Máster en Ingeniería de Sistemas y de Control

Trabajo Fin de Máster

Optimización de envíos a corto plazo en la planta de Mercedes Benz de España

---

Jorge Salazar Beitia

Dirección

Eva Besada Portas

Jose Antonio López Orozco

Año académico: 2024-2025

Convocatoria: Febrero

Resumen

A lo largo de este Trabajo de Fin de Máster, se ha desarrollado una solución para optimizar la gestión logística del envío de vehículos a los diferentes destinos desde la fábrica de Mercedes-Benz en Vitoria-Gasteiz, utilizando algoritmos genéticos multiobjetivo. Para ello, se ha diseñado un modelo de optimización con tres objetivos principales: la ocupación eficiente de la campa de almacenamiento de la fábrica, la minimización de los costes de almacenamiento en los puertos y la regularidad en el número de vehículos transportados. Estos objetivos están interrelacionados entres sí y pueden entrar en conflicto.

El problema planteado es de naturaleza compleja, multiobjetivo y no lineal, con variables discretas y restricciones interdependientes. Debido a esta complejidad, los métodos de optimización tradicionales no resultan adecuados, lo que justifica la elección de algoritmos genéticos. Estas técnicas permiten explorar espacios de búsqueda extensos, evitar óptimos locales y aproximarse a soluciones cercanas al óptimo global.

A través de un enfoque en múltiples fases, se han implementado dos algoritmos genéticos basados en una función de aptitud (*fitness*) y operadores de selección, cruce y mutación, permitiendo iterar sobre una población de soluciones con el fin de encontrar la más adecuada para cada escenario, considerando las prioridades definidas por el Departamento de Logística de Envíos de la empresa.

Los resultados obtenidos con las implementaciones de los algoritmos han sido satisfactorios, tanto en términos de calidad de las soluciones generadas como en el rendimiento de los algoritmos. Por ello, se considera que los algoritmos genéticos son una herramienta eficaz para abordar este tipo de problemas, permitiendo obtener soluciones de alta calidad que, combinadas con la experiencia de los expertos, pueden contribuir a la toma de decisiones estratégicas óptimas.

PALABRAS CLAVE: Logística, gestión óptima de envíos, optimización, programación no lineal multiobjetivo, algoritmos genéticos.

Índice de contenidos

Índice de figuras

Índice de tablas

CAPÍTULO 1

INTRODUCCIÓN

La función logística ha sido considerada un proceso rutinario durante muchos años, que consistía, únicamente, en hacer llegar los productos al cliente final desde los centros de producción. Con la llegada de la globalización de la economía y la consiguiente apertura de nuevos mercados distanciados geográficamente, la logística ha ido evolucionando hacia nuevas dimensiones y se ha convertido en un proceso transversal complejo que permite posicionar a las empresas y mejorar considerablemente la satisfacción del cliente. En este sentido, una buena gestión logística, además de reducir tiempos de entrega y optimizar costes, es considerada como un elemento clave para generar valor para los clientes, los proveedores y los accionistas de la empresa [1].

En el ámbito de la logística, los modelos de optimización constituyen un instrumento eficaz para la toma de decisiones informadas basadas en algoritmos avanzados y datos. Estos modelos permiten encontrar soluciones óptimas a problemas complejos como la planificación de rutas de transporte, la gestión de la cadena de suministro, la gestión de inventario, la disposición de depósitos de mercancías, etc., mejorando la eficiencia y reduciendo costes.

1.1. Objetivos

El objetivo general del Trabajo Fin de Máster (TFM en adelante) es la adaptación y aplicación de un modelo de optimización del proceso logístico en el Departamento de Envíos de la fábrica de Mercedes-Benz en Vitoria. A nivel general, se plantea realizar un análisis detallado de un proceso complejo y dinámico para proponer una solución que permita alcanzar la eficiencia operativa, minimizar costes y garantizar la entrega sin demoras de los vehículos a los destinos finales.

Para llevar a cabo este objetivo general, se plantean los siguientes objetivos específicos:

- Estudiar el problema de optimización del proceso logístico y su formulación matemática, especificando sus objetivos y prestando especial atención a la compresión de los elementos que lo constituyen y las restricciones a las que está sujeto.

- Familiarizarse con las características y en el manejo de la metodología propuesta para resolver el problema de optimización: los algoritmos genéticos.
- Implementar dos tipos diferentes de algoritmos genéticos, evaluar sus resultados, analizar la eficacia y la eficiencia de estos dos algoritmos en la solución del problema propuesto.
- Realizar un análisis comparativo del desempeño de los algoritmos implementados.
- Interpretar los resultados del análisis realizado teniendo en cuenta sus limitaciones.

1.2. Organización del trabajo

Tras este capítulo introductorio, el trabajo se organiza en los siguientes capítulos:

- El capítulo 2 presenta el contexto en el que se enmarca el problema. Por un lado, se presenta la empresa y se realiza una breve descripción del problema de optimización a desarrollar y, por otro lado, se exponen las principales características de la metodología que se utilizará en la búsqueda de la solución óptima: los algorítmos genéticos.
- En el capítulo 3 se detalla el problema de optimización del proceso logístico. Se describe el problema y se definen los objetivos. Se realiza además la descripción matemática del problema de optimización de la red logística desde la fábrica de mercedes Benz a los diferentes destinos, que incluye, las variables de decisión, los parámetros y las constantes, las restricciones y las funciones objetivo.
- En el capítulo 4 se describen las diferentes etapas en la implementación de los algoritmos genéticos empleados en la resolución del problema de optimización descrito previamente.
- En el capítulo 5 se presentan los resultados y se analiza la adecuación, utilidad y facilidad de configuración y programación de los métodos utilizados.
- El capítulo 6 se presentan las conclusiones y las principales limitaciones del análisis realizado.

CAPÍTULO 2

EL CONTEXTO

2.1. La empresa

El Trabajo Fin de Máster se ha desarrollado en el marco del Convenio de Colaboración entre la Universidad Complutense de Madrid y la empresa Mercedes-Benz AG para la realización del Trabajo Fin de Máster, durante el periodo comprendido entre el 1 de octubre de 2023 y el 30 de abril de 2024.

Mercedes-Benz AG es una empresa alemana que se dedica a la fabricación de vehículos. Mercedes-Benz AG es una subsidiaria de la compañía Mercedes-Benz Group y es ampliamente conocida por la fabricación de automóviles de lujo, deportivos, autobuses, camiones, vehículos utilitarios deportivos y furgonetas.

La planta de Mercedes-Benz en Vitoria, donde se ha realizado este TFM, es la segunda planta de fabricación de furgonetas de Mercedes-Benz AG a nivel mundial y es la planta líder en la producción de furgonetas de tamaño medio a nivel global, incluyendo diferentes modelos. Creada en 1954 y con una expansión continua desde entonces hasta convertirse en un centro de producción de vanguardia, la fábrica de Vitoria desempeña un papel crucial en la coordinación de la producción y distribución a nivel internacional, abarcando, desde la estrategia de producción, hasta la gestión logística y de calidad. Actualmente, la planta vitoriana es uno de los centros industriales más destacados del País Vasco, contribuyendo de manera significativa al desarrollo económico de la región, dando empleo a alrededor de 4.800 personas.

La fábrica vitoriana es un complejo industrial compuesto por diversas áreas de producción y gestión que desempeñan un papel crucial en la creación de vehículos de alta calidad. El área de producción se divide en tres etapas fundamentales: Montaje Bruto, Pintura y Montaje Final del vehículo, cada una de las cuales tiene su propia ubicación en naves separadas. Un aspecto interesante de esta fábrica es que no se involucra en el proceso de estampación de piezas, ya que las partes esenciales, como las puertas, portones delanteros y traseros, y los largueros laterales, llegan a la planta preformadas y listas para ensamblar.

El Montaje Bruto es el primer paso en la línea de producción, donde se lleva a cabo la unión de las piezas de la carrocería utilizando técnicas de soldadura, pegado y sellado. Una vez que la carrocería toma forma, se traslada a la nave de Pintura, donde recibe un trata-

miento superficial para prepararla para el proceso de pintura, se aplican capas de pintura que proporcionan protección contra la corrosión y el desgaste. La carrocería pintada se dirige a la nave de Montaje Final, donde se realiza el ensamblaje de los diversos componentes que constituyen el vehículo. En esta etapa, los trabajadores realizan operaciones como la instalación de interiores, sistemas eléctricos, neumáticos y otros componentes esenciales. Además del ensamblaje, se llevan a cabo tareas adicionales, como el llenado de líquidos, la aplicación de ceras protectoras y los retoques finales. Se realiza una prueba de estanqueidad para garantizar que el vehículo no tenga fugas y esté en condiciones óptimas. Por último, la fábrica realiza controles exhaustivos e inspecciones finales para asegurarse de que cada vehículo cumple con los estándares de calidad establecidos. Una vez superadas estas pruebas, los vehículos están listos para ser expedidos al mercado.

En la Figura 2.1 mostramos las diversas entradas y salidas del proceso.

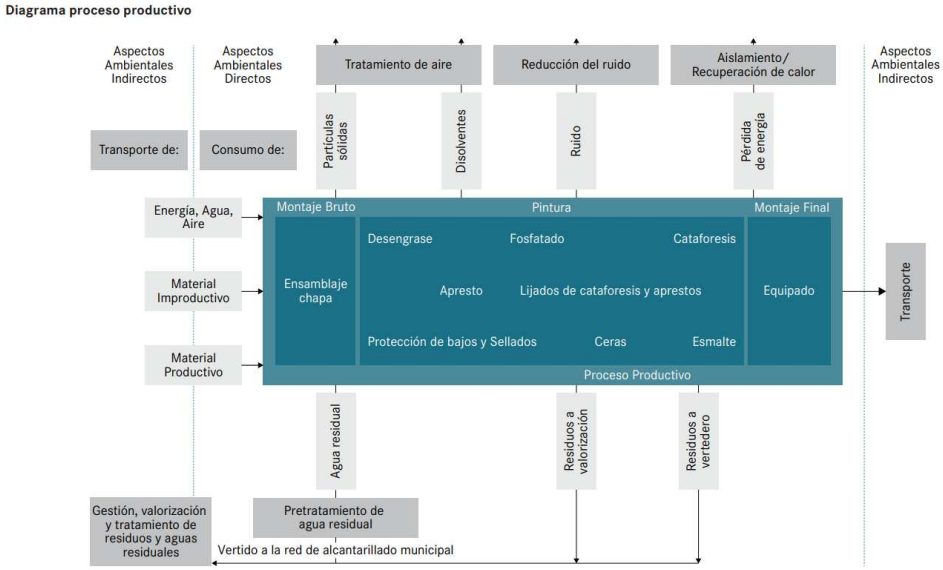

Figura 2.1: Esquema(Organigrama) entradas y salidas del proceso.

2.1.1. La cadena logística

La fabricación y distribución de furgonetas implica una serie de operaciones complejas que van mucho más allá del mero ensamblaje. La Logística es la parte del proceso de gestión de la cadena de suministro encargada de la planificación, implementación y control eficiente del flujo de materiales y productos terminados, así como el flujo de información relacionada desde el punto de origen hasta el punto de destino cumpliendo al máximo con las necesidades de los clientes y generando los mínimos costes operativos.

Como se puede observar en la Figura 2.2, en la estructura Mercedes Benz Vans, la cadena logística integra varios tipos de logística con funciones diferenciadas:

- **Logística de Disposición de Materiales**: esta función se encarga de gestionar la adquisición de piezas a través de sistemas informáticos de gestión de necesidades. Los

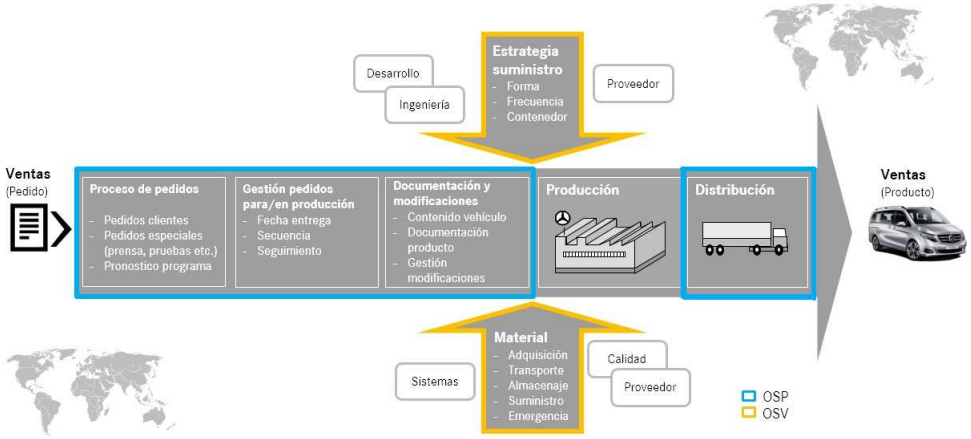

Figura 2.2: Esquema de la cadena logística

programas de suministro se envían a los proveedores de acuerdo a las necesidades de producción, especificando la cantidad requerida. Además, se realiza un seguimiento para asegurar la entrega en la cantidad y fecha programadas, utilizando sistemas informáticos compartidos con los proveedores para garantizar la trazabilidad y la transparencia de datos e información.

- **Logística de Transporte de Materiales**: es la que se encarga de organizar el transporte de las piezas desde el proveedor hasta la fábrica.
- **Logística interna (Flujo de Materiales)**: hace referencia a los flujos internos de los materiales dentro de la fábrica (descarga, recepción, almacenamiento, suministro, etc.).
- **Logística de embalajes**: se encarga de planificar, diseñar, fabricar y dimensionar los embalajes donde los proveedores van a suministrar diferentes piezas. A la hora de adjudicar un embalaje a una pieza, se debe utilizar la matriz que define el camino a seguir para llegar a un contenedor pequeño, a uno universal o a uno especial, en función de diferentes criterios (tamaño de la pieza, alcance, peso, etc.).
- **Logística inversa**: denominada también Expedición de Vacíos, se encarga de planificar los flujos de retorno de los embalajes vacíos hacia el proveedor.
- **Logística de transporte de vehículos**: se encarga de organizar el transporte de los vehículos terminados desde la fábrica hasta los centros de distribución y, de estos, al cliente final.

Si bien no existe un tipo de logística que pueda considerarse más importante que otro dentro de la cadena, el final de dicha cadena, la logística de transporte de vehículos, se puede considerar como una de las partes más críticas, ya que, por un lado, se trabaja con el producto terminado, con un mayor valor para la empresa y, por otro, la proximidad al cliente es mayor.

