Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Herramienta de visualización de indicadores y soluciones del Cross-Dock Door Assignment Problem (CDAP)

Cross-Dock Door Assignment Problem (CDAP) indicators and solutions visualization tool

Karina Kalwani Israni

La Laguna, 13 de julio de 2023

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. **Francisco Román Ángel-Bello Acosta**, con N.I.F. 54.951.451-N contratado María Zambrano 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:

"Herramienta de visualización de indicadores y soluciones del Cross-Dock Door Assignment Problem (CDAP)"

ha sido realizada bajo su dirección por Dña. **Karina Kalwani Israni**, con N.I.F. 55.501.769-D.

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 julio de 2023

Agradecimientos

A mi tutora Belén, por proponer este tema de trabajo y por su orientación y apoyo constante. Este trabajo hubiera sido muy complicado sin su ayuda, ya sea por los materiales proporcionados o por las aportaciones realizadas.

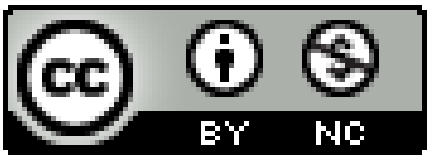
Al co-tutor Francisco, por proporcionar los materiales necesarios, relacionados principalmente con los resultados computacionales que se utilizan en el presente trabajo.

A todos y cada uno de los profesores que han impartido asignaturas en la carrera, ya que los conocimientos y experiencias proporcionadas por ellos me ayudaron bastante en desarrollar habilidades necesarias para este trabajo.

A mis amigos por regalarme esta experiencia universitaria inolvidable.

Y por último (pero no menos importante) a mis padres y mi familia por siempre estar a mi lado y apoyarme en todo lo que he hecho en mi vida.

Licencia



Resumen

Un centro cross-dock es una instalación logística que se encarga de transferir la mercancía proveniente de un camión de entrada a otro camión de salida, sin requerir ningún almacenamiento intermedio. Existen varios problemas asociados a esta estrategia, uno de ellos es el de la asignación óptima de camiones a puertas del centro cross-dock, de tal manera que el coste interno sea mínimo. El objetivo del presente proyecto es desarrollar una herramienta que permita analizar los algoritmos que resuelven este problema, así como la calidad de los resultados generados. Para ello, se van a usar varias métricas asociadas a los algoritmos y las soluciones. Se pretende que la herramienta proporcione información valiosa para la toma de decisiones logísticas en centros cross-dock.

Palabras clave: cross-docking, logística, estrategia, optimización, centro de distribución, análisis, algoritmos

Abstract

A cross-dock facility is a logistics installation that transfers goods directly from an incoming truck to an outgoing truck, without requiring any intermediate storage. There are several problems associated with this strategy, one of which is the Cross-Dock Door Assignment Problem (CDAP), which has an aim of minimizing internal costs. The objective of this project is to develop a tool for analyzing the algorithms that solve this problem, as well as the quality of the generated results. Various metrics related to the algorithms and solutions will be used. The tool aims to provide valuable information for logistical decision-making in cross-dock centers.

Keywords: cross-docking, logistics, strategy, optimization, distribution center, analysis, algorithms

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En España, el sector logístico está en continuo crecimiento. Según el Instituto Nacional de Estadística (INE), el sector de transporte y almacenamiento creció un 18.9 % en el año 2021 con respecto al año 2020, donde el sector de transporte de mercancías por carretera aumentó un 10 % del 2020 al 2021. Una de las razones de este crecimiento podría ser el aumento del uso de las plataformas de comercio electrónico (e-commerce), que cada día están ganando más popularidad y se están convirtiendo en la opción preferida de los compradores (Coppola (2021)).

Con el incremento del volumen de ventas del *e-commerce* y, consecuentemente, la competencia en este sector, es muy importante usar una estrategia lógistica que ayude a entregar la mercancía al consumidor en tiempo y forma, ya que hoy en día muchas empresas están prometiendo a sus consumidores entrega de productos comprados al día siguiente o, incluso, el mismo día, siendo ésta una estrategia competitiva (Boysen et al. (2019)).

Tradicionalmente se usaba la estrategia de *warehousing*, es decir, se recibía la mercancía y se guardaba en un almacén (*warehouse* en inglés) y, cuando un cliente realizaba un pedido, se recogían los productos para cargarlos en un camión. Este proceso se puede resumir en cuatro pasos: recepción, almacenamiento, preparación de pedidos y transporte, siendo el almacenamiento y la preparación de pedidos los pasos más costosos (Van Belle et al. (2012)).

El problema anterior se podría resolver usando, como alternativa al *warehousing*, la estrategia *cross-docking*. En esta alternativa, los productos son enviados a los clientes el mismo día, acortando así el tiempo de entrega, el costo de almacenamiento y de la mano de obra.

En un centro de *cross-docking* se encuentran, por un lado las puertas de entrada y, en el lado opuesto, las puertas de salida. La estrategia se puede resumir en los siguientes pasos:

- **Recepción**: Un camión cargado de mercancía llega al centro y se posiciona en la puerta de entrada que se le ha asignado.
- **Descarga y Carga**: Se descargan las mercancías del camión de entrada, se ordenan según sus destinos e inmediatamente se transfieren a los camiones de salida para

ser transportadas a sus respectivos destinos.

- **Envío**: Los camiones de salida, posicionados en las puertas de salida asignadas, reciben mercancía de uno o varios camiones de entrada y proceden a su siguiente destino.

La diferencia entre el *warehousing* y el *cross-docking* se encuentra en el manejo de la mercancía. En el primer caso, se debe almacenar en el centro y recuperar cuando el cliente realiza un pedido, mientras que en el segundo no se require el almacenamiento previo de la mercancía, sino que es inmediatamente transferida a los camiones de salida. Por lo tanto, se podría interpretar a un centro *cross-docking* como una zona de tránsito (Nduwayo (2020)). A veces algunos productos pueden requerir un tiempo mínimo de almacenamiento, pero éste suele ser menos de 24 horas. Esto ayuda a reducir el tiempo de entrega y el coste de inventario.

En la literatura, se han estudiado varios tipos de problemas relacionados con la estrategia de *cross-docking*, como por ejemplo la ubicación y la forma del centro, la capacidad y diseño del área de descarga/carga, etc (Buijs et al. (2014)). Uno de estos problemas es el de la asignación de camiones a puertas, conocido como *Cross-dock Door Assignment Problem* , o *CDAP* (Tsui and Chang (1990), Zhu et al. (2009), Nduwayo (2020)).

1.1. Objetivos

El objetivo global de este trabajo es diseñar una herramienta de visualización para los resultados computacionales del problema de CDAP, que permita al decisor tomar decisiones frente a un conjunto de algortimos posibles. Para ello, se considerarán los siguientes objetivos parciales:

1. Visualizar y analizar los resultados computacionales proporcionados por un conjunto de algoritmos existentes para la resolución del CDAP.
2. Visualizar y analizar los resultados obtenidos por la simulación realizada, mediante la cual se determina la factibilidad de las soluciones cuando la capacidad de las puertas se modifica.
3. Generar un visualizador de soluciones que permita comprender la estructura de las soluciones de mayor calidad.

Para la resolución de este problema existen varios algoritmos, entre ellos están, por un lado los algoritmos exactos, y por el otro lado los algoritmos metaheurísticos (Pentico (2007)), (Yamada and Nasu (2000)). En este trabajo se han utilizado los algoritmos basados en búsquedas por entornos variables (Hansen et al. (2010)) que han sido diseñados e implementados por investigadores del Grupos de Computación Inteligente y Ciencia de Datos de la Universidad de La Laguna.

Con los algoritmos exactos se garantiza la obtención de una solución óptima para el problema. Si no existe una solución factible para una instancia, se puede demostrar

usando un algoritmo exacto. Sin embargo, estos algoritmos requieren un tiempo de cómputo bastante elevado, así como un consumo alto de memoria (Fakhravar (2022)).

Una alternativa a los algoritmos exactos son los algoritmos metaheurísticos, que, aunque no garantizan la obtención de una solución óptima, resuelven el problema en un tiempo de cómputo inferior.

En este trabajo se van a analizar los resultados obtenidos por estos algoritmos a través de métricas relacionadas con las soluciones generadas, así como la factibilidad de éstas. Además, se va a desarrollar una aplicación web que permita ver las soluciones de una manera visual, ya que esto simplifica el análisis de las soluciones obtenidas.

1.2. Revisión de Literatura

La estrategia de *cross-docking*, aunque tenga varias ventajas, no siempre es una estrategia logística óptima ya que depende del producto tratado y su demanda. En Apte and Viswanathan (2000), los autores consideran dos factores clave para determinar la estrategia a implementar: la ratio de la demanda del producto y el costo unitario de desabastecimiento. Si la demanda del producto es estable, es más fácil manejar el flujo de la mercancía; mientras que el costo unitario de desabastecimiento es preferible que sea menor ya que en el *cross-docking* la posibilidad de que un producto se agote es alta. En la figura 1.1, proporcionada por Van Belle et al. (2012) se resume esta información.

Unit stock-out costs	*Product demand rate*: Stable and constant	Unstable or fluctuating
High	Cross-docking can be implemented with proper systems and planning tools	Traditional warehousing preferred
Low	Cross-docking preferred	Cross-docking can be implemented with proper systems and planning tools

Figura 1.1: Adaptación óptima del cross-docking (Van Belle et al. (2012))

En la literatura existen muchas definiciones de una implementación efectiva de esta estrategia, una de ellas es la creación de una red de *cross-docks*. En Buijs et al. (2014), los autores explican la importancia de incluir la definición de una red *cross-docking* amplia, donde consideran dos tipos de operaciones: *cross-docking* local y a nivel de red. En las operaciones de *cross-docking* local se incluyen todas aquellas operaciones que se realizan en un centro, mientras que en las de nivel de red se incluyen todas aquellas que se realizan en otro lugar dentro de la red. Los autores definen una red *cross-docking* como un “subsistema de una cadena de suministro formado por uno o más *cross-docks*, sus rutas de transporte entrantes y salientes, y los stakeholders conectados a los *cross-docks* a través de esas rutas” (Buijs et al. (2014)). En este caso los stakeholders podrían ser proveedores, clientes, otros centros de distribución, etc.

Hay varios problemas de optimización que se deben resolver para una implementación efectiva de la estrategia de *cross-docking*. Estos problemas normalmente se dividen en tres categorías: estratégicios, tácticos y operacionales. En este trabajo se van a analizar

los algoritmos que resuelven un problema operacional, conocido como el Cross-dock Door Assignment Problem (Problema de asignación de camiones a puertas).

Existen varias soluciones para el problema CDAP. En Tsui and Chang (1990) se propone una solución que tiene como restricción la asignación de solamente un camión a una puerta de entrada. Luego en Zhu et al. (2009) los autores ampliaron la solución anterior permitiendo la asignación de más de un camión a una puerta, pero en este caso usando como restricción la capacidad de las puertas. Hay muchas soluciones disponibles para este problema (Tarhini et al. (2016); Nassief et al. (2016); Guemri et al. (2019)).

Algunos de los trabajos actuales tratan varios aspectos relacionados con el CDAP. En Wang and Alidaee (2019) se describe una solución planteada para una variante del CDAP, el Multi-floor Cross-dock Door Assignment Problem (MCDAP), es decir, el problema de asignación de camiones a puertas para centros *cross-dock* de varias plantas, que requiere una planificación mucho más complicada, ya que en este caso se deben usar ascensores u otro instrumento para transportar la mercancía entre plantas.

En la mayoría de los estudios se han considerado entornos deterministas, es decir, donde se asume un conocimiento preciso y perfecto de los parámetros y restricciones del problema, pero esto no es posible en casos de *cross-docking* reales, ya que hay diversos factores que generan incertidumbre en los parámetros del problema. Estos factores pueden ser congestión de tráfico, fallos técnicos en los camiones, avería del transpaleta y otros fallos imprevistos que complican la predicción de la hora de llegada y salida de los camiones, o el tiempo de servicio. En el trabajo Essghaier et al. (2021) los autores proponen soluciones para el CDAP, tanto para entornos deterministas como para tratar las incertidumbres asociadas, que luego son probadas para confirmar el efecto positivo de éstas.

La presente memoria está organizada de la siguiente manera. El capítulo 2 describe detalladamente el funcionamiento de un centro *cross-dock* y el problema de asignación de camiones a puertas, así como dos formulaciones matemáticas que resuelven el problema. En el capítulo 3 se analizan los resultados obtenidos por los algoritmos que resuelven el problema CDAP, teniendo en cuenta el tamaño del problema y el número de iteraciones. El capítulo 4 realiza un análisis de factibilidad de las soluciones generadas en un proceso de simulación, cuyos resultados se encuentran en una serie de ficheros de texto. El capítulo 5 describe la aplicación web creada para visualizar las soluciones de una manera gráfica. En el capítulo 6 se presenta una breve conclusión al trabajo, así como una propuesta de líneas futuras. En el capítulo 7 se incluye un resumen extendido y las conclusiones en inglés. Finalmente, el capítulo 8 ofrece un presupuesto asociado a este trabajo.

Capítulo 2

Descripción del Cross-Dock Door Assignment Problem

Para entender mejor el problema de asignación de camiones a puertas, primero se debe explicar con más detalles cómo funciona la estrategia de *cross-docking* en un centro *cross-dock*.

Muchas veces, la estructura física de una terminal *cross-dock* suele ser similar a la de un rectángulo, larga y estrecha (en la literatura se le suele denominar "*terminal en forma de I*"), pero también existen otras formas (L, T, X, etc) (Bartholdi and Gue (2004)).

En el centro, se encuentran disponibles múltiples puertas, que se clasifican en dos categorías: puertas de entrada y puertas de salida, ambas ubicadas en extremos opuestos de la terminal. Cuando los camiones cargados llegan al centro, se les asigna una puerta de entrada específica donde proceden a descargar la mercancía. Luego, con la ayuda de equipamiento especializado, como carretillas elevadoras o transpaletas, se traslada la mercancía a los camiones de salida ubicados en las puertas correspondientes. Finalmente, se carga la mercancía en los camiones para que sea transportada a su próximo destino.

Este proceso no requiere una infraestructura especial, ya que normalmente los productos no se almacenan para un largo periodo de tiempo (menos de 24 horas, como se mencionó en el capítulo anterior).

Teniendo el cuenta este funcionamiento, el Cross-dock Door Assignment Problem (CDAP) consiste en asignar los camiones de entrada a puertas de entrada, y los camiones de salida a puertas de salida. Recibe como parámetro el conjunto de camiones de entrada y salida, y tiene como objetivo lograr una asignación óptima de camiones a puertas, de tal manera que se minimice el coste de manipulación de materiales (distancia total recorrida de los dispositivos utilizados para el transporte interno de mercancías, es decir, entre las puertas de entrada y salida).

Para que una solución se considere factible, debe satisfacer una o varias restricciones, que normalmente suelen ser relacionadas a la asignación y/o capacidad de las puertas.

2.1. Formulaciones Matemáticas

Para resolver el problema de asignación de camiones a puertas existen varias formulaciones matemáticas. A continuación, se presentan dos de ellas, siendo la primera propuesta por Tsui and Chang (1990) y la segunda por Zhu et al. (2009) (extensión de Tsui and Chang (1990), como se explicó en el apartado 1.2).

2.1.1. Formulación Cross-dock Door Assignment Problem de Tsui y Chang

Esta formulación fue propuesta en Tsui and Chang (1990).

- **Parámetros**:
  - M : Numero de proveedores (camiones de entrada)
  - N : Número de clientes (camiones de salida)
  - I : Número de puertas de entrada
  - J : Número de puertas de salida
  - w_{mn} : Número de viajes requeridos por la transpaleta (u otro dispositivo de transporte interno de productos) para llevar la mercancía del proveedor m al cliente n
  - d_{ij} : Distancia entre la puerta de entrada i y la puerta de salida j
- **Variables de Decisión**:
  - $x_{mi} = 1$ si el proveedor m es asignado a la puerta de entrada i, en otro caso $x_{mi} = 0$
  - $y_{nj} = 1$ si el cliente n es asignado a la puerta de salida j, en otro caso $y_{nj} = 0$
- **Formulación**:

$$\min_{z} \quad \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{m=1}^{M}\sum_{n=1}^{N} d_{ij} w_{mn} x_{mi} y_{nj} \tag{2.1a}$$

$$\text{s.t.} \quad \sum_{m=1}^{M} x_{mi} = 1 \qquad i = 1, \ldots, I, \tag{2.1b}$$

$$\sum_{i=1}^{I} x_{mi} = 1 \qquad m = 1, \ldots, M, \tag{2.1c}$$

$$\sum_{n=1}^{N} y_{nj} = 1 \qquad j = 1, \ldots, J, \tag{2.1d}$$

$$\sum_{j=1}^{J} y_{nj} = 1 \qquad n = 1, \ldots, N, \tag{2.1e}$$

$$x_{mi} = 0 \quad or \quad 1 \qquad m = 1, \ldots, M; \quad i = 1, \ldots, I, \tag{2.1f}$$

$$y_{nj} = 0 \quad or \quad 1 \qquad n = 1, \ldots, N; \quad j = 1, \ldots, J \tag{2.1g}$$

En 2.1b se está asegurando que cada puerta de entrada tiene asignado solamente un camión proveedor, y 2.1c obliga que cada camión proveedor esté asignado a solamente una puerta de entrada. De la misma forma, la restricción 2.1d asegura que cada puerta de salida tenga asignada solamente un camión cliente, y en 2.1e se obliga que cada cliente esté asignado a solamente una puerta de salida.

En esta formulación matemática se está obligando a que una puerta pueda servir solamente a un camión, es decir, el número de proveedores M debe ser igual que el número de puertas de entrada I ($M = I$), y el número de clientes N debe ser igual que el número de puertas de salida J ($N = J$), para poder satisfacer las restricciones planteadas. Con este planteamiento se podría concluir que la solución no es muy práctica, ya que no se puede usar en aquellos casos en los que el número de camiones sea superior al número de puertas.

