$

$$\sum_{\forall k \in M} s_k x_{ki} \leq a_i \qquad (\forall I \in I) \tag{2.10}$$

$$\sum_{\forall l \in N} r_l y_{lj} \leq b_j \qquad (\forall j \in J) \tag{2.11}$$

$$\sum_{\forall j \in J} z_{kij} = s_k x_{ki} \qquad (\forall k \in M, \forall i \in I) \tag{2.12}$$

$$\sum_{\forall i \in I} z_{kij} = \sum_{\forall l \in N} f_{kl} y_{lj} \qquad (\forall k \in M, \forall j \in J) \tag{2.13}$$

$$\begin{aligned} x_{ki} &\in \{0,1\} && (\forall k \in M, \forall i \in I) \\ y_{lj} &\in \{0,1\} && (\forall l \in N, \forall j \in J) \\ z_{kij} &\geq 0 && (\forall k \in M, \forall i \in I), \forall j \in J) \end{aligned} \tag{2.14}$$

Las restricciones (2.12) y (2.13) unen las variables de decisión x_{ki} y y_{lj} con las variables de flujo z_{kij}. Las restricciones (2.12) aseguran que todos los *pallets* desde el proveedor *k* que se reciben en las puertas de entrada *i* se envían a sus clientes, mientras que las restricciones (2.13) asegura que todos los *pallets* que demandan los clientes al proveedor *k* en la puerta de salida *j* se reciben.

Capítulo 3

Generador de instancias

Para poder probar los diferentes algoritmos que se han implementado con el fin de obtener soluciones para el CDAP, se necesita contar con distintas instancias atendiendo a una serie de consideraciones, muchas de las cuales han sido originalmente descritas por Nassief et al. (2016). Uno de los parámetros considerados es la holgura, que constituye la capacidad adicional que tiene una puerta para recibir *pallets*, expresada en %. Por ejemplo, si el número total de *pallets* que envían los proveedores es de 50, y solo se cuenta con una puerta de entrada, con holgura 5 % la puerta dispone de capacidad 52,5. Así pues, a mayor holgura, las puertas contarán con mayor capacidad.

En el trabajo de Nassief et al. (2016) se realizan pruebas con un total de 50 instancias, el cual es un número limitado. Es por esto que se decidió generar un conjunto de instancias más amplio, con diferentes parámetros, para analizar la eficacia y eficiencia de los algoritmos diseñados.

3.1. Consideraciones

- El número de *pallets* que envía el proveedor $k \in M$ al cliente $l \in N$ es un entero generado aleatoriamente y obtenido mediante una distribución uniforme $U[10, 50]$ hasta alcanzar la densidad correspondiente ($25\,\%$, $35\,\%$, $50\,\%$, y $75\,\%$).
- Hay al menos un número entero no nulo en cada fila y columna de la matriz de flujo F. Esto significa que cada camión de entrada envía *pallets* al menos a un camión de salida y cada camión de salida recibe *pallets* de al menos un camión de entrada.
- Como especifican Sayed et al. (2020), se generan cuatro conjuntos de instancias de dificultad creciente que difieren en términos de la densidad de la matriz de flujo: $25\,\%$, $35\,\%$, $50\,\%$ y $75\,\%$. La densidad de la matriz de flujos se define como la relación entre el número de *pallets* que deben enviar los camiones de entrada a los de salida y el número máximo que podría transferirse en el escenario en el que todos los camiones de entrada tienen *pallets* que deben enviarse a cada uno de los camiones de salida.
- En este trabajo no se consideran los tiempos de procesamiento.
- En este trabajo se asume que no hay límite en el número de *pallets* que se pueden mover en cada viaje (q).

- La matriz de distancias se genera con números del intervalo $[8, 8+|I|-1]$ indicando que una distancia directa entre dos puertas (es decir, dos puertas enfrentadas) es de 8, y luego se añade un incremento de 1 unidad para la siguiente puerta indirecta.
- El número de camiones de entrada/salida considerados es 8, 9, 10, 11, 12, 15, 20 y 50.
- El número de puertas de entrada/salida considerados es 4, 5, 6, 7, 10, y 30.
- Las capacidades de todas las puertas son idénticas y se calculan dividiendo el flujo total procedente de todos los orígenes entre el número total de puertas de entrada, y añadiendo después una holgura de capacidad de 5, 10, 15, 20 y $30\,\%$.
- Como se especificó previamente, se utilizan los parámetros descritos por Nassief et al. (2016) y, para cada combinación, se generarán 5 instancias.

3.2. Implementación

Para generar las instancias, se hizo un programa en el lenguaje de programación Python y un *script* en Bash con el fin de automatizar la generación de todas las posibles instancias.

3.2.1. Programa en Python

El programa en Python recibe como parámetros de entrada los datos esenciales del problema, siendo éstos: el nombre del fichero de salida donde volcar los datos (en formato JSON [1]) y la cantidad de proveedores, clientes, puertas de entrada y puertas de salida. Opcionalmente, también se pueden indicar mínima y máxima cantidad de *pallets* que puede enviar un proveedor a un cliente, mínima y máxima capacidad de una puerta de entrada y de salida, la distancia mínima de una puerta de entrada a una de salida, la densidad de la matriz de flujos y el valor de holgura de las puertas. Los datos de entrada se indican por línea de comandos siguiendo las directrices POSIX (Pos, 2018).

Los datos opcionales tendrán un valor por defecto a excepción de la holgura, la cual, si se indica, se asignará a todas las puertas la misma capacidad, calculada con la ecuación (3.1), obviando las capacidades mínimas y máximas indicadas. Esta ecuación tiene en cuenta el número total de *pallets* que envían todos los proveedores, y se divide esta cantidad entre el número de puertas de entrada, obteniendo una capacidad base que se incrementa en función de la holgura (*slackness*).

$$a_i = b_j = \frac{\sum_{\forall k \in M} s_k}{|I|}(1 + slackness) \qquad (\forall i \in I, \forall j \in J) \tag{3.1}$$

La salida será un fichero en formato JSON con todos los datos necesarios para representar el problema. Cabe destacar que, de acuerdo a las consideraciones definidas en la sección 3.1, entre distintas ejecuciones con la misma entrada variará la instancia,

[1] https://www.json.org

debido a que existe un factor de aleatoriedad al generarla, concretamente a qué clientes un proveedor envía mercancía, y el número de *pallets* que se envían, lo cual afecta a la matriz de flujos y las capacidades de las puertas.

3.2.2. Script en Bash

Con el fin de automatizar la tarea de generar las instancias para el análisis, y teniendo en cuenta las consideraciones de la sección 3.1, se hizo un programa simple que, al ejecutarlo, genera instancias aleatorias, empleando el programa en Python e iterando sobre varias listas que contienen los valores a usar. Específicamente, el *script* consiste en un bucle que va iterando sobre los valores de número de camiones y de puertas, holgura y densidades descritos en la sección 3.1, y en todas las iteraciones se asignará una cantidad de *pallets* mínima de 10 y máxima de 50 y una distancia de 8 entre las puertas enfrentadas.

Por último, para diferenciar entre las instancias generadas e identificarlas correctamente, el nombre de los ficheros JSON resultantes seguirán la siguiente convención:

*Camiones*x*Puertas*x*Holgura*x*Densidad*_*Instancia*.json

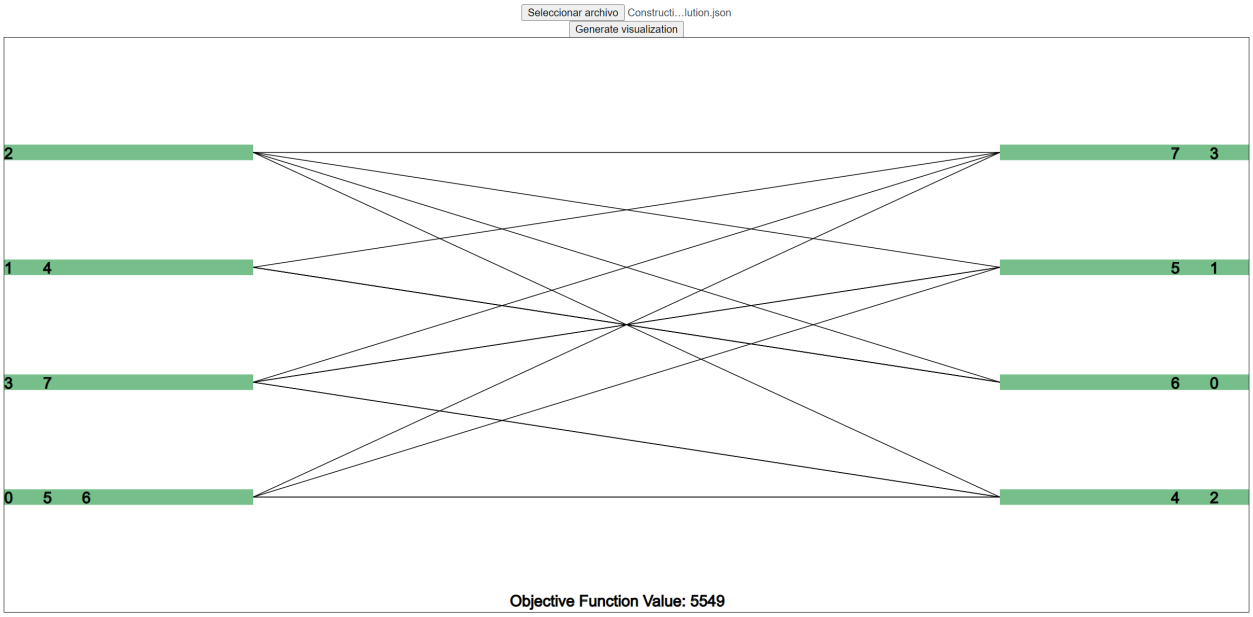
Por ejemplo, la primera instancia consistirá de 8 camiones y 4 puertas, con valor de holgura 5 % y densidad 25, por lo que se almacenará en el fichero *8x4x5x25_1.json*.

Capítulo 4

Visualizador de soluciones

Con el fin de poder visualizar las soluciones gráficamente, se desarrolló una aplicación *web* en el lenguaje de programación *JavaScript* utilizando el *framework Vue.js*. Se puede observar un ejemplo en la figura 4.1 de una solución generada con el algoritmo constructivo de asignación a la puerta con máxima capacidad, para un problema con 4 puertas de entrada/salida y 8 proveedores/clientes.

Figura 4.1: Visualizador de soluciones



Las puertas están representadas visualmente con rectángulos de color verde, concretamente el color #42b983 en formato hexadecimal, sobreponiendo el identificador de los camiones asignados a dicha puerta en texto de color negro (#000000 en formato hexadecimal). Cabe destacar que este contraste del texto sobre el fondo supera el mayor nivel de conformidad (AAA) de las directrices de accesibilidad al contenido *web* WCAG definida por el W3C[1]. El lado izquierdo representa las puertas de entrada donde estarán asignados los proveedores, y el derecho las puertas de salida donde estarán asignados los clientes.

[1] https://www.w3.org/WAI/WCAG2AAA-Conformance

Además, se dibujan una serie de líneas que representan los envíos de *pallets*. Una línea que une dos puertas simboliza que algún proveedor de la puerta de entrada envía mercancía a alguno de los clientes de la correspondiente puerta de salida. Por ejemplo, en el caso de la figuran, 4.1 el proveedor con identificador 4 está asignado a la puerta de entrada 1 (los identificadores comienzan en 0). La puerta de entrada 1 tiene dos líneas, que llevan a las puertas de salida 0 y 2. Esto se debe a que el proveedor 4 envía *pallets* al cliente 3 (puerta 0) y a los clientes 0 y 6 (puerta 2). Por debajo de la representación, y de manera centrada, se encuentra el valor de la función objetivo.