El modelo de optimización que se propone en este TFM, se enfoca en uno de los procesos de esta última fase, la cual pertenece a la división denominada VAN/OSO (Mercedes

Benz Vans Operations, Outbound and Global Logistics), división responsable de todas las operaciones necesarias para que los vehículos sean distribuidos desde las distintas factorías de la compañía hasta su destino final sin demoras. En la fábrica de Vitoria, esta división está representada por un único departamento, el departamento de Logística de Transporte de Vehículos (VAN/OSO/LPE) que, como veremos más adelante, es responsable de la primera etapa del proceso logístico de los vehículos fabricados en la planta, y el resto de departamentos de la división que participan en dicho proceso están ubicados en Alemania.

2.1.2. El departamento de Logística de Transporte de Vehículos

El departamento de Logística de Transporte de Vehículos de la fábrica de Vitoria (VAN/OSO/LPE) desempeña diversas funciones que son fundamentales para garantizar un flujo eficiente y sin contratiempos en el traslado de furgonetas desde la fábrica, hacia puertos y carroceros primero, y hacia su destino final en países de todos los continentes, posteriormente (Figura 2.3).

Figura 2.3: Esquema del transporte de vehículos desde la fábrica al destino final.

A continuación se detallan algunas de las principales responsabilidades y áreas de acción de este departamento:

- **Organización de Transporte:**
  - Gestión y disposición de transportes de furgonetas desde fábricas, puertos y carroceros hasta el destino, siguiendo los Incoterms (reglas internacionales que clarifican las responsabilidades y obligaciones del comprador y el vendedor en el transporte de mercancías) establecidos. Esta labor implica una cuidadosa coordinación para garantizar la fluidez en la entrega.
  - Planificación del transporte multimodal, abarcando camiones, barcos, ferrocarriles y aviones, junto con el desarrollo y optimización de toda la red logística. Aquí, se lleva a cabo una planificación estratégica para asegurar la eficiencia en los diferentes modos de transporte.
- **Calidad de Transporte:**
  - Formación y auditorías de carga en diversas etapas, desde las fábricas hasta los carroceros. El departamento se compromete a garantizar estándares de calidad durante todo el proceso logístico.
  - Gestión de daños de transporte, asegurando la integridad y calidad de los vehículos a lo largo de todo el proceso logístico. Se implementan medidas para minimizar y gestionar cualquier daño.

- **Gestión de Aduanas:** que abarca la gestión de aduanas, tanto de exportación como de importación, el sistema de declaración y procesamiento electrónico de rutas nacionales e internacionales y los sitemas de declaraciones Intrastat / Extrastat, donde se recopilan y gestionan datos estadísticos relacionados con los movimientos de mercancías.

- **Gestión de costes y otros:** que incluye la gestión de proyectos, la planificación de presupuestos, facturación, proveedores, gestión de operador logístico, la configuración y gestión de sistemas informáticos para optimizar los procesos logísticos, etc.

La dimensión operativa de la división se revela a través de diversos indicadores significativos. En este sentido, cabe desatacar que el suministro se extiende a prácticamente 100 países de los 5 continentes y se gestionan alrededor de 100 proveedores que abarcan camiones, barcos, trenes y transporte aéreo (ver Figura 2.4). Estos proveedores tienen la tarea de descargar en aproximadamente 1.600 destinos, englobando tanto la fábrica Vitoria como la fábrica Barcelona, junto con diversos carroceros en toda Europa. La coordinación de una red extensa de transporte es esencial.

El 50 % del volumen de fabricación se distribuye directamente a concesionarios en países como Alemania, Francia, España, Portugal y Austria. Diariamente, se cargan alrededor de 125 camiones en la fábrica, gestionando un flujo constante de aproximadamente 6.500 vehículos en tránsito en promedio.

El 85 % del volumen total de fabricación se transporta de manera multimodal, empleando una combinación eficiente de diferentes medios de transporte como camiones, barcos, ferrocarriles y transporte aéreo. El 15 % restante se transporta en camiones directamente desde la fábrica hasta el destino final en España, Portugal, Francia y Suiza principalmente.

La ruta marítima de Pasajes a Zeebrügge (Bélgica), atendida por 3-4 barcos semanales, se destaca al transportar alrededor de 2.000 vehículos por semana. Adicionalmente, con el objetivo de diversificar y flexibilizar la cadena de suministro, se cuenta con otros 4 puertos de salida, que incluyen Barcelona, Santander, y Valencia.

En términos de operaciones aduaneras, se realizan aproximadamente 40.000 exportaciones de vehículos, 1.000 importaciones de piezas, 13.000 tránsitos NCTS y 7.500 exportaciones de piezas.

La gestión financiera de este extenso entramado logístico implica un presupuesto sustancial de aproximadamente 70 millones de euros anuales.

Estos datos revelan la importancia estratégica del conjunto de operaciones logísticas y ponen de manifiesto la necesidad de implementar una gestión logística del transporte y la distribución que garantice que los vehículos lleguen a los clientes de la forma más eficiente posible. Con esto, no solo se satisface al cliente, sino que también se pueden lograr ahorros significativos en costes y tiempo e importantes ventajas competitivas para la empresa.

2.2. La gestión óptima de procesos logísticos

La gestión de los procesos logísticos implica planificar, coordinar y ejecutar el flujo de bienes y servicios desde el origen hasta el destino de forma que se optimice el uso de

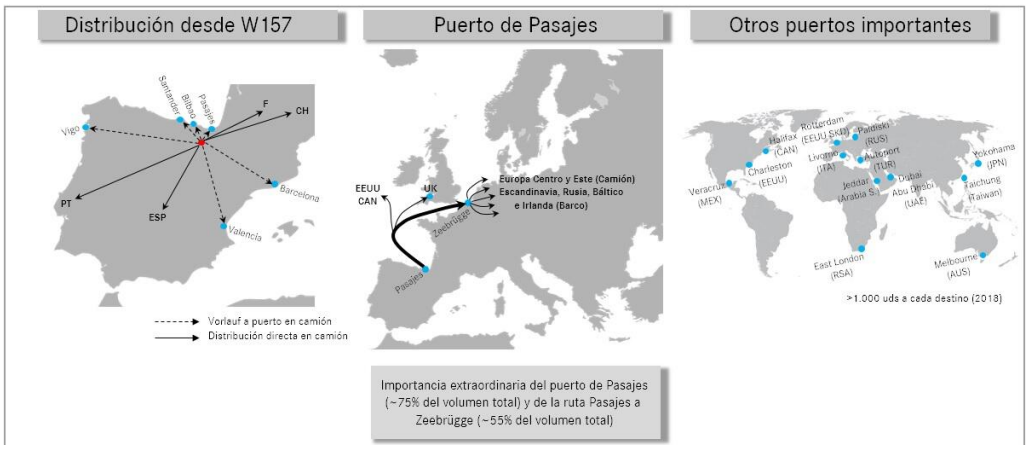

Figura 2.4: La cadena de distribución de la planta de Mercedes Benz en Vitoria

recursos, como modos de transporte, rutas, almacenes y stocks, al tiempo que satisface la demanda de los clientes y minimiza los costes, consiguiendo de esta manera la eficiencia operativa. Una red logística se puede describir a través de un grafo que consiste en un conjunto de *nodos o vértices* y *aristas o arcos* [2]. Los nodos se utilizan para representar objetos o agentes (fábricas, almacenes, centros de distribución, cliente final, etc.) y los arcos para representar la relación que existe entre los nodos (modos de transporte). Los grafos que representan una red logística son, normalmente, grafos dirigidos (los arcos tienen una dirección) y pueden tener múltiples arcos entre los mismos pares de nodos, uno por cada modo de transporte.

La optimización de una red logística hace referencia al conjunto de decisiones que se deben tomar para garantizar un funcionamiento óptimo de la misma [3]. Estos sistemas de decisión se pueden plantear como problemas de optimización matemática en los que el objetivo es maximizar o minimizar una o varias funciones objetivo sujeto a un número finito de restricciones sobre las variables de decisión del problema.

Al igual que la gestión logística, muchos problemas reales involucran la optimización de varios objetivos que a menudo están en conflicto y compiten entre sí y, por ese motivo, los problemas de optimización multiobjetivo son muy comunes en diferentes áreas (ingeniería, física, economía,...). El problema de optimización multiobjetivo puede definirse como el problema de encontrar [4]: "un vector de variables de decisión que satisface restricciones y optimiza una función vectorial cuyos elementos representan las funciones objetivo. Estas funciones forman una descripción matemática de criterios de desempeño que normalmente están en conflicto entre sí. Por lo tanto, el término optimizar significa encontrar una solución que proporcione valores aceptables de todas las funciones objetivo para quien toma las decisiones".

Por su naturaleza y complejidad, los problemas de optimización multiobjetivo requieren el uso de técnicas diferentes a las técnicas estándar utilizadas en optimización con un solo objetivo. Existen diferentes metodologías que se pueden utilizar para resolver problemas de optimización matemática. Estos métodos se pueden clasificar en dos grandes grupos. El primero de ellos engloba los métodos clásicos como la optimización lineal, lineal entera mixta, no lineal, estocástica, dinámica, etc., que suelen ser eficientes para problemas que tienen una estructura bien definida (funciones lineales o convexas), no implican un gran

número de variables o de restricciones complejas y proporcionan una solución exacta. En el segundo grupo encontramos los métodos metaheurísticos que intentan aprovechar diferentes algoritmos en lugar de formulaciones matemáticas explícitas y suelen ser más adecuados para problemas complejos con múltiples objetivos y restricciones no lineales, tienen muchas variables, tanto continuas como discretas, y obtienen soluciones aproximadas en un tiempo razonable. Entre estos métodos se incluyen los algoritmos evolutivos (genéticos entre otros), el método del recocido simulado, las búsquedas heurísticas, etc. Cada uno de ellos presenta una serie de ventajas y desventajas, por lo que la elección de la metodología adecuada dependerá de la naturaleza del problema, del tiempo y los recursos disponibles y de la necesidad de precisión.

En el contexto de redes logísticas de transporte, se plantean, normalmente, problemas de optimización multiobjetivo que permitan encontrar soluciones que equilibren objetivos, como pueden ser, minimizar costes, reducir tiempos de entrega, y maximizar el nivel de servicio al cliente, entre otros. Estos objetivos pueden entrar en conflicto, ya que, por ejemplo, para reducir los costes de transporte se deberían agrupar envíos y utilizar la máxima capacidad del transporte, lo que podría aumentar los tiempos de entrega.

Para mejorar la eficiencia del proceso logístico de entregas en la fábrica de Mercedes Benz de Vitoria, en este trabajo se desarrolla un modelo de optimización cuyo objetivo principal es la gestión óptima de la campa de almacenamiento temporal de vehículos, lugar donde se almacenan los vehículos antes de su salida de la fabrica a diferentes destinos intermedios y finales. Para alcanzar este objetivo general, es necesario optimizar varias funciones objetivo simultáneamente, concretamente tres, teniendo en cuenta una serie de restricciones representadas a través de inecuaciones que las variables de decisión deben cumplir para que la solución al problema sea válida. Las funciones objetivo, como se verá más adelante, hacen referencia a los costes, a la ocupación de la campa y a la regularidad del tráfico, y las restricciones hacen referencia, fundamentalmente, a limitaciones temporales y de capacidad, muy comunes en los problemas de optimización en logística. Tal y como se detalla en el próximo capítulo, el problema de optimización que se plantea es un problema de optimización multiobjetivo no lineal dinámico, con un alto grado de complejidad.

Tras el análisis del problema de optimización planteado, se ha considerado que los algoritmos genéticos son el método apropiado para su resolución. A continuación se hace una breve descripción de dicha metodología y se justifica su elección.

2.3. Algoritmos genéticos

Como se ha mencionado anteriormente, los métodos heurísticos constituyen una alternativa general a los métodos clásicos, que mediante diferentes mecanismos buscan una solución a problemas de optimización complejos que se pueda considerar "buena", pero no necesariamente óptima, en un tiempo razonable.

Existen muchos métodos heurísticos de naturaleza muy diferente que hacen difícil su clasificación: de descomposición, inductivos, constructivos, de búsqueda local, etc. Los dos últimos constituyen la base de los procedimientos metaheurísticos, que han aparecido en los últimos años con el objetivo de obtener mejores resultados que los heurísticos tradicionales. Los métodos metaheurísticos se pueden definir como «métodos sofisticados o heurísticas que tienen como objetivo localizar, inducir o seleccionar una heurística que pueda ofrecer

la mejor solución posible a un problema de optimización, incluso en ausencia de datos suficientes o cuando los recursos computacionales son limitados. Los Metaheurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos [5]».

En estos momentos existe un gran desarrollo y crecimiento de los métodos heurísticos. En este trabajo vamos a limitarnos a algoritmos genéticos (GA) que se utilizan en diferentes ámbitos y se consideran una herramienta muy eficiente para encontrar soluciones óptimas en una gran variedad de problemas en diferentes campos del conocimiento. Además, se ha demostrado que dichos algoritmos son muy potentes y generalmente aplicables para resolver problemas difíciles de un solo objetivo y que sus estructuras algorítmicas fundamentales también se pueden aplicar para resolver muchos problemas multiobjetivo[6] [7].

Los algoritmos genéticos (GA) son métodos de búsqueda y optimización basados en poblaciones inspirados en la genética y en la selección natural de los individuos y la supervivencia de aquellos que mejor se adaptan al medio ambiente de Charles Darwin [8]. Los GA son parte de una clase más amplia de técnicas de Computación Evolutiva [9] y fueron desarrollados en la década de 1960 por John Holland y un grupo de investigadores de la Universidad de Michigan, quienes establecen las bases de desarrollos posteriores. En 1975, el propio Holland publica el libro *Adaptación en Sistemas Naturales y Artificiales* [10] en el que se presenta sistemática y rigurosamente el concepto de sistemas digitales adaptativos utilizando las operaciones de mutación, la selección y el cruce, y simulando el proceso de la evolución biológica como estrategia para resolver problemas. Estos trabajos pioneros generaron un interés generalizado en la computación evolutiva y fue David Goldberg en 1989 quien los formalizó y sistematizó [11]. Un Algoritmo Genético consiste básicamente en una población de soluciones o individuos codificados de forma similar a cromosomas, de forma que el cromosoma o cromosomas que representa a un individuo contengan toda la información necesaria para describirle. Cada uno de los individuos se evalúa a través de una función de ajuste o *fitness*, que permite cuantificar su validez como solución al problema. En función del valor asignado, el individuo tiene más o menos oportunidades de reproducción o cruce, promoviendo así la exploración de soluciones óptimas en el espacio de búsqueda. Además, con cierta probabilidad se realizan mutaciones de los individuos para garantizar la diversidad de soluciones. Así, usualmente, un GA tiene cinco componentes básicos [8]:

- Una configuración paramétrica (tamaño de la población, probabilidad de cruzamiento, probabilidad de mutación, etc.).
- Una población de posibles soluciones al problema (*mapa de búsqueda*).
- Un procedimiento para crear una población inicial de posibles soluciones (mediante un proceso aleatorio o aleatorizado).
- Una forma de clasificar (*fitnnes value*) y evaluar cada una de las posibles soluciones (*selection operator*).
- Un conjunto de operadores de evolución que alteran la composición de los individuos de la población a través de las generaciones: operador cruce o *crossover operator* que combina soluciones y operador de mutaciones, *mutation operator*, que evita la pérdida de diversidad en las soluciones.