2.1.2. Formulación Cross-dock Door Assignment Problem de Zhu

En Zhu et al. (2009) los autores proponen una solución que resuelve el problema anterior usando como restricción la capacidad de las puertas, y permitiendo la asignación de más de un camión a una puerta.

- **Parámetros**:
  - M : Numero de proveedores (camiones de entrada)
  - N : Número de clientes (camiones de salida)
  - I : Número de puertas de entrada
  - J : Número de puertas de salida
  - w_{mn} : Número de viajes requeridos por la transpaleta (u otro dispositivo de transporte interno de productos) para llevar la mercancía del proveedor m al cliente n
  - d_{ij} : Distancia entre la puerta de entrada i y la puerta de salida j
  - s_m : Volumen de mercancía desde el proveedor m
  - S_i : Capacidad de la puerta de entrada i
  - r_n : Demanda de mercancía del cliente n
  - R_j : Capacidad de la puerta de salida j
- **Variables de Decisión**:
  - $x_{mi} = 1$ si el proveedor m es asignado a la puerta de entrada i, en otro caso $x_{mi} = 0$
  - $y_{nj} = 1$ si el cliente n es asignado a la puerta de salida j, en otro caso $y_{nj} = 0$

- **Formulación**:

$$\min_{z} \quad \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{m=1}^{M}\sum_{n=1}^{N} d_{ij} w_{mn} x_{mi} y_{nj} \tag{2.2a}$$

$$\text{s.t.} \quad \sum_{m=1}^{M} s_m x_{mi} \leq S_i \qquad i = 1, \ldots, I, \tag{2.2b}$$

$$\sum_{i=1}^{I} x_{mi} = 1 \qquad m = 1, \ldots, M, \tag{2.2c}$$

$$\sum_{n=1}^{N} r_n y_{nj} \leq R_j \qquad j = 1, \ldots, J, \tag{2.2d}$$

$$\sum_{j=1}^{J} y_{nj} = 1 \qquad n = 1, \ldots, N, \tag{2.2e}$$

$$x_{mi} = 0 \quad or \quad 1 \qquad m = 1, \ldots, M; \quad i = 1, \ldots, I, \tag{2.2f}$$

$$y_{nj} = 0 \quad or \quad 1 \qquad n = 1, \ldots, N; \quad j = 1, \ldots, J \tag{2.2g}$$

En este caso, con 2.2b se asegura que el volumen de mercancía del camión proveedor m (s_m) no exceda de la capacidad máxima de la puerta de entrada i (S_i), 2.2d hace lo mismo para el camión cliente n (r_n) en la puerta de salida j (R_j). Por otro lado, las restricciones 2.2c y 2.2e garantizan que un camión de entrada (o salida) esté asignado solamente a una puerta de entrada (o salida), aunque a una misma puerta sí pueden estar asignados varios camiones.

2.2. Conclusión

En el presente capítulo se explicó detalladamente el funcionamiento interno de un centro *cross-dock* y el problema de asignación de camiones a puertas (Cross-dock Door Assignment Problem) asociado a éste.

Además, se presentaron dos formulaciones matemáticas para el CDAP: la formulación propuesta por Tsui y Chang en Tsui and Chang (1990) y la propuesta por Zhu en Zhu et al. (2009). La formulación de Tsui y Chang considera que cada puerta podrá servir solamente a un camión, con lo cual ésta será aplicable solamente cuando el número de camiones sea igual al número de puertas. Por otro lado, la formulación de Zhu permite la asignación de más de un camión a una puerta, y utiliza como restricción la capacidad de las puertas. Estas formulaciones son de gran importancia para desarrollar algoritmos que permitan encontrar soluciones eficientes para este problema.

Capítulo 3

Análisis de Algoritmos

Durante los años, diversos autores han desarrollado algoritmos para resolver el problema del CDAP, y muchos de ellos han usado instancias de prueba para valorar la efectividad de las soluciones (Zhu et al. (2009), Guignard et al. (2012), Guemri et al. (2019)).

Como se explicó en el apartado 1.1, hay dos tipos de algoritmos que resuelven este problema: algoritmos exactos y metaheurísticos. Los algoritmos exactos garantizan la obtención de una solución óptima, pero tardan bastante en calcularla; mientras que con los metaheurísticos se puede obtener una solución en un tiempo menor, pero no se puede asegurar la optimalidad de ésta.

En este apartado, se van a analizar los resultados computacionales obtenidos por cuatro algoritmos metaheurísticos basados en multiarranque: Random Variable Neighborhood Descent ($C3RVND$), Composite Local Search ($C3CLS$), Modified Variable Neighborhood Descent ($C3ModVND$), y Original Variable Neighborhood Descent ($C3OrVND$).

Específicamente, se hará uso de cuatro algoritmos multiarranque, para lo que se generaron tres grupos de soluciones factibles iniciales, formados por $2 \times M \times I$, $4 \times M \times I$ y $8 \times M \times I$ soluciones, respectivamente, donde M es el número de camiones e I es el número de puertas. Concretamente, en primer lugar, se generaron $K_{max} = M \times I$ asignaciones para cada lado del centro de cross-docking (puertas de entrada/puertas de salida). La mitad de las asignaciones de puertas para cada lado, correspondientes a cualquiera de los proveedores o clientes, se obtuvieron con un algoritmo constructivo, que sólo tiene en cuenta un lado del cross-dock (entrada/salida) y asigna camiones a la puerta con mayor capacidad de forma prioritaria, mientras que la otra mitad se obtuvieron con un algoritmo constructivo que también tiene en cuenta un lado del cross-dock, pero asigna camiones a la puerta con menor capacidad en primer lugar. Luego, cada asignación lateral se utiliza como entrada para el tercer algoritmo constructivo, que produce una solución inicial completa. Por lo tanto, se generó un grupo de $2 \times M \times I$ soluciones para llevar a cabo la ejecución de los cuatro algoritmos multiarranque. El proceso se repite considerando $K_{max} = 2 \times M \times I$ y $K_{max} = 4 \times M \times I$, para obtener los tres conjuntos de soluciones iniciales para ejecutar los cuatro algoritmos.

Se usarán como métricas el tamaño del problema, la solución obtenida, el tiempo de cómputo y la diferencia entre el valor objetivo de la solución obtenida y la óptima.

Para ello, se ha decidido utilizar la herramienta *Microsoft Power BI*. Se trata de una

plataforma de análisis y visualización de datos con unas ventajas que la convierten en una herramienta apta para este caso. Una de estas ventajas es que permite conectar múltiples orígenes de datos para crear visualizaciones. Dichas visualizaciones pueden ser de cualquier tipo: gráfico de barras, gráfico de evolución,...., además de otros elementos que ayudan a crear un entorno visual personalizable. Las gráficas que se muestran en este apartado también se pueden visualizar a través de este enlace.

3.1. Resultados Obtenidos por los Algoritmos

Los resultados obtenidos por los distintos algoritmos se encuentran en un fichero con extensión *.xlsx*, es decir, en una hoja de cálculo de *Microsoft Excel*, en la que se resumen.

En dicha hoja de cálculo, se encuentran los valores generados por distintos algoritmos (suma de las distancias recorridas por los transpaletas para llevar toda la mercancía de los proveedores a los clientes), así como el tiempo de cómputo y la diferencia entre el valor generado y el óptimo, para cada problema (instancia). En total hay 44 instancias con sus respectivas soluciones obtenidas por los algoritmos. Cada una de éstas están basadas en tres datos:

- **Número de proveedores/clientes**: El número de camiones de entrada y salida.
- **Número de puertas**: El número de puertas de entrada y salida que dispone el centro *cross-dock*.
- **Porcentaje de Holgura**: Representa el porcentaje de capacidad adicional que se suma a la capacidad mínima de cada puerta, obtenida como la división entre la suma total del número de pallets a enviar y el número de puertas disponibles.

Un ejemplo del nombre de instancia es *12x5S10*, que tiene el siguiente significado:

- 12 camiones de proveedores y 12 de clientes.
- 5 puertas de entrada y 5 de salida.
- Porcentaje de holgura 10, es decir, un $10\,\%$ de capacidad adicional.

Las datos utilizados para generar gráficas en *Microsoft Power BI* y, con ello, analizar los algoritmos, provienen de la misma hoja de cálculo.

Para crear las gráficas se han tenido en cuenta distintos parámetros: primero se analizan los algoritmos metaheurísticos que proporcionan una solución para un tamaño de problema en concreto, luego se analiza lo mismo pero teniendo en cuenta también el número de iteraciones, y finalmente se intenta analizar la eficiencia general de los algoritmos para cualquier tamaño de problema.

3.2. Algoritmos Según Tamaño del Problema

El proceso de búsqueda de una solución, en un algoritmo metaheurístico consiste en generar una solución inicial y luego, iterativamente, mejorar la calidad de las soluciones calculadas previamente. Para ello, es necesario establecer una condición de parada, que normalmente suele ser un número máximo de iteraciones, o un tiempo máximo de cómputo (Alancay et al. (2016)). En este caso se ha establecido como condición de parada un número máximo de iteraciones, denotado como K_{max}, que representa el número de soluciones iniciales generadas para la ejecución de los algoritmos multiarranque.

Este parámetro es muy crucial, ya que determinará cuántas veces iterará el algoritmo. Si se elige un valor grande, hay más posibilidades de obtener una solución óptima, pero también aumenta el tiempo de cómputo. Por lo tanto, es importante encontrar un equilibrio entre un valor K_{max} lo suficientemente grande para encontrar una solución de alta calidad y un tiempo de cómputo aceptable.

3.2.1. Análisis Valor

En este apartado se va a analizar la calidad de la solución proporcionada por un algoritmo multiarranque, teniendo en cuenta la condición de parada K_{max} y el tamaño del problema.

Como se explicó anteriormente, los algoritmos metaheurísticos no garantizan la obtención de una solución óptima. Por lo tanto, para analizar la calidad de la solución de un problema dado, se va a tener en cuenta el valor óptimo calculado por un algoritmo exacto para ese mismo problema.

Una cosa que cabe destacar en este caso es que, el porcentaje de holgura que se explicó anteriormente no se va a tener en cuenta para el análisis de los algoritmos, ya que el factor que va a influir más en las soluciones será el tamaño del problema, que estará definido por dos valores:

- El número de camiones de proveedores y clientes (en este caso, se han considerado 8, 9, 10, 11, 12 y 15.
- El número de puertas de entrada y salida (en este caso, se han considerado 4, 5, 6 y 7.

Tamaño: 9x4	**M*I**	**2*M*I**	**4*M*I**
C3CLS	5979.6	5978.2	5978.2
C3RVND	5978.2	5978.2	5978.2
C3ModVND	5978.2	5978.2	5978.2
C3OrVND	5982.2	5982.2	5981.2
OptVal	5978.2	5978.2	5978.2

Tabla 3.1: Valores obtenidos para el problema de tamaño 9x4

En la figura 3.1 se pueden apreciar los resultados obtenidos por los algoritmos *C3CLS*, *C3RVND*, *C3ModVND* y *C3OrVND*, para un problema de tamaño *9x4*, con $K_{max} = M * I$,

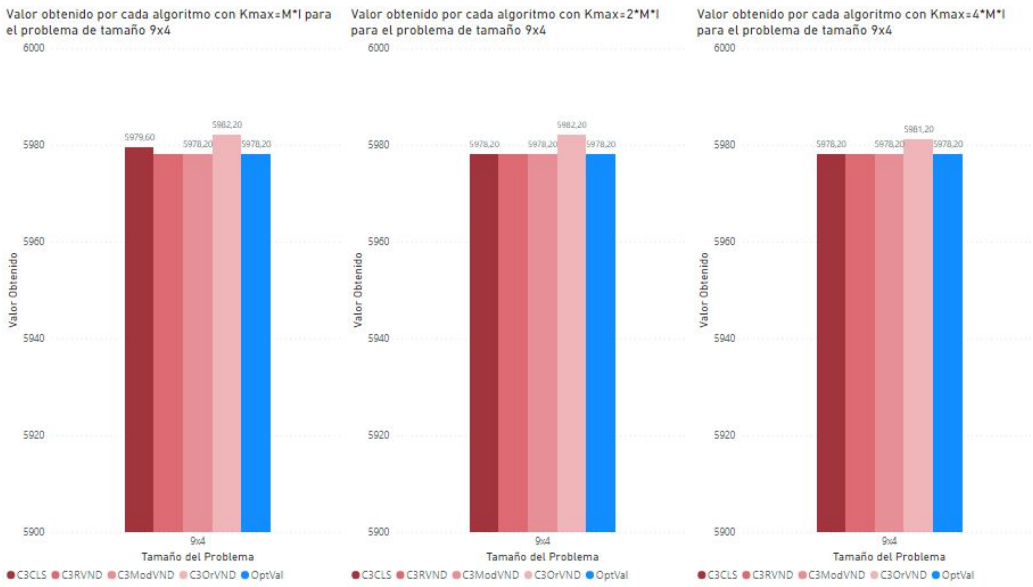


Figura 3.1: Gráfica de valores obtenidos para el problema de tamaño 9x4

$K_{max} = 2 * M * I$ y $K_{max} = 4 * M * I$, siendo M el número de camiones (en este caso 9) e I el número de puertas (en este caso 4). Caben destacar dos observaciones:

- El algoritmo *C3CLS* no proporcionó la solución óptima con $K_{max} = M * I$, pero al aumentar este parámetro si logró llegar a ella. Una justificación para este comportamiento podría ser la que se proporcionó en el apartado anterior sobre la importancia del valor K_{max}
- El algoritmo *C3OrVND* no logra llegar al valor óptimo. La solución mejora con $K_{max} = 4 * M * I$, pero no consigue proporcionar la óptima.

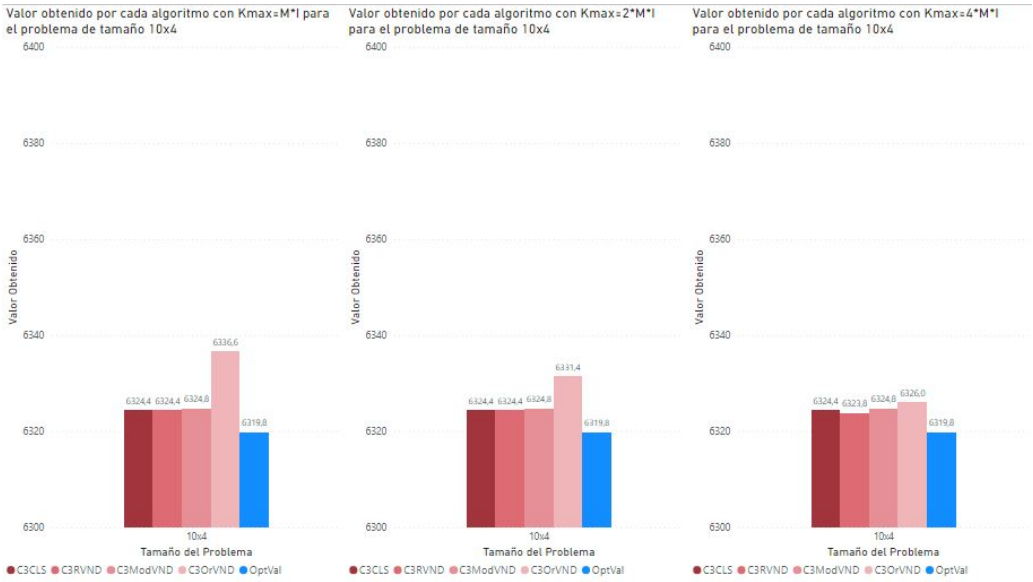


Figura 3.2: Gráfica de valores obtenidos para el problema de tamaño 10x4

En cuanto a los resultados obtenidos para el problema de tamaño *10x4* (figura 3.2), se puede observar que ningún algoritmo proporciona la solución óptima. Concretamente:

Tamaño: 10x4	M*I	2*M*I	4*M*I
C3CLS	6324.4	6324.4	6324.4
C3RVND	6324.4	6324.4	6323.8
C3ModVND	6324.8	6324.8	6324.8
C3OrVND	6336.6	6331.4	6326
OptVal	6319.8	6319.8	6319.8

Tabla 3.2: Valores obtenidos para el problema de tamaño 10x4

- Con el algoritmo *C3RVND* y $K_{max} = 4 * M * I$ se logra mejorar la solución, pero no llega a ser la óptima.
- En el *C3OrVND* la solución mejora gradualmente con el aumento del número de iteraciones, pero tampoco proporciona la óptima.