Al entrar en la aplicación *web*, el usuario se encuentra con un título, un selector de archivos y un botón para generar la visualización. El archivo debe estar en formato JSON y contener los datos esenciales para representar la solución mediante este visualizador, concretamente: número de puertas de entrada y salida, número de proveedores y clientes, asignaciones de proveedores y clientes, matriz de flujo de *pallets* y valor de la función objetivo. Para la figura 4.1, el fichero JSON contendría los datos mostrados en el fichero 4.1.

Fichero 4.1: Solución en formato JSON

```json
{
  "in_doors": 4,
  "out_doors": 4,
  "suppliers": 8,
  "customers": 8,
  "suppliers_assigned": [3, 1, 0, 2, 1, 3, 3, 2],
  "customers_assigned": [2, 1, 3, 0, 3, 1, 2, 0],
  "deliveries": [
    {"2": 26},
    {"0": 22},
    {"0": 41, "1": 32, "3": 50, "4": 30},
    {"4": 10, "5": 31},
    {"0": 40, "3": 44, "6": 50},
    {"1": 47, "3": 31},
    {"4": 43},
    {"1": 44, "2": 31, "7": 31}
  ],
  "objective_function_value": 5549.0
}
```

Para el fichero que contiene la solución, cada posición del vector de asignaciones de los camiones representa el identificador del correspondiente camión, y el valor corresponde a la puerta asignada. Por ejemplo, el proveedor 0 está asignado a la puerta con identificador 3. Además, en el vector de envíos cada posición representa el proveedor y su valor es un objeto donde las propiedades son los clientes que reciben mercancía de dicho proveedor y el valor de cada propiedad es el número de *pallets*. Por ejemplo, la línea 14 del fichero 4.1 simboliza que el proveedor con identificador 5 envía 47 *pallets* al cliente 1 y 31 *pallets* al cliente 3.

Capítulo 5

Algoritmos constructivos

Durante el presente capítulo se presentarán cuatro algoritmos constructivos para resolver el CDAP. Se dice que un algoritmo es constructivo cuando la solución al problema se forma de manera secuencial, en este caso asignando un proveedor/cliente (camión) a una puerta de entrada/salida en cada paso, haciendo uso de medidas heurísticas para elegir la puerta. Los algoritmos están descritos en pseudo-código para facilitar su lectura y abstraerlos del lenguaje de programación específico, pero el lenguaje de programación escogido para su implementación fue Java en su versión 17.0.1 [1].

b

El primer algoritmo implementado fue el de asignación a las puertas con mayor capacidad, debido a que parecía una heurística adecuada, pero en la implementación se fallaba en obtener soluciones para varias instancias, especialmente en las instancias con valores de holgura más ajustados. Esto fue lo que motivó a cambiar el criterio al de utilizar las puertas con capacidad más ajustada, esto es, aquellas que tuvieran la menor capacidad posible para colocar a un camión cumpliendo las restricciones de capacidad de la puerta, obteniendo más soluciones factibles en las instancias de menor holgura.

Un ejemplo sería la instancia *SetA_10x4S5* extraída de Nassief et al. (2016), que no producía una solución factible con el algoritmo de asignación a puertas de mayor capacidad, pero con la de menor capacidad sí que se obtuvo una solución factible con valor objetivo 6948, sabiendo que el valor óptimo obtenido a través de un algoritmo exacto es de 6518.

No obstante, dado que los algoritmos constructivos mencionados son deterministas, se diseñó un algoritmo que añade aleatoriedad en la elección de la puerta para cada uno de ellos, con el fin de obtener soluciones distintas en cada ejecución.

Por último, la solución inicial obtenida con los algoritmos anteriores no era lo suficientemente cercana a las soluciones óptimas ya conocidas. Es por esto que, con el fin de obtener una solución inicial de mayor calidad, se diseñó un algoritmo donde la asignación de los proveedores dependía de la de los clientes. Sin embargo, se descartó este algoritmo de cara al análisis computacional debido a que, aún obteniendo mejores valores de función objetivo, también existían problemas de factibilidad con las holguras más ajustadas.

[1] https://openjdk.java.net/projects/jdk/17/

5.1. Asignación a la puerta con mayor capacidad

El algoritmo 5.1 consiste en asignar tanto a proveedores como a clientes a la puerta que tenga mayor capacidad en ese instante. El algoritmo funciona de la misma manera para los dos grupos: proveedores o camiones de entrada y clientes o camiones de salida, pero en los algoritmos se muestran las asignaciones de los proveedores con el fin de simplificar la lectura. Primero, se inicializan las capacidades de las puertas, que son todas iguales por la manera en la que están generadas las instancias.

Data: CDAP Instance
Result: CDAP Solution
1 $Cap_i \leftarrow a_i, \forall i \in I$;
2 sort the trucks in a list L in decreasing order according to the s_k values;
3 **while** *there are trucks in L* **do**
4 take the truck *k* in the first position of *L*;
5 assign *k* to the door *i* with the biggest current capacity;
6 $Cap_i \leftarrow Cap_i - s_k$;
7 delete *k* from *L*;
8 **end**

Figura 5.1: Máxima Capacidad

A continuación, se ordenan los camiones en una lista atendiendo a la cantidad de *pallets* que envían/reciben en orden decreciente. Como estructura de datos se elige una lista, dado que permite tener un grupo de elementos ordenado y es dinámica, lo que permite añadir y eliminar elementos fácilmente.

Una vez ordenada la lista, se asignan los camiones que envían/reciben mayor número de *pallets* a la puerta con mayor capacidad en ese instante. Para ello, existe un bucle cuyo cuerpo se ejecutará mientras la lista de camiones no esté vacía, es decir, mientras haya camiones por asignar a puertas. Como están ordenados por *pallets*, en cada iteración se selecciona el primer camión de la lista, se asigna a la puerta con mayor capacidad en ese instante, se actualiza la capacidad de la puerta atendiendo a los *pallets* y, por último, se elimina el primer elemento de la lista, puesto que el camión ya ha sido asignado.

Data: CDAP Instance
Result: CDAP Solution
1 $Cap_i \leftarrow a_i, \forall i \in I$;
2 sort the trucks in a list L in decreasing order according to the s_k values;
3 **while** *there are trucks in L* **do**
4 take the truck *k* in the first position of *L*;
5 assign *k* to the door *i* with the lowest current capacity where $Cap_i - s_k \geq 0$;
6 $Cap_i \leftarrow Cap_i - s_k$;
7 delete *k* from *L*;
8 **end**

Figura 5.2: Algoritmo Constructivo Mínima Capacidad

5.2. Asignación a la puerta con menor capacidad

El algoritmo 5.2 sigue el mismo procedimiento que el anterior, cambiando el cuerpo del bucle *while* para que los camiones se asignen a la puerta con menor capacidad en cada instante. Cabe destacar que se requiere de una comprobación adicional para asegurar la factibilidad de la solución y cumplir las restricciones de capacidad del problema (restricciones 2.10 y 2.11).

5.3. Aleatoriedad en la elección de puertas

Con el fin de poder contar con distintas soluciones en cada ejecución para usarlas en las metaheurísticas, el algoritmo 5.3 cuenta con aleatoriedad en la elección de la puerta a la que se asigna el correspondiente camión. En este caso, la entrada también contempla un parámetro *n* que corresponde al número de candidatos que estarán en la lista de puertas que elegir.

Data: CDAP Instance, n
Result: CDAP Solution
$Cap_i \leftarrow a_i, \forall i \in I$;
sort the trucks in a list L in decreasing order according to the s_k values;
while *there are trucks in L* **do**
    take the truck *k* in the first position of *L*;
    sort the doors in a list *M* in decreasing order according to the Cap_i values;
    assign *k* to the door *i* selected randomly from the *n* first elements of *M*;
    $Cap_i \leftarrow Cap_i - s_k$;
    delete *k* from *L*;
end

Figura 5.3: Máxima Capacidad con Aleatoriedad

En el caso del algoritmo 5.3, las puertas se ordenan de mayor a menor capacidad para después poder elegir entre las *n* puertas con mayor capacidad, pero también se probó el algoritmo utilizando las puertas con menor capacidad. Además, para la elección del número de candidatos *n* se utilizó la fórmula 5.1. Por ejemplo, para una instancia con 10 puertas de entrada, el algoritmo escogerá una puerta aleatoria entre las 4 de mayor capacidad en ese instante.

$$\text{mín}(5, \lceil |I| * 0{,}4 \rceil) \tag{5.1}$$

5.4. Asignación de los proveedores en función de los clientes

Los anteriores algoritmos constructivos generan una solución donde las asignaciones de proveedores y clientes son independientes. No obstante, si se quiere partir de un mejor valor de función objetivo, el algoritmo 5.4 toma en cuenta la asignación de los clientes para colocar los proveedores.

Data: CDAP Instance
Result: CDAP Solution
assign customers to outbound doors;
$Cap_i = a_i, \forall i \in i$;
$B = M$;
sort the customers in a list L_1 in decreasing order according to the r_l values;
while *there are customers in* L_1 **do**
    select the customer *l* in the first position of L_1;
    select all suppliers of customer *l* that are currently in *B* and arrange them in a list L_2 in decreasing order according to the f_{kl} values;
    select the inbound door i closest to the outbound door *j* where the customer *l* was assigned, that has available capacity to assign the first supplier in L_2;
    while *there is enough capacity available at the inbound door i* **do**
        select the supplier *k* in the first position in L_2 and assign it to the inbound door *i*;
        identify the customers of supplier *k* and the outbound doors to which they have been assigned;
        update the r_l values for all customers of supplier *k*;
        remove from L_1 the customers with $r_l = 0$;
        $Cap_i \leftarrow Cap_i - s_k$;
        delete the supplier *k* from B and from L_2;
    end
    If list L_2 is not empty, then delete suppliers from L_2;
end

Figura 5.4: Proveedores Dependientes de Clientes

Primero, se asignarán los clientes siguiendo cualquiera de los anteriores algoritmos y se inicializarán: las capacidades de las puertas, una lista *B* que corresponde al conjunto de proveedores que no han sido asignados a puertas (inicialmente coincide con el conjunto de todos los proveedores) y una lista L_1 que corresponde a los clientes ordenados de manera decreciente por número de *pallets* que reciben.

A continuación, se entra en un bucle que se ejecutará mientras haya clientes en L_1, seleccionando aquellos que mayor número de *pallets* reciben y colocando todos los proveedores que envían al primer cliente *l* de L_1 en una lista L_2, también ordenados de acuerdo al número de *pallets* que envían. Una vez se obtiene L_2, se selecciona la puerta *i* más cercana al cliente *l* con suficiente capacidad para asignar al primer proveedor de L_2, y se colocan a los proveedores de L_2 en orden mientras haya capacidad en *i*, actualizando todos los datos correspondientes.

Capítulo 6

Búsquedas por entornos

Las búsquedas por entornos surgen como primera aproximación para llevar a cabo la mejora de las soluciones generadas por los algoritmos constructivos descritos en el capítulo anterior. Para diseñar e implementar estos algoritmos de búsqueda, es necesario definir los movimientos que se aplican a las soluciones para así, obtener los algoritmos de búsqueda local correspondientes.

Una búsqueda local es un método heurístico para resolver problemas de optimización computacionalmente difíciles, como es el caso del CDAP. Este tipo de algoritmos aplican cambios (movimientos) a la solución de partida y se analiza el resultado para ver si ha habido una mejora, usualmente esperando que la solución obtenida tras realizar los movimientos se aproxime lo máximo posible a la solución óptima (Michiels et al., 2018).