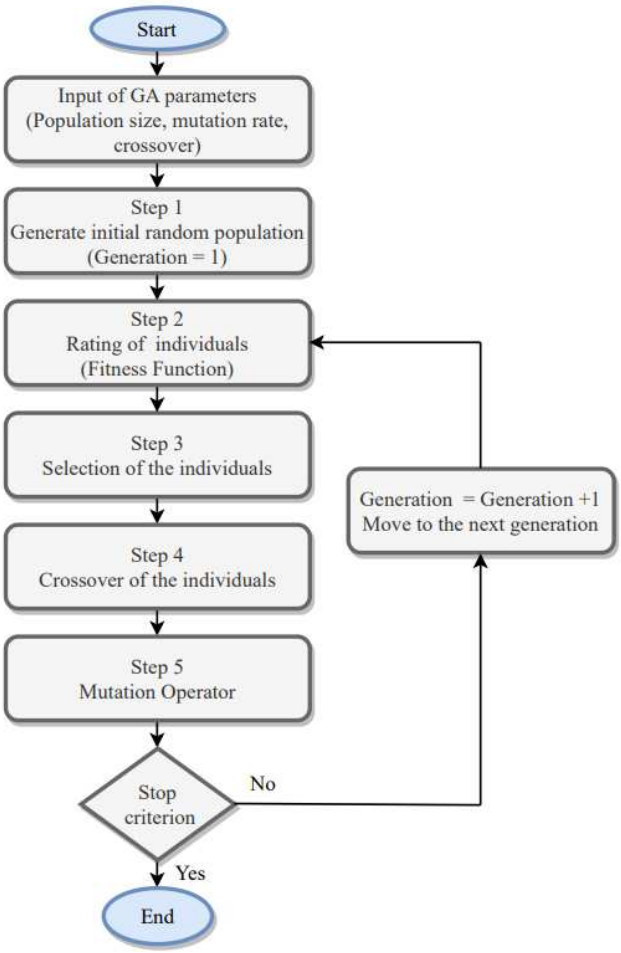

Figura 2.5: Diagrama de flujo de un GA [12].

La Figura 2.5 [12] muestra el diagrama de flujo de un GA en un problema de optimización o búsqueda numérica. Las diferentes etapas de implementación del algoritmo que a partir de una lista, tal vez infinita, de posibles soluciones, permite encontrar la mejor solución al problema. Estas etapas son las siguientes [13]:

- Representación del problema (Codificación): establece cómo se va a modelar el problema dentro del marco del GA. Se establece la representación adecuada de las posibles soluciones del problema (llamadas individuos) para que puedan ser manipuladas por operadores genéticos posteriormente. Las representaciones pueden ser binarias, enteras, reales o permutaciones.

- Formación de la población inicial: para diseñar un espacio inicial de búsqueda de soluciones, el GA crea una población con un número predeterminado de individuos, cada uno de los cuales es una posible solución al problema. Cada individuo lleva asociado un conjunto de variables con valores asignados aleatoriamente en esta etapa para garantizar que ocupen posiciones diferentes en el espacio de búsqueda de la solución óptima al problema. Estas son las variables que manipula el GA durante el proceso de optimización.

- Evaluación (Función de *fitness*): evalúa a cada individuo de la población mediante una función de *fitness* (o aptitud), que calcula el valor que proporciona la solución representada por ese individuo en función de los objetivos planteados.

- Operadores Genéticos: permiten analizar el espacio de soluciones con el objetivo de mejorar la población.

  - Selección: eligen individuos de la población actual para reproducirse, dando preferencia a los individuos con un valor mayor en la función *fitness*, los más aptos.
  - Cruce: combinan partes de dos individuos seleccionados para generar nuevos. Hay numerosos tipos de cruces, como el de un punto, el de dos puntos, etc., que se utilizan en función de la codificación del individuo.
  - Mutación: realizan pequeños cambios para introducir diversidad en la población, explorar nuevas áreas en el espacio de búsqueda y evitar el estancamiento en óptimos locales.

  la Figura 2.6 ([14], [15], [16]) recoge diferentes operadores que se utilizan para la selección, cruce y mutación. La elección adecuada de los mismos dependerá de la naturaleza del problema.

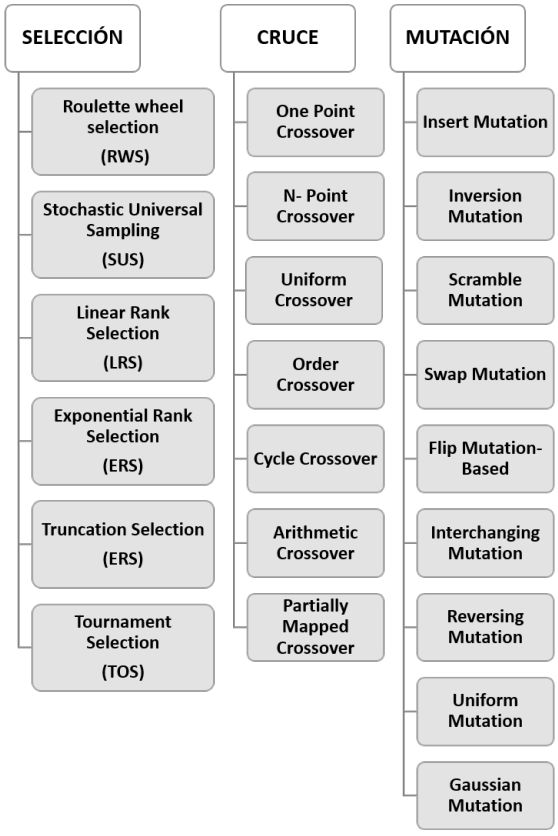

Figura 2.6: Operadores utilizados en un GA [14], [15], [16].

- Reemplazo: reemplazan a los individuos menos aptos de la población actual por los nuevos individuos generados a través del cruce y de la mutación, creando así una nueva generación que supera a la anterior.
- Condición de Parada: el proceso iterativo prosigue hasta que se cumple una condición de parada establecida, como alcanzar un número máximo de generaciones o encontrar una solución con un valor en la función fitness satisfactorio.

Al ser un enfoque basado en la población, los GA son muy adecuados para resolver problemas de optimización multiobjetivo. La capacidad de los GA de buscar simultáneamente diferentes regiones de un espacio de soluciones permite encontrar un conjunto diverso de soluciones para problemas difíciles con espacios de soluciones no convexos, discontinuos y multimodales, lo que les ha convertido en uno de los enfoques heurísticos más populares para los problemas de diseño y optimización multiobjetivo. [17].

La elección de los GA para resolver el problema de optimización que se presenta en este trabajo se basa, principalmente, en la naturaleza del mismo:

- Multiobjetivo: el problema tiene múltiples objetivos que entran en conflicto ya que, por ejemplo, mantener mucho tiempo los vehículos en la campa puede generar cuellos de botella en la producción, mientras que forzar su salida puede generar costes de almacenamiento en los destinos intermedios. Los GA, frente a algunos métodos clásicos y a otros métodos metaheurísticos, están diseñados específicamente para resolver problemas multiobjetivo como el que se plantea en este trabajo, manteniendo una diversidad de soluciones, aproximándose al óptimo y permitiendo resolver el problema en un tiempo razonable [7].
- No linealidad y discontinuidad: se trata de un problema en el que las variables son discretas y las funciones objetivo y las restricciones pueden ser no lineales y discontinuas o presentar reglas sujetas a condiciones complejas. Al contrario que los métodos clásicos, los algoritmos genéticos no requieren que la función objetivo sea continua, diferenciable o lineal, ya que pueden trabajar sin necesidad de aplicar cálculos diferenciales o transformar el problema en una aproximación lineal. Los GA exploran directamente el espacio de soluciones factibles y las evalúan en las funciones objetivo tal y como son, sin necesidad de transformarlas, y, una vez evaluadas, seleccionan las mejores soluciones.
- Dinámico: en cada momento del tiempo, las variables dependen de las decisiones anteriores y las decisiones actuales afectan a las variables futuras. Existe, por lo tanto, una interdependencia temporal. Los GA son una de las mejores opciones para resolver problemas dinámicos como el que se analiza en este trabajo, en comparación con otros métodos de optimización. En este sentido, pueden ajustarse a los cambios en el tiempo incorporando la evolución dinámica del sistema. En contraste, los métodos clásicos que se utilizan en problemas de optimización dinámicos, como la programación dinámica, pueden presentar problemas cuando se manejan decisiones dinámicas con variables discretas.

La implementación de los GA es relativamente sencilla y proporcionan resultados robustos que evitan los óptimos locales, particularmente en entornos no lineales, dinámicos y con

variables discretas de gran escala, manteniendo una diversidad de soluciones, aproximándose al óptimo y permitiendo resolver problemas complejos en un tiempo razonable [11]. Los GA son versátiles y flexibles, ya que se pueden adecuar a nuevos escenarios mediante ajustes en las funciones y/o las restricciones y requieren menos información que otras metodologías, tanto en lo que a la formulación matemática del problema se refiere como a la precisión y completitud de los datos [5]. Son, además, una de las metodologías más utilizadas para resolver problemas de optimización en logística, no solo a nivel teórico [18], sino también práctico. Algunas empresas como DHL y FedEx han utilizado algoritmos genéticos para optimizar rutas de distribución, reduciendo tiempos de entrega y costes operativos significativamente [19]. Por estos motivos se les ha considerado como una herramienta eficaz para resolver el problema planteado.

Existe una gran variedad de Algoritmos Genéticos Multiobjetivo (MOEAs). El primero de ellos, denominado VEGA (Algoritmo genético evaluado por vectores), fue propuesto por Schaffer [20] en 1985. Posteriormente, se han desarrollado una gran variedad de ellos, que pueden agruparse en dos grandes grupos: los métodos que utilizan un ranking de Pareto para evaluar la Función de *fitness*; y los métodos de agregación que transforman un problema de optimización multiobjetivo en una colección de problemas de un solo objetivo. En la Tabla 2.1, elaborada por Konaka, Coitb y Smithc [17], se muestran los aspectos más destacados de los algoritmos más conocidos con sus ventajas e inconvenientes.

Tal y como se explicará en el capítulo 4, el problema de optimización planteado se aborda utilizando dos MOEAs diferentes que representan los dos enfoques citados anteriormente.

En primer lugar se implementará un algoritmo genético cuya función de *fitness* incorpora pesos o ponderaciones para manejar múltiples objetivos (WBGA). Estos algoritmos son computacionalmente eficientes y fáciles de implementar en este tipo de problemas. La elección de este algoritmo está condicionada principalmente a la necesidad de la empresa de priorizar aquellos objetivos que sean más relevantes según el contexto o las necesidades específicas de cada situación, permitiendo ajustar el enfoque de la solución hacia metas que maximicen sus objetivos en escenarios particulares.

En segundo lugar, se implementará un algoritmo genético basado en el frente de Pareto, el NSGA-II. Se trata de un algoritmo reconocido por su eficiencia en el manejo de grandes poblaciones y su capacidad para mantener la diversidad entre soluciones. Propuesto por Deb et.al en 2002 [21], avanzando en el concepto introducido por Goldberg 1989 [11], NSGA-II es uno de los algoritmos más utilizados para resolver problemas de optimización multiobjetivo. Su principal ventaja es que que proporciona un conjunto de soluciones consideradas igualmente buenas, de modo que el usuario puede elegir la mejor en función de las necesidades específicas de cada situación particular.

En lo que a los diferentes operadores genéticos se refiere, se utilizarán los siguientes:

- Selección: operador por Ruleta o *Wheel Selection* en el caso del algoritmo WBGA y dominancia y distancia de aglomeración (crowding distance) en el caso del NSGA-II.
- Cruce: cruce porcentual y por segmentos.
- Mutación: mutación gaussiana, también conocida como mutación basada en perturbaciones normales

MOEA	Asignación de *Fitness*	Ventajas	Inconvenientes
MOGA	Ranking de Pareto	Extensión simple del GA de un solo objetivo	Convergencia lenta.
VEGA	Cada subpoblación se evalúa con respecto a un objetivo diferente	Primer MOGA. Implementación sencilla	Tiende a converger hacia los extremos de cada objetivo
WBGA	Promedio ponderado de objetivos normalizados	Extensión simple del GA de un solo objetivo	Dificultades en espacios de funciones objetivo no convexos. Pesos definidos
NPGA	Sin asignación de fitness, selección por torneo	Proceso de selección simple con torneos	Parámetro adicional para selección por torneo
RWGA	Promedio ponderado de objetivos normalizados	Eficiente y fácil de implementar	Dificultades en espacios de funciones objetivo no convexos
PESA	Sin asignación de fitness	Fácil de implementar. Eficiente computacionalmente	Su rendimiento depende del tamaño de las regiones de búsqueda. Requiere información previa sobre el espacio objetivo
PAES	Dominancia de Pareto para reemplazar a un padre si el descendiente domina	Estrategia aleatoria de escalada por mutación. Fácil de implementar. Eficiente computacionalmente	No está basado en poblaciones. Su rendimiento depende del tamaño de las regiones de búsqueda
NSGA	Clasificación basada en ordenación no dominada	Convergencia rápida	Problemas con el parámetro de tamaño de nicho
NSGA-II	Clasificación basada en ordenación no dominada	Bien probado. Eficiente	La distancia de hacinamiento solo funciona en el espacio objetivo
SPEA	Clasificación basada en archivo externo de soluciones no dominadas	Bien probado	Algoritmo de agrupamiento complejo. Sin parámetro para agrupamiento
SPEA-2	Fuerza de los dominadores	SPEA mejorado	Cálculo de fitness y densidad computacionalmente costoso.
RDGA	Reducción a un problema bi-objetivo con ranking de soluciones y densidad como objetivos	Actualización dinámica de regiones de búsqueda. Robusto respecto al número de objetivos	Más difícil de implementar que otros
DMOEA	Clasificación basada en regiones de búsqueda	Técnicas eficientes para actualizar densidades de regiones de búsqueda. Métodos adaptativos para ajustar los parámetros	Más difícil de implementar que otros

Tabla 2.1: Algoritmos genéticos multiobjetivo más conocidos [17].