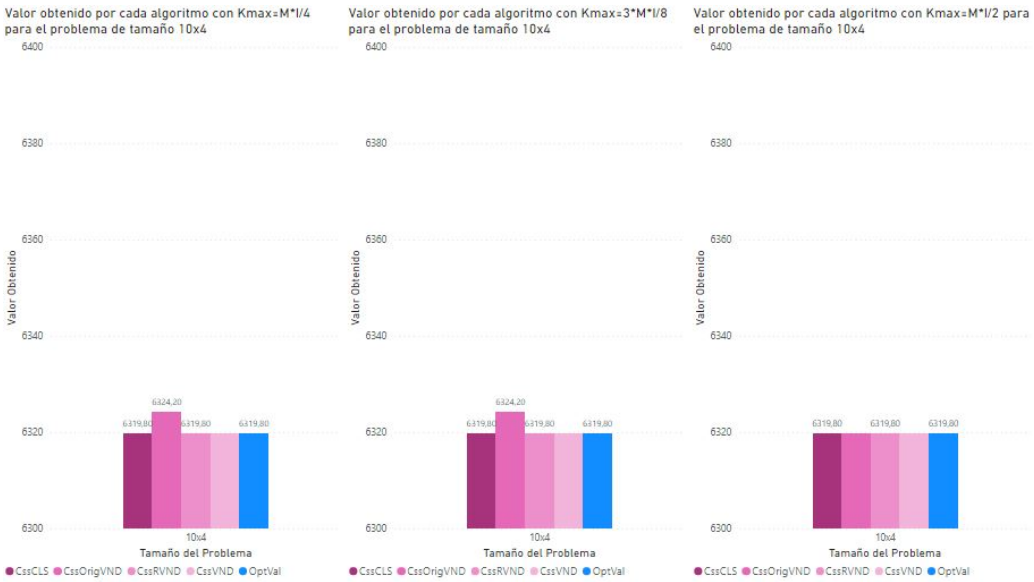


Figura 3.3: Gráfica de valores obtenidos para el problema de tamaño 10x4

Tamaño: 10x4	M*I/4	3*M*I/8	M*I/2
CssCLS	6319.8	6319.8	6319.8
CssOrigVND	6324.2	6324.2	6319.8
CssRVND	6319.8	6319.8	6319.8
CssVND	6319.8	6319.8	6319.8
OptVal	6319.8	6319.8	6319.8

Tabla 3.3: Valores obtenidos para el problema de tamaño 10x4

En la figura 3.3 se pueden ver los resultados obtenidos por el segundo grupo de algoritmos y K_{max}, compuesto por los algoritmos *CssRVND, CssCLS, CssVND y CssOrigVND*, y K_{max} $M \times I/4$, $3 \times M \times I/8$ y $M \times I/2$, para el problema de tamaño *10x4*. En este caso sí se logra llegar a la solución óptima, incluso con un K_{max} menor que en el caso anterior (figura 3.2). El algoritmo *CssOrigVND* proporciona la solución óptima con $K_{max} = M * I/2$.

En las figuras 3.4 y 3.5 se pueden observar los resultados para el mismo número de camiones que el caso anterior, pero aumentando el número de puertas. En la primera

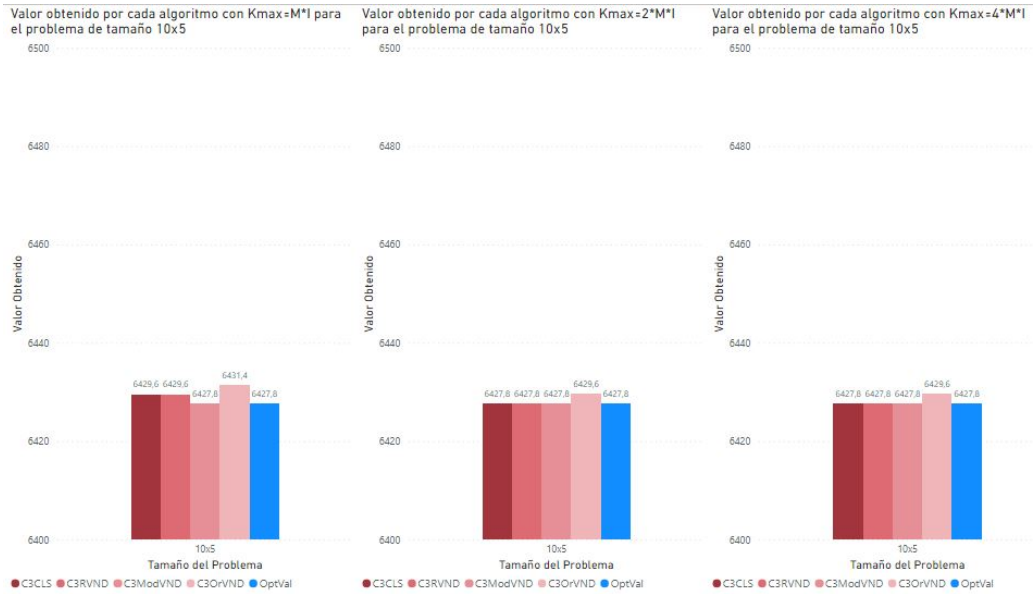


Figura 3.4: Gráfica de valores obtenidos para el problema de tamaño 10x5

Tamaño: 10x5	**M*I**	**2*M*I**	**4*M*I**
C3CLS	6429.6	6427.8	6427.8
C3RVND	6429.6	6427.8	6427.8
C3ModVND	6427.8	6427.8	6427.8
C3OrVND	6431.4	6429.6	6429.6
OptVal	6427.8	6427.8	6427.8

Tabla 3.4: Valores obtenidos para el problema de tamaño 10x5

Tamaño: 10x5	**M*I/4**	**3*M*I/8**	**M*I/2**
CssCLS	6427.8	6427.8	6427.8
CssOrigVND	6427.8	6427.8	6427.8
CssRVND	6427.8	6427.8	6427.8
CssVND	6427.8	6427.8	6427.8
OptVal	6427.8	6427.8	6427.8

Tabla 3.5: Valores obtenidos para el problema de tamaño 10x5

gráfica sí se puede apreciar un cambio importante, ya que todos los algoritmos, excepto el *C3OrVND*, logran proporcionar una solución óptima. El *C3ModVND* calcula el valor óptimo con un menor número de iteraciones ($K_{max} = M * I$) en comparación con los demás algoritmos, que lo hacen con $K_{max} = 2 * M * I$. Mientras que en la segunda gráfica los algoritmos calculan la solución óptima con $K_{max} = M * I/4$, algo que con un tamaño de problema *10x4* (Figura 3.3) no se pudo lograr. Una posible justificación para este comportamiento podría ser el aumento del número de puertas, lo que facilitaría el proceso de asignación de camiones a puertas.

En la siguiente figura (3.6), se muestran los resultados proporcionados para un problema de tamaño *12x5* (manteniendo el número de puertas y aumentando la cantidad de camiones, en comparación con el caso anterior). Se puede observar que el único algoritmo que logra calcular la solución óptima con un menor número de iteraciones es el *C3RVND*,

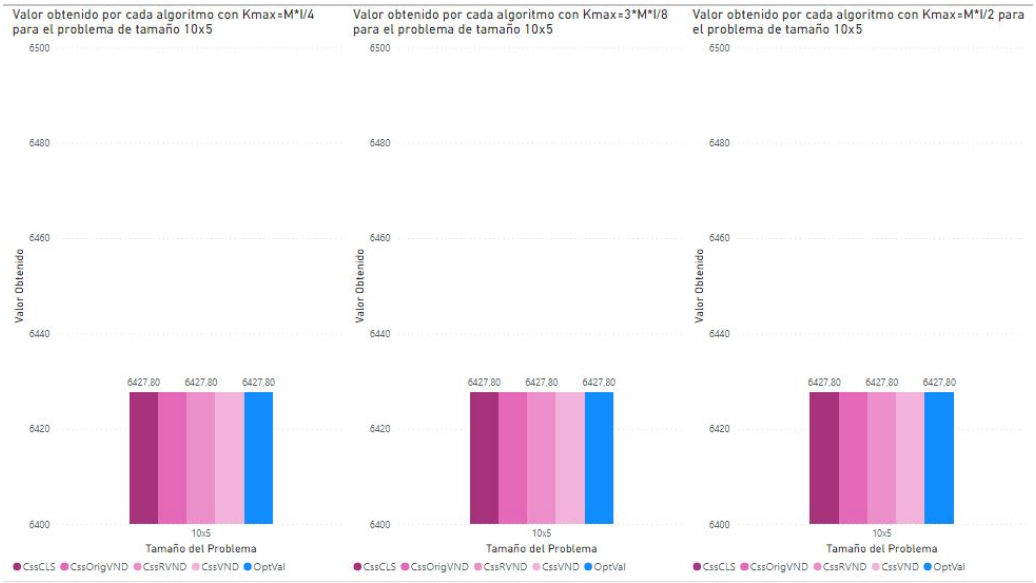


Figura 3.5: Gráfica de valores obtenidos para el problema de tamaño 10x5

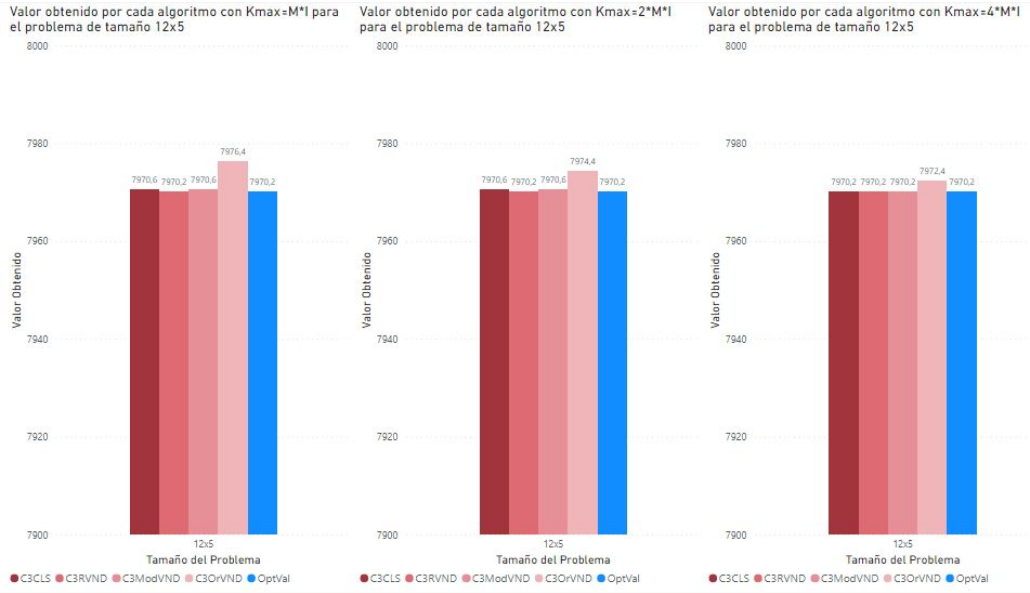


Figura 3.6: Gráfica de valores obtenidos para el problema de tamaño 12x5

Tamaño: 12x5	**M*I**	**2*M*I**	**4*M*I**
C3CLS	7970.6	7970.6	7970.2
C3RVND	7970.2	7970.2	7970.2
C3ModVND	7970.6	7970.6	7970.2
C3OrVND	7976.4	7974.4	7972.4
OptVal	7970.2	7970.2	7970.2

Tabla 3.6: Valores obtenidos para el problema de tamaño 12x5

mientras que los demás (exceptuando el *C3OrVND*) lo hacen con $K_{max} = 4 * M * I$.

En cuanto a la figura 3.7, se puede observar que los algoritmos llegan al valor óptimo con un menor número de iteraciones (comparado con el caso anterior donde el tamaño del problema era *12x5*, un número de puertas inferior al actual), lo calculan con $K_{max} =$

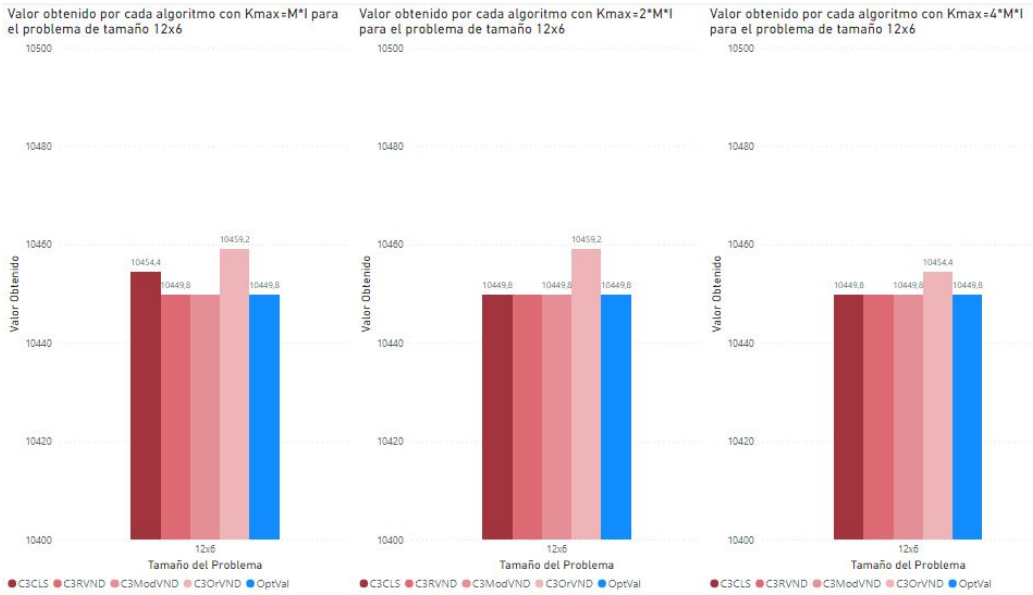


Figura 3.7: Gráfica de valores obtenidos para el problema de tamaño 12x6

Tamaño: 12x6	**M*I**	**2*M*I**	**4*M*I**
C3CLS	10454.4	10449.8	10449.8
C3RVND	10449.8	10449.8	10449.8
C3ModVND	10449.8	10449.8	10449.8
C3OrVND	10459.2	10459.2	10454.4
OptVal	10449.8	10449.8	10449.8

Tabla 3.7: Valores obtenidos para el problema de tamaño 12x6

$2 * M * I$

Tamaño: 15x6	**M*I**	**2*M*I**	**4*M*I**
C3CLS	13781	13763.6	13759.2
C3RVND	13780	13762.2	13760
C3ModVND	13770	13760.6	13759.2
C3OrVND	13797.8	13793	13784.8
OptVal	13756.4	13756.4	13756.4

Tabla 3.8: Valores obtenidos para el problema de tamaño 15x6

Al aumentar el tamaño de problema a *15x6*, se puede observar que llegar a la solución óptima con algoritmos metaheurísticos es más complicado. En el caso de la figura 3.8, ninguno de los algoritmos logra alcanzar la solución óptima.

Si se aumenta el número de puertas (problema de tamaño *15x7*), en la figura 3.9 se puede observar que con ningún algoritmo se logra obtener la solución óptima, es decir, el aumento del número de puertas en este caso no ayuda a facilitar la resolución del problema.

En cambio, en el segundo grupo de algoritmos (3.10), el aumento del número de puertas sí facilita la asignación, ya que los algoritmos logran calcular la solución óptima, exceptuando el *CssOrigVND*.

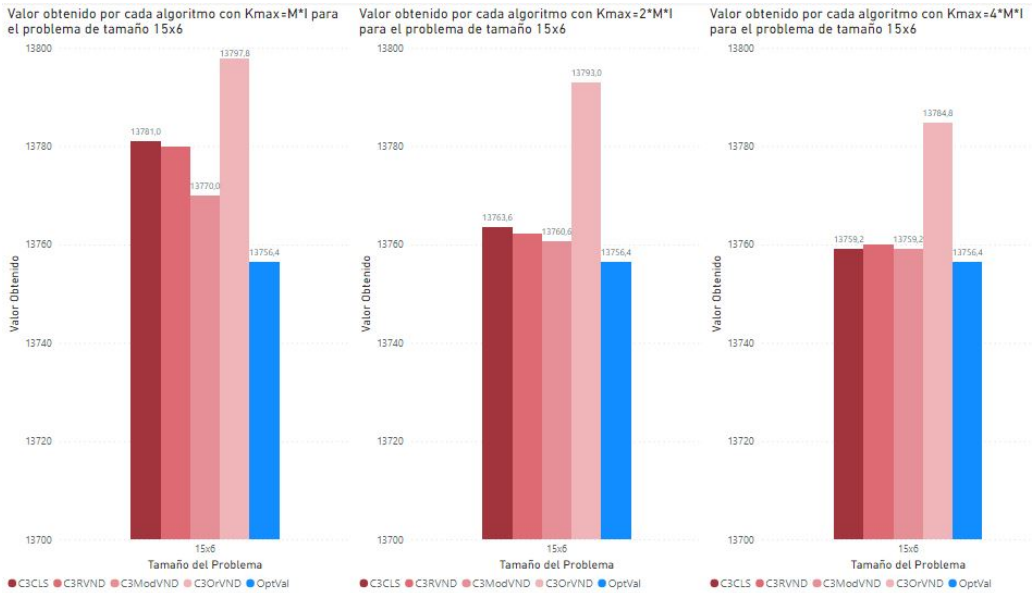


Figura 3.8: Gráfica de valores obtenidos para el problema de tamaño 15x6

Tamaño: 15x7	**M*I**	**2*M*I**	**4*M*I**
C3CLS	14722.2	14709.6	14691.8
C3RVND	14711.2	14696.4	14691.8
C3ModVND	14714.2	14701.2	14694.6
C3OrVND	14712	14701.6	14695.4
OptVal	14688.8	14688.8	14688.8

Tabla 3.9: Valores obtenidos para el problema de tamaño 15x7

Tamaño: 15x7	**M*I/4**	**3*M*I/8**	**M*I/2**
CssCLS	14688.8	14688.8	14688.8
CssOrigVND	14693	14692.8	14689.4
CssRVND	14689.4	14688.8	14688.8
CssVND	14688.8	14688.8	14688.8
OptVal	14688.8	14688.8	14688.8

Tabla 3.10: Valores obtenidos para el problema de tamaño 15x7

En conclusión, se puede observar que, cuanto más grande sea el tamaño del problema, más complicado va a ser para un algoritmo metaheurítico calcular la solución óptima. Esto podría ser un gran problema ya que en este caso se están evaluando los algoritmos con instancias de prueba, que tienen un tamaño bastante inferior en comparación con los casos reales.

Un factor que influye bastante es la diferencia entre la cantidad de camiones y el número de puertas, ya que cuanto menor sea este número, más facil va a ser para los algoritmos obtener una solución óptima.