El conjunto de distintas soluciones generadas a partir de una inicial corresponde al entorno de la solución, y en este trabajo se han definido un total de 6 estructuras de entorno para el CDAP; relocalización de proveedor/cliente a otra puerta, intercambio de proveedores/clientes e intercambio de puertas de entrada/salida.

6.1. Movimientos

6.1.1. Relocalización de camiones

El movimiento de relocalización de un camión (tanto para proveedores como para clientes), consiste en escoger un camión asignado a una puerta y colocarlo en otra, siempre y cuando se satisfagan las restricciones de capacidad.

Por ejemplo, la figura 6.1 es la representación gráfica de una solución inicial generada usando el algoritmo constructivo de máxima capacidad descrito en la sección 5.1, y correspondiente al fichero 4.1. Una posible relocalización sería la representada en la figura 6.2: mover el proveedor 1 de la puerta 1 a la 2, lo cual decrementa el valor de la función objetivo en 22 unidades. Esto se debe a que el proveedor 1 envía 22 *pallets* al cliente 0, inicialmente asignado a una puerta con distancia 9 respecto a la del cliente. Con la relocalización, el proveedor está a distancia 8 del cliente; *5549 - (9 * 22) + (8 * 22) = 5549 - 22 = 5527*. Cabe destacar que las distancias fueron definidas en la sección 3.1, específicamente en la consideración de que las puertas enfrentadas tienen distancia 8 y por cada puerta indirecta a partir de la enfrentada, se incrementa en una unidad.

Figura 6.1: Solución inicial

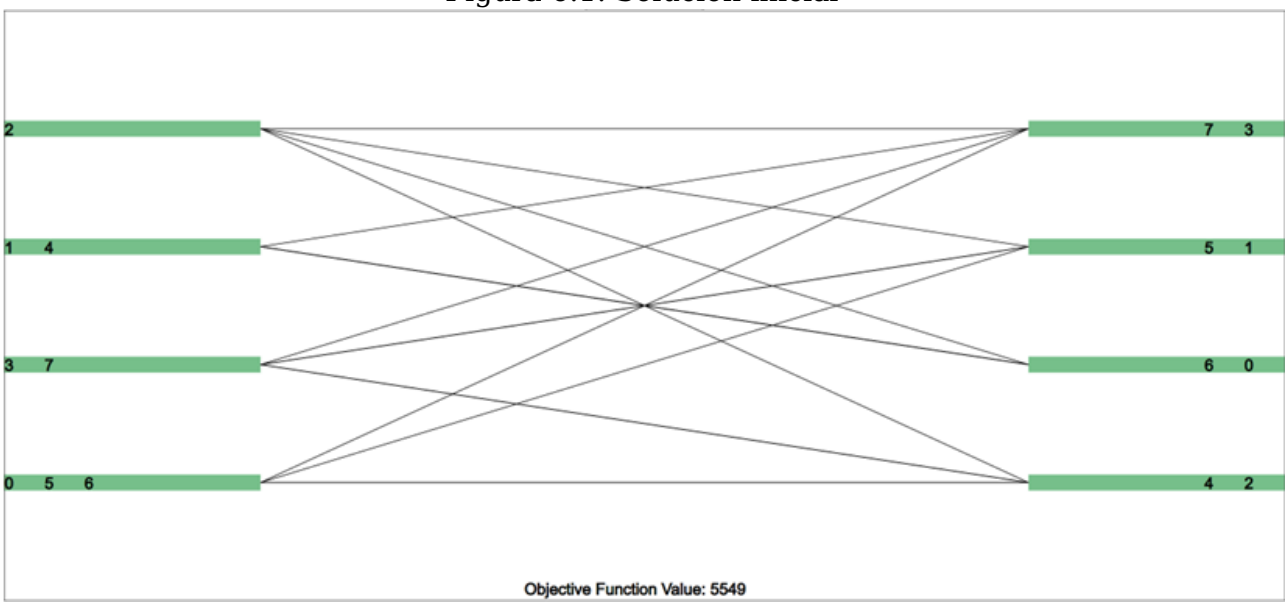
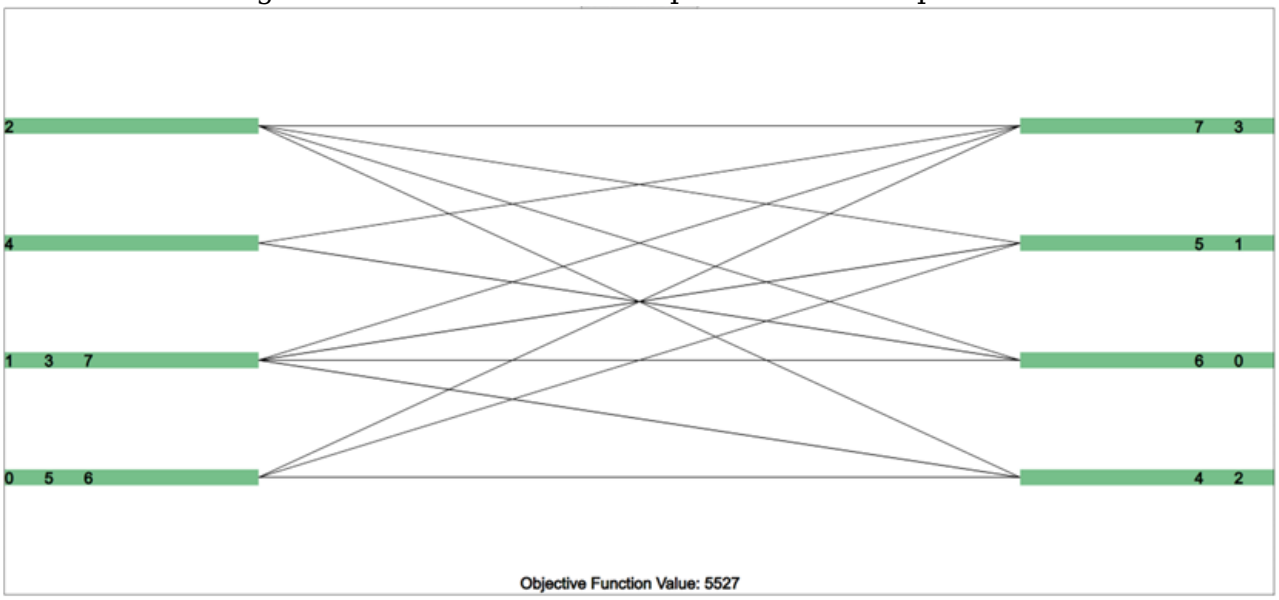


Figura 6.2: Relocalización del proveedor 1 a la puerta 2



De manera genérica, el cambio en la función objetivo viene dado por las ecuaciones 6.1 y 6.2, y una mejora de la solución corresponde a un valor de Δ negativo. En el ejemplo descrito en el párrafo anterior $\Delta = -22$.

- Relocalización de un proveedor k_1 de una puerta i_1 a otra i_2:

$$\Delta = \sum_{\forall l : f(k_1,l)>0} f(k_1, l)(d(i_2, j_l) - d(i_1, j_l)) \tag{6.1}$$

- Relocalización de un cliente l_1 de una puerta j_1 a otra j_2:

$$\Delta = \sum_{\forall k : f(k,l_1)>0} f(k, l_1)(d(i_k, j_2) - d(i_k, j_1)) \tag{6.2}$$

6.1.2. Intercambio de camiones

El movimiento de intercambio de camiones consiste en escoger un camión asignado a una puerta e intercambiarlo por otro camión asignado a otra puerta. Realmente se trataría de dos relocalizaciones de camiones simultáneas, por lo que el cambio de la función objetivo sería sumar ambas relocalizaciones, descritos para proveedores en la ecuación 6.3 y para los clientes en la 6.4.

- Intercambio entre el proveedor k_1 asignado a la puerta de entrada i_1 y el proveedor k_2 asignado a la puerta de entrada i_2:

$$\begin{aligned}\Delta_1 &= \sum_{\forall l: f(k_1,l)>0} f(k_1,l)(d(i_2,j_l) - d(i_1,j_l)) \\ \Delta_2 &= \sum_{\forall l: f(k_2,l)>0} f(k_2,l)(d(i_1,j_l) - d(i_2,j_l)) \end{aligned} \tag{6.3}$$

$$\Delta = \Delta_1 + \Delta_2$$

- Intercambio entre el cliente l_1 asignado a la puerta de salida j_1 y el cliente l_2 asignado a la puerta de salida j_2:

$$\begin{aligned}\Delta_1 &= \sum_{\forall k: f(k,l_1)>0} f(k,l_1)(d(i_k,j_2) - d(i_k,j_1)) \\ \Delta_2 &= \sum_{\forall k: f(k,l_2)>0} f(k,l_2)(d(i_k,j_1) - d(i_k,j_2)) \end{aligned} \tag{6.4}$$

$$\Delta = \Delta_1 + \Delta_2$$

6.1.3. Intercambio de puertas

El último entorno corresponde al descrito por los movimientos de intercambio de puertas, donde se escogerán dos puertas, y todos los camiones de la primera se asignan a la segunda y viceversa. Por ejemplo, si a la solución de la figura 6.1 se le aplica un intercambio entre las puertas 1 y 2, el resultado sería el que está representado en la figura 6.3, resultando en un $\Delta = -133$.

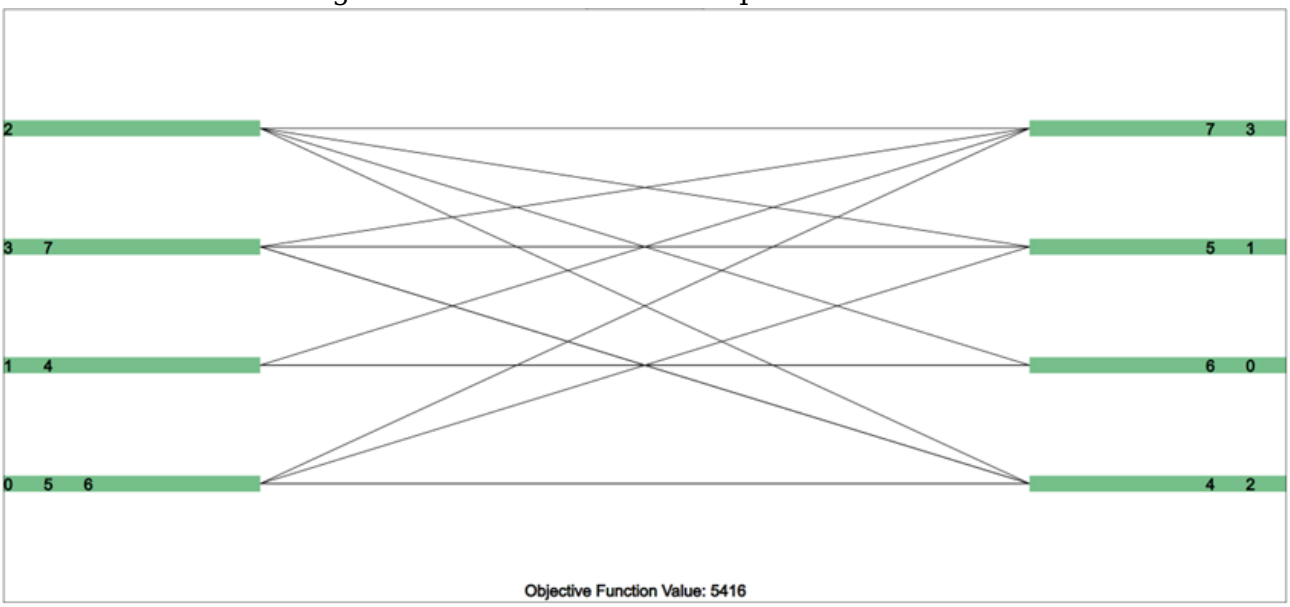
Para calcular el valor de Δ, se suman todas las relocalizaciones de camiones que se hacen de una puerta a otra, resultando en las ecuaciones 6.5 y 6.6 para los proveedores y clientes, respectivamente.