Una vez presentadas las características principales del problema de optimización y de la metodología que se va a utilizar para su resolución, a continuación se realiza una presentación detallada del modelo de optimización que representa el problema planteado, para posteriormente implementar los GA para encontrar la solución al mismo.

CAPÍTULO 3

MODELO DE OPTIMIZACIÓN

La aplicación de un modelo de optimización a un caso concreto requiere comprender el problema a resolver y definir unos objetivos claros, determinar los parámetros y las variables de control, establecer las restricciones y definir la función o funciones objetivo. A continuación se explica cómo se puede aplicar el modelado de optimización a este escenario concreto paso a paso.

3.1. Descripción del problema y definición de objetivos

La optimización del proceso logístico de entregas que abarca el transporte de furgonetas desde la planta de fabricación de Mercedes-Benz en Vitoria hasta diversas ubicaciones geográficas presenta un desafío logístico crucial que requiere una planificación eficiente de diferentes tareas (gestión de inventario, seguimiento de los vehículos y rutas de entrega, etc.) que puede ayudar a reducir errores humanos, agilizar los tiempos de respuesta, reducir costes y mejorar la eficiencia operativa en general, garantizando la entrega oportuna a los destinos finales.

La complejidad del problema radica en la consideración de múltiples variables interrelacionadas entre sí, como son la capacidad específica de cada transportista, la gestión de la campa de almacenamiento temporal, y la infraestructura portuaria, entre otras.

Los transportistas desempeñan un papel crucial. Cada transportista tiene capacidades específicas de carga, limitaciones de espacio y costes asociados por kilómetro recorrido. La asignación óptima de furgonetas a estos transportistas debe realizarse considerando la capacidad total de producción de la fábrica. La eficiencia y la reducción de costes serán clave en la toma de decisiones sobre la combinación de transportistas y la definición de rutas.

La logística portuaria agrega una capa adicional de complejidad. La elección de puertos cercanos a la planta de Vitoria implica considerar capacidades de carga y descarga específicas, así como costes logísticos asociados. Los tiempos de operación y posibles restricciones temporales en los puertos deben integrarse en la planificación para garantizar la eficiencia de la cadena de suministro. La gestión adecuada de la interacción entre los transportistas

y los puertos será crucial para minimizar los tiempos de espera y optimizar la carga y descarga de furgonetas.

En el núcleo del problema logístico se encuentra la gestión de la campa de almacenamiento temporal en la fábrica de Vitoria, con una capacidad limitada. Este área juega un papel vital en el flujo de furgonetas, actuando como un punto intermedio antes del transporte. La capacidad de la campa debe ser gestionada cuidadosamente para evitar cuellos de botella en la producción y distribución. La planificación debe tener en cuenta la necesidad de almacenamiento temporal y coordinar la entrada y salida de furgonetas de manera eficiente.

Para abordar este problema, se propone un problema de optimización cuyo objetivo principal es minimizar los costes operativos asociados a una deficiente organización del tráfico, al mismo tiempo que se realiza una ocupación eficiente de la campa de almacenamiento temporal y se minimizan los costes derivados de las estancias en los puertos. En dicho problema, se deben incorporar todas las variables que intervienen en el mismo, establecer la relación que existe entre dichas variables y tener en cuenta las limitaciones o restricciones que se definen sobre las mismas. En este sentido, la optimización del transporte de furgonetas desde la planta de Mercedes-Benz en Vitoria hasta destinos finales implica una cuidadosa consideración de los flujos de entrada de furgonetas a la campa, la capacidad de almacenamiento de la misma y de los transportistas, la infraestructura portuaria y los costes de estancia en los puertos. La implementación de un enfoque integral de optimización es esencial para garantizar la eficiencia operativa, minimizar costes y cumplir con las demandas de entrega de manera efectiva en el corto plazo.

En este trabajo afrontamos el problema de optimización de envíos de furgonetas desde la planta de fabricación de Mercedes-Benz en Vitoria hasta diversas ubicaciones portuarias (puertos de Pasajes, Barcelona, Valencia y Santander) y otros destinos (resto de España y Portugal, Francia y Suiza). En este sentido, no se incluye en el problema de optimización la parte del proceso que se desarrolla una vez que las furgonetas están cargadas en los barcos, ya que esta última fase, si bien es crucial para para la entrega de los vehículos a tiempo, no se planifica desde la fábrica de Vitoria, sino desde un departamento de la división ubicado en Alemania.

Para poder entender el problema planteado definimos en primer lugar, la red logística correspondiente (ver Figura 3.1). Los vehículos salen de la fábrica y, dependiendo de su destino, irán a un segmento de la campa de la planta o a otro para su posterior transporte. Aquellos con dirección al puerto de Pasajes (P) irán a la primera zona, la zona M1, y el resto (Barcelona (B), Valencia (V), Santander (S), resto de España y Portugal (E), Francia (F) y Suiza (SW)) a la segunda, la zona M2. Posteriormente, los vehículos se montan en camiones con destino a los diferentes destinos. Aquellos que llegan a los puertos, se almacenan en campas portuarias hasta que llegue la naviera que pueda trasladarlos al puerto destino.

Teniendo en cuenta las características de la red, podemos clasificarla como una red de transporte con una planificación guiada por la oferta, en la que se operan con los datos de una previsión de suministro que tiene que procesarse y transportarse a los puntos alternativos de demanda con el menor coste [3].

Este enfoque integral requiere tener en cuenta consideraciones adicionales y aspectos clave que van más allá del problema de optimización que se plantea en este trabajo, pero que son fundamentales para garantizar la eficiencia del proceso logístico y deberían trabajarse,

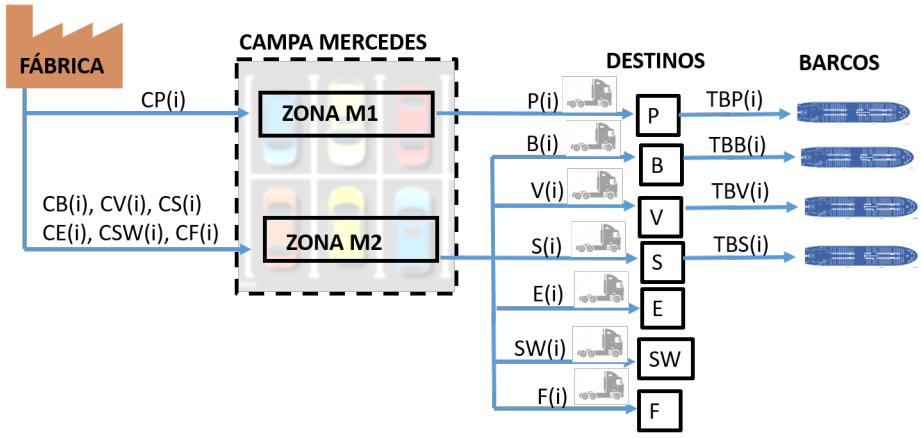

Figura 3.1: Red logística del transporte desde la fábrica a los puertos

o bien paralelamente, o bien a partir de una extensión del mismo. Estos se detallan a continuación:

1. Coordinación con los diferentes departamentos ubicados en Alemania: la comunicación estrecha y el trabajo conjunto con los departamentos de Logística de Transporte de Vehículos que son responsables de las diferentes fases del proceso logístico de entrega de vehículos es fundamental para que el proceso sea eficiente.

2. Coordinación con Proveedores de los servicios de transporte y Destinatarios: la colaboración estrecha con los transportistas, puertos y destinatarios es esencial. Se deben establecer canales de comunicación eficientes para garantizar una ejecución de la planificación exenta de problemas.

3. Tecnología de Seguimiento y Gestión de contingencias: la implementación de tecnologías de seguimiento en tiempo real, como sistemas de telemetría y GPS, permitirá monitorear y ajustar la planificación en función de eventos imprevistos. En este sentido, se deben desarrollar planes de contingencia robustos para hacer frente a situaciones imprevistas, como pueden ser las congestiones de tráfico o los retrasos en puertos derivadas de las inclemencias meteorológicas o de otra índole. La capacidad de reaccionar rápidamente es indispensable para garantizar la continuidad de la cadena de suministro.

4. Sostenibilidad y Eficiencia Energética: dada la creciente importancia de la sostenibilidad, se deben tener en cuenta consideraciones de eficiencia energética en la selección de rutas y transportistas. La minimización del impacto ambiental debe ser un factor adicional en la toma de decisiones.

5. Integración de Sistemas de Información: integración de sistemas de información, como software de gestión de flotas, bases de datos de puertos y herramientas de planificación logística, facilitará una gestión más eficiente y una toma de decisiones informada.

6. Evaluación Continua y Mejora: la implementación exitosa de la solución no debe considerarse como un fin, sino como un proceso continuo. La recopilación de datos y

la evaluación regular del rendimiento permitirán realizar ajustes y mejoras constantes en el sistema de optimización.

7. Consideraciones Geopolíticas y Regulatorias: deben incorporarse en la planificación factores geopolíticos y regulaciones específicas del transporte internacional para garantizar el cumplimiento normativo y evitar sorpresas regulatorias que conlleven sanciones o incrementen costes.

Una vez descrito el problema, a continuación se detallan los elementos claves del modelo matemático que se propone para resolverlo: las variables y los parámetros, las restricciones y las funciones objetivo.

3.2. Variables, Parámetros y Constantes

Las variables a tener en cuenta en un problema de optimización representan los elementos del modelo matemático que son controlables directa (variables de decisión) o indirectamente (variables intermedias). Representan, por lo tanto, las decisiones que se pueden tomar y que afectan al valor de la función objetivo.

Los parámetros y las constantes hacen referencia a los valores numéricos que presentan algunos elementos del problema que pueden estar presentes, tanto en las funciones objetivo como en las restricciones.

A continuación, en primer lugar se describen los diferentes parámetros y constantes que afectan al modelo, ya que de dichos elementos dependen algunas de las variables intermedias del mismo.

3.2.1. Parámetros y Constantes

Para el correcto planteamiento del problema de optimización, es fundamental considerar los diferentes parámetros y constantes que pueden influir en las posibles soluciones. Los valores de los parámetros, tales como el número de coches a cada destino y la capacidad de los recursos disponibles (transportistas, barcos, etc.) se obtienen de los informes y pueden variar según la semana escogida para realizar el estudio de optimización. Las constantes, tales como la capacidad máxima de las diferentes zonas de la campa o los costes de almacenamiento en puertos, son, sin embargo, valores fijos. Tanto los parámetros como las constantes condicionarán las soluciones del problema de optimización.

La lista de constantes a considerar es la siguiente:

- **En relación a la capacidad de la campa**: cada una de las zonas de la campa de almacenamiento temporal tiene una capacidad máxima.
  - **CM1**: Capacidad máxima de la zona M1 (1350).
  - **CM2**: Capacidad máxima de la zona M2 (643).
- **En relación a las cuotas de envíos a puertos**: el número de vehículos que se pueden transportar semanalmente a los diferentes destinos se pacta anualmente con los transportistas y puede diferir de una semana a otra.

- **CuotaP**: Cuota semanal de envíos a Pasajes.
- **CuotaB**: Cuota semanal de envíos a Barcelona.
- **CuotaV**: Cuota semanal de envíos a Valencia.
- **CuotaS**: Cuota semanal de envíos a Santander.
- **CuotaE**: Cuota semanal de envíos a resto de España y Portugal.
- **CuotaSW**: Cuota semanal de envíos a Suiza.
- **CuotaF**: Cuota semanal de envíos a Francia.

- **En relación a la capacidad operativa diaria de transporte hacia los puertos**: el máximo de vehículos que pueden salir de la campa hacia un destino determinado está limitado por cuestiones operativas, ya que, por ejemplo, el número de camiones que pueden transitar y cargar vehículos diariamente en las diferentes zonas está limitado por el espacio, los accesos, etc.

  - **MoP**: Máximo operativo diario para Pasajes (600).
  - **MoB**: Máximo operativo diario para Barcelona (164).
  - **MoV**: Máximo operativo diario para Valencia (72).
  - **MoS**: Máximo operativo diario para Santander (36).
  - **MoE**: Máximo operativo diario para resto de España y Portugal (97).
  - **MoSW**: Máximo operativo diario para Suiza(93).
  - **MoF**: Máximo operativo diario para Francia(110)
  - **Mo2**: Máximo operativo diario para la zona M2(400).

  En el caso de la zona M2 de la campa, además de un máximo operativo para dicha zona, se establece un máximo operativo por destino (Barcelona, Valencia, Santander, resto de España y Portugal, Suiza y Francia).

- **En relación a los costes de utilización de los puertos**: sobrepasar un número de días determinado desde la llegada de los vehículos a los puertos hasta su salida en barco, supone un coste diario a pagar. Este número de días es diferente para cada puerto.

  - **LP**: número de días a partir del cual se paga un coste por tener los coches almacenados en Pasajes (4).
  - **LB**: número de días a partir del cual se paga un coste por tener los coches almacenados en Barcelona (22).
  - **LV**: número de días a partir del cual se paga un coste por tener los coches almacenados en Valencia (12).
  - **LS**: número de días a partir del cual se paga un coste por tener los coches almacenados en Santander (22).

  - **CostoP**: Coste al día por tener los coches en Pasajes a partir del día LP (0,77 €/vehículo).

- **CostoB**: Coste al día por tener los coches en Barcelona a partir del día LB (1,5 €/vehículo).
- **CostoV**: Coste al día por tener los coches en Valencia a partir del día LV (1,38 €/vehículo).
- **CostoS**: Coste al día por tener los coches en Santander a partir del día LS (1,5 €/vehículo).