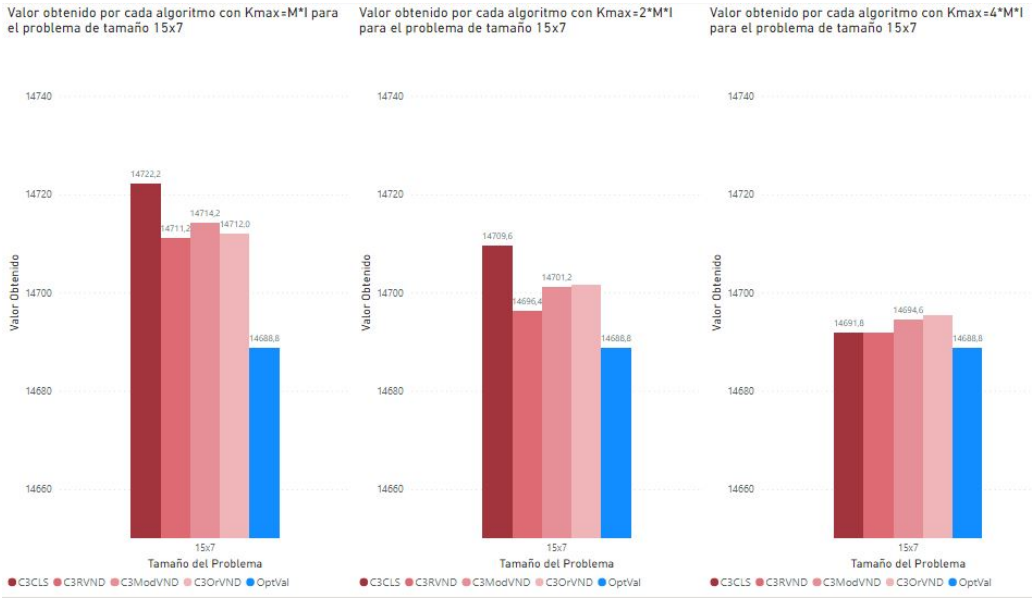


Figura 3.9: Gráfica de valores obtenidos para el problema de tamaño 15x7

3.2.2. Análisis Valor-Tiempo

En el apartado anterior se analizaron los valores proporcionados por los algoritmos metaheurísticos, y en éste se va a intentar enriquecer el análisis anterior incorporando otra métrica, el tiempo de cómputo.

Como se ha explicado en apartados anteriores, el elemento diferenciador clave entre un algoritmo exacto y uno metaheurístico (aplicado en este problema) es el tiempo de cómputo. Por lo tanto, es muy importante analizar este parámetro a la hora de evaluar los algoritmos metaheurísticos.

Se van a usar los mismos problemas del apartado anterior (con los mismos valores), añadiendo, como parámetro, el tiempo que tardó un algoritmo en proporcionar el valor.

En el caso del problema de tamaño *9x4* (figura 3.11), se puede observar el crecimiento del tiempo de cómputo cuando aumenta el número de iteraciones máximas. Por lo tanto, es necesario establecer un valor K_{max} lo suficientemente grande como para poder proporcionar una solución óptima, pero no debe ser muy grande, ya que podría influenciar negativamente al tiempo de cómputo.

El algoritmo *C3RVND* y *C3ModVND* logran llegar a la solución óptima con $K_{max} = M*I$, pero el *C3ModVND* lo hace en un menor tiempo (aunque la diferencia no sea muy alta), por lo tanto se podría decir que este algoritmo es el más apto para el problema.

Si se aumenta un camión más al tamaño del problema anterior, se puede observar en la figura 3.12 que ninguno de los algoritmos logra calcular la solución óptima.

En cambio, el segundo grupo de algoritmos y K_{max} (figura 3.13) sí lo consigue hacer, pero el tiempo que tarda en calcular el valor óptimo es relativamente mayor que en el primer grupo.

En este caso, los algoritmos que llegan a la solución óptima en un tiempo de cómputo inferior son el *CssCLS* y *CssVND*, ambos con $K_{max} = M * I/4$.

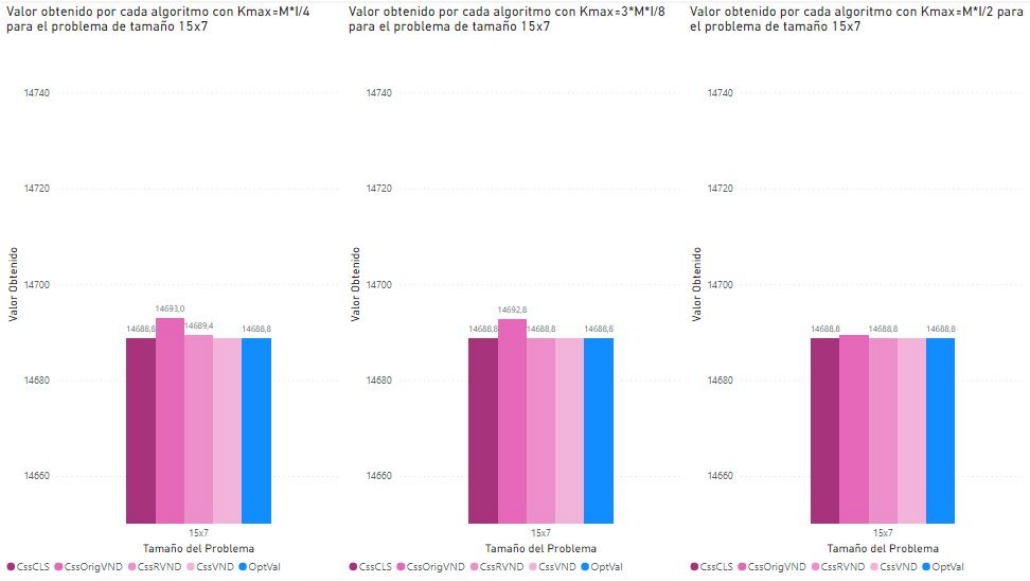


Figura 3.10: Gráfica de valores obtenidos para el problema de tamaño 15x7

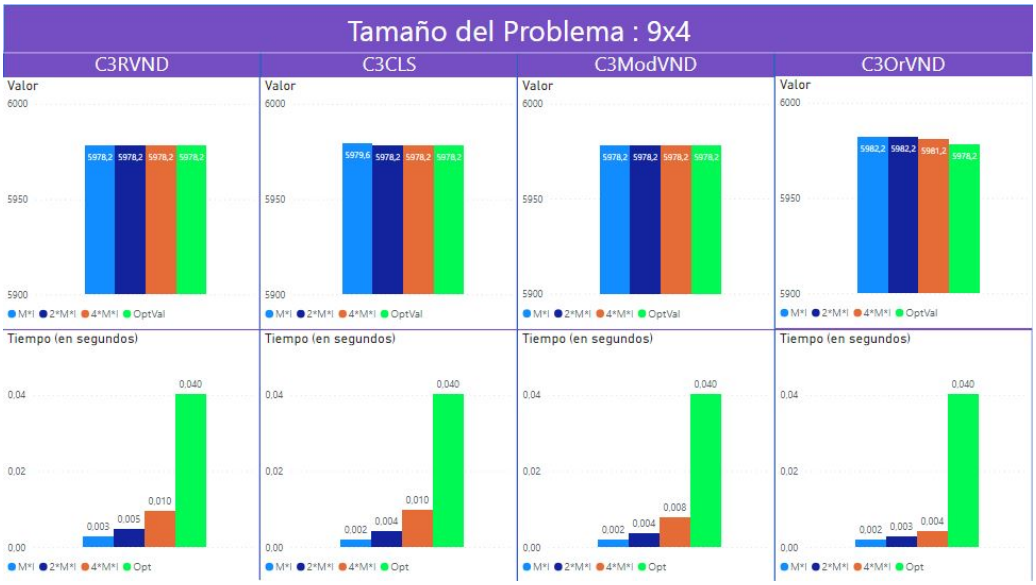


Figura 3.11: Gráfica de valor/tiempo para el problema de tamaño 9x4

Si se aumenta el número de puertas (tamaño del problema *10x5*), se puede observar en la figura 3.14 que el primer grupo de algoritmos consigue llegar al valor óptimo, tal y como se explicó en el apartado anterior, el *C3ModVND* calcula el valor óptimo en un menor número de iteraciones. En este caso, se puede apreciar además el tiempo de cómputo del algoritmo para ese valor.

Para el problema de tamaño *12x5*, se puede observar en la figura 3.15 que el algoritmo que proporciona la solución óptima en un tiempo inferior es el *C3RVND* con un $K_{max} = M * I$.

En el segundo grupo de algoritmos (figura 3.16) ninguno consigue calcular la solución óptima con $K_{max} = M * I/4$ (esto podría ser por la diferencia entre la cantidad de camiones y puertas), y con $K_{max} = 3 * M * I/8$ el algoritmo que llega a ella en un tiempo de cómputo menor es el *CssCLS*.

Tamaño: 9x4	**M*I**	**2*M*I**	**4*M*I**
C3RVND			
Valor	5978.2	5978.2	5978.2
Tiempo	0.003	0.005	0.01
C3CLS			
Valor	5979.6	5978.2	5978.2
Tiempo	0.002	0.004	0.01
C3ModVND			
Valor	5978.2	5978.2	5978.2
Tiempo	0.002	0.004	0.008
C3OrVND			
Valor	5982.2	5982.2	5981.2
Tiempo	0.002	0.003	0.004
Opt			
Valor	5978.2	5978.2	5978.2
Tiempo	0.04	0.04	0.04

Tabla 3.11: Valor/Tiempo para el problema de tamaño 9x4

Tamaño: 10x4	**M*I**	**2*M*I**	**4*M*I**
C3RVND			
Valor	6324.4	6324.4	6323.8
Tiempo	0.004	0.006	0.012
C3CLS			
Valor	6324.4	6324.4	6324.4
Tiempo	0.005	0.006	0.012
C3ModVND			
Valor	6324.8	6324.8	6324.8
Tiempo	0.003	0.005	0.010
C3OrVND			
Valor	6336.6	6331.4	6326
Tiempo	0.002	0.003	0.005
Opt			
Valor	6319.8	6319.8	6319.8
Tiempo	0.062	0.062	0.062

Tabla 3.12: Valor/Tiempo para el problema de tamaño 10x4

Por lo tanto, de entre los dos grupos de algoritmos, el que proporciona la solución óptima en un tiempo inferior es el *C3RVND*.

En cuanto al tamaño de problema *12x6*, cabe destacar la diferencia del tiempo de cómputo entre los dos grupos de algoritmos (figura 3.17 y 3.18). En este caso, el algoritmo más apto en cuanto al tiempo sería el *C3ModVND* con $K_{max} = M * I$.

Para el problema con 15 camiones y 6 puertas, no es posible calcular la solución óptima con los algoritmos metaheurísticos (figura 3.19).

Si se aumenta el número de puertas, el primer grupo de algoritmos no logra calcular el

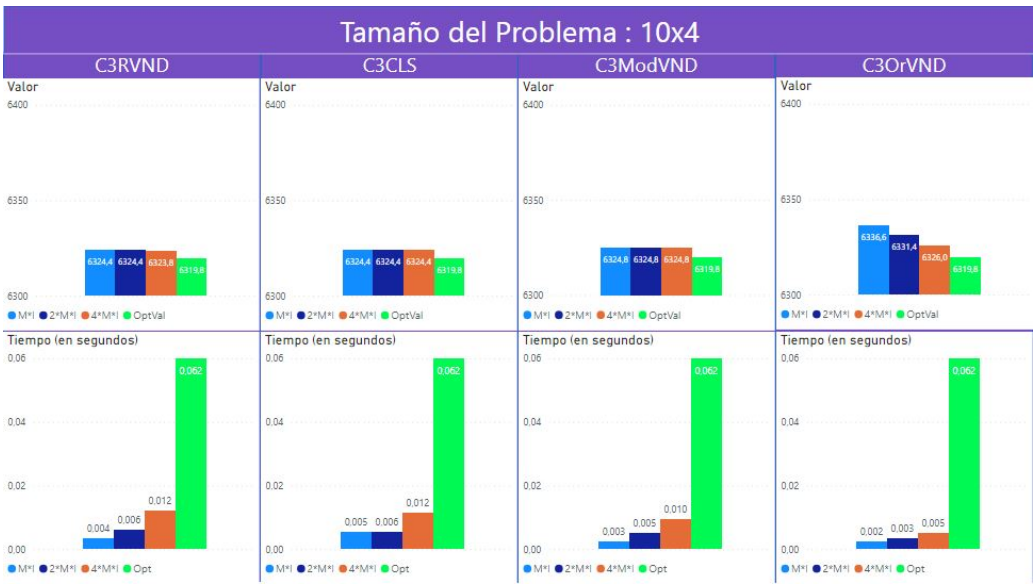


Figura 3.12: Gráfica de valor/tiempo para el problema de tamaño 10x4

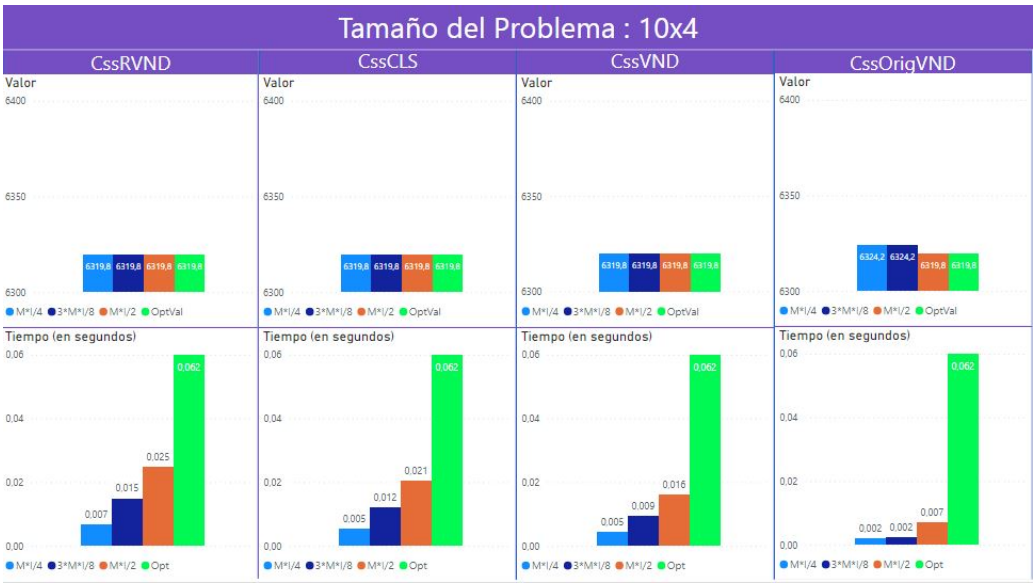


Figura 3.13: Gráfica de valor/tiempo para el problema de tamaño 10x4

valor óptimo (figura 3.20), pero el segundo grupo sí (figura 3.21), y el algoritmo que tarda menos en proporcionar la solución es el *CssVND*.

En conclusión, se puede observar el aumento del tiempo con el crecimiento del tamaño del problema. Una cosa que cabe destacar es la diferencia drástica entre el tiempo de cómputo de un algoritmo exacto y el de un algoritmo metaheurístico. Esta es la razón por la que muchas empresas están optando por usar algoritmos metaheurísticos, ya que en un caso real el tamaño de problema suele ser muy grande, y por lo tanto el tiempo de cómputo de un algoritmo exacto también aumenta.

Tamaño: 10x4	M*I/4	3*M*I/8	M*I/2
CssRVND			
Valor	6319.8	6319.8	6319.8
Tiempo	0.007	0.015	0.025
CssCLS			
Valor	6319.8	6319.8	6319.8
Tiempo	0.005	0.012	0.021
CssVND			
Valor	6319.8	6319.8	6319.8
Tiempo	0.005	0.009	0.016
CssOrigVND			
Valor	6324.2	6324.2	6319.8
Tiempo	0.002	0.002	0.007
Opt			
Valor	6319.8	6319.8	6319.8
Tiempo	0.062	0.062	0.062

Tabla 3.13: Valor/Tiempo para el problema de tamaño 10x4

Tamaño: 10x5	M*I	2*M*I	4*M*I
C3RVND			
Valor	6429.6	6427.8	6427.8
Tiempo	0.005	0.01	0.02
C3CLS			
Valor	6429.6	6427.8	6427.8
Tiempo	0.004	0.009	0.017
C3ModVND			
Valor	6427.8	6427.8	6427.8
Tiempo	0.004	0.008	0.015
C3OrVND			
Valor	6431.4	6429.6	6429.6
Tiempo	0.003	0.005	0.008
Opt			
Valor	6427.8	6427.8	6427.8
Tiempo	0.101	0.101	0.101

Tabla 3.14: Valor/Tiempo para el problema de tamaño 10x5

3.3. Análisis General

En los apartados anteriores se analizaron los algoritmos metaheurísticos para tamaños de problemas específicos, pero en la práctica los problemas suelen ser mucho más grandes. Por lo tanto, es complicado nombrar el mejor algoritmo metaheurístico ya que, como se pudo apreciar en el apartado anterior, para cada tamaño de problema había un algoritmo diferente que proporcionaba el valor óptimo en un tiempo inferior. Con lo cual, se va a intentar analizar el rendimiento de los dos grupos de algoritmos sin tener en cuenta el tamaño del problema. Para ello, se va a usar como métrica la diferencia entre el valor generado por cada algoritmo y el óptimo para una instancia, lo que se conoce como el