- Intercambio de todos los proveedores asignados a la puerta de entrada i_1 con todos los proveedores asignados a la puerta de entrada i_2:

$$\begin{aligned}\Delta_1 &= \sum_{\forall k \text{ in } i_1} \sum_{\forall l: f(k,l)>0} f(k,l)(d(i_2,j_l) - d(i_1,j_l)) \\ \Delta_2 &= \sum_{\forall k \text{ in } i_2} \sum_{\forall l: f(k,l)>0} f(k,l)(d(i_1,j_l) - d(i_2,j_l)) \end{aligned} \tag{6.5}$$

$$\Delta = \Delta_1 + \Delta_2$$

Figura 6.3: Intercambio de la puerta 1 con la 2



- Intercambio de todos los clientes asignados a la puerta de salida j_1 con todos los clientes asignados a la puerta de salida j_2:

$$\Delta_1 = \sum_{\forall l \text{ in } j_1} \sum_{\forall k: f(k,l)>0} f(k,l)(d(i_k, j_2) - d(i_k, j_1))$$
$$\Delta_2 = \sum_{\forall l \text{ in } j_2} \sum_{\forall k: f(k,l)>0} f(k,l)(d(i_k, j_1) - d(i_k, j_2)) \tag{6.6}$$
$$\Delta = \Delta_1 + \Delta_2$$

6.2. Generador de movimientos

Con el fin de facilitar el análisis de las soluciones contenidas en el entorno de una solución inicial (soluciones vecinas), se desarrolló un generador de movimientos. Dicho generador recibe una solución, y devuelve una lista de movimientos, descritos por su valor de Δ y los datos necesarios para el movimiento correspondiente (por ejemplo, para una relocalización de un camión a otra puerta es necesario saber el camión que se va a relocalizar y la puerta donde ocurrirá la nueva asignación).

La lista de movimientos generada contendrá todos los posibles, siempre y cuando resulten en una solución factible. Por ejemplo, para la solución representada en la figura 6.1, los posibles movimientos de intercambio de puertas son los siguientes, con las puertas a intercambiar y el valor de Δ: *D0<->D1 = 15.0, D0<->D2 = -22.0, D0<->D3 = 118.0, D1<->D2 = -133.0, D1<->D3 = 70.0, D2<->D3 = 56.0*. Como los movimientos están almacenados en una lista, se pueden ordenar fácilmente de menor a mayor valor de la función objetivo y saber cuál es el mejor movimiento (mejor solución vecina). En este caso, el intercambio de la puerta de entrada 1 con la 2 sería la mejor solución vecina, con un Δ de -133. Esto significa que hay una mejora notable en el valor de la función objetivo, por lo que es de interés realizar ese movimiento para obtener una mejor solución.

6.3. Tipos de búsqueda local

6.3.1. Voraz

Una búsqueda local voraz consiste en explorar todo el entorno de una solución y escoger el mejor movimiento, como está descrito en el algoritmo 6.4. Si aplicando el mejor movimiento se consigue mejorar el valor de la función objetivo, se genera el entorno de la solución resultante de aplicar el movimiento, y se explora. De lo contrario, el resultado obtenido sería un óptimo local.

```
   Data: initialSolution (CDAP Solution)
   Result: bestSolution (CDAP Solution)
 1 bestSolution ← initialSolution;
 2 solutionImproved ← true;
 3 while solutionImproved do
 4 |   solutionImproved = false;
 5 |   generate all movements of bestSolution and order them by Δ increasingly in a
   |    list L;
 6 |   select first movement m from L;
 7 |   if m improves bestSolution then
 8 |   |   bestSolution ← apply movement m to bestSolution;
 9 |   |   solutionImproved ← true;
10 |   end
11 end
```

Figura 6.4: Voraz

La ventaja de esta búsqueda local es que, ya que se explora todo el entorno, se sabe con certeza que se ha escogido la mejor solución del entorno. No obstante, la generación de todos los movimientos para cada solución podría suponer un coste computacional elevado.

6.3.2. Ansiosa

El algoritmo 6.5 describe la búsqueda local ansiosa, la cual consiste en explorar el entorno hasta encontrar el primer movimiento que mejore el valor de la función objetivo, tras lo cual se aplica el movimiento, resultando en una solución que volverá a ser explorada. Esto sucederá hasta que, para una solución, ninguno de los movimientos mejore el valor de función objetivo.

```
   Data: initialSolution (CDAP Solution)
   Result: bestSolution (CDAP Solution)
 1 bestSolution ← initialSolution;
 2 solutionImproved ← true;
 3 while solutionImproved do
 4 |   solutionImproved = false;
 5 |   L ← {};
 6 |   while there are possible movements of bestSolution not contained in L do
 7 |   |   m ← generate movement of bestSolution;
 8 |   |   L ← L ∪ m;
 9 |   |   if m improves bestSolution then
10 |   |   |   bestSolution ← apply movement m to bestSolution;
11 |   |   |   solutionImproved ← true;
12 |   |   |   break;
13 |   |   end
14 |   end
15 end
```

Figura 6.5: Ansiosa

Este algoritmo proporciona la ventaja de no tener que generar todos los movimientos para cada entorno, pero la exploración no resulta tan exhaustiva como en la búsqueda local voraz, por lo que es posible que se desechen soluciones mejores.

6.3.3. Por sub-entornos

La última búsqueda local consiste en generar sub-entornos con el fin de juntar las mejores características de las búsquedas voraz y ansiosa. Se define el sub-entorno, y se generan todos los posibles movimientos, escogiendo el que mejor valor de Δ tenga. Si con dicho movimiento se mejora la solución, se efectúa y se vuelve a hacer la búsqueda por sub-entornos. De lo contario, se analizaría el siguiente sub-entorno.

Por ejemplo, para el movimiento de relocalización de un proveedor a otra puerta, un sub-entorno podría ser aquel definido por escoger una puerta de entrada y analizar todas las relocalizaciones de proveedores asignados a otras puertas de entrada. Si se halla una relocalización que mejore el valor de la función objetivo, se efectúa y se vuelve a analizar la primera puerta. De lo contrario, se genera el siguiente sub-entorno. El óptimo local se alcanza una vez analizadas todas las puertas de entrada sin encontrar un movimiento que mejore la solución.

Capítulo 7

GRASP

En este capítulo se describirá el algoritmo metaheurístico GRASP, implementado para conseguir una solución de alta calidad para el CDAP. El procedimiento GRASP fue descrito por primera vez por Feo and Resende (1995) y se describe de manera genérica en el algoritmo 7.1. GRASP corresponde a las siglas de *Greedy Randomized Adaptive Search Procedure* o procedimiento de búsqueda adaptativa aleatoria voraz en español.

```
   Data: problem (CDAP Instance)
   Result: bestSolution (CDAP Solution)
1  preprocessing phase;
2  while stopping criteria not fulfilled do
3  |  constructive phase;
4  |  postprocessing phase;
5  |  update bestSolution;
6  end
```

Figura 7.1: Procedimiento GRASP

Cabe destacar que el GRASP es un proceso iterativo que consiste de dos fases (además de la de pre-procesamiento que se efectúa de manera opcional previa al bucle): fase constructiva y fase de post-procesamiento (también conocida como fase de mejora), donde la primera consiste en generar una solución por medio de una heurística determinada, y la segunda en mejorar dicha solución explorando su espacio de búsqueda. El GRASP es un algoritmo metaheurístico debido a que se construyen diferentes soluciones y se exploran, llegando a tener un espacio de soluciones amplio con el fin de encontrar una solución de alta calidad, y quedarnos con la que mejore el valor de la función objetivo, en este caso con aquella que tenga menor valor al tratarse de un problema de minimización.

En este trabajo, el criterio de parada es un número de iteraciones fijas, habiendo probado distintas combinaciones de cara al análisis computacional. Además, se añadió un número de iteraciones sin mejora, que se obtiene multiplicando el número de iteraciones fijas por un factor de 0,01. Por ejemplo, si se emplean 10.000 iteraciones, tras 100 iteraciones donde el valor de la función objetivo no mejora, se detiene el procedimiento.

Por otro lado, la fase de pre-preprocesamiento consiste en generar una solución factible de partida con el fin de inicializar la mejor solución. La solución de partida será

aleatoria, dependiendo del algoritmo constructivo que se escoja. La fase constructiva es similar, con la diferencia de tener en cuenta las soluciones que ya han sido visitadas. Para esto, se utiliza un conjunto donde se van añadiendo las soluciones únicas. En este caso, se definen como soluciones únicas aquellas cuyo valor función objetivo coincide, sin tener en cuenta las asignaciones de los camiones.

Por último, existe una fase de post-procesamiento o mejora donde se explora la solución obtenida en la fase constructiva por medio de búsquedas. Para esta fase, se utilizaron dos algoritmos: el algoritmo de Búsqueda Variable Descendiente o VND (*Variable Neighborhood Descent*) y una versión alternativa del VND donde se hace uso de aprendizaje automático.

7.1. VND

El algoritmo VND se basa en el hecho de que un óptimo local obtenido para un entorno N_1 no tiene por qué coincidir con el óptimo local obtenido en otro entorno N_2, por lo que puede ser ventajoso combinar varios entornos. En la figura 7.2 se describe el algoritmo VND básico, siendo la entrada una solución de partida (obtenida con el algoritmo constructivo en este caso), y un conjunto de entornos *N*.

```
   Data: x (CDAP Solution), N
   Result: bestSolution (CDAP Solution)
 1 while no improvement obtained do
 2 |   l ← 1;
 3 |   while l ≤ l_max do
 4 |   |   find the best neighbor x' of x (x' ∈ N_l);
 5 |   |   if x' is better than x then
 6 |   |   |   x ← x';
 7 |   |   |   l ← 1;
 8 |   |   else
 9 |   |   |   l ← l + 1;
10 |   |   end
11 |   end
12 |   bestSolution ← x;
13 end
```

Figura 7.2: VND Básico

En este trabajo se emplean 6 estructuras de entorno, definidas por los movimientos descritos en la sección 6.1; relocalización de proveedores, relocalización de clientes, intercambio de proveedores, intercambio de clientes, intercambio de puertas de entrada e intercambio de puertas de salida.

Por otro lado, para mejorar el VND básico se podría aprovechar las búsquedas locales definidas en 6.3 para que, una vez se encuentre una solución vecina que mejore el valor de la función objetivo, obtener un óptimo local a partir de ésta. Es por ello que para este trabajo se planteó el algoritmo 7.3, donde se realiza una búsqueda local de la mejor

solución vecina (si ésta ha mejorado el valor de la función objetivo). En concreto, se usó la búsqueda local voraz para encontrar un óptimo local.

```
   Data: x (CDAP Solution), N
   Result: bestSolution (CDAP Solution)
 1 while no improvement obtained do
 2 |   l ← 1;
 3 |   while l ≤ l_max do
 4 |   |   find the best neighbor x' of x (x' ∈ N_l);
 5 |   |   if x' is better than x then
 6 |   |   |   x ← localSearch(x');
 7 |   |   |   l ← 1;
 8 |   |   else
 9 |   |   |   l ← l + 1;
10 |   |   end
11 |   end
12 |   bestSolution ← x;
13 end
```

Figura 7.3: VND con búsqueda local

7.1.1. Orden de las estructuras de entorno

Una de las incógnitas que surge a la hora de implementar el VND es en qué orden poner las estructuras de entorno, debido a que el algoritmo se ejecuta de manera secuencial, y una estructura de entorno podría encaminar una solución hacia un espacio de soluciones determinado. En este trabajo se contemplan dos órdenes:

- RSD
  1. Relocalización de camiones
  2. Intercambio de camiones
  3. Intercambio de puertas
- DRS
  1. Intercambio de puertas
  2. Relocalización de camiones
  3. Intercambio de camiones

Cada ítem corresponde a un par de estructuras de entorno, analizando primero los proveedores y después los clientes. Así pues, la opción RSD tendrá el siguiente conjunto de entornos:

- N_1: Relocalización de proveedores
- N_2: Relocalización de clientes

- N_3: Intercambio de proveedores
- N_4: Intercambio de clientes
- N_5: Intercambio de puertas de entrada
- N_6: Intercambio de puertas de salida

7.1.2. Aprendizaje automático

El orden de las estructuras de entorno podría cambiar en función de la instancia del CDAP, atendiendo a diferentes parámetros como la holgura o la densidad de la matriz de flujo. Es por esto que se introdujo el algoritmo 7.4, que aplica aprendizaje automático, con el fin de escoger las estructuras de entorno en función del problema específico que se trate.