Los parámetros del modelo son los siguientes:

- **En relación al flujo de entrada desde la fábrica a la campa**: cada día de la semana (indicado por *i*) se fabrica un número determinado de vehículos con diferentes destinos. Aquellos cuyo destino es el puerto de Pasajes se almacenan en la zona M1 de la campa y el resto en la zona M2.
  - **CP(i)**: Coches fabricados el día i que van a la zona M1 para Pasajes.
  - **CB(i)**: Coches fabricados el día i que van a la zona M2 para Barcelona.
  - **CV(i)**: Coches fabricados el día i que van a la zona M2 para Valencia.
  - **CS(i)**: Coches fabricados el día i que van a la zona M2 para Santander.
  - **CE(i)**: Coches fabricados el día i que van a la zona M2 para resto de España y Portugal.
  - **CSW(i)**: Coches fabricados el día i que van a la zona M2 para Suiza.
  - **CF(i)**: Coches fabricados el día i que van a la zona M2 para Francia.
- **En relación a la salida de puerto**: el número de vehículos que se pueden transportar diariamente desde cada uno de los puertos está condicionado por la disponibilidad de barcos y por su capacidad.
  - **TBP(i)**: Coches que puede transportar el barco el día i en el puerto de Pasajes.
  - **TBB(i)**: Coches que puede transportar el barco el día i en el puerto de Barcelona.
  - **TBV(i)**: Coches que puede transportar el barco el día i en el puerto de Valencia.
  - **TBS(i)**: Coches que puede transportar el barco el día i en el puerto de Santander.

  Si no hay barco en un determinado puerto un determinado día de la semana, el valor del parámetro es 0, por ejemplo, TBP(2)=0 en el caso de que no haya barco en el puerto de Pasajes el martes.

3.2.2. Variables

A continuación, se describen las principales variables que se han seleccionado para capturar la complejidad en la toma de decisiones de la planificación logística y su función en el modelo.

3.2.2.1. Variables de decisión

Las variables de decisión son las variables que el decisor puede controlar para conseguir uno o varios objetivos. Para realizar una gestión óptima de la campa de almacenamiento temporal (objetivo principal) es necesario decidir cuál es el número de vehículos que saldrán de la misma hacia diferentes destinos, teniendo en cuenta los valores de los diferentes parámetros y constantes que generan las restricciones del modelo y limitan el conjunto de decisiones factibles. Las variables de decisión son, por lo tanto:

- **P(i)**: Coches que salen de la zona M1 con destino a Pasajes.
- **B(i)**: Coches que salen de la zona M2 con destino a Barcelona.
- **V(i)**: Coches que salen de la zona M2 con destino a Valencia.
- **S(i)**: Coches que salen de la zona M2 con destino a Santander.
- **E(i)**: Coches que salen de la zona M2 con destino a resto de España y Portugal.
- **SW(i)**: Coches que salen de la zona M2 con destino a Suiza.
- **F(i)**: Coches que salen de la zona M2 con destino a Francia.

3.2.2.2. Variables intermedias

Las variables intermedias son variables que no se controlan directamente pero que dependen de las variables de decisión y proporcionan información relevante para la elección de los valores óptimos. En este problema de optimización se definen las siguientes variables intermedias:

- **En relación a los vehículos almacenados en la campa**: el número de vehículos almacenados en una zona de la campa con un destino determinado se calcula teniendo en cuenta los almacenados el día anterior en esa zona y con ese destino, los fabricados ese día que han ido a esa zona y con ese destino y los que salen ese día desde esa zona y hacia ese destino.
  - **BCM1P(i)**: Coches almacenados en la zona M1 con destino Pasajes.
  - **BCM2B(i)**: Coches almacenados en la zona M2 con destino Barcelona.
  - **BCM2V(i)**: Coches almacenados en la zona M2 con destino Valencia.
  - **BCM2S(i)**: Coches almacenados en la zona M2 con destino Santander.
  - **BCM2E(i)**: Coches almacenados en la zona M2 con destino España y Portugal.
  - **BCM2SW(i)**: Coches almacenados en la zona M2 con destino Suiza.
  - **BCM2F(i)**: Coches almacenados en la zona M2 con destino Francia.

  Donde el número de vehículos almacenados en una determinada zona de la campa con un destino determinado se obtiene como:

$$BCM1P(i) = BCM1P(i-1) + CP(i) - P(i) \tag{3.1}$$

$$BCM2B(i) = BCM2B(i-1) + CB(i) - B(i) \tag{3.2}$$
$$BCM2V(i) = BCM2V(i-1) + CV(i) - V(i) \tag{3.3}$$
$$BCM2S(i) = BCM2S(i-1) + CS(i) - S(i) \tag{3.4}$$
$$BCM2E(i) = BCM2E(i-1) + CE(i) - E(i) \tag{3.5}$$
$$BCM2SW(i) = BCM2SW(i-1) + CSW(i) - SW(i) \tag{3.6}$$
$$BCM2F(i) = BCM2F(i-1) + CF(i) - F(i) \tag{3.7}$$

- **En relación a los vehículos almacenados en los puertos**: aunque la capacidad de almacenamiento en los puertos sea suficientemente grande para garantizar la recepción de los vehículos, es necesario conocer cuántos vehículos hay almacenados un día determinado en cada puerto.

  - **DP(i)**: Coches almacenados en Pasajes.
  - **DB(i)**: Coches almacenados en Barcelona.
  - **DV(i)**: Coches almacenados en Valencia.
  - **DS(i)**: Coches almacenados en Santander.

  Además, es necesario saber el número de días que llevan allí para calcular el coste que esto supone.

  - **DP(i,a)**: Coches almacenados en Pasajes que llevan *a* días allí.
  - **DB(i,a)**: Coches almacenados en Barcelona que llevan *a* días allí.
  - **DV(i,a)**: Coches almacenados en Valencia llevan *a* días allí.
  - **DS(i,a)**: Coches almacenados en Santander llevan *a* días allí.

Las variables intermedias consideradas son almacenes y su índice *i = 0,...,7* son los días de la semana, de forma que el *i=0* es el almacenamiento inicial antes de comenzar la semana. El índice *a* de las variables intermedias, que hacen referencia al número de vehículos almacenados en los puertos, toma valores entre 1 e infinito y representa un buffer para que el coste de almacenamiento se cobre a partir del día establecido por dichos puertos.

Teniendo en cuenta que uno de los objetivos es minimizar el coste de almacenamiento en los puertos, la carga de vehículos en los barcos se realiza en orden descendente, comenzando con los que llevan más tiempo almacenados. Para calcular el número de vehículos almacenados el día *i* en un puerto que llevan *a* días allí, definimos la siguiente variable intermedia:

- Coches que llevan *a* días en Pasajes que se pueden cargar el día *i* en el barco:

$$\textbf{SP(i,a)} = TBP(i) - \sum_{b=a+1}^{\infty} DP(i,b)$$

- Coches que llevan *a* días en Barcelona que se pueden cargar el día *i* en el barco:

$$\textbf{SB(i,a)} = TBB(i) - \sum_{b=a+1}^{\infty} DB(i,b)$$

- Coches que llevan *a* días en Valencia que se pueden cargar el día *i* en el barco:

$$\textbf{SV(i,a)} = TBV(i) - \sum_{b=a+1}^{\infty} DV(i,b)$$

- Coches que llevan *a* días en Santander que se pueden cargar el día *i* en el barco:

$$\textbf{SS(i,a)} = TBS(i) - \sum_{b=a+1}^{\infty} DS(i,b)$$

Hay que tener en cuenta que los coches que llevan almacenados *a* días en un puerto, por ejemplo en Pasajes, solo se pueden cargar en el barco si después de haber cargado los que llevan más días, $\sum_{b=a+1}^{\infty} DP(i,b)$, aún queda sitio en el barco, $SP(i,a) = TBP(i) - \sum_{b=a+1}^{\infty} DP(i,b) > 0$, en caso contrario no se podrá cargar ninguno de ellos, $SP(i,a) = 0$, quedando todos ellos almacenados en el puerto al día siguiente (*i+1*).

El número de vehículos que llevan *a* días almacenados en un determinado puerto el día *i* son aquellos que el día anterior, *i-1*, llevaban *a-1* días y no se cargaron en el barco:

$$DP(i,a) = \begin{cases} DP(i-1,a-1) - SP(i-1,a-1) & \text{si } SP(i-1,a-1) \geq 0 \\ DP(i-1,a-1) & \text{de otro modo} \end{cases} \tag{3.8}$$

$$DB(i,a) = \begin{cases} DB(i-1,a-1) - SB(i-1,a-1) & \text{si } SB(i-1,a-1) \geq 0 \\ DB(i-1,a-1) & \text{de otro modo} \end{cases} \tag{3.9}$$

$$DV(i,a) = \begin{cases} DV(i-1,a-1) - SV(i-1,a-1) & \text{si } SV(i-1,a-1) \geq 0 \\ DV(i-1,a-1) & \text{de otro modo} \end{cases} \tag{3.10}$$

$$DS(i,a) = \begin{cases} DS(i-1,a-1) - SS(i-1,a-1) & \text{si } SS(i-1,a-1) \geq 0 \\ DS(i-1,a-1) & \text{de otro modo} \end{cases} \tag{3.11}$$

Siendo $DP(i-1,0) = P(i-1)$, $DB(i-1,0) = B(i-1)$, $DV(i-1,0) = V(i-1)$ y $DS(i-1,0) = S(i-1)$.

Así, el número de coches almacenados en un determinado puerto el día i se calcula como:

$$DP(i) = \sum_{a=1}^{\infty} DP(i,a)$$

$$DB(i) = \sum_{a=1}^{\infty} DB(i,a)$$

$$DV(i) = \sum_{a=1}^{\infty} DV(i,a)$$

$$DS(i) = \sum_{a=1}^{\infty} DS(i,a)$$

Una vez definidos los parámetros, las constantes y las variables, se pueden establecer, tanto las restricciones como las funciones objetivo.

3.3. Restricciones

Para el correcto planteamiento del problema de optimización es fundamental considerar las restricciones que limitan las posibles soluciones. Las restricciones de un modelo de optimización se representan a través de expresiones matemáticas que limitan los valores de las variables de decisión o las relaciones entre dichas variables. Estas restricciones son inherentes a las características del problema y pueden variar según las condiciones específicas de cada semana en que se realice el estudio, ya que, por ejemplo, la cuota semanal de los trayectos se pacta cada año. La adecuada definición y consideración de estas restricciones garantizará que las soluciones propuestas sean factibles, cumplan con los requisitos operativos y logísticos y sean aplicables en el contexto real del problema. A continuación, se describen las principales restricciones que deben considerarse en la formulación del modelo matemático:

- **Restricciones de Capacidad de los Transportistas**: no superar la cuota semanal pactada con los transportistas en las diferentes rutas. Cada transportista tiene una capacidad limitada de carga de furgonetas que se pacta para cada semana. Se establecen restricciones para asegurar que la asignación de furgonetas a cada transportista respete dicha cuota semanal, evitando sobrecargas y garantizando un transporte eficiente. Estas cuotas son específicas para cada ruta.

$$\sum_{i=1}^{7} P(i) \leq CuotaP \tag{3.12}$$

$$\sum_{i=1}^{7} B(i) \leq CuotaB \tag{3.13}$$

$$\sum_{i=1}^{7} S(i) \leq CuotaS \tag{3.14}$$

$$\sum_{i=1}^{7} V(i) \leq CuotaV \tag{3.15}$$

$$\sum_{i=1}^{7} E(i) \leq CuotaE \tag{3.16}$$

$$\sum_{i=1}^{7} SW(i) \leq CuotaSW \tag{3.17}$$

$$\sum_{i=1}^{7} F(i) \leq CuotaF \tag{3.18}$$

- **No superar el máximo operativo de las diferentes zonas de la campa**: para mantener una operación eficiente y evitar cuellos de botella, es crucial no superar el máximo operativo de cada zona de la campa. Hay que tener en cuenta que en la zona

M2, además de no sobrepasar el máximo operativo para cada destino, no se puede superar el máximo operativo de dicha zona.

$$P(i) \leq MoP; \forall i = 1, ..,7 \tag{3.19}$$

$$B(i) \leq MoB; \forall i = 1, ..,7 \tag{3.20}$$

$$V(i) \leq MoV; \forall i = 1, ..,7 \tag{3.21}$$

$$S(i) \leq MoS; \forall i = 1, ..,7 \tag{3.22}$$

$$E(i) \leq MoE; \forall i = 1, ..,7 \tag{3.23}$$

$$SW(i) \leq MoSW; \forall i = 1, ..,7 \tag{3.24}$$

$$F(i) \leq MoF; \forall i = 1, ..,7 \tag{3.25}$$

$$B(i) + S(i) + V(i) + E(i) + F(i) + SW(i) \leq Mo2; \forall i = 1, ..,7 \tag{3.26}$$

- **No superar la capacidad de almacenamiento de las diferentes zonas de la campa**: El número de vehículos almacenados en cada zona no puede superar la capacidad de dicha zona.

$$BCM1P(i) \leq CM1; \forall i = 1, ..,7 \tag{3.27}$$

$$BCM2(i) \leq CM2; \forall i = 1, ..,7 \tag{3.28}$$

Donde

$$BCM2(i) = BCM2B(i) + BCM2V(i) + BCM2S(i) + BCM2E(i)+$$

$$+BCM2SW(i) + BCM2F(i)$$

- **Restricciones temporales de estancia en la campa**: los vehículos con destino a Suiza y a Francia presentan necesidades específicas de 1 día de preparación en la campa antes de su salida. Estas restricciones temporales deben tenerse en cuenta para garantizar la coherencia en la planificación.

$$F(1) \leq BCM2F(0) \tag{3.29}$$

$$F(i) \leq CF(i-1) + BCM2F(i-1); \forall i = 2, ..., 7 \tag{3.30}$$

$$SW(1) \leq BCM2SW(0) \tag{3.31}$$

$$SW(i) \leq CSW(i-1) + BCM2SW(i-1); \forall i = 2, ..., 7 \tag{3.32}$$

- **Restricciones de Capacidad de los barcos**: no superar la capacidad diaria de los barcos en los diferentes puertos.

$$TBP(i) \leq DP(i) + P(i); \forall i = 1, ..., 7 \tag{3.33}$$

$$TBB(i) \leq DB(i) + B(i); \forall i = 1, ..., 7 \tag{3.34}$$

$$TBV(i) \leq DV(i) + V(i); \forall i = 1, ..., 7 \tag{3.35}$$

$$TBS(i) \leq DS(i) + S(i); \forall i = 1, ..., 7 \tag{3.36}$$

- Restricciones de no negatividad de las variables: tanto el número de vehículos enviados a cada destino como los almacenados en la campa y en los puertos deben ser superiores o iguales a cero para asegurar que sean factibles.

$$P(i) \geq 0; \forall i = 1, ..,7 \tag{3.37}$$

$$B(i) \geq 0; \forall i = 1, ..,7 \tag{3.38}$$

$$V(i) \geq 0; \forall i = 1, ..,7 \tag{3.39}$$

$$S(i) \geq 0; \forall i = 1, ..,7 \tag{3.40}$$

$$E(i) \geq 0; \forall i = 1, ..,7 \tag{3.41}$$

$$SW(i) \geq 0; \forall i = 1, ..,7 \tag{3.42}$$

$$F(i) \geq 0; \forall i = 1, ..,7 \tag{3.43}$$

$$BCM1P(i) \geq 0; \forall i = 0, ..,7 \tag{3.44}$$

$$BCM2B(i) \geq 0; \forall i = 0, ..,7 \tag{3.45}$$

$$BCM2V(i) \geq 0; \forall i = 0, ..,7 \tag{3.46}$$

$$BCM2S(i) \geq 0; \forall i = 0, ..,7 \tag{3.47}$$

$$BCM2E(i) \geq 0; \forall i = 0, ..,7 \tag{3.48}$$

$$BCM2SW(i) \geq 0; \forall i = 0, ..,7 \tag{3.49}$$

$$BCM2F(i) \geq 0; \forall i = 0, ..,7 \tag{3.50}$$

$$DP(i, a) \geq 0; \forall i = 0, ..., 7; a = 1, ...\infty \tag{3.51}$$

$$DB(i, a) \geq 0; \forall i = 0, ..., 7; a = 1, ...\infty \tag{3.52}$$

$$DV(i, a) \geq 0; \forall i = 0, ..., 7; a = 1, ...\infty \tag{3.53}$$

$$DS(i, a) \geq 0; \forall i = 0, ..., 7; a = 1, ...\infty \tag{3.54}$$

3.4. Funciones objetivo

Una función objetivo es una expresión matemática que representa el funcionamiento del sistema y que se persigue optimizar (maximizar o minimizar). En un problema de optimización las funciones objetivo son fundamentales para evaluar y dirigir las posibles soluciones al problema ya que representan los objetivos que se desean alcanzar.