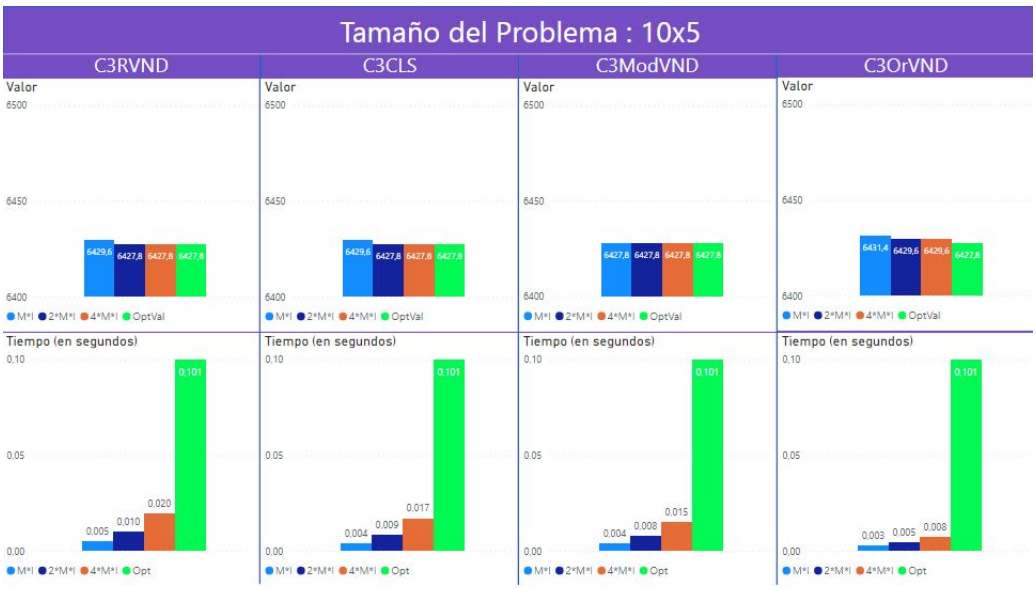


Figura 3.14: Gráfica de valor/tiempo para el problema de tamaño 10x5

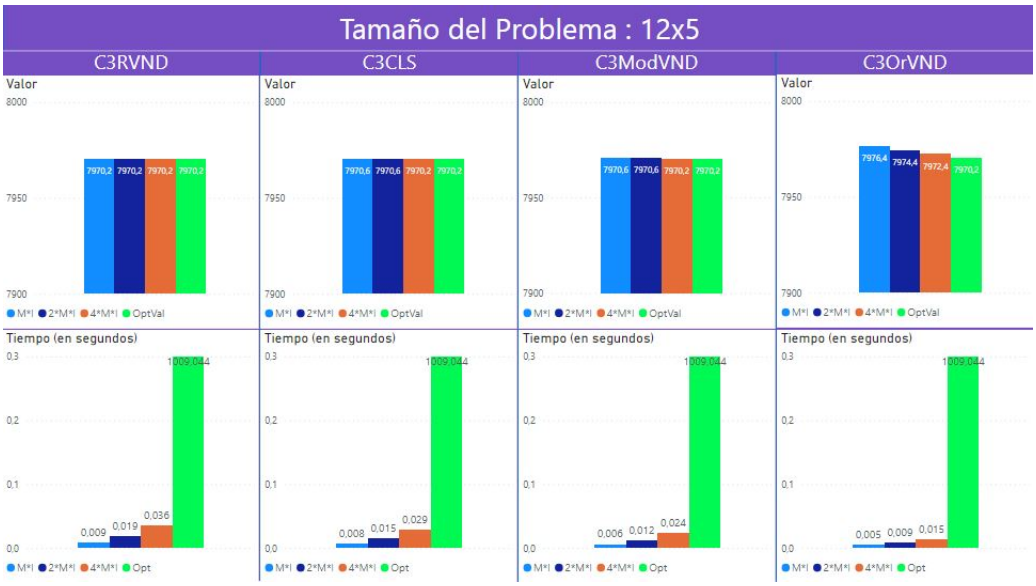


Figura 3.15: Gráfica de valor/tiempo para el problema de tamaño 12x5

"Gap".

Como lo que se está intentando analizar es el rendimiento general de los algoritmos sin tener en cuenta las instancias, se va a representar el valor medio de *Gap* para cada algoritmo.

En la figura 3.22 se puede observar que, al aumentar el número de iteraciones, se reduce la diferencia entre el valor generado por los algoritmos y el óptimo (*Gap*), es decir, se mejora la solución. Pero también aumenta el tiempo de cómputo.

Concreatemente, el algoritmo *C3OrVND* es el que tiene los tiempos mínimos para cada número de iteraciones, pero al mismo tiempo el *Gap* es bastante alto.

Mientras que en la figura 3.23 se reflejan los valores para el segundo grupo de algoritmos, donde el *Gap* es bastante inferior comparado con el primer grupo pero, al mismo tiempo, el tiempo de cómputo es muy alto.

Tamaño: 12x5	**M*I**	**2*M*I**	**4*M*I**
C3RVND			
Valor	7970.2	7970.2	7970.2
Tiempo	0.009	0.019	0.036
C3CLS			
Valor	7970.6	7970.6	7970.2
Tiempo	0.008	0.015	0.029
C3ModVND			
Valor	7970.6	7970.6	7970.2
Tiempo	0.006	0.012	0.024
C3OrVND			
Valor	7976.4	7974.4	7972.4
Tiempo	0.005	0.009	0.015
Opt			
Valor	7970.2	7970.2	7970.2
Tiempo	1009.044	1009.044	1009.044

Tabla 3.15: Valor/Tiempo para el problema de tamaño 12x5

Tamaño: 12x5	**M*I/4**	**3*M*I/8**	**M*I/2**
CssRVND			
Valor	7970.6	7970.2	7970.2
Tiempo	0.029	0.065	0.114
CssCLS			
Valor	7972.4	7970.2	7970.2
Tiempo	0.024	0.056	0.094
CssVND			
Valor	7970.6	7970.6	7970.2
Tiempo	0.019	0.042	0.071
CssOrigVND			
Valor	7988	7974.6	7972.6
Tiempo	0.011	0.012	0.04
Opt			
Valor	7970.2	7970.2	7970.2
Tiempo	1009.044	1009.044	1009.044

Tabla 3.16: Valor/Tiempo para el problema de tamaño 12x5

Concretamente, el *CssOrigVND* tiene un tiempo de cómputo bastante inferior que los demás algoritmos pero, al igual que el algoritmo *C3OrVND* en la figura 3.22, la diferencia entre el valor generado y el óptimo es bastante elevado.

Por lo tanto, es complicado determinar cuál es el mejor algoritmo (y con cuántas iteraciones máximas), ya que depende de las prioridades específicas del problema. Por ejemplo, cuando el tamaño del problema no es muy grande, se podría priorizar una solución de mayor calidad en lugar de optimizar el tiempo de cómputo, ya que no existirá una diferencia notable en este parámetro.

En cambio, si se trata de un problema más grande, el impacto del K_{max} en la calidad

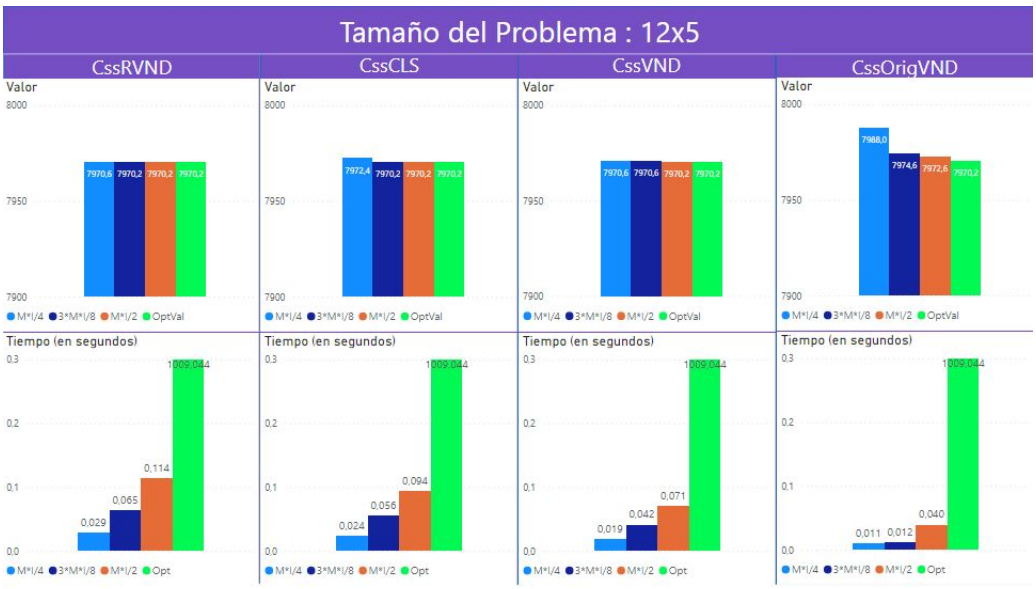


Figura 3.16: Gráfica de valor/tiempo para el problema de tamaño 12x5

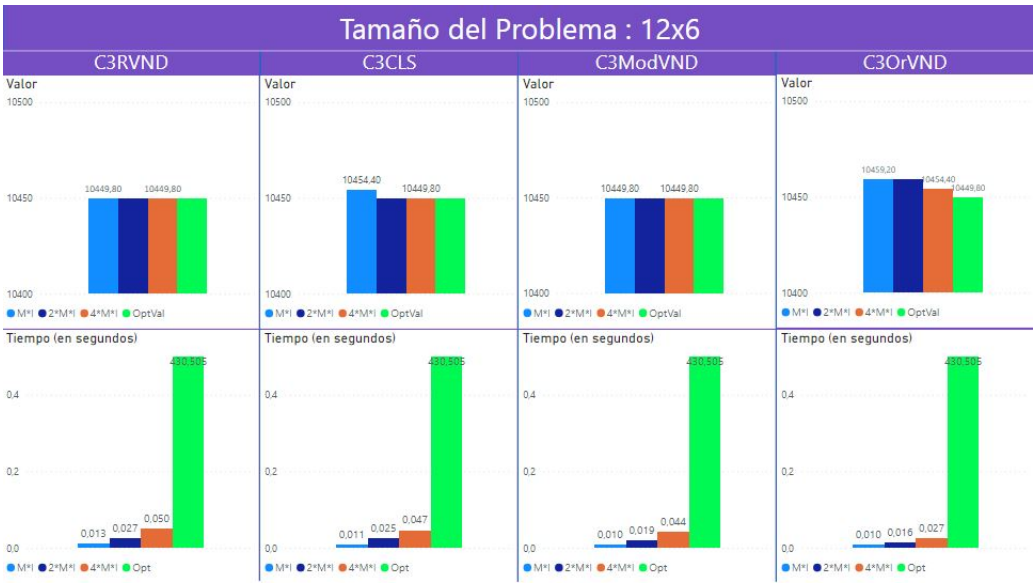


Figura 3.17: Gráfica de valor/tiempo para el problema de tamaño 12x6

de la solución podría ser más grande. En este caso, habría que evaluar las necesidades y restricciones asociadas al problema específico (complejidad, recursos, etc).

En definitiva, la elección del algoritmo y el número máximo de iteraciones depende del problema específico, es decir, si se quiere priorizar más el tiempo de cómputo o la calidad de la solución.

Si la prioridad es obtener una solución de mejor calidad, hay cuatro algoritmos que tienen la mejor solución según el *Gap* (tabla 3.22). Entre ellos, el que proporciona la solución en un tiempo inferior es el algoritmo **CssCLS** con $K_{max} = 3 * M * I/8$.

En cambio, si se prioriza más el tiempo de cómputo, hay dos algoritmos que tienen el menor tiempo (tabla 3.23). Entre ellos, el algoritmo que proporciona una solución mejor es el *C3ModVND* con $K_{max} = M * I$.

Tamaño: 12x6	M*I	2*M*I	4*M*I
C3RVND			
Valor	10449.8	10449.8	10449.8
Tiempo	0.013	0.027	0.05
C3CLS			
Valor	10454.4	10449.8	10449.8
Tiempo	0.011	0.025	0.047
C3ModVND			
Valor	10449.8	10449.8	10449.8
Tiempo	0.01	0.019	0.044
C3OrVND			
Valor	10459.2	10459.2	10454.4
Tiempo	0.01	0.016	0.027
Opt			
Valor	10449.8	10449.8	10449.8
Tiempo	430.5048	430.5048	430.5048

Tabla 3.17: Valor/Tiempo para el problema de tamaño 12x6

Tamaño: 12x6	M*I/4	3*M*I/8	M*I/2
CssRVND			
Valor	10449.8	10449.8	10449.8
Tiempo	0.045	0.108	0.194
CssCLS			
Valor	10449.8	10449.8	10449.8
Tiempo	0.036	0.078	0.144
CssVND			
Valor	10449.8	10449.8	10449.8
Tiempo	0.029	0.066	0.114
CssOrigVND			
Valor	10452	10449.8	10449.8
Tiempo	0.019	0.024	0.075
Opt			
Valor	10449.8	10449.8	10449.8
Tiempo	430.5048	430.5048	430.5048

Tabla 3.18: Valor/Tiempo para el problema de tamaño 12x6

3.4. Conclusión

En este capítulo se han analizado los resultados computacionales obtenidos por algunos de los algoritmos metaheurísticos que resuelven el problema CDAP. Se observó que el tamaño del problema, compuesto por el número de camiones y puertas, influencia bastante en el valor generado por el algoritmo, ya que, a medida que aumentaba el tamaño del problema, era más dificil para los algoritmos metaheurísticos encontrar la solución óptima.

Además, se realizó un análisis del valor generado por un algoritmo frente al tiempo

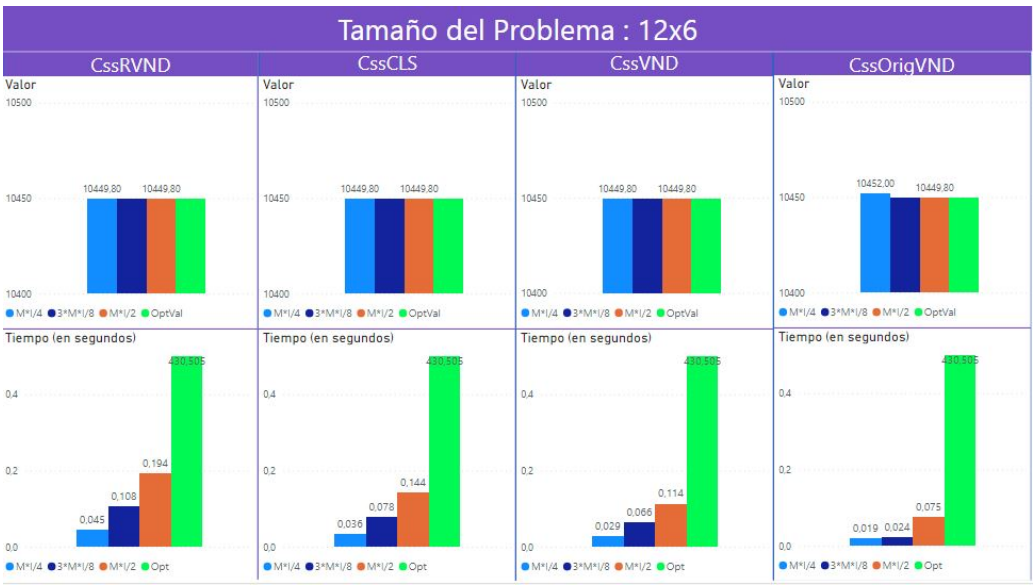


Figura 3.18: Gráfica de valor/tiempo para el problema de tamaño 12x6

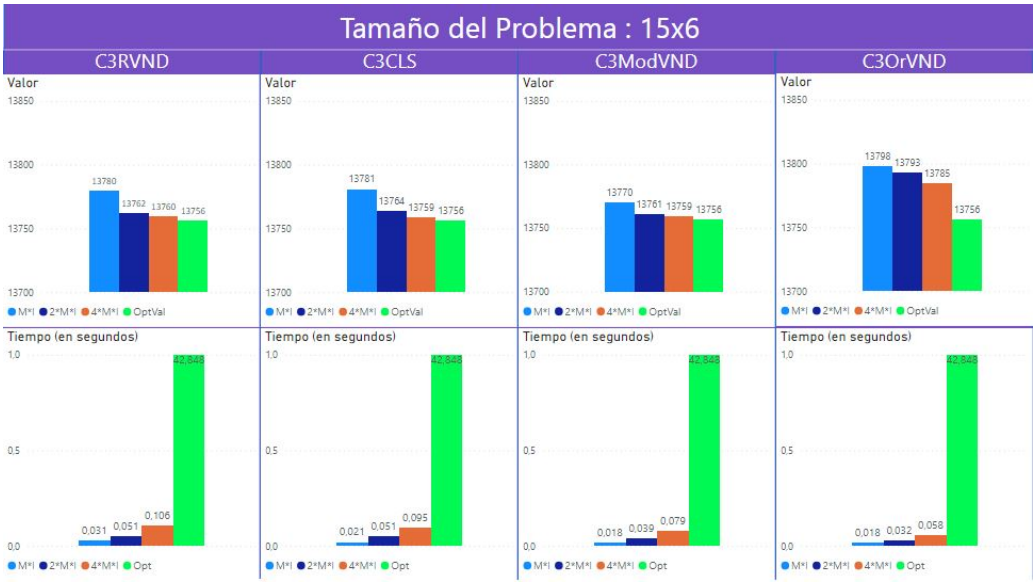


Figura 3.19: Gráfica de valor/tiempo para el problema de tamaño 15x6

de cómputo, y se observó que el tiempo de cómputo también aumenta con el tamaño del problema. Por lo tanto, se encontró que la elección del algoritmo depende de las prioridades específicas del problema, es decir, si se quiere priorizar la calidad de la solución o el tiempo de cómputo.