Data: x (CDAP Solution), N
Result: bestSolution (CDAP Solution)
while *no improvement obtained* **do**
  $l \leftarrow$ select neighborhood $\in N$ according to weights and recency;
  find the best neighbor *x'* of *x* ($x' \in N_l$);
  if *x' is better than x* **then**
    $x \leftarrow localSearch(x')$;
    $N_l \leftarrow$ increase weight;
  $bestSolution \leftarrow x$;
end

Figura 7.4: VND con aprendizaje automático

Se observa que no es necesario iterar sobre las estructuras de entorno, ya que se escogerán en función de su peso. El peso se define con la fórmula 7.1, estableciendo una probabilidad base para cada estructura de entorno que siempre será de $\frac{1}{|N|} * 0{,}3 = \frac{1}{6} * 0{,}3 = 0{,}05$.

$$weight(x) = \frac{success(x)}{\sum_{\forall l \in N} success(l)} * 0{,}7 + \frac{1}{|N|} * 0{,}3 \tag{7.1}$$

Adicionalmente, existe otra probabilidad que dependerá del número de veces que una estructura de entorno haya mejorado la solución y será la que mayor impacto tenga a la hora de seleccionarla (en la primera iteración se escogerá una estructura de entorno aleatoria donde todas tengan la misma probabilidad, ya que el número de mejoras totales será 0). El numerador *success(x)* correspondería al número de veces que la estructura de entorno *x* ha mejorado una solución, y el denominador corresponde al número total de mejoras de la solución por parte de cualquiera de las estructuras de entorno definidas.

Otro aspecto a tener en cuenta es lo reciente que ha sido la estructura de entorno. Por ejemplo, no tendría sentido escoger la misma estructura de entorno que se empleó en la iteración anterior, debido a que ya se obtuvo un óptimo local con ella. Es por ello que

el código contempla una lista de estructuras de entorno activadas, desactivándolas cada vez que se escoge. Si en la siguiente iteración se encuentra otra estructura que mejore la solución, se activa la anterior y desactiva la nueva. Esto también permite saber cuándo finalizar la búsqueda: una vez todas las estructuras de entorno estén desactivadas.

Por ejemplo, si N_1 ha mejorado la solución 2 veces, y N_5 la ha mejorado 1 vez, hay un total de 3 mejoras, y tendríamos los siguientes pesos: $weight(N_1) \approx 52\,\%$, $weight(N_5) \approx 28\,\%$ y $weight(N_i) = 5\,\%, \forall i \in \{2, 3, 4, 6\}$. No obstante, si se acaba de mejorar la solución utilizando N_5, la desactivamos y los pesos se ajustarían para que sumen 100 pero se mantenga la proporción: $weight(N_1) \approx 72\,\%$ y $weight(N_i) \approx 7\,\%, \forall i \in \{2, 3, 4, 6\}$.

Capítulo 8

Resultados computacionales

En este capítulo, se analizan los resultados obtenidos con los distintos algoritmos, teniendo en cuenta que la implementación fue en el lenguaje de programación Java 17.0.1 y se ejecutó en un ordenador con 16 GB de RAM y procesador Intel Core i5-9600K con 3.7 GHz.

Se escogieron 6 algoritmos GRASP para el análisis computacional, analizando 2 algoritmos para la fase constructiva: asignación a puertas de máxima y mínima capacidad y tres VND; orden RSD, orden DRS y el VND con aprendizaje automático.

8.1. Factibilidad

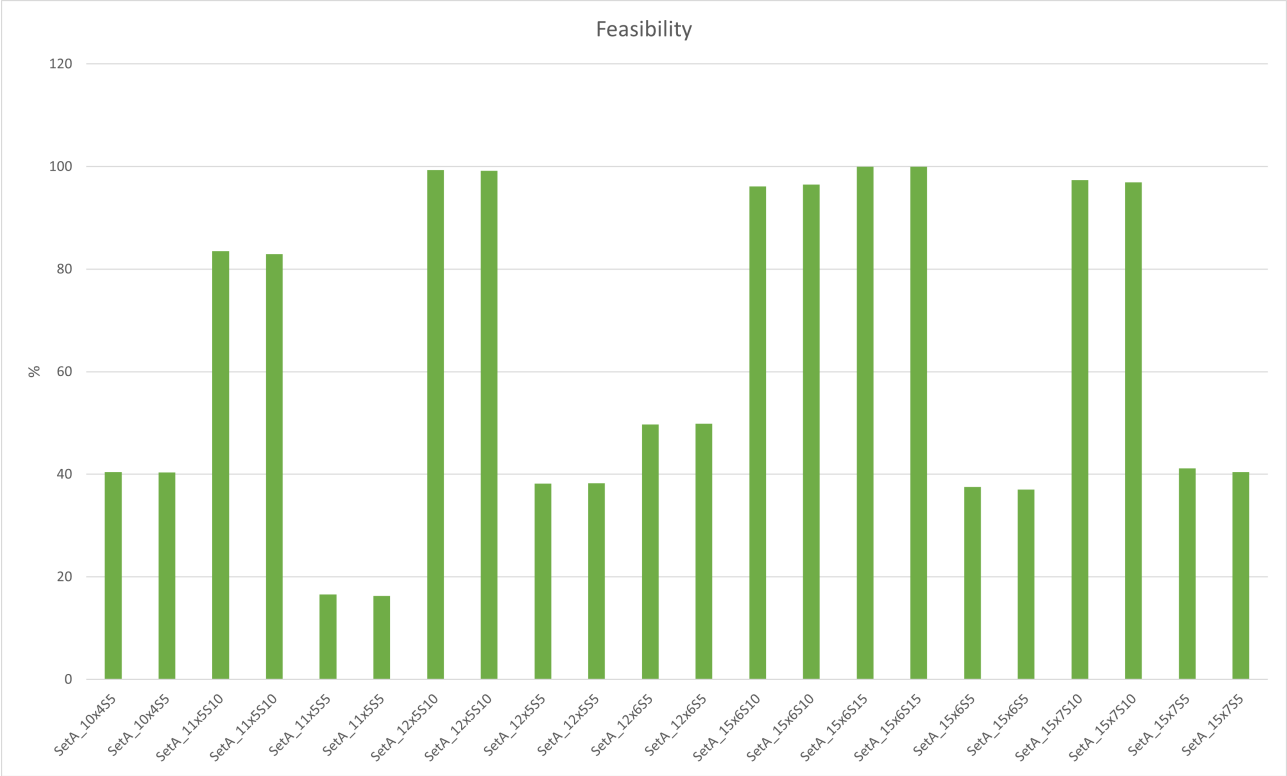
Los algoritmos 5.1, 5.2 y 5.4 son deterministas, es decir, siempre dan la misma solución en cada ejecución. Esto puede resultar un problema ya que no aseguran obtener una solución factible. Guignard et al. (2012) generaron un conjunto de 50 instancias para el CDAP denominado *SetA*, las cuales se utilizaron en este trabajo con el fin de comprobar los distintos algoritmos implementados. Por ejemplo, se describe un *SetA_10x4S5* para el cual no se obtiene una solución factible con el algoritmo constructivo de asignación a la puerta con mayor capacidad.

Así pues, se realizó un análisis de factibilidad de las 50 instancias para saber si, con los algoritmos que añaden aleatoriedad, se obtiene una solución factible en alguna de las ejecuciones. En la tabla 8.1 se puede ver aquellas instancias (11 de las 50) con las que no se conseguía factibilidad en el 100 % de las ejecuciones de los algoritmos de asignación a puertas de máxima capacidad (Max) y mínima capacidad (Min) con aleatoriedad. Se realizaron 10.000 ejecuciones para cada uno, y también se analizó el número de soluciones únicas que se obtienen, definiendo como única aquella cuyo valor de la función objetivo ya ha sido obtenido (independientemente de la asignación final).

Cabe destacar que los dos algoritmos tienen resultados bastante similares, por lo que no parece relevante la elección de éste de cara a la factibilidad. También, se puede observar que la holgura influye en la factibilidad, ya que aquellas que no superan el 50 % son de holgura 5, y el resto de 10 o 15, obteniendo soluciones factibles con las instancias con holgura 20 y 30 en todas las ejecuciones.

Con estos resultados se puede concluir que estos dos algoritmos son adecuados para generar soluciones, pudiendo conseguir soluciones factibles para las 50 instancias.

Figura 8.1: Factibilidad



Cuadro 8.1: Análisis de factibilidad

Instance	**Algorithm**	**Feasible Solutions**	**%**	**Unique Solutions**
SetA_10x4S5	Max	4045	40,45	582
SetA_10x4S5	Min	4033	40,33	578
SetA_11x5S10	Max	8352	83,52	898
SetA_11x5S10	Min	8294	82,94	904
SetA_11x5S5	Max	1658	16,58	529
SetA_11x5S5	Min	1629	16,29	510
SetA_12x5S10	Max	9930	99,3	918
SetA_12x5S10	Min	9920	99,2	919
SetA_12x5S5	Max	3819	38,19	815
SetA_12x5S5	Min	3823	38,23	805
SetA_12x6S5	Max	4969	49,69	1020
SetA_12x6S5	Min	4986	49,86	1021
SetA_15x6S10	Max	9616	96,16	1327
SetA_15x6S10	Min	9649	96,49	1348
SetA_15x6S15	Max	9997	99,97	1416
SetA_15x6S15	Min	9998	99,98	1378
SetA_15x6S5	Max	3753	37,53	1108
SetA_15x6S5	Min	3701	37,01	1122
SetA_15x7S10	Max	9733	97,33	1526
SetA_15x7S10	Min	9692	96,92	1529
SetA_15x7S5	Max	4115	41,15	1217
SetA_15x7S5	Min	4046	40,46	1195

Además, también se consigue un número considerable de soluciones únicas, por lo que existirá un espacio de soluciones amplio con el que trabajar en las metaheurísticas.

8.2. Resultados GRASP

8.2.1. Iteraciones

El algoritmo GRASP tiene como condición de parada un número de iteraciones fijo. Es de interés saber qué valor asignar a esta variable, debido a que podría influir en la solución obtenida.