En el problema de optimización de este TFM, teniendo en cuenta que el objetivo principal es la gestión óptima de la campa de almacenamiento temporal de vehículos, se han definido tres funciones objetivo principales, cada una orientada a mejorar distintos aspectos operativos de dicha gestión. La correcta formulación y consideración de estas funciones garantizará que las soluciones propuestas no solo sean óptimas desde un punto de vista teórico, sino también aplicables y beneficiosas en la práctica. A continuación, se describe cada una de ellas:

1. **Función de Regularidad**: Nuestro objetivo es lograr que el tráfico sea lo más homogéneo posible a lo largo de la semana. De esta manera se contribuye a una distribución equilibrada de la carga de trabajo entre los diferentes actores involucrados en la cadena de suministro y se elimina la incertidumbre, principalmente de los transportistas. Como veremos en el capítulo siguiente, para alcanzar este objetivo se minimiza de la desviación estándar del tráfico a los diferentes destinos entre distintos días. Matemáticamente, esto se expresa como:

$$\text{mín}[\sigma_P + \sigma_B + \sigma_V + \sigma_S + \sigma_{SW} + \sigma_E + \sigma_F]$$

   donde σ representa la desviación estándar de las salidas de vehículos desde la campa hacia los diferentes destinos (Pasajes, Barcelona, etc.).

2. **Ocupación Mínima de la Campa**: Deseamos que la ocupación de las diferentes zonas de la campa de almacenamiento temporal sea la mínima posible. De esta manera se consigue que el tráfico sea más fluido y se evita que se llegue a sus capacidades máximas, lo que podría generar cuellos de botella importantes. La función matemática que recoge este objetivo es:

$$\text{mín}[BCM1P(i) + BCM2(i)]; \forall i = 1, ..., 7$$

3. **Minimización del Coste de Estancia en los Puertos**: Los coches que permanecen en los diferentes puertos más de un determinado número de días generan un coste diario adicional. Nuestro objetivo es minimizar este coste, asegurando así una mayor eficiencia en la rotación de vehículos y reduciendo gastos innecesarios. Como se ha señalado anteriormente, este objetivo puede entrar en conflicto con el anterior ya que puede suponer que, para reducir estos costes, los vehículos permanezcan en la campa y no se minimice, por lo tanto, su ocupación. Contrariamente, minimizar la ocupación de la campa para evitar cuellos de botella en la producción, puede suponer que los vehículos salgan de la fábrica mucho antes de la salida de los barcos, lo que incrementa el coste de estancia en los puertos. Matemáticamente, esto se expresa como:

$$\text{mín} \sum_{i=1}^{7} [\sum_{a=LP}^{\infty} DP(i,a)CostoP + \sum_{a=LB}^{\infty} DB(i,a)CostoB + \\ + \sum_{a=LV}^{\infty} DV(i,a)CostoV + \sum_{a=LS}^{\infty} DS(i,a)CostoS]$$

Estas funciones objetivo permitirán orientar el proceso de optimización hacia soluciones prácticas y eficientes, alineadas con los intereses operativos y económicos del proyecto. Se han definido teniendo en cuenta las prioridades del Departamento de Entrega de Vehículos. Adicionalmente, la mejora de la eficiencia global del proceso implicaría tener en cuenta otros objetivos que están fuera del alcance de este TFM y que incrementarían considerablemente la complejidad del problema. Los objetivos que se podrían incluir serían, entre otros:

- Minimización de Costes Totales: reducir al mínimo los costes asociados con el transporte de furgonetas, considerando los costes de los transportistas y cualquier otro coste operativo relevante con una función objetivo diseñada para lograr esta minimización global.

- Optimización de la Utilización de Recursos: optimizar la asignación de furgonetas a transportistas y la elección de rutas para garantizar la máxima utilización de los recursos disponibles, como la capacidad de producción de la fábrica, las capacidades de carga de los transportistas y los puertos, y la capacidad de la campa de almacenamiento temporal.

- Minimización de Tiempos de Transporte: minimizar los tiempos totales de transporte, incluyendo los tiempos de viaje, los tiempos de carga y descarga en los puertos, etc. La reducción de los tiempos de transporte contribuirá a una entrega más rápida y eficiente.

- Consideraciones Ambientales: en función de los objetivos estratégicos de Mercedes-Benz, estudiar la posibilidad de incluir la minimización del impacto ambiental como un objetivo adicional. Esto podría implicar la elección de rutas más sostenibles desde el punto de vista ambiental.

3.5. Planificación de la siguiente semana

El modelo se ha desarrollado para una semana concreta; no obstante, permite obtener soluciones para las semanas posteriores. En ese caso, los parámetros de entrada de cada semana se corresponderían con los datos resultantes de la semana anterior. Así, por ejemplo, la situación inicial (el lunes) de la campa en una semana determinada sería la obtenida en la solución del modelo en la semana previa, y esta información se utilizaría como punto de partida para planificar la siguiente semana. En el caso de Pasajes, por ejemplo, el número de coches almacenados en la campa el lunes se obtendría como:

$$BCM1P(1) = BCM1P(0) + CP(1) - P(1) \tag{3.55}$$

donde $BCM1P(0)$ es el número de coches almacenados en la campa el domingo de la semana anterior.

Lo mismo ocurre en los almacenes de los puertos:

$$DP(1,a) = \begin{cases} DP(0,a-1) - SP(0,a-1) & \text{si } SP(0,a-1) \geq 0 \\ DP(0,a-1) & \text{de otro modo} \end{cases} \tag{3.56}$$

El número de vehículos almacenados en Pasajes el lunes que llevan *a* días allí, son los que el domingo de la semana anterior llevaban *a-1* días ($DP(0,a-1)$) menos los que llevando *a-1* días allí, se cargaron en el barco ese día ($SP(0,a-1)$), en el caso de que se cargara alguno ($SP(0,a-1) \geq 0$). Si no se cargó ninguno el domingo ($SP(0,a-1) < 0$), todos los que quedaron almacenados en el puerto ese día, permanecerán almacenados el lunes ($DP(1,a) = DP(0,a-1)$).

Este desarrollo del modelo nos permite realizar predicciones a medio-largo plazo. Sin embargo, es preciso disponer de toda la información necesaria. Por ejemplo, es necesario conocer cuándo salen los barcos de cada puerto y qué capacidad tienen para poder hacer una buena planificación que minimize los costes a medio-largo plazo. Esta información, en muchos casos, no está disponible. En ese caso, se podrían hacer estimaciones y, posteriormente, realizar correcciones en los valores de entrada y en los parámetros para ajustarse a

lo ocurrido realmente y a las necesidades específicas del momento, mejorando la capacidad de respuesta frente a posibles cambios en las condiciones logísticas.

Una vez descrito el modelo, en el capítulo siguiente se detalla cómo se implementan los GA para resolver el problema de optimización expuesto en este capítulo. Posteriormente, se analizan los resultados obtenidos y se discute la adecuación de los GA así como su utilizad y facilidad de configuración y programación.

CAPÍTULO 4

IMPLEMENTACIÓN DEL ALGORITMO GENÉTICO

En este capítulo se describen las diferentes etapas en la implementación de los algoritmos genéticos empleados para resolver el problema de optimización descrito previamente. Así, se han implementado dos GA que permite encontrar soluciones cercanas al óptimo para este tipo de problemas complejos y de gran escala.

A continuación se presenta una descripción del funcionamiento de cada uno de los algoritmos, las librerías utilizadas, los operadores seleccionados y las diferentes etapas de la implementación. Para una mejor comprensión del funcionamiento dichos algoritmos puede consultarse los Apéndices 4 y 5, en los que se presenta el código comentado, para cada uno de ellos.

4.1. Algoritmo genético basado en ponderaciones (WBGA)

El diagrama que se muestra en la Figura 4.1 explica el funcionamiento del algoritmo genético basado en ponderaciones creado para resolver el problema de optimización. Este esquema muestra las etapas principales del proceso, desde la creación inicial de la población hasta alcanzar la solución óptima, destacando los operadores genéticos utilizados y las decisiones importantes en cada fase. En particular, se destacan las siguientes fases:

1. **Inicialización**: Creación aleatoria de la población inicial que actúa como punto de partida.

2. **Evaluación**: Cálculo de la aptitud de cada individuo según la función objetivo (*fintess*) definida.

3. **Selección**: Selección de los individuos más adecuados para reproducirse, utilizando el método de selección apropiado.

4. **Cruzamiento y Mutación**: Uso de operadores genéticos para investigar y mejorar el espacio de soluciones.

5. **Reemplazo**: Creación de una nueva población para la siguiente generación.

6. **Criterio de Terminación**: Comprobación del cumplimiento de las condiciones para parar el proceso, como llegar a un número máximo de generaciones o encontrar una solución adecuada.

Este diagrama ayuda a entender cómo funciona el algoritmo, mostrando sus partes principales y cómo interactúan mientras se busca la mejor solución.

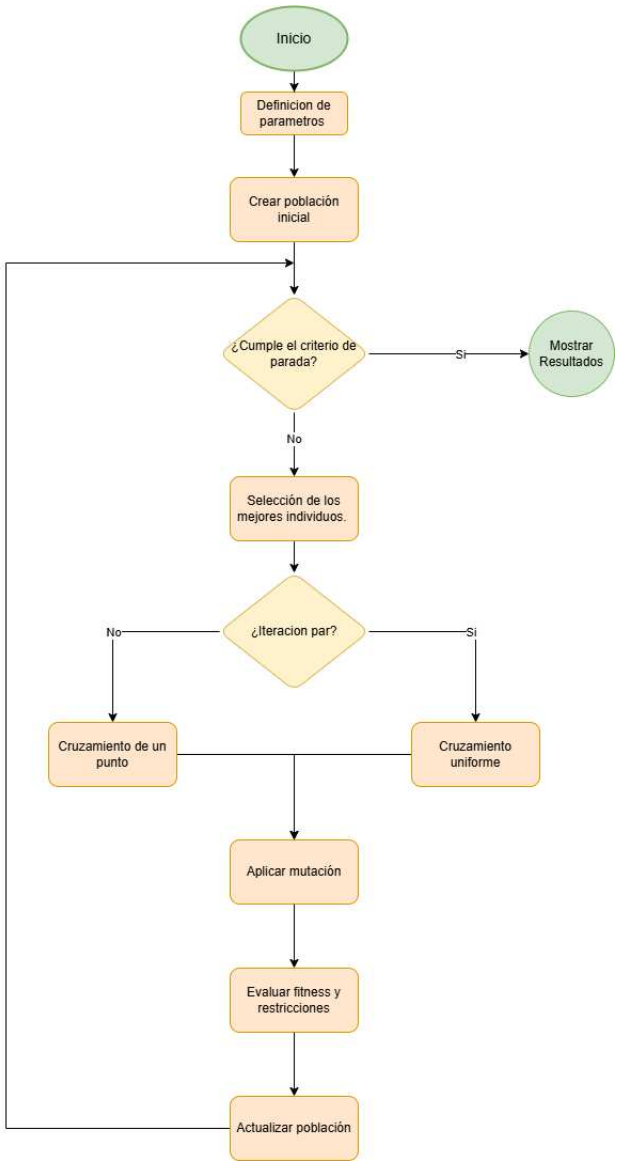

Figura 4.1: Diagrama de flujo

4.1.1. Implementación: Uso de Python y Librerías

Para llevar a cabo el algoritmo genético desarrollado se ha elegido **Python** por su simpleza, claridad y la amplia gama de librerías que proporciona para el análisis y la

visualización de datos. Python es perfecto para este tipo de proyectos por su fácil sintaxis, lo que facilita su implementación y corrección y el mantenimiento del código.

Las principales librerías utilizadas son:

- **Pandas**(Versión: 2.2.3): Permite el manejo eficaz de grandes cantidades de datos a través de *DataFrames*. Se emplea para manejar la población de individuos y guardar los resultados de cada repetición del algoritmo.
- **NumPy** (Versión: 2.1.2): Es crucial para llevar a cabo cálculos matemáticos y estadísticos de forma efectiva. Se utiliza realizar diferentes operaciones matemáticas y permite una generación y manejo de datos muy rápido.
- **Random**: Hace más fácil la creación de números al azar, que es esencial para elegir, combinar y cambiar individuos en cada generación.
- **Matplotlib** (Versión :3.10.0): Se utiliza para hacer gráficos fijos, permite ver cómo avanza el algoritmo, el desarrollo del *fitness*, a través de las iteraciones.
- **Seaborn** (Versión: 0.13.2): Basada en Matplotlib, ofrece una manera más fácil y atractiva de hacer gráficos estadísticos, mejorando la interpretación visual de los resultados.