Tamaño: 15x6	**M*I**	**2*M*I**	**4*M*I**
C3RVND			
Valor	13780	13762.2	13760
Tiempo	0.031	0.051	0.106
C3CLS			
Valor	13781	13763.6	13759.2
Tiempo	0.021	0.051	0.095
C3ModVND			
Valor	13770	13760.6	13759.2
Tiempo	0.018	0.039	0.079
C3OrVND			
Valor	13797.8	13793	13784.8
Tiempo	0.018	0.032	0.058
Opt			
Valor	13756.4	13756.4	13756.4
Tiempo	42.8478	42.8478	42.8478

Tabla 3.19: Valor/Tiempo para el problema de tamaño 15x6

Tamaño: 15x7	**M*I**	**2*M*I**	**4*M*I**
C3RVND			
Valor	14711.2	14696.4	14691.8
Tiempo	0.036	0.076	0.145
C3CLS			
Valor	14722.2	14709.6	14691.8
Tiempo	0.029	0.068	0.125
C3ModVND			
Valor	14714.2	14701.2	14694.6
Tiempo	0.027	0.053	0.105
C3OrVND			
Valor	14712	14701.6	14695.4
Tiempo	0.024	0.042	0.077
Opt			
Valor	14688.8	14688.8	14688.8
Tiempo	139.8116	139.8116	139.8116

Tabla 3.20: Valor/Tiempo para el problema de tamaño 15x7

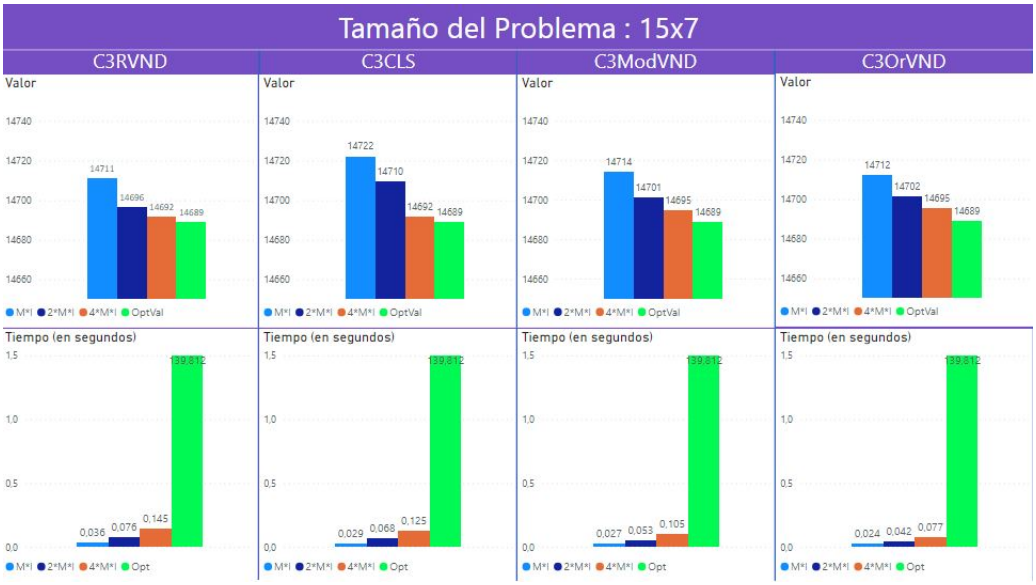


Figura 3.20: Gráfica de valor/tiempo para el problema de tamaño 15x7

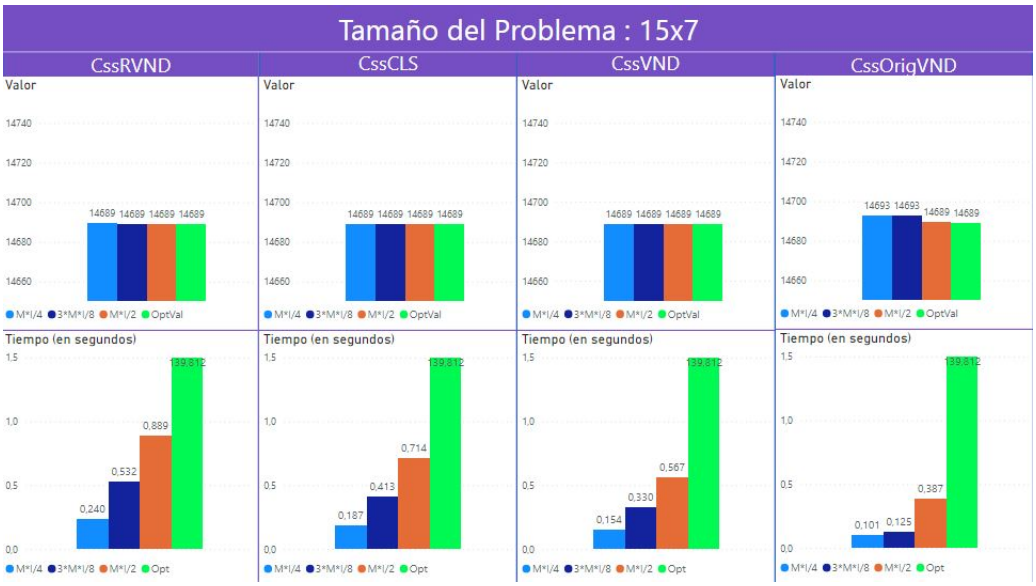


Figura 3.21: Gráfica de valor/tiempo para el problema de tamaño 15x7

Tamaño: 15x7	M*I/4	3*M*I/8	M*I/2
CssRVND			
Valor	14689.4	14688.8	14688.8
Tiempo	0.24	0.532	0.889
CssCLS			
Valor	14688.8	14688.8	14688.8
Tiempo	0.187	0.413	0.714
CssVND			
Valor	14688.8	14688.8	14688.8
Tiempo	0.154	0.33	0.567
CssOrigVND			
Valor	14693	14692.8	14689.4
Tiempo	0.101	0.125	0.387
Opt			
Valor	14688.8	14688.8	14688.8
Tiempo	139.8116	139.8116	139.8116

Tabla 3.21: Valor/Tiempo para el problema de tamaño 15x7

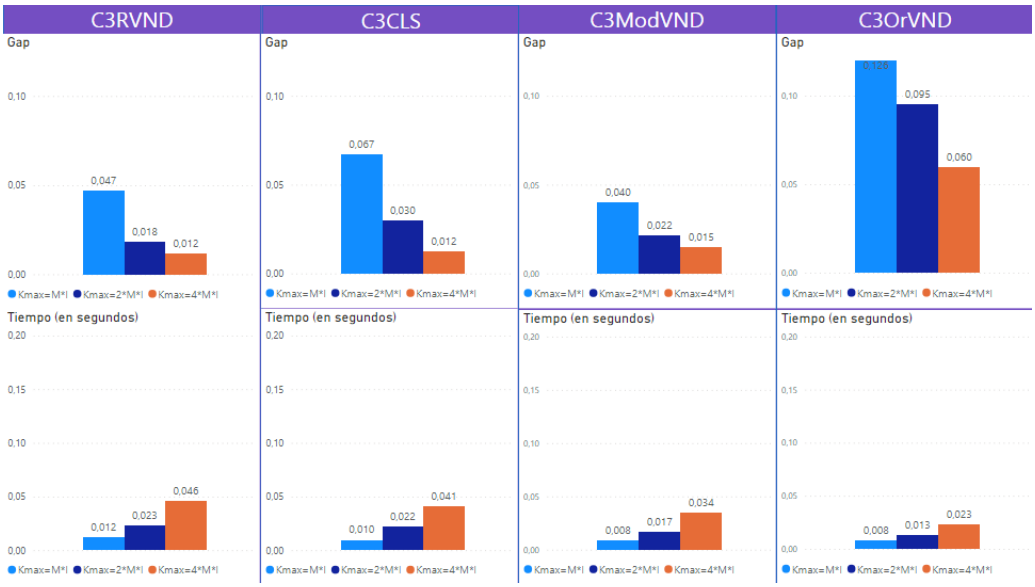


Figura 3.22: Gráfica para el análisis general de algoritmos.

Algoritmo	Kmax	Gap	Tiempo (segundos)
CssRVND	**M*I/2**	0.002	0.208
CssCLS	**3*M*I**/8	0.002	0.093
CssCLS	**M*I/2**	0.002	0.169
CssVND	**M*I/2**	0.002	0.132

Tabla 3.22: Algoritmos con mejor Gap

Algoritmo	Kmax	Tiempo (segundos)	Gap
C3ModVND	**M*I**	0.008	0.04
C3OrVND	**M*I**	0.008	0.126

Tabla 3.23: Algoritmos con mejor tiempo de cómputo

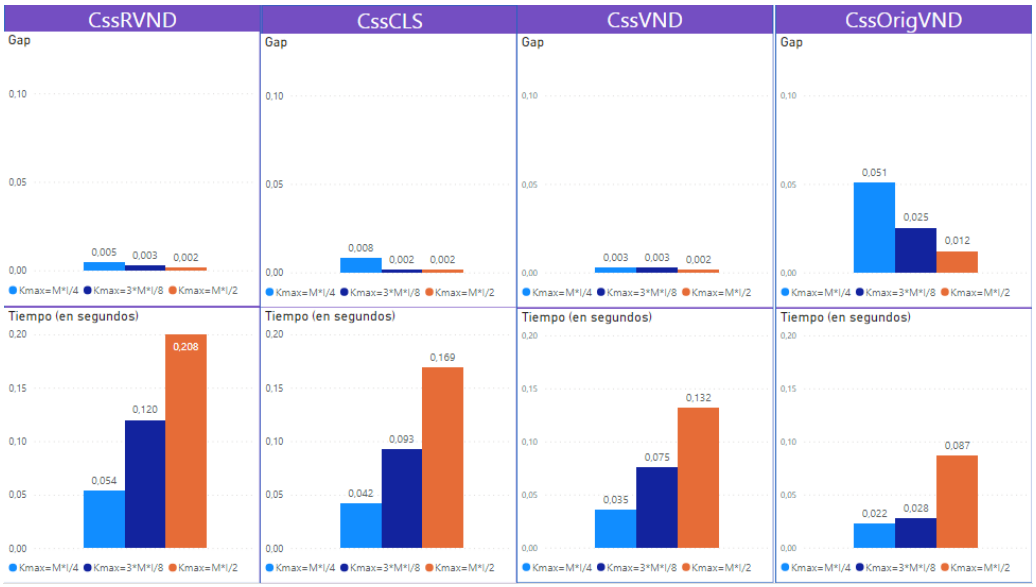


Figura 3.23: Gráfica para el análisis general de algoritmos.

Capítulo 4

Análisis de la Simulación de las Soluciones

En el apartado anterior se intentó determinar qué algoritmo es el más adecuado para resolver el problema de asignación de camiones a puertas, definiendo la calidad de la solución como la diferencia entre el valor calculado por un algoritmo y el valor óptimo. Pero cabe destacar que una solución de calidad también se podría definir utilizando la factibilidad de ésta, es decir, el nivel de flexibilidad de la solución calculada.

En un centro *cross-dock*, las puertas tienen una capacidad determinada, que los algoritmos deben tener en cuenta a la hora de asignar un camión a una puerta. Cuando se habla de capacidad se refiere a la cantidad de *pallets* que puede gestionar una puerta. Por ejemplo, si una puerta tiene una capacidad de 20 *pallets*, el algoritmo debe asignar un camión que contenga un número de *pallets* inferior o igual a la capacidad.

El problema viene cuando, en un determinado momento y por causas ajenas, una puerta no puede proporcionar la capacidad que se le había asignado inicialmente. Esto podría ocurrir, por ejemplo, cuando un trabajador del centro no se encuentra presente en ese momento, o si falla un dispositivo de transporte interno de mercancía (transpaleta). Si la solución no se puede ajustar en este tipo de situaciones de último momento, se consideraría no factible.

Por lo tanto, la solución calculada debe tener un cierto nivel de robustez para soportar estos eventos disruptivos. Teniendo esto en cuenta, se han creado una serie de gráficas (una para cada problema), para analizar la factibilidad de las soluciones generadas en un proceso de simulación, en el que la capacidad de las puertas se define como una variable aleatoria que sigue una distribución uniforme.

4.1. Análisis de Factibilidad

El primer paso para analizar la factibilidad de las soluciones sería realizar una simulación de éstas considerando la capacidad de las puertas como una variable aleatoria y extraer los resultados. Estos resultados se encuentran en un fichero con extensión *.txt-out* y existe un fichero para cada problema, cuyo nombre incluye el número total de camiones y puertas, el valor de Holgura y un ID de la instancia.

Por ejemplo, si el nombre del fichero es `SolA_8x4S10_1.txt-out`, significa que el

problema tiene 8 camiones, 4 puertas, el valor de holgura es 10, y se trata de una instancia con ID 1.

En total hay 220 instancias distintas, que están definidas por los siguientes valores:

- **Número de camiones**: en este caso se han considerado 8, 9, 10, 11, 12, 15 y 20.
- **Número de puertas**: en este caso se han considerado 4, 5, 6, 7 y 10.
- **Valor de holgura**: en este caso se han considerado 10, 15, 20 y 30.
- **ID de la solución**: la cantidad de soluciones de cada tamaño de problema depende del número de puertas, es decir, si el problema tiene 4 puertas, en total habrá 4 soluciones para ese tamaño del problema, 2 generadas por calidad y 2 por dispersión.

Para analizar la factibilidad de una solución, el procedimiento que se sigue es el siguiente:

- Una vez que se genera la solución inicial, se reduce la capacidad de las puertas. Por ejemplo, si la capacidad inicial de las puertas era de 166 *pallets*, se le reduce un porcentaje de capacidad, por ejemplo, $-1\,\%$. Entonces, se ejecutan simulaciones en las que la capacidad de la puerta tiene un valor en el rango $[164{,}34, 166]$, es decir, el valor mínimo del rango será el valor inicial de capacidad menos el porcentaje del mismo. Los porcentajes considerados para cada instancia serán: $0\,\%, -1\,\%, -2\,\%, ..., -20\,\%$.
- Para cada porcentaje, se van a realizar 100 iteraciones, y en cada iteración se va a seleccionar una capacidad aleatoria, cuyo valor se encuentre dentro del rango calculado.
- Finalmente, se va a analizar si la solución generada inicialmente sigue siendo factible.
- El resultado de todo este procedimiento se introduce en el fichero con extensión *.txt-out* que se explicó anteriormente. En dicho fichero, para cada porcentaje se menciona, entre otros valores, la capacidad aleatoria seleccionada, y un valor *booleano* que indica si la solución sería factible para la capacidad seleccionada.

Al final de cada fichero se especifica, de las 100 iteraciones realizadas para cada porcentaje, cuántas veces la solución ha sido factible y cuántas veces ha sido infactible. Este es el dato que se mostrará en las gráficas, con lo cual es necesario crear un programa que extraiga estos datos de los 220 ficheros y los ponga en una hoja de cálculo de *Microsoft Excel* para que se puedan cargar estos datos en *Microsoft Power BI* y crear las gráficas para analizar la factibilidad. Dichas gráficas se pueden visualizar en este enlace.

Para crear el programa se ha optado por usar el lenguaje de programación *Python*[1]. Este lenguaje es ampliamente utilizado en el campo de ciencia de datos gracias a su simplicidad y flexibilidad. En este caso se optó por utilizar *Python* ya que ofrece una gran variedad de bibliotecas específicas para el análisis de datos y la manipulación de hojas de cálculo, lo cual era necesario para el desarrollo del programa. La biblioteca que se

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

utilizó para generar la hoja de cálculo de *Microsoft Excel* es *openpyxl*[2]. Se trata de una biblioteca para leer y escribir ficheros de *Excel* (*.xlsx*).

	A	B	C	D	E	F	G
1	Instance	val-0%	fact-0%	nofact-0%	val-1%	fact-1%	nofact-1%
2	10x4S10_1	205	100	0	203.969	50	50
3	10x4S10_2	205	100	0	203.911	50	50
4	10x4S15_1	215	100	0	214.02	100	0
5	10x4S15_2	215	100	0	213.964	100	0
6	10x4S20_1	224	100	0	222.877	100	0
7	10x4S20_2	224	100	0	222.854	93.5	6.5
8	10x4S30_1	243	100	0	241.761	50	50
9	10x4S30_2	243	100	0	241.746	100	0
10	10x5S10_1	165	100	0	164.189	100	0
11	10x5S10_2	165	100	0	164.194	74	26
12	10x5S15_1	172	100	0	171.166	100	0
13	10x5S15_2	172	100	0	171.091	100	0
14	10x5S20_1	180	100	0	179.088	79	21
15	10x5S20_2	180	100	0	179.176	75	25
16	10x5S30_1	194	100	0	193.12	100	0
17	10x5S30_2	194	100	0	193	100	0

Figura 4.1: Ejemplo de la hoja de cálculo con el número de soluciones factibles e infactibles.

En la figura 4.1, se puede observar un extracto de la hoja de cálculo generada por el programa mencionado anteriormente. Una cosa que cabe destacar es que, teniendo en cuenta la simplicidad que deberían tener las gráficas que se generen posteriormente utilizando estos datos, se optó por unificar las distintas soluciones de un problema en dos grupos, las que generan soluciones de calidad y las que generan soluciones de dispersión.

4.2. Visualización de Factibilidad

El siguiente paso sería, utilizando los datos de factibilidad (explicado en el apartado anterior), generar las gráficas para analizar si las soluciones generadas son factibles o infactibles.

Para el problema *8x4S10* (figura 4.2), se pueden observar dos gráficas, una para mostrar la cantidad de soluciones de calidad factibles e infactibles , y otra para las soluciones de dispersión. La solución de calidad seguirá siendo factible incluso cuando las puertas dispongan de un 6 % menos de capacidad, mientras que la de dispersión solo soporta una fluctuación de capacidad de hasta un 3 %.