Cuadro 8.2: Número de iteraciones

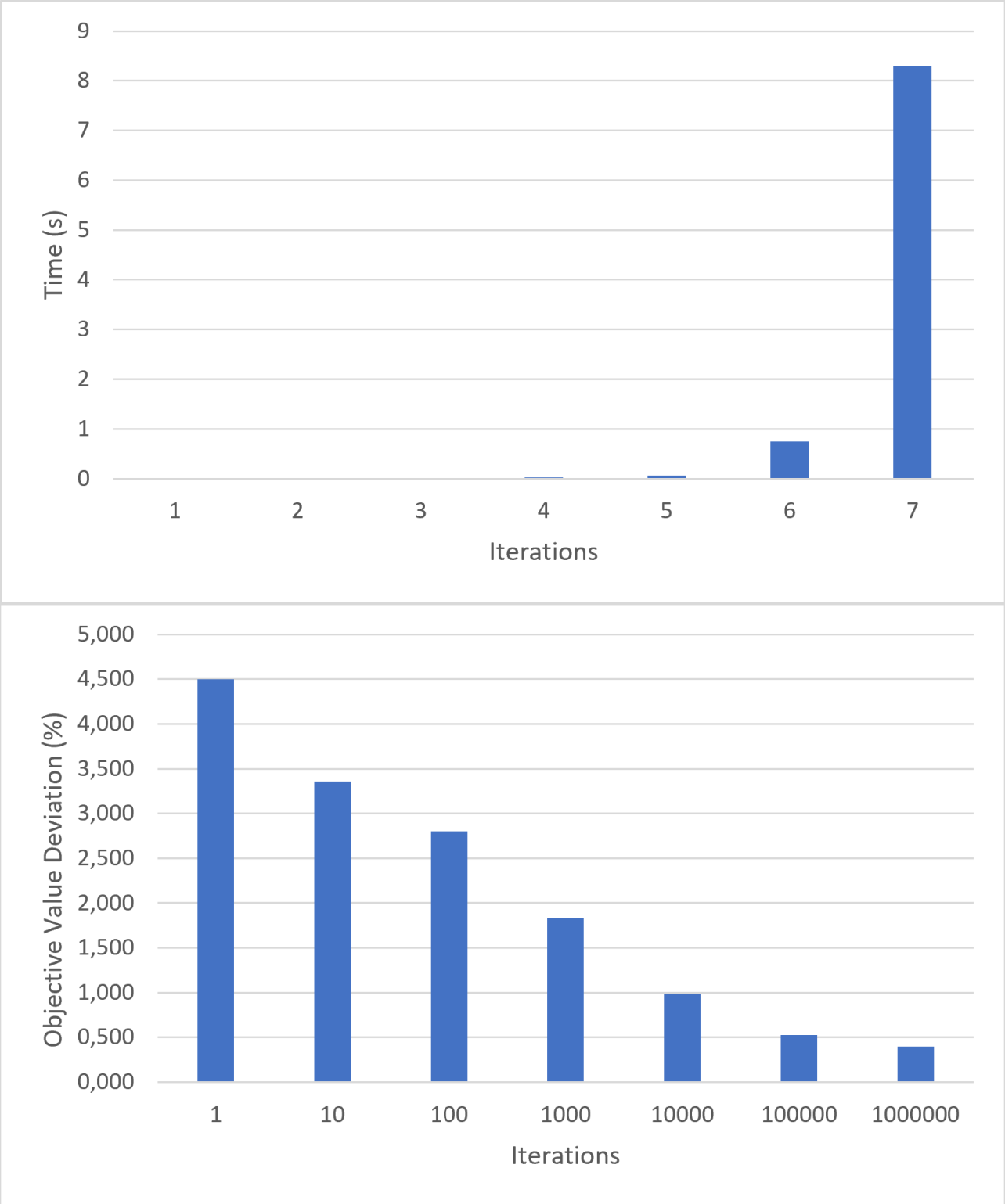
Iterations	Mean ObjVal	Mean BKV	%	Mean Time(s)	Mean BKT
1	11308.773	10741.84	4.501	0.0112	164.016
10	11178.240	10741.84	3.356	0.0135	164.016
100	11111.733	10741.84	2.803	0.0157	164.016
1000	11004.847	10741.84	1.832	0.0251	164.016
10000	10903.773	10741.84	0.990	0.061	164.016
100000	10842.677	10741.84	0.528	0.745	164.016
1000000	10819.973	10741.84	0.399	8.296	164.016

En la tabla 8.2 se puede observar que se han definido 7 valores de número de iteraciones. Para cada uno, se analiza el valor de la función objetivo promedio obtenido con los 6 algoritmos metaheurísticos, el mejor valor conocido (BKV), el promedio de desviación del valor de la función objetivo obtenido respecto al mejor valor conocido (%), el tiempo de CPU promedio en segundos y el tiempo que tardó el algoritmo que proporcionó el mejor valor conocido (BKT). Cabe destacar que se habla de mejor valor conocido en vez de valor óptimo debido a que, para las instancias de tamaño *20x10* no se pudo obtener solución óptima por las limitaciones de tiempo de cómputo que tiene el algoritmo exacto, por lo que para esta tabla se usaron los valores obtenidos por Guemri et al. (2019), que emplean otros métodos metaheurísticos, concretamente dos búsquedas tabú probabilísticas.

De estos datos, y mirando las gráficas obtenidas (figura 8.2), se puede sacar la siguiente conclusión: a mayor número de iteraciones, se requiere de mayor tiempo de cómputo para ejecutar el algoritmo, pero se obtiene una solución de mejor calidad. No obstante, la diferencia respecto a 100.000 y 1.000.000 iteraciones es bastante notable respecto al tiempo de cómputo, pero tan solo de un 0.014 % respecto a la desviación de la función objetivo, por lo que escoger 100.000 iteraciones parece el mejor valor para obtener una solución de calidad sin esperar demasiado tiempo.

En los posteriores análisis se ejecutan los algoritmos utilizando 100.000 iteraciones, ya que sería un valor de número de iteraciones realista.

Figura 8.2: Número de iteraciones



8.2.2. Mejor algoritmo promedio

Resulta de interés saber cuál de los 6 algoritmos implementados es el más polivalente, es decir, cuál se puede usar de manera genérica para cualquier instancia. Es por ello que con la tabla 8.3 se realiza un estudio similar al del número de iteraciones, analizando en este caso qué algoritmo se utiliza en la fase constructiva y qué algoritmo VND se utiliza en la fase de post-procesamiento del GRASP.

Cuadro 8.3: Algoritmos

Constructive	**VND**	**Mean ObjVal**	**Mean BKV**	**%**	**Time(s)**	**Mean BKT**
Max	DRS	10836.02	10741.84	0.507	0.835	164.016
Max	RSD	10845.92	10741.84	0.547	0.754	164.016
Max	Learning	10851.08	10741.84	0.562	0.740	164.016
Min	DRS	10837.18	10741.84	0.502	0.803	164.016
Min	RSD	10842.28	10741.84	0.519	0.647	164.016
Min	Learning	10843.58	10741.84	0.533	0.693	164.016

Se puede observar que el algoritmo constructivo de asignación a puertas de mínima capacidad es mejor en promedio en cuanto a tiempo y valor de la función objetivo respecto a su contraparte de máxima capacidad. Además, el mejor VND en cuanto a valor de función objetivo es el de orden DRS, aunque requiere de mayor tiempo de cómputo que los otros algoritmos VND.

Por otra parte, el algoritmo en el que se usa aprendizaje automático no parece aportar mejora ni en tiempo de cómputo ni en valor de la función objetivo respecto a los otros dos, por lo que priorizar una estructura de entorno que mejora la solución con frecuencia no parece ser la mejor decisión en la mayoría de casos.

Figura 8.3: Algoritmos

8.2.3. Mejor algoritmo según holgura

El análisis anterior se hizo sin tener en cuenta características individuales de las instancias. Uno de los valores que más parecen influir a la hora de obtener una solución es el valor de holgura de las puertas, posiblemente porque existe menor cantidad de movimientos factibles que realizar. Con el fin de saber qué algoritmo funciona mejor para cada valor de holgura, se realizaron las tablas 8.4, 8.5, 8.6, 8.7 y 8.8, con sus respectivas gráficas.

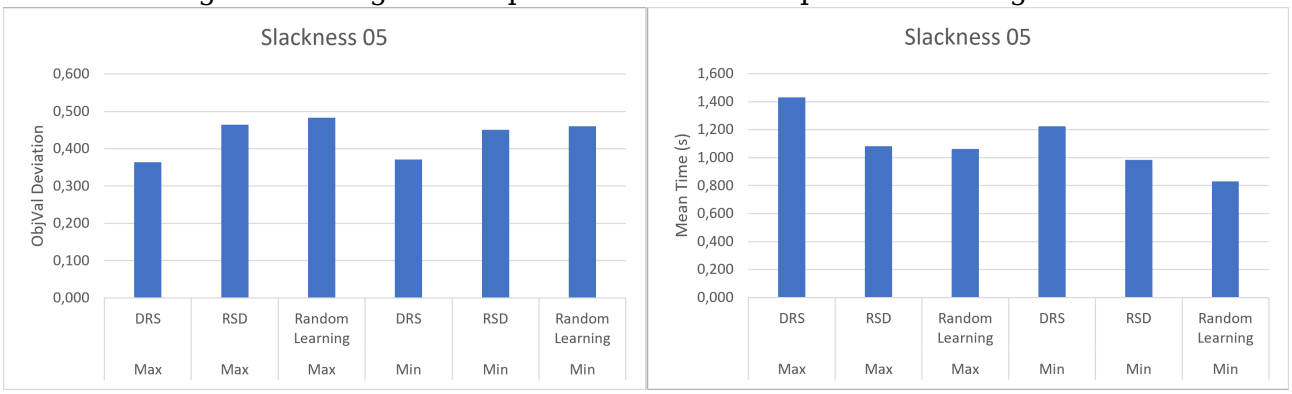
Una de las primeras observaciones que se puede hacer es que los resultados obtenidos con la holgura 15 condicionaron el valor promedio obtenido en el apartado anterior, debido a que en 4 de los 5 valores de holgura el mejor valor de función objetivo promedio se obtiene con un algoritmo constructivo de máxima capacidad, pero en la que tiene valor 15 la diferencia entre algoritmo de máxima y mínima capacidad es notable.

Otro aspecto destacable es que con el algoritmo VND que emplea aprendizaje automático se obtienen peores resultados en cuanto a valor de función objetivo en las menores holguras, pero se obtienen mejores tiempos de cómputo. Sin embargo, en valores de holgura mayores, y especialmente en aquellos de 20, se obtienen muy buenos resultados en cuanto a valor de función objetivo para el algoritmo VND con aprendizaje automático, especialmente utilizando el constructivo de máxima capacidad.