El uso de Python y las diferentes librerías permite una puesta en marcha efectiva, con un control simplificado de información y cálculos, además de una presentación clara y exacta de los resultados obtenidos en cada repetición. Esto ayuda tanto al análisis como a la selección de decisiones durante el proceso de mejora.

4.1.2. Inicialización y evaluación de población inicial

La etapa de inicio y formación de la población es esencial en cualquier algoritmo genético, ya que establece el comienzo de la búsqueda de soluciones. En este caso, el algoritmo se crea para distribuir recursos a diferentes destinos y cumplir con restricciones logísticas. A continuación, se muestra un resumen del proceso de formación de la población.

La población inicial está compuesta por un grupo de individuos, cada uno representando una solución posible al problema. Los individuos se organizan en un *DataFrame*, donde cada fila corresponde a un individuo y cada columna muestra una asignación específica para un destino en un día particular.

Cada individuo se representa utilizando una codificación entera. En esta codificación, cada valor en el cromosoma es un número entero que indica la cantidad de recursos asignados a un destino en un día particular. Cada cromosoma es un vector de valores enteros, donde cada valor x_i representa la cantidad de recursos (vehículos) asignados al destino correspondiente en el día i (ver Figura 4.2).

Los valores asignados a cada individuo se establecen de forma que se satisfagan las restricciones de capacidad. Así, por ejemplo, para el destino Pasajes, cuyo límite máximo es 600, se generan valores enteros aleatorios en el rango de 0 a 600 para cada uno de los 7 días considerados. Lo mismo ocurre con el resto de los destinos. Este método de codificación entera con valores límite garantiza que las soluciones iniciales sean discretas y se ajusten a las restricciones de capacidad de los destinos.

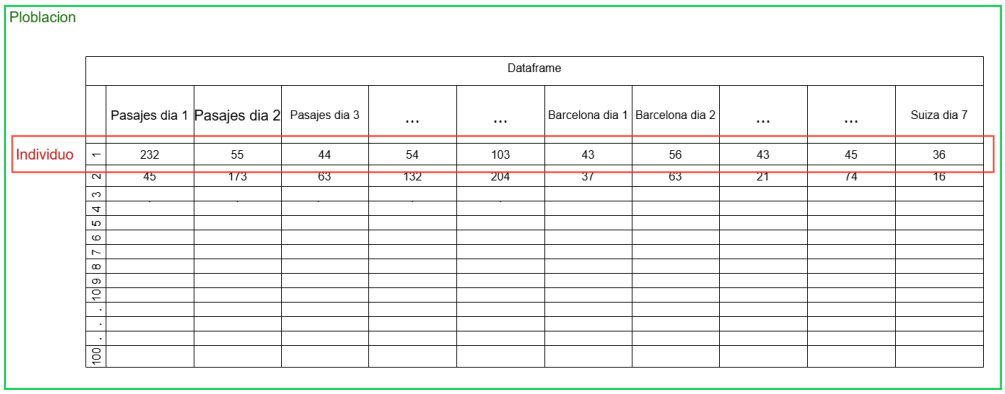

Figura 4.2: Ejemplo de población

Una vez creada la población, cada individuo es evaluado mediante funciones de *fitness* y de cumplimiento de las restricciones. La función de *fitness* calcula el valor de la solución basado en los objetivos establecidos en relación a los costes, la ocupación y la regularidad. Adicionalmente, se verifica el cumplimiento de restricciones específicas del problema, y se calcula una métrica combinada (*Fitness+Rest*) que penaliza a los individuos que no cumplen con estas restricciones. Además, se identifican dinámicamente las columnas relacionadas con las asignaciones utilizando patrones regulares. Esto permite organizar de manera flexible la población y facilita su manipulación en las etapas siguientes de cruzamiento y mutación. Esto se explicará con más detalle en el apartado de evaluación.

Durante la ejecución del algoritmo, la población inicial irá creciendo, evolucionando y mutando, dando lugar a nuevas generaciones que mejoran los resultados anteriores, hasta encontrar la solución adecuada o alcanzar un límite de generaciones.

4.1.3. Selección de los individuos a cruzar

La elección de individuos para el cruce es una de las fases importantes en el funcionamiento de los algoritmos genéticos. Se debe garantizar que los mejores individuos tienen una mayor posibilidad de reproducirse frente a los individuos menos buenos. Se debe dar también una oportunidad a los individuos menos buenos para mantener la diversidad genética en el proceso de reproducción.

La estrategia de selección que se ha escogido en este caso es la selección por ruleta (*Roulette Wheel Selection* (RWS)). Este procedimiento intenta encontrar a los individuos más capaces de la población presente al mismo tiempo que se mantiene la diversidad genética, aplicando un método de selección aleatoria fundamentado en la probabilidad acumulada de capacidad normalizada. De esta forma los individuos con mayor (menor) aptitud tienen más (menos) posibilidades de ser seleccionados, pero no garantiza que sean siempre escogidos (rechazados). Esto ayuda a mantener una diversidad genética en las primeras generaciones, explorando más soluciones potenciales. Se trata de un método simple de implementar y, a diferencia de otros métodos de selección (por torneo o elitismo), permite que individuos menos aptos tengan la posibilidad de influir en la solución, lo que puede evitar la convergencia prematura hacia un óptimo local.

Los pasos esenciales del proceso de selección se explican a continuación:

- **Asignación de probabilidades a cada individuo**: se asigna a cada individuo de la población una probabilidad p_i de ser seleccionado que es proporcional a su *fitness*:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

  donde *n* denota el tamaño de la población en términos del número de individuos. la función de *fitness* se detallará posteriormente.

- **Generación de umbrales aleatorios**: Se crean dos números al azar r_1 y r_2 en el rango [0, 1]. Estos valores son los límites de selección y establecen los puntos de corte en la distribución acumulada de habilidad de los individuos, afectando así la probabilidad de elección.

- **Copia de datos y cálculo de indicadores**: Se hace una copia de la información de la población, añadiendo columnas extras que muestran si la probabilidad total de adaptación de cada individuo supera r_1 o r_2.

- **Identificación de padres**:A partir de las columnas creadas, se reconocen los valores más bajos que sobrepasan cada límite (r_1 y r_2). Los individuos que corresponden a estos valores más bajos son seleccionados como **padres** de la futura generación.

- **Preparación de los padres para el cruce**: Los padres elegidos se sacan de la población y se organizan para la reproducción quitando columnas adicionales como identificadores y medidas específicas (*Fitness*, *Restricciones*, etc.) que no son necesarias para el proceso de cruce.

Como se ha señalado anteriormente, las principales ventajas de la selección por ruleta son:

- **Mantenimiento de diversidad y aptitud**: Este enfoque garantiza variedad en la elección al utilizar valores al azar en cada repetición, al mismo tiempo que favorece a los individuos más capacitados gracias a la aplicación del *Fitness* total como estándar de selección.

- **Prevención de la selección de grupos dominantes**: Se evita la selección repetida de un único grupo poderoso, lo que deja abierto un mayor espacio para descubrir soluciones.

4.1.4. Cruce de Individuos

Una vez seleccionados los padres, se cruzan usando técnicas de cruce particulares. El cruce es el proceso a través del cual se crean nuevos individuos (**hijos**) a partir de los padres elegidos, uniendo sus rasgos para descubrir nuevas soluciones en el área de búsqueda. A continuación, se explican los dos métodos de cruce utilizados en el algoritmo genético implementado:

- **Cruce Porcentual**: este método permuta columnas entre los padres según un porcentaje determinado (Figura 4.3):

- Se determina la cantidad de columnas a cruzar,n, multiplicando el porcentaje de cruzamiento (`porcentaje_cruzamiento`) por el total de columnas del individuo.
- Las columnas elegidas para el cruce son las primeras n columnas de los dos padres.
- Dos nuevos individuos (hijos) se inicializan como copias de sus padres. El primer hijo es una copia del primer padre y el segundo del segundo padre.
- Se intercambian las columnas impares seleccionadas: Las columnas del segundo padre reemplazan las del primero en el hijo 1. Las columnas del primer padre reemplazan las del segundo en el hijo 2.
- Finalmente, los hijos se transforman en listas de posiciones para ser usados en las siguientes etapas.

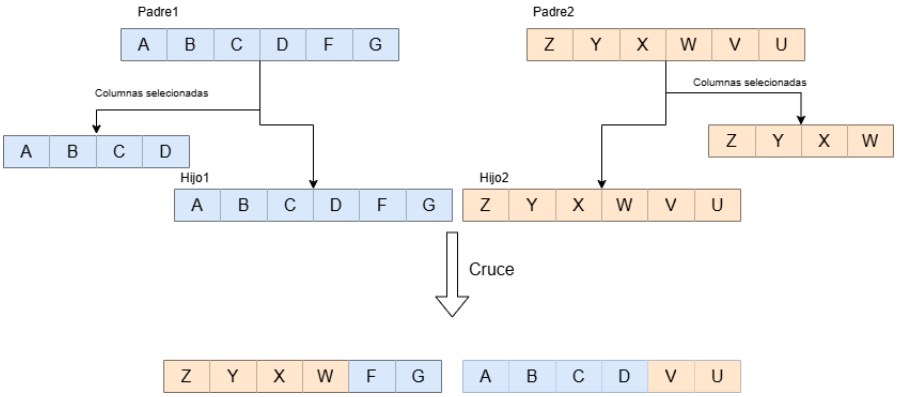

Figura 4.3: Cruce Porcentual

- **Cruce por Segmentos**: Este método intercambia bloques de variables, segmentos, entre los padres para formar los nuevos individuos (Figura 4.4):

  - Se define el tamaño de segmento para dividir la información de los padres en bloques. En este caso, el tamaño del segmento es 7 (días de la semana) de forma que cada bloque contiene los valores semanales de cada destino (Pasajes, Barcelona, etc.).
  - Se alternan los segmentos: los segmentos impares del primer padre se asignan al primer hijo y los del segundo padre, al segundo hijo. Los segmentos pares del segundo padre se asignan al primer hijo, y los pares del primer padre al segundo hijo.
  - Los bloques concatenados forman los nuevos individuos (hijos).

Si bien se podrían utilizar otros tipos de cruce y diferentes variaciones de los cruces propuestos, los que se han utilizado en el algoritmo implementado nos permiten mantener la variedad genética al facilitar nuevas combinaciones de rasgos de los progenitores y ofrecen un equilibrio entre explorar y usar el espacio de búsqueda.

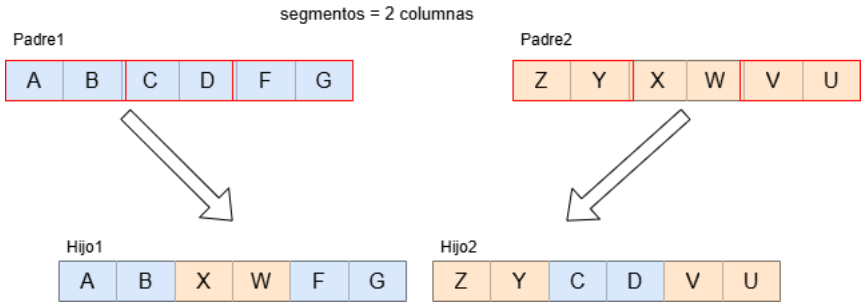

Figura 4.4: Cruce por Segmentos

4.1.5. Mutación de Individuos

El proceso de mutación añade diversidad en la población al cambiar al azar las posiciones de ciertos individuos. Este mecanismo facilita la exploración de nuevas áreas del espacio de búsqueda y previene la convergencia temprana hacia soluciones locales. A continuación, se explican los principales pasos de este proceso.

En el caso específico de este algoritmo, se utiliza mutación gaussiana, también conocida como mutación basada en perturbaciones normales. Este tipo de mutación implica agregar un valor aleatorio generado a partir de una distribución normal con media cero y desviación estándar controlada a cada valor de un individuo seleccionado aleatoriamente. El proceso se desarrolla en diferentes etapas:

- **Creación del *DataFrame* de Mutantes:**
  - Se inicializa un *DataFrame* vacío (`popm`) para almacenar los individuos mutantes.
  - Los identificadores de los mutantes se asignan de forma secuencial a partir del índice actual de la población.
  - Para cada dimensión y destino, se crean columnas que inicialmente se rellenan con valores `NaN`.
- **Selección de Individuos Base para Mutación:**
  - Se selecciona aleatoriamente un individuo de la población existente como base para cada mutante.
  - Los valores del individuo base se copian para aplicar las mutaciones.
- **Aplicación de la Mutación:**
  - Se define un vector de desviación estándar (σ) proporcional a los límites superiores (`ub`) para ajustar la magnitud de las mutaciones.
  - Para cada posición con una probabilidad determinada por la tasa de mutación (μ), se agrega un valor al azar, creado a partir de una distribución normal con media 0 y desviación estándar σ.
  - Las posiciones finales se ajustan y transforman en números enteros. También se establecen límites mínimos y máximos (`lb` y `ub`) para garantizar que las mutaciones sean válidas.

Una vez realizadas las mutaciones, los individuos mutados se añaden a la población para poder ser evaluados a continuación junto con el resto de los individuos de nueva creación resultantes del cruce.

El proceso de mutación apoya la selección y el cruce al añadir una fuente extra de diversidad, aumentando de esta forma la habilidad del algoritmo genético para encontrar soluciones creativas. Este método garantiza que la población no se quede atrapada en un óptimo local al hacer pequeñas alteraciones en los individuos actuales.

4.1.6. Evaluación de Individuos y Restricciones

Una vez generados los nuevos individuos (**hijos**) y (**mutantes**) a través de los operadores de cruce y mutación, se debe analizar su rendimiento en relación con el problema a resolver. Este proceso abarca el cálculo de medidas importantes, la organización de la población y el manejo de la convergencia del algoritmo. A continuación, se describen las etapas clave.

4.1.6.1. Identificación de los Nuevos Individuos

La nueva generación de individuos de identifica de la siguiente manera:

- A cada nuevo individuo generado se le asigna un identificador único basado en el índice de iteración actual y su posición relativa en la población.
- Los datos de los individuos son agregados a un nuevo conjunto temporal de población (`popc_it`).