Una cosa que cabe destacar, en este caso y en los demás que se verán más adelante es que, a medida que aumenta el porcentaje, la factibilidad va decreciendo, pero no de manera continua. Por ejemplo, en el caso de la gráfica de calidad, se puede observar

[2]https://openpyxl.readthedocs.io/en/stable/

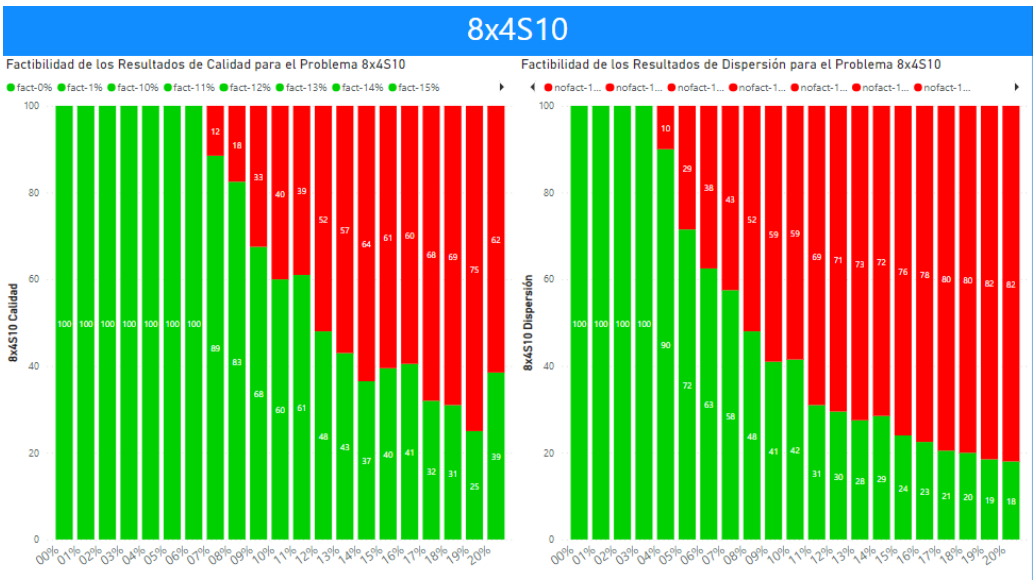


Figura 4.2: Soluciones factibles/infactibles para el problema 8x4S10

que cuando se reduce la capacidad en 14 %, la cantidad de instancias factibles es 37, pero cuando este porcentaje llega al 15 %, la cantidad de instancias factibles sube a 40. Una razón para este comportamiento podría ser la aleatoriedad de los valores, ya que en cada iteración se escoge un valor aleatorio para la capacidad. Por lo tanto, en algunas ocasiones esta aleatoriedad podría influir negativamente en el análisis de la solución.

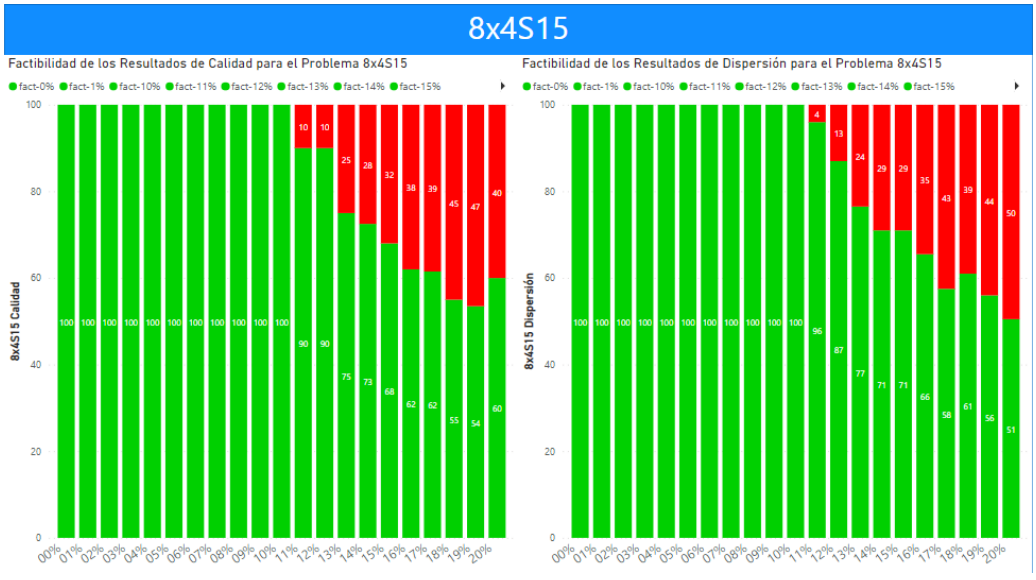


Figura 4.3: Soluciones factibles/infactibles para el problema 8x4S15

Si se aumenta el valor de holgura a 15 (figura 4.3), la factibilidad de las soluciones también aumenta. En este caso la solución generada seguirá siendo factible si la capacidad de las puertas se reduce hasta un 10 %, incluso en general también la factibilidad es mejor en comparación con la instancia *8x4S10* (figura 4.2).

En el caso de la figura 4.4 se puede observar que el aumento de la cantidad de camiones (*9x4S15*) influye negativamente en la factibilidad de la solución (en comparación con el problema *8x4S15*, figura 4.3), ya que la solución se vuelve viable si la capacidad se reduce hasta un 3 %, a partir de este porcentaje la proporción de soluciones infactibles va

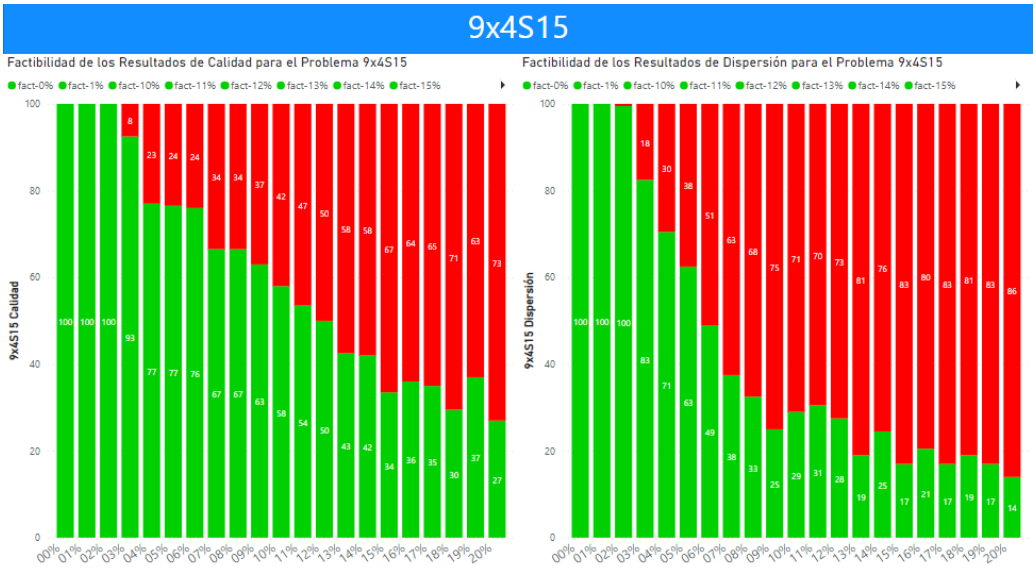


Figura 4.4: Soluciones factibles/infactibles para el problema 9x4S15

aumentando.

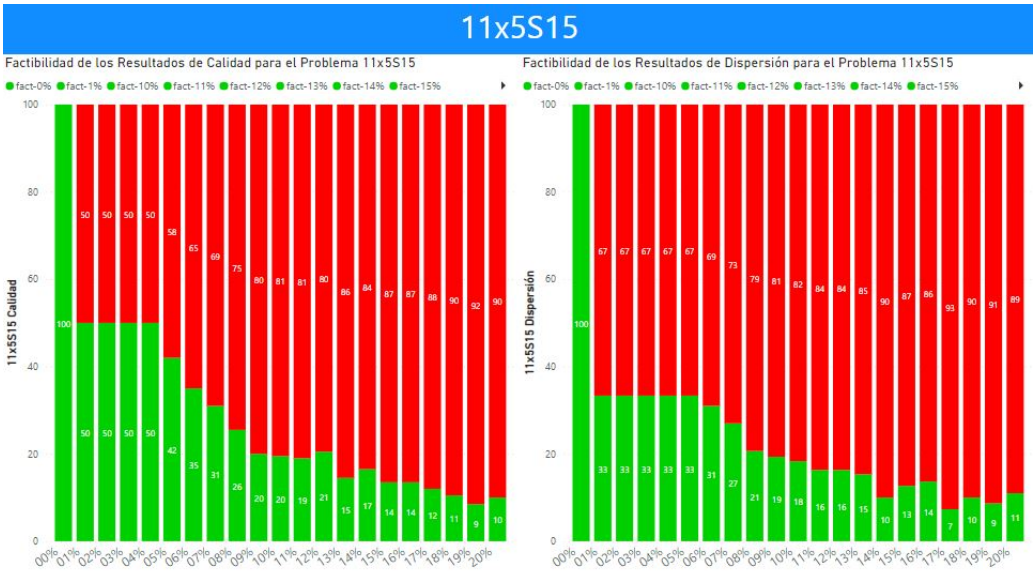


Figura 4.5: Soluciones factibles/infactibles para el problema 11x5S15

Si se examina la solución proporcionada para el problema *11x5S15* (figura 4.5), se puede notar que no permite una reducción considerable de capacidad. Incluso con una disminución mínima, comienzan a aparecer más instancias infactibles que factibles. Por lo tanto, se podría decir que el nivel de factibilidad de esta solución es baja.

Al analizar el problema *11x5S30* (figura 4.6), se puede observar que el aumento del valor de holgura influye positivamente en el cálculo de la solución, aunque éste no sea muy significativo. Esto puede ser debido a que, un valor de holgura mayor permite una mayor flexibilidad en la asignación de camiones a puertas, y esto puede influir en la factibilidad de la solución ya que las puertas de entrada y salida disponen de un mayor margen, lo que aumenta las posibilidades de encontrar una asignación factible.

En cuanto al problema *12x5S10*, se puede observar en la figura 4.7 un comportamiento

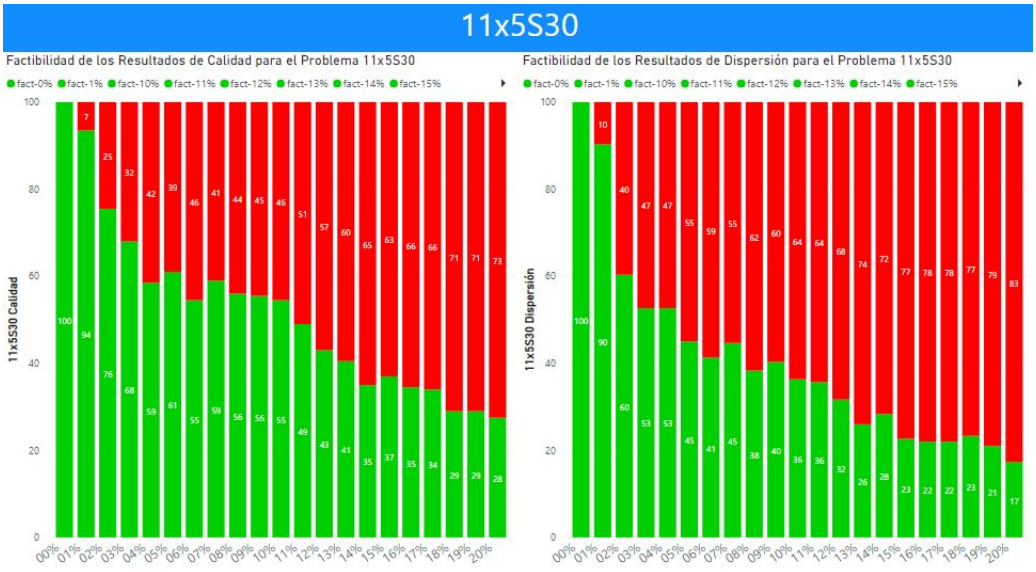


Figura 4.6: Soluciones factibles/infactibles para el problema 11x5S30

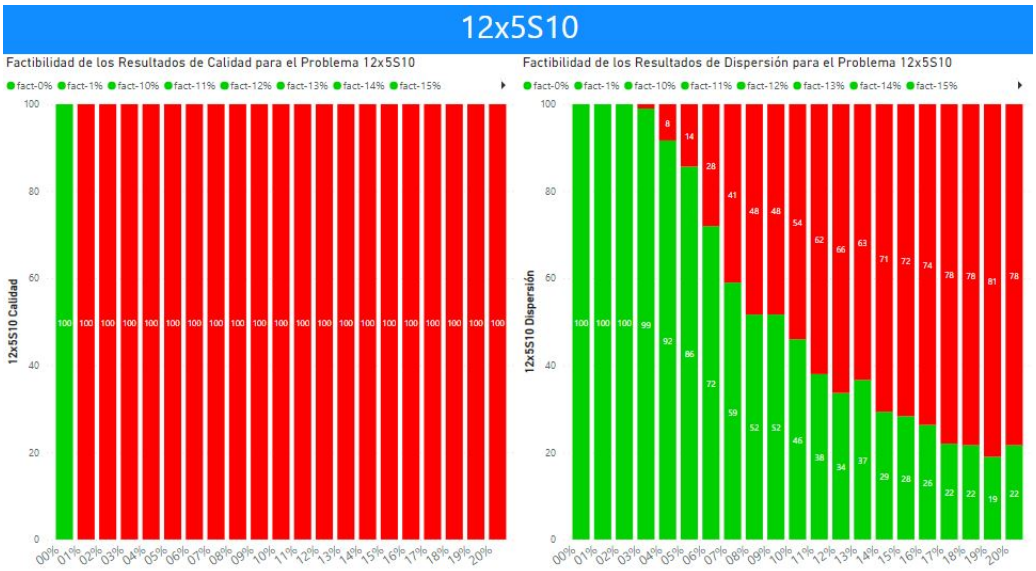


Figura 4.7: Soluciones factibles/infactibles para el problema 12x5S10

inusual, ya que a partir de una reducción mínima de capacidad, no hay ninguna instancia factible entre las 100 realizadas. Una razón podría ser la aleatoriedad del valor de la capacidad (explicado anteriormente en este apartado), pero también podría ser por otros parámetros y características específicas del problema, por ejemplo por la cantidad de *pallets* que se deben gestionar de un camión en concreto.

En cuanto al problema *20x10S10* (figura 4.8), se puede observar que la cantidad de instancias infactibles es relativamente mayor a las factibles. Una de las razones de este comportamiento podría ser el tamaño del problema, que es considerablemente grande. A medida que el tamaño del problema aumenta, la complejidad de éste también sube, con lo cual se dificulta la obtención de una solución factible.

En conclusión, hay muchos factores que afectan a la factibilidad de las soluciones. En primer lugar, el tamaño del problema influye bastante en la factibilidad de las soluciones

Figura 4.8: Soluciones factibles/infactibles para el problema 20x10S10

obtenidas, ya que a medida que aumenta el tamaño del problema, también sube la complejidad y la cantidad de restricciones.

Además, el parámetro anterior también podría verse afectado por la capacidad inicial asignada a cada puerta. Por ejemplo, si el tamaño del centro es pequeño y la capacidad de las puertas de entrada y salida es suficiente para manejar la descarga/carga de la mercancía, se pueden encontrar soluciones de mayor calidad. Sin embargo, un tamaño de centro grande y una capacidad inferior puede afectar a la factibilidad de la solución debido al aumento de la complejidad del problema. Otra cosa que se se observó en el análisis de la factibilidad es que, un valor de holgura mayor permite aumentar la flexibilidad a la hora de asignar camiones a puertas.

4.3. Conclusión

Una de las cualidades que debe tener una solución de calidad es un alto nivel de flexibilidad, es decir, debe ser factible en situaciones de último momento donde la capacidad de las puertas se ve reducida. Por lo tanto, en este capítulo se realiza un análisis de factibilidad de las soluciones generadas.

Se observa que la factibilidad de las soluciones está influenciada por varios factores, como el tamaño del problema, la capacidad de las puertas y el porcentaje de holgura. Por lo tanto, es importante considerar estos factores a la hora de diseñar algoritmos para el CDAP.

Capítulo 5

Visualización de Soluciones

Hasta ahora se han aboradado diversos aspectos relacionados con la solución del problema de Cross-dock Door Assignment Problem (CDAP), como el algoritmo utilizado para generarla, el tiempo requerido para ello, su nivel de optimización, la cantidad de iteraciones realizadas y la factibilidad de ésta.

Una vez hecho lo anterior, el siguiente paso sería representar la solución gráficamente, es decir, la asignación de camiones a puertas. Para ello, se ha creado una herramienta de visualización que ilustre cómo se han asignado los proveedores a las puertas de entrada, y los clientes a las puertas de salida. Además, si un camión de entrada debe enviar mercancía a un camión de salida, ambos estarán conectados visualmente, indicando así la ruta que debe seguir la mercancía desde la puerta de entrada donde se encuentra el proveedor hasta la puerta de salida donde se encuentra el cliente. Esto permitirá mostrar de manera clara y concisa cómo se ha establecido el flujo interno de mercancías en el centro *cross-dock*.

Dicha herramienta se implementará como una aplicación *web* usando el *framework ReactJS*. Una de las razones para la elección de éste fue su amplia comunidad de desarrolladores y una gran variedad de librerías y herramientas. La aplicación *web* solicitará al usuario dos archivos: uno que contenga la definición del problema y otro que contenga la solución generada para ese problema.

5.1. Estructura de la Definición del Problema

El archivo que contiene la definición del problema debe incluir la siguiente información:

- Número total de proveedores y clientes, así como la cantidad de puertas de entrada y salida de los que dispone el centro.
- Una matriz de tamaño $M \times N$, donde M será el número de camiones de entrada y N será el número de camiones de salida. En dicha matriz se debe representar la cantidad de *pallets* que el proveedor m quiere enviar al cliente n, siendo m el ID del proveedor y n el ID del cliente.
- La matriz de distancias de tamaño $I \times J$, donde I será la cantidad de puertas de

entrada y J la cantidad de puertas de salida. Se debe representar la distancia entre la puerta de entrada i y la puerta de salida j.

- Número total de *pallets* que lleva cada proveedor.
- Cantidad de *pallets* que puede gestionar cada puerta de entrada (capacidad).
- Número total de *pallets* que debe llevar cada cliente después de cargar la mercancía correspondiente.
- Cantidad de *pallets* que puede gestionar cada puerta de salida (capacidad).