Cuadro 8.4: Algoritmos para instancias con puertas de holgura 05

Constructive	VND	Mean ObjVal	Mean BKV	%	Time	Mean BKT
Max	DRS	11086,1	11001,8	0,364	1,431	21,81
Max	RSD	11105,6	11001,8	0,465	1,083	21,81
Max	Random Learning	11104,3	11001,8	0,483	1,063	21,81
Min	DRS	11083,0	11001,8	0,371	1,224	21,81
Min	RSD	11104,3	11001,8	0,451	0,985	21,81
Min	Random Learning	11102,5	11001,8	0,460	0,831	21,81

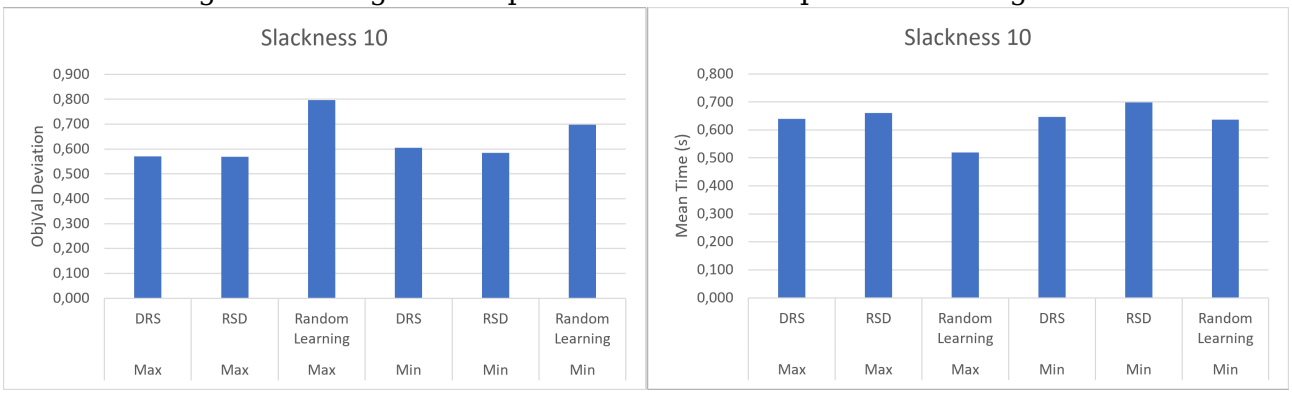
Figura 8.4: Algoritmos para instancias con puertas de holgura 05



Cuadro 8.5: Algoritmos para instancias con puertas de holgura 10

Constructive	VND	Mean ObjVal	Mean BKV	%	Time	Mean BKT
Max	DRS	10887,8	10785,2	0,571	0,640	20,391
Max	RSD	10902,1	10785,2	0,569	0,661	20,391
Max	Random Learning	10935,4	10785,2	0,797	0,519	20,391
Min	DRS	10896,5	10785,2	0,604	0,646	20,391
Min	RSD	10903,8	10785,2	0,584	0,699	20,391
Min	Random Learning	10917,5	10785,2	0,698	0,637	20,391

Figura 8.5: Algoritmos para instancias con puertas de holgura 10



Cuadro 8.6: Algoritmos para instancias con puertas de holgura 15

Constructive	VND	Mean ObjVal	Mean BKV	%	Time	Mean BKT
Max	DRS	10841,0	10717,3	0,723	0,588	24,494
Max	RSD	10825,8	10717,3	0,601	0,620	24,494
Max	Random Learning	10846,1	10717,3	0,709	0,608	24,494
Min	DRS	10819,5	10717,3	0,515	0,777	24,494
Min	RSD	10819,5	10717,3	0,563	0,475	24,494
Min	Random Learning	10808,1	10717,3	0,517	0,666	24,494

Cuadro 8.7: Algoritmos para instancias con puertas de holgura 20

Constructive	VND	Mean ObjVal	Mean BKV	%	Time	Mean BKT
Max	DRS	10750,9	10650,1	0,504	0,743	739,012
Max	RSD	10742,1	10650,1	0,518	0,621	739,012
Max	Random Learning	10727,3	10650,1	0,378	0,705	739,012
Min	DRS	10756,2	10650,1	0,601	0,709	739,012
Min	RSD	10746,8	10650,1	0,565	0,570	739,012
Min	Random Learning	10747,5	10650,1	0,541	0,671	739,012

Cuadro 8.8: Algoritmos para instancias con puertas de holgura 30

Constructive	VND	Mean ObjVal	Mean BKV	%	Time	Mean BKT
Max	DRS	10614,3	10554,8	0,371	0,776	14,374
Max	RSD	10654,0	10554,8	0,582	0,786	14,374
Max	Random Learning	10642,3	10554,8	0,443	0,804	14,374
Min	DRS	10630,7	10554,8	0,420	0,658	14,374
Min	RSD	10637,0	10554,8	0,429	0,505	14,374
Min	Random Learning	10642,3	10554,8	0,449	0,659	14,374

Figura 8.6: Algoritmos para instancias con puertas de holgura 15

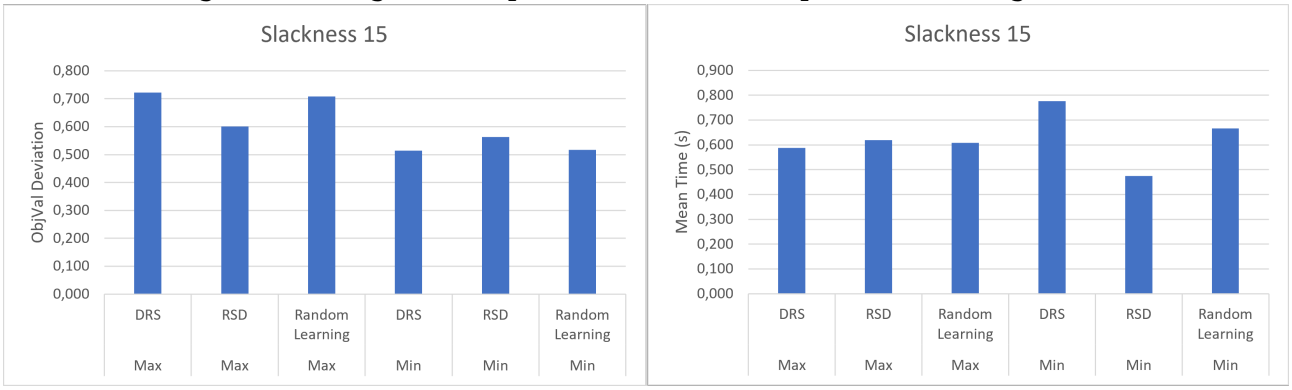


Figura 8.7: Algoritmos para instancias con puertas de holgura 20

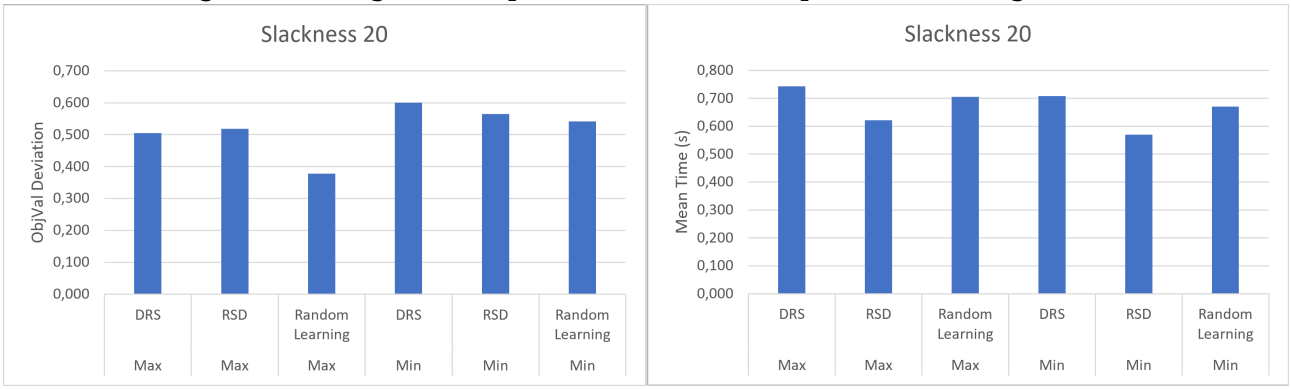
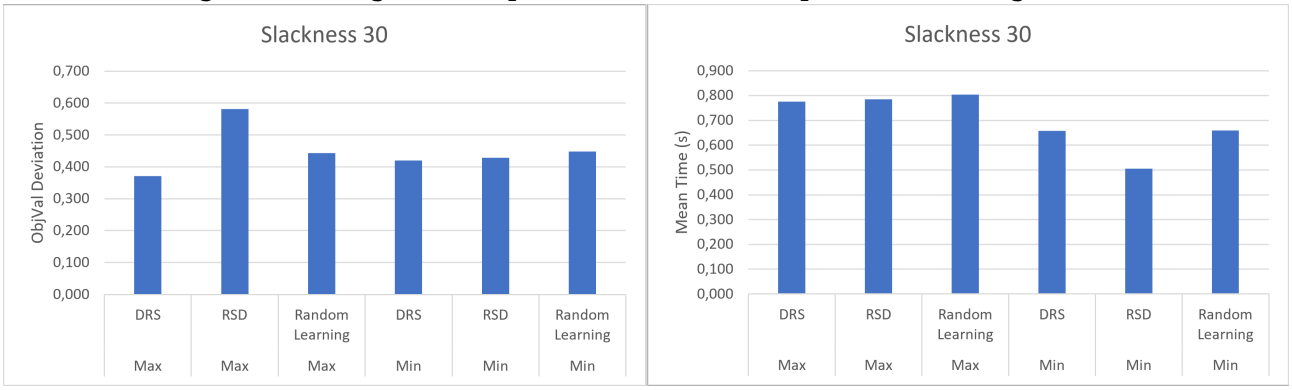


Figura 8.8: Algoritmos para instancias con puertas de holgura 30



Capítulo 9

Conclusiones y líneas de trabajo futuras

En este capítulo se presentan las conclusiones obtenidas tras realizar el trabajo de fin de grado, así como posibles maneras de ampliar éste.

9.1. Conclusiones

En este trabajo, se han diseñado e implementado un total de 6 algoritmos metaheurísticos para resolver el problema de asignación de camiones a puertas de un *cross-dock* (CDAP), de gran interés para empresas que se dedican a la distribución de mercancías. Específicamente, se han diseñado algoritmos GRASP con dos alternativas para la fase constructiva; asignación de camiones a las puertas con mayor capacidad y asignación a las de mínima capacidad, y tres alternativas para la fase de mejora, utilizando como base el algoritmo VND. Las estructuras de entorno empleadas para el VND son: relocalización de proveedores, relocalización de clientes, intercambio de proveedores, intercambio de clientes, intercambio de puertas de entrada e intercambio de puertas de salida.

Dos de los algoritmos propuestos para la fase de mejora son VND, cambiando el orden de las estructuras de entorno, con el fin de ver si este factor influye en el resultado final. Además, el último algoritmo desarrollado es una variación del VND, al que se le aplica aprendizaje automático en la selección de la estructura de entorno a utilizar en cada paso del algoritmo.

Estos 6 algoritmos logran obtener soluciones de alta calidad en un tiempo de cómputo razonable, lo cual es especialmente relevante si se cuenta con un centro de distribución complejo con varias puertas de entrada y donde intervienen múltiples proveedores y clientes. El mejor número de iteraciones entre los analizados resultó ser el de 100.000 debido a la relación entre la calidad de la solución objetivo y el tiempo de cómputo empleado.

En promedio, y para las 50 instancias analizadas, el mejor algoritmo resultó ser el que emplea la asignación de camiones a las puertas de mínima capacidad para la fase constructiva, y el VND con el orden de intercambio de puertas, relocalización de camiones e intercambio de camiones para la fase de mejora.

Por otro lado, el análisis del rendimiento de los distintos algoritmos según la holgura de las puertas muestra que es de utilidad seleccionar un algoritmo u otro atendiendo a

las especificidades de cada instancia del problema. Con valores de holgura de 5 y $30\,\%$, el algoritmo con el que se obtiene mejor valor de la función objetivo es el que utiliza para la fase constructiva la asignación a puertas de máxima capacidad (Max) y el orden DRS para el VND de la fase de mejora. Con valores de holgura 10 y $20\,\%$, se obtienen mejores resultados también con el algoritmo constructivo Max, usando el orden RSD para la fase de mejora con holgura 10 % y el algoritmo con aprendizaje automático en el caso de la holgura de 20 %. Por último, para los valores de holgura del 15 %, los mejores resultados se obtienen con el algoritmo constructivo de asignación a puertas de mínima capacidad, y el orden DRS para la fase de mejora.

Además, se generó un conjunto de instancias del CDAP, atendiendo a múltiples consideraciones, con el fin de ampliar las instancias que han servido de análisis en la literatura científica asociada al problema.