4.1.6.2. Cálculo de *Fitness* y Restricciones

Se evalúa la aptitud de cada individuo como solución a través de la función de *fitness* y del cumplimiento de las restricciones:

- El *fitness* de cada individuo se calcula mediante una función objetivo que combina los tres objetivos principales planteados en el Capítulo 3:
  1. **Función de ocupación**: `occupation_value`. Evalúa la eficiencia del uso de las diferentes zonas de la campa.
  2. **Función de regularidad**:`regularity_value`. Mide la homogeneidad del tráfico a lo largo de la semana con el objetivo de lograr una distribución equilibrada de la carga de trabajo
  3. **Gestión de puertos**: `gestion_puer`. Evalúa los costes asociados al almacenamiento en puertos considerando buffers y límites de capacidad con el objetivo de hacer un uso eficiente de los recursos:
- La puntuación final de *fitness* se calcula como una suma ponderada de estas tres funciones:

$$\texttt{fitness\_value} = M_1 \cdot \texttt{occupation\_value} + M_2 \cdot \texttt{regularity\_value} + \\ + M_3 \cdot \texttt{gestion\_puer} \tag{4.1}$$

Donde M_1, M_2 y M_3 son las ponderaciones que establece el usuario o experto en función de las prioridades organizacionales o las condiciones del entorno. Hay que tener en cuenta, que el objetivo del problema de optimización es la minimización de las tres funciones objetivo. Por este motivo, los individuos con mayor aptitud serán aquellos con un menor valor del `fitness_value`.

- Las restricciones del problema son evaluadas mediante un conjunto de penalizaciones. Como se ha señalado en el Capítulo 3, el conjunto de restricciones hacen referencia a limitaciones operativas, temporales y de capacidad, así como de no negatividad de las variables del modelo.
- Se penalizan las soluciones que no cumplen las restricciones mediante un factor de prioridad (`restr_prio`):

$$\texttt{Fitness+Rest} = \texttt{Fitness} + \texttt{restr\_prio} \cdot \texttt{Restricciones} \tag{4.2}$$

En el caso de que se cumplan las restricciones $\texttt{restr\_prio} = 0$.

4.1.6.3. Actualización de la Población

La actualización se desarrolla siguiendo el proceso que se detalla a continuación:

- Los nuevos individuos se agregan a la población existente.
- La población completa se ordena en función de la métrica `Fitness+Rest` para priorizar los mejores individuos.
- Los mejores individuos son seleccionados para formar la nueva población, mientras que los restantes son descartados.

4.1.6.4. Almacenamiento de los Mejores Resultados

Una vez actualizada la población se almacenan todos los resultados:

- Se almacena el mejor individuo sin considerar restricciones (`pob_res`) para referencia del usuario.
- El mejor individuo considerando restricciones (`xMejor`) se guarda como la solución actual.
- La peor solución también se registra para análisis de desempeño.

4.1.6.5. Control de Convergencia

El control de convergencia es el mecanismo que controla y regula el proceso evolutivo para garantizar que el algoritmo progrese adecuadamente hacia una solución óptima. Para llevar a cabo el control de convergencia:

- Se analiza si el *Fitness+Rest* del mejor individuo no mejora después de un número definido de iteraciones (`rep`), el algoritmo detiene su ejecución.

- Se utiliza un contador de control para monitorear las iteraciones sin mejora.

Si la convergencia se considera adecuada o si no hay mejoras, se detiene la ejecución del algoritmo.

4.1.6.6. Registro de Iteraciones

Se guarda información sobre el estado del algoritmo en cada iteración o generación, incluyendo información del mejor individuo (*Fitness*, *Restricciones*, y posiciones de las variables).

Este proceso permite comprobar que la comunidad avance hacia mejores soluciones en cada repetición al mismo tiempo que respeta las limitaciones del problema.

4.1.6.7. Muestreo y Obtención de la Solución Final

El proceso de selección de muestras y la elección de la solución final son pasos importantes para resumir los resultados obtenidos durante las iteraciones del algoritmo genético. Este procedimiento permite analizar la convergencia y obtener las mejores soluciones encontradas. A continuación, se presenta el flujo de trabajo:

- **Registro de Métricas por Iteración:** Para cada iteración, se almacena un resumen de las métricas clave en un *DataFrame* denominado `resumen_iteraciones`. Las métricas incluyen:
  - El mejor valor de *fitness*.
  - El mejor valor combinado de *Fitness+Rest*.
  - Los valores de las restricciones.
  - Las posiciones correspondientes al mejor individuo.
- **Visualización de la Evolución del *Fitness*:** Se generan gráficos lineales utilizando `seaborn` para visualizar la evolución del mejor *fitness* y *Fitness+Rest* a lo largo de las iteraciones. Los gráficos incluyen:
  - Un gráfico de línea azul para *fitness* sin restricciones.
  - Un gráfico de línea verde para *Fitness+Rest*, que incluye las penalizaciones por restricciones.

  Estos gráficos permiten identificar tendencias y evaluar la convergencia del algoritmo.
- **Exportación de Resultados:** Al finalizar las iteraciones, los datos del resumen se exportan a un archivo Excel (`resumen_iteraciones.xlsx`) para su posterior análisis y validación. Se incluyen los mejores resultados encontrados con y sin restricciones en las últimas filas del *DataFrame*.
- **Identificación de la Solución Óptima:** Se selecciona como solución final el mejor individuo de la última iteración, considerando tanto el *Fitness* como el cumplimiento de las restricciones. Una solución que no cumple las restricciones no se considera una solución válida; no obstante, se guarda el mejor resultado sin restricciones como referencia adicional porque puede servir para realizar mejoras o modificaciones ne los recursos disponibles.

- **Interacción con el Usuario:** Finalmente, el algoritmo muestra los gráficos generados y espera una confirmación del usuario antes de finalizar la ejecución.

Este procedimiento garantiza que el usuario pueda ver y entender los resultados del algoritmo genético de forma clara y organizada, ayudando en la interpretación de los datos y en la elección de las soluciones sugeridas. En este sentido, el método permite analizar una solución concreta obtenida con los parámetros específicos actuales. Dicho análisis se realizará en el siguiente capítulo.

4.2. Algoritmo genético basado en frentes de Pareto (NSGA-II)

El algoritmo genético de ordenamiento no dominado II (NSGA-II) es un algoritmo ampliamente utilizado para resolver problemas de optimización multiobjetivo. Su principal característica es que busca soluciones que no sean dominadas por otras [1].

El algoritmo genético NSGA-II desarrollado se ha implementado también con **Python** y además de las librerías **NumPy** (Versión: 2.1.2), **Random**, **Matplotlib** (Versión :3.10.0) y **Seaborn** comentadas anteriormente, se ha utilizado **DEAP** (Distributed Evolutionary Algorithms in Python , Versión :1.4.0). DEAP es un framework basado en el lenguaje de programación Python para cálculo evolutivo. Trabaja fácilmente con paralelización y multiprocesamiento. Su objetivo es proporcionar herramientas prácticas para la creación rápida de prototipos de algoritmos evolutivos personalizados, donde cada paso del proceso sea lo más explícito (como un pseudocódigo) y fácil de leer y entender como sea posible [22]. Además, nos brinda un conjunto de herramientas que nos permite implementar y testear rápidamente algoritmos genéticos .

El mecanismo de funcionamiento del NSGA-II se puede ver a partir de su diagrama de flujo (ver Figura 4.5 [23])

A continuación se describen las diferentes etapas del proceso [24]:

1. **Generación de la población inicial P_0:** Se genera aleatoriamente una población inicial de individuos. Cada individuo representa una solución, un vector de variables de decisión.

2. **Evaluación de las funciones de aptitud:** Se evalúa la aptitud de cada individuo en función del valor alcanzado en cada una de las funciones objetivo del problema, teniendo en cuenta el cumplimiento de las restricciones. Así, las soluciones que no cumplen las restricciones se penalizan de forma que su aptitud se reduce, con una reducción mayor cuanto mayor sea el número de restricciones incumplidas.

3. **Clasificación por dominancia:** Se ordena la población en niveles de dominancia (frentes): El primer frente F_1 está formado por los individuos que no son dominados por ningún otro individuo. El segundo frente, F_2, está compuesto por los individuos que solo están dominados por los del primer frente, F_1, el F_3 por el F_2y así sucesivamente.

[1] En un problema de optimización multiobjetivo, decimos que una solución domina a otra cuando responde mejor en al menos uno de los objetivos y no peor en los demás.

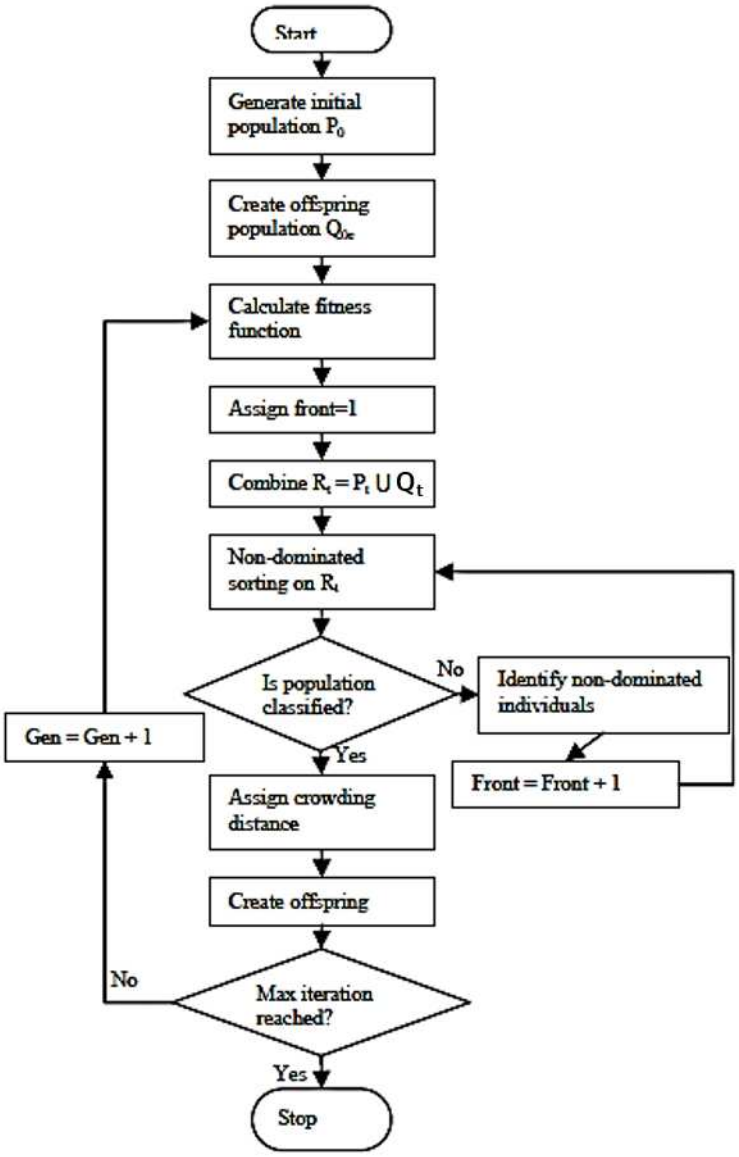

Figura 4.5: Diagrama de flujo del NSGA-II [22].

4. **Cálculo de la distancia de aglomeración:** Se calcula la distancia de aglomeración (crowding distance) de cada individuo dentro de cada frente. Esta distancia mide la densidad de individuos en el espacio objetivo. Individuos con mayor distancia de aglomeración tienen mayor diversidad y se consideran preferibles.

5. **Selección de los mejores padres**: Sobre la población actual se seleccionan N parejas de soluciones escogidas aleatoriamente. De cada pareja de soluciones, se elige la alternativa que pertenezca al frente con menor rango. Si las alternativas en competencia pertenecen al mismo frente, entonces se elige aquella con una distancia de aglomeración mayor, ya que así se introduce un mayor grado de diversidad en la generación en construcción. Los N individuos elegidos, P_0, son la base para obtener descendencia.

6. **Aplicación de operadores de cruce y mutación:** Se aplican los operadores de cruce y mutación para generar la población descendiente (Q_0). En este caso, se han utilizado los mismos operadores que en el WSGA.

7. **Preselección y preservación de las soluciones de élite:** se reúnen los padres y los descendientes obtenidos por medio de los operadores de selección, cruce y mutación, creando un conjunto de tamaño $2N$: $R_0 = P_0 \cup Q_0$. Este conjunto de soluciones se clasifica en sus respectivos frentes de dominancia y se seleccionan los individuos que pertenezcan a los frentes de menor rango hasta obtener una nueva generación de N individuos, P_1. Si no es posible seleccionar todas las alternativas de un frente determinado, se eliminan aquellos individuos con una menor distancia de aglomeración (ver Figura 4.6[25])

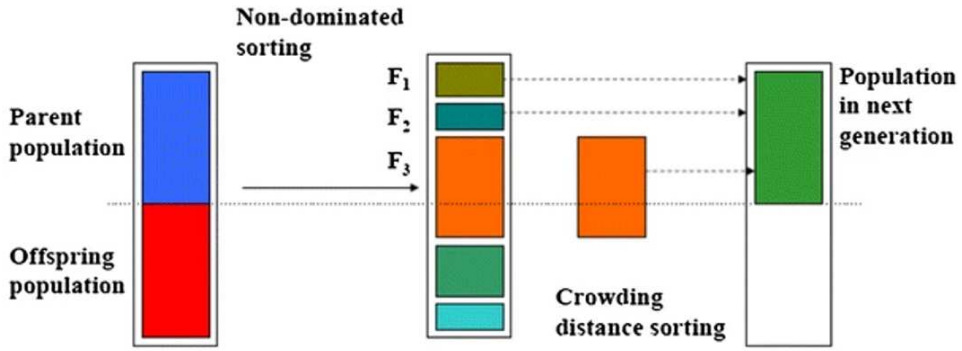

Figura 4.6: Mecanismo de evolución del NSGA-II [24].

8. **Repetición y fin del proceso:** se repite el proceso de selección, cruce, mutación y actualización generacional ($P_2, P_3, ...$) hasta que se cumplan los criterios de finalización, como llegar a un número máximo de generaciones o alcanzar un umbral de convergencia.

La población final contiene las soluciones Pareto-óptimas que representan mejores soluciones en un equilibrio de los objetivos en conflicto, permitiendo que el usuario seleccione la mejor según sus necesidades. Señalar que las soluciones obtenidas satisfacen siempre las restricciones, ya que las penalizaciones introducidas por el incumplimiento de las mismas, hacen que las soluciones que no las cumplen, solo