Archivo 1 Definición del problema

```
8 4 4 8

0 0 16 13 0 0 0 0
0 0 34 0 0 10 0 0
0 37 0 0 0 0 0 0
27 0 0 0 0 0 0 0
0 0 0 0 27 16 0 0
0 0 0 0 0 0 41 44
0 0 10 0 44 0 44 0
0 0 14 0 35 0 15 0

8 9 10 11
9 8 9 10
10 9 8 9
11 10 9 8

29 44 37 27 43 85 98 64

113 113 113 113

27 37 74 13 106 26 100 44

113 113 113 113
```

En el archivo 1 se muestra un ejemplo del archivo que contiene la definición de un problema. De este archivo se puede sacar la siguiente información:

- El problema contiene 8 proveedores, 4 puertas de entrada, 4 puertas de salida y 8 clientes.
- La matriz de tamaño 8×8 que contiene la cantidad de *pallets* que un proveedor envía a un cliente. Por ejemplo: el proveedor 1 envía 16 *pallets* al cliente 3 y 13 *pallets* al cliente 4.
- La matriz de distancias de tamaño 4×4. La distancia entre la puerta de entrada 2 y la puerta de salida 4 es de 10 unidades.

- Número de *pallets* que lleva cada proveedor. Por ejemplo: el proveedor 6 tiene 85 *pallets*.
- Capacidad de cada puerta, que en este caso es 113.
- Número de *pallets* que debe llevar cada cliente. Por ejemplo: el cliente 5 debe llevar 106 *pallets*.

De este fichero se extraerá, principalmente, la matriz que contiene la cantidad de mercancía que un proveedor envía a un cliente. Esto ayudará a determinar la relación entre proveedores y clientes.

Es importante tener en cuenta que, al igual que ocurre en situaciones reales, un proveedor puede tener mercancía para más de un cliente, y un cliente también puede recibir mercancía de más de un proveedor.

5.2. Estructura de la Solución Generada

El archivo que contiene la solución del problema debe incluir el nombre de la instancia, el valor óptimo obtenido, el tiempo tardado, la asignación del proveedor a la puerta de entrada y del cliente a la puerta de salida. Estos datos serán fundamentales para visualizar la asignación de camiones a puertas en la aplicación *web*.

Archivo 2 Solución del problema

```
Instance name: 8x4x5x25_1.txt
Optimal value: 3633
CPU time: 0.055

Proveedor: 1 2 3 4 5 6 7 8
Puerta:    1 1 1 2 4 2 3 4
Cliente: 1 2 3 4 5 6 7 8
Puerta:  2 1 1 2 4 2 3 2 Asignacion de proveedores a puertas de entrada
Proveedores en la puerta 1: 1 2 3
Proveedores en la puerta 2: 4 6
Proveedores en la puerta 3: 7
Proveedores en la puerta 4: 5 8

Asignacion de clientes a puertas de salida
Clientes en la puerta 1: 2 3
Clientes en la puerta 2: 1 4 6 8
Clientes en la puerta 3: 7
Clientes en la puerta 4: 5
*****************************
```

En el archivo 2 se puede ver un ejemplo de la solución generada para el problema, cuyas especificaciones se encuentran en el archivo 1 descrito en el apartado anterior. Tiene el siguiente contenido:

- Nombre de la instancia, que es *8x4x5x25_1.txt*
- El valor óptimo del problema, que en este caso es 3633.
- El tiempo de cómputo, que es $0{,}055$ segundos.
- ID de los proveedores.
- Asignación de puertas de entrada a cada proveedor. Por ejemplo: a los proveedores con ID 1, 2 y 3 se les ha asignado la puerta 1.
- ID de los clientes.
- Asignación de puertas de salida a cada cliente. Por ejemplo: a los clientes con ID 1, 4, 6 y 8 se les ha asignado la puerta 2.

5.3. Visualización

En la figura 5.1 se muestra lo primero que el usuario puede visualizar al entrar a la aplicación *web*, es decir, el título y un selector de archivos donde el usuario debe seleccionar el archivo que contiene las especificaciones del problema, cuya estructura debe ser la que se especificó en el apartado 5.1.

Una vez hecho lo anterior, el programa se encarga de leer el archivo y extraer la información necesaria, que en este caso sería la matriz que representa la cantidad de *pallets* que cada proveedor debe enviar a cada cliente. Si todo se ha procesado correctamente, se muestra al usuario otro selector de archivos. En este caso, debe especificar el archivo que contiene la solución generada para el problema anterior, cuya estructura debe ser la que se especificó en el apartado 5.2.

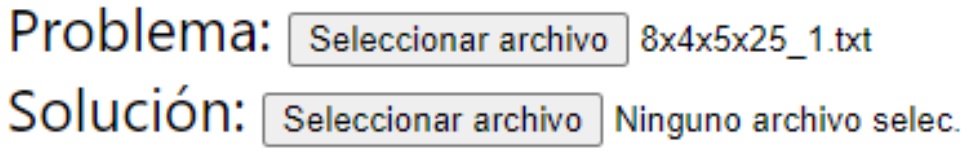


Figura 5.1: Visualizador de soluciones: selección del archivo de solución

Finalmente, si ambos archivos se han leído correctamente, se muestra la asignación de camiones a puertas de manera gráfica, como se puede observar en la figura 5.2. Cada puerta se representa en forma de un rectángulo, las que se encuentran en el lado izquierdo de la visualización son las puertas de entrada, mientras que las del lado derecho son puertas de salida. Dentro de cada rectángulo se especifican los ID de los camiones que se han asignado a la puerta correspondiente. Por ejemplo: a la primera puerta de entrada se han asignado los camiones de entrada (proveedores) con ID 1, 2 y 3, y a la tercera puerta de salida se ha asignado el camión de salida (cliente) con ID 7.

Si un proveedor tiene mercancía para un cliente, se muestra una flecha que une a éstos. Por ejemplo: el proveedor con ID 1, que se encuentra en la primera puerta de entrada,

tiene mercancía para los clientes con ID 3 (ubicado en la primera puerta de salida) e ID 4 (ubicado en la segunda puerta de salida), por lo tanto la primera puerta de entrada estará unida mediante una flecha con la primera y segunda puerta de salida.

Ademas, se puede observar en la parte baja de la visualización, información adicional sobre ésta, como el nombre de la instancia, el valor óptimo calculado para la solución y el tiempo que se tardó en generarla.

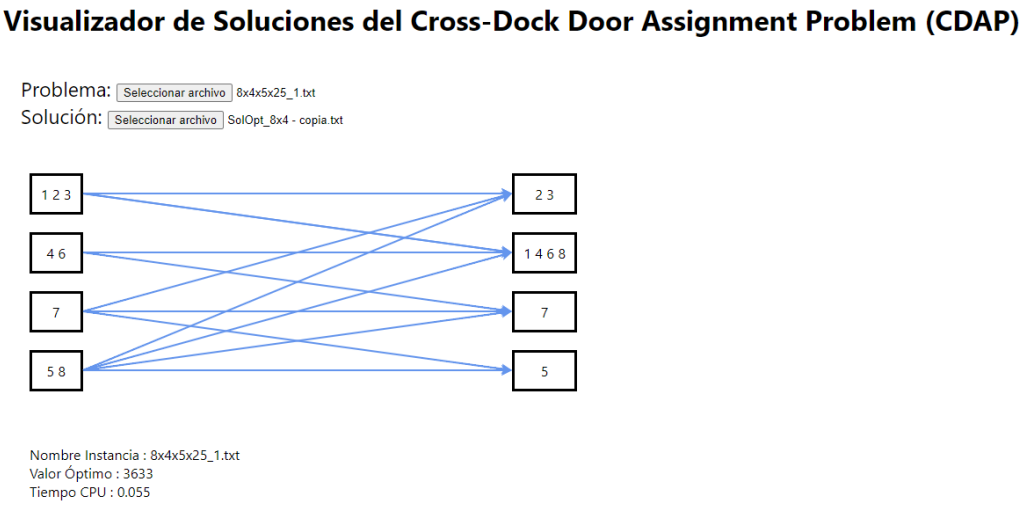


Figura 5.2: Visualizador de soluciones: visualización de la solución del problema

5.4. Conclusión

En este capítulo se explica la herramienta desarrollada para visualizar las soluciones de una manera gráfica, es decir, la asignación de camiones a puertas de un centro *cross-dock*. La aplicación web, implementada con el *framework ReactJS*, permite al usuario cargar dos archivos: uno que contiene la definición del problema y otro que contiene la solución generada para ese problema. Una vez que se cargan correctamente estos archivos, se muestra la asignación de camiones a puertas y las relaciones entre los proveedores y clientes que tienen mercancía correspondiente.

Esta herramienta permite analizar y comprender, de manera intuitiva, las soluciones generadas, que podría ser de gran utilidad para tomar decisiones y optimizar la operación lógistica en los centros.

Capítulo 6

Conclusiones y líneas futuras

En este trabajo, se ha evaluado el rendimiento de los algoritmos metaheurísticos usando diferentes datos y métricas, como son el tamaño del problema, el valor objetivo de la solución obtenida, el tiempo de cómputo y el gap entre la solución obtenida y la óptima. En total se han evaluado 44 instancias con diferentes tamaños de problema, realizando análisis específico para cada uno de éstos. Una vez hecho lo anterior se observó que la cantidad de camiones y puertas influye bastante, tanto en la obtención de una solución óptima como en el tiempo de cómputo necesario para llegar a ese valor. Por lo tanto es muy necesario controlar estos aspectos ya que en casos reales el tamaño del problema suele ser bastante grande.

En el análisis general de los algoritmos se intentó analizar el rendimiento de los algoritmos sin tener en cuenta el tamaño del problema. Para obtener un valor único de cada algoritmo, se calculó el promedio de los valores generados por cada algoritmo para cada instancia, y de esos valores se utilizó el *Gap*, es decir, la diferencia entre la solución generada por cada algoritmo y la óptima. Se encontró que algunos algoritmos destacaban por sus soluciones de mejor calidad, mientras que otros por un tiempo de cómputo inferior. Por lo tanto, la elección del algoritmo depende de las necesidades y restricciones del problema.

Además, se realizó un análisis de la factibilidad de las soluciones al ser simuladas, considerando la capacidad de las puertas como una variable aleatoria con distribución de probabilidad uniforme. De esta manera, se analizó la robustez de las soluciones ante la aparición eventos disruptivos que afecten a la capacidad máxima de las puertas. Se observó que el tamaño del problema afecta en la factibilidad de las soluciones, ya que a medida que aumenta el tamaño, también crece la complejidad y la cantidad de restricciones. Otro factor que también influye es la capacidad inicial asignada a cada puerta.

Finalmente, se desarrolló una aplicación *web* para facilitar la visualización de las soluciones generadas. Esta herramienta permite a los usuarios cargar instancias del problema y la solución generada por el algoritmo, y visualizar esta información de manera gráfica.

En cuanto a líneas de trabajo futuras, hay varios trabajos que se podrían llevar a cabo para continuar y ampliar este proyecto. En este trabajo se observó que, para tamaños de problemas mayores, los algoritmos analizados proporcionaban soluciones de peor

calidad. Una solución podría ser aumentar el número de iteraciones, pero esto requerirá un mayor tiempo de cómputo. Por lo tanto, se podría trabajar en mejorar estos algoritmos, de tal manera que puedan proporcionar soluciones de mejor calidad para problemas con un mayor tamaño. También se podrían generar nuevos algoritmos que resuelvan este problema.

En este caso también se evaluó la efectividad de los algoritmos usando instancias de prueba. Por lo tanto, otra línea de trabajo podría ser el análisis de los algoritmos usando un caso real de un centro *cross-dock*. Se podrían generar soluciones para estos casos usando los algoritmos metaheurísticos, y comparar el resultado con asignaciones reales que se estén realizando en ese momento en el centro.

Capítulo 7

Summary and Conclusions

In this work, the performance of metaheuristic algorithms has been evaluated using different data and metrics, such as problem size, objective value of the obtained solution, computation time, and the gap between the obtained solution and the optimal solution. A total of 44 instances with varying problem sizes were evaluated, conducting specific analyses for each of them. It was observed that the number of trucks and gates significantly influence both the attainment of an optimal solution and the computation time required to reach that value. Therefore, it is crucial to control these aspects as real-life problem sizes tend to be quite large.

In the overall analysis of the algorithms, an attempt was made to assess their performance regardless of the problem size. To obtain a single value for each algorithm, the average of the values generated by each algorithm for each instance was calculated, and the Gap, which represents the difference between the solution generated by each algorithm and the optimal solution, was utilized. It was found that some algorithms stood out for their solutions of higher quality, while others excelled in terms of shorter computation time. Therefore, the choice of algorithm depends on the needs and constraints of the problem.

Furthermore, a feasibility analysis of the solutions was conducted by simulating them, considering the gate capacity as a random variable with a uniform probability distribution. This allowed for an examination of the robustness of the solutions in the face of disruptive events affecting the maximum capacity of the gates. It was observed that problem size affects the feasibility of the solutions, as larger sizes lead to increased complexity and a greater number of constraints. Another influencing factor is the initial capacity assigned to each gate.

Finally, a web application was developed to facilitate the visualization of the generated solutions. This tool enables users to upload problem instances and the solution generated by the algorithm, and visualize this information in a graphical manner.

As for future lines of work, several possibilities exist to continue and expand upon this project. It was observed in this work that the analyzed algorithms provided lower-quality solutions for larger problem sizes. One possible solution could be to increase the number of iterations, although this would require more computation time. Therefore, efforts could be made to enhance these algorithms, allowing them to provide higher-quality solutions for larger problems. Additionally, new algorithms could be developed to address this

problem.

In this case, the effectiveness of the algorithms was also evaluated using test instances. Hence, another line of work could involve analyzing the algorithms using a real-life case of a cross-docking center. Solutions could be generated for these cases using metaheuristic algorithms and compared with actual assignments being carried out at the center.

Capítulo 8

Presupuesto

El presupuesto estimado para el desarrollo del trabajo se ha calculado teniendo en cuenta, principalmente, la duración total del proyecto, que fue de 300 horas. Esta cifra está en línea con lo establecido en la Resolución de 21 de marzo de 2011, de la Universidad de La Laguna, donde se determina una duración de 12 créditos ECTS para la asignatura *Trabajo Fin de Grado* del *Grado en Ingeniería Informática*. Se considera que cada crédito ECTS equivale a 25 horas de trabajo, por lo que se obtiene un total de 300 horas para el proyecto (12 *créditos ECTS* $\times$ 25 *horas* $=$ 300 *horas*).

En cuanto al coste total del personal involucrado, se ha tenido en cuenta un salario de 17€ por hora, con lo cual se obtiene un coste total de 5100€ (300 *horas* $\times$ 17€ $=$ 5100€)

Bibliografía

Alancay, N., Villagra, S., Villagra, N.A., 2016. Algoritmos metaheurísticos trayectoriales para optimizar problemas combinatorios. Informe Científico Técnico UNPA 8, 56–75.

Apte, U.M., Viswanathan, S., 2000. Effective cross docking for improving distribution efficiencies. International Journal of Logistics 3, 291–302.

Bartholdi, J.J., Gue, K.R., 2004. The best shape for a crossdock. Transportation science 38, 235–244.

Boysen, N., de Koster, R., Weidinger, F., 2019. Warehousing in the e-commerce era: A survey. European Journal of Operational Research 277, 396–411.

Buijs, P., Vis, I.F., Carlo, H.J., 2014. Synchronization in cross-docking networks: A research classification and framework. European Journal of Operational Research 239, 593–608.

Coppola, D., 2021. Annual retail e-commerce sales growth worldwide from 2017 to 2025. Statista .

Essghaier, F., Allaoui, H., Goncalves, G., 2021. Truck to door assignment in a shared cross-dock under uncertainty. Expert Systems with Applications 182, 114889.

Fakhravar, H., 2022. Combining heuristics and exact algorithms: A review. arXiv preprint arXiv:2202.02799 .

Guemri, O., Nduwayo, P., Todosijević, R., Hanafi, S., Glover, F., 2019. Probabilistic tabu search for the cross-docking assignment problem. European Journal of Operational Research 277, 875–885.

Guignard, M., Hahn, P.M., Pessoa, A.A., da Silva, D.C., 2012. Algorithms for the cross-dock door assignment problem, in: Proceedings of the fourth international workshop on model-based metaheuristics, pp. 145–162.

Hansen, P., Mladenović, N., Moreno Pérez, J.A., 2010. Variable neighbourhood search: Methods and applications. Annals of Operations Research 175, 367 – 407.

Nassief, W., Contreras, I., As' Ad, R., 2016. A mixed-integer programming formulation and lagrangean relaxation for the cross-dock door assignment problem. International Journal of Production Research 54, 494–508.

Nduwayo, P., 2020. Mathematical programming formulations and algorithms for the cross-dock door assignment problem. Ph.D. thesis. Université Polytechnique Hauts-de-France.

Pentico, D.W., 2007. Assignment problems: A golden anniversary survey. European Journal of Operational Research 176, 774–793.

Tarhini, A.A., Yunis, M.M., Chamseddine, M., 2016. Natural optimization algorithms for the cross-dock door assignment problem. IEEE Transactions on intelligent transportation systems 17, 2324–2333.

Tsui, L.Y., Chang, C.H., 1990. A microcomputer based decision support tool for assigning dock doors in freight yards. Computers & Industrial Engineering 19, 309–312.

Van Belle, J., Valckenaers, P., Cattrysse, D., 2012. Cross-docking: State of the art. Omega 40, 827–846.

Wang, H., Alidaee, B., 2019. The multi-floor cross-dock door assignment problem: Rising challenges for the new trend in logistics industry. Transportation Research Part E: Logistics and Transportation Review 132, 30–47.

Yamada, T., Nasu, Y., 2000. Heuristic and exact algorithms for the simultaneous assignment problem. European Journal of Operational Research 123, 531–542.

Zhu, Y.R., Hahn, P.M., Liu, Y., Guignard, M., 2009. New approach for the cross-dock door assignment problem, in: in Proceedings of the XLI Brazilian Symposium on Operations Research.