Por último, cabe destacar que se desarrolló una aplicación *web* funcional que permite representar soluciones del CDAP gráficamente, para que cualquier persona pueda visualizar y entender una posible asignación de los camiones a las puertas.

9.2. Líneas de trabajo futuras

Este trabajo presenta varias líneas futuras prometedoras que se pueden perseguir para ampliarlo, empezando por el visualizador de soluciones, que no es suficientemente completo. Esto es debido a que, en su estado actual, no se visualizan las capacidades restantes de las puertas ni la cantidad de *pallets* que envía un proveedor a un cliente. Además, se propone integrar todo el desarrollo en una aplicación web, que permita construir una instancia de CDAP de manera interactiva, seleccionar el algoritmo para resolverlo, y mostrar la solución al usuario.

Por otra parte, una posible mejora al generador de movimientos consistiría en aplicar un solo movimiento en cada llamada, añadiendo estados al generador con el fin de saber qué movimientos ha efectuado. Esto sería de utilidad si se quisiera implementar un algoritmo que no recorre todo el entorno, sino que se queda con la primera solución que mejore el valor de la función objetivo (por ejemplo, el algoritmo de búsqueda local ansioso), lo cual reduce la complejidad espacial al no tener que generar todos los movimientos.

Además, una línea de trabajo sería implementar y analizar los otros algoritmos de búsqueda local descritos, concretamente el ansioso y el de sub-entornos, con el fin de observar si ofrecen alguna mejora respecto al voraz descrito en este trabajo.

En cuanto al aprendizaje automático, dado que los resultados no fueron muy satisfactorios, se podría idear otro algoritmo o modificar los pesos con el fin de ver si se obtiene alguna mejoría notable. Además, existe la posibilidad de diseñar un algoritmo de aprendizaje automático que mantenga el conocimiento entre distintas instancias, puesto que el desarrollado en este trabajo realiza un aprendizaje único en cada instancia y se pierde en la siguiente ejecución.

Por último, se propone realizar un análisis más profundo de los algoritmos metaheurísticos descritos en este trabajo, teniendo en cuenta otros aspectos como: número de camiones, número de puertas o densidad de la matriz de flujo.

Capítulo 10

Summary and conclusions

This chapter contains the summary and conclusions that can be extracted from the final degree project. Additionally, few promising future lines of work are proposed to extend this work.

10.1. Summary

In this project, a total of 6 metaheuristic algorithms were used to solve the Cross-Docking Assignment Problem (CDAP), which is of great interest to companies involved in the distribution of goods. Specifically, GRASP algorithms with two possibilities for the constructive phase; assigning trucks to the doors with the highest capacity and assigning trucks to those with the lowest capacity, and three possibilities for the post-processing phase, using the VND algorithm as a basis were designed. The neighborhood structures used for the VND are: suppliers relocation, customers relocation, suppliers swap, customers swap, inbound doors swap and outbound doors swap.

Two of the algorithms proposed for the post-processing phase are VND, where the order of the neighborhood structures was changed in order to analyze if this factor affects the final result. The last algorithm developed is a variation of the VND that applies machine learning in the selection of the neighborhood structure.

The proposed algorithms manage to obtain high quality solutions in a reasonable computational time, which is especially relevant if there is a complex distribution centre with several doors and where multiple suppliers and customers are involved. The best number of iterations among those analysed turned out to be 100,000, due to the relationship between the quality of the solution obtained and the computational time spent.

From the 50 analyzed instances taken from Nassief et al. (2016), we can conclude that, among the 6 algorithms implemented, the best GRASP algorithm in general is the one with a constructive phase where the trucks are assigned to the minimum capacity door, and a post-processing phase where a Variable Neighbourhood Descent algorithm is performed in the order doors swap, trucks relocation and trucks swap.

On the other hand, the analysis of the performance of the different algorithms according to the slackness of the doors shows that it is useful to select an algorithm according to the specificities of each instance of the problem. With slackness values of 5 and 30 %,

the algorithm with the best value of the objective function is the one that uses the assignment to doors of maximum capacity (Max) for the constructive phase, and the DRS order of the VND for the post-processing phase. With slackness values of 10 and 20 %, better results are also obtained with the Max constructive algorithm, using the RSD order for the post-processing phase with 10 % slackness, and the algorithm with machine learning in the case of 20 % slackness. Finally, for slackness values of 15 %, the best results are obtained with the constructive algorithm of assigning the trucks to the minimum capacity door, and the DRS order for the post-processing phase.

In addition, a set of CDAP instances was generated, based on multiple considerations to extend the ones that have been analyzed in the scientific literature associated with the problem.

Additionally, a functional web application was developed to represent CDAP solutions graphically, so that anyone can visualize and understand a possible assignment of trucks to the corresponding doors.

10.2. Future work

This work presents several future lines of work that can be pursued to extend it, starting with the solution viewer, which can be completed. This is because, in its current state, it does not visualise the remaining capacities of the doors or the amount of pallets that a provider sends to a client. Furthermore, it is proposed to integrate the whole development into a web application, which would allow to build an instance of CDAP interactively, select the algorithm to solve it, and display the solution to the user.

On the other hand, a possible improvement to the movement generator would be to apply a single movement in each call, adding states to the generator in order to know which movements it has already performed. This would be useful if there was an algorithm that does not traverse the entire environment, but instead takes the first solution that improves the value of the objective function (e.g., the anxious local search algorithm), which reduces the spatial complexity by not having to generate all the moves.

In addition, one line of work would be to implement and analyse the other local search algorithms described, namely the anxious and the sub-environment algorithms, in order to see if they offer any improvement over the greedy one described in this paper.

As for machine learning, given that the results were not very satisfactory, another algorithm could be designed or the weights could be modified in order to see if any noticeable improvement is obtained. In addition, there is the possibility of designing a machine learning algorithm that maintains the knowledge between different instances, since the one developed in this work performs a unique learning in each instance and is lost in the next execution.

Finally, it is proposed to carry out a deeper analysis of the metaheuristic algorithms described in this work, taking into account other aspects such as: number of trucks, number of doors or density of the flow matrix.

Capítulo 11

Presupuesto

El presupuesto de este trabajo consta principalmente del coste de personal, teniendo en cuenta el número de horas requeridas para el desarrollo del mismo. El cálculo del tiempo se realiza en base a la duración del proyecto; 300 horas, acorde con lo establecido en la Resolución de 21 de marzo de 2011, de la Universidad de La Laguna, por la que se publica el plan de estudios de Graduado en Ingeniería Informática, donde se establece una duración de 12 créditos ECTS para el Trabajo de Fin de Grado. Un crédito ECTS equivale a 25 horas de trabajo; *25 * 15 = 300*.

El coste total de personal equivale a 4800€, asumiendo un coste por hora de 16€.

Bibliografía

L. Sparks, “The changing structure of distribution in retail companies: An example from the grocery trade,” *Transactions of the Institute of British Geographers*, vol. 11, no. 2, p. 147, 1986.

M. R. Galbreth, J. A. Hill, and S. Handley, “An investigation of the value of cross-docking for supply chain management,” *Journal of Business Logistics*, vol. 29, no. 1, p. 225–239, 2008.

L. Okdinawati, T. M. Simatupang, and Y. Sunitiyoso, “Modelling collaborative transportation management: Current state and opportunities for future research,” *Journal of Operations and Supply Chain Management (JOSCM)*, vol. 8, no. 2, pp. 96–119, 2015.

D. Agustina, C. Lee, and R. Piplani, “A review: Mathematical models for cross docking planning,” *International Journal of Engineering Business Management*, vol. 2, no. 2, pp. 47–54, 2010.

N. Boysen and M. Fliedner, “Cross dock scheduling: Classification, literature review and research agenda,” *Omega*, vol. 38, no. 6, pp. 413–422, 2010.

J. Van Belle, P. Valckenaers, and D. Cattrysse, “Cross-docking: State of the art,” *Omega*, vol. 40, no. 6, pp. 827–846, 2012.

E. González-La Rotta and M. Becerra-Fernández, “Cross-docking with vehicle routing problem. a state of art review [plataformas de intercambio con ruteo de vehículos. una revisión del estado del arte],” *DYNA (Colombia)*, vol. 84, no. 200, pp. 271–280, 2017.

R. Mavi, M. Goh, N. Mavi, F. Jie, K. Brown, S. Biermann, and A. Khanfar, “Cross-docking: A systematic literature review,” *Sustainability (Switzerland)*, vol. 12, no. 11, 2020.

J. Bartholdi III and K. Gue, “The best shape for a crossdock,” *Transportation Science*, vol. 38, no. 2, pp. 235–244, 2004.

W. Nassief, I. Contreras, and R. As’ad, “A mixed-integer programming formulation and lagrangean relaxation for the cross-dock door assignment problem,” *International Journal of Production Research*, vol. 54, no. 2, pp. 494–508, 2016.

O. Guemri, P. Nduwayo, R. Todosijević, S. Hanafi, and F. Glover, “Probabilistic tabu search for the cross-docking assignment problem,” *European Journal of Operational Research*, vol. 277, 03 2019.

S. Gelareh, F. Glover, O. Guemri, S. Hanafi, P. Nduwayo, and R. Todosijević, “A comparative study of formulations for a cross-dock door assignment problem,” *Omega*, vol. 91, p. 102015, 2020.

S. I. Sayed, I. Contreras, J. A. Diaz, and D. E. Luna, “Integrated cross-dock door assignment and truck scheduling with handling times,” *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, vol. 28, no. 3, pp. 705–727, 2020.

C. Blum and A. Roli, “Metaheuristics in combinatorial optimization,” *ACM Computing Surveys*, vol. 35, no. 3, p. 268–308, 2003.

C. S. Sung and W. Yang, “An exact algorithm for a cross-docking supply chain network design problem,” *Journal of the Operational Research Society*, vol. 59, no. 1, pp. 119–136, 2008. [Online]. Available: https://doi.org/10.1057/palgrave.jors.2602328

N. Boysen, D. Briskorn, and M. Tschöke, “Truck scheduling in cross-docking terminals with fixed outbound departures,” *OR Spectrum*, vol. 35, no. 2, p. 479–504, 2012.

F. Chen and C.-Y. Lee, “Minimizing the makespan in a two-machine cross-docking flow shop problem,” *European Journal of Operational Research*, vol. 193, no. 1, p. 59–72, 2009.

M. Wen, J. Larsen, J. Clausen, J.-F. Cordeau, and G. Laporte, “Vehicle routing with cross-docking,” *Journal of the Operational Research Society*, vol. 60, no. 12, p. 1708–1718, 2009.

A.-L. Ladier and G. Alpan, “Cross-docking operations: Current research versus industry practice,” *Omega (United Kingdom)*, vol. 62, pp. 145–162, 2016.

A. Rijal, M. Bijvank, and R. de Koster, “Integrated scheduling and assignment of trucks at unit-load cross-dock terminals with mixed service mode dock doors,” *European Journal of Operational Research*, vol. 278, no. 3, pp. 752–771, 2019.

“Ieee standard for information technology–portable operating system interface (posix(tm)) base specifications, issue 7,” *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*, pp. 213–218, 2018.

W. Michiels, E. H. L. Aarts, and J. Korst, *Theory of Local Search*. Cham: Springer International Publishing, 2018, pp. 299–339.

T. A. Feo and M. G. Resende, “Greedy randomized adaptive search procedures,” *Journal of Global Optimization*, vol. 6, no. 2, p. 109–133, 1995.

M. Guignard, P. M. Hahn, A. A. Pessoa, and D. C. da Silva, “Algorithms for the cross-dock door assignment problem,” in *Proceedings of the fourth international workshop on model-based metaheuristics*, 2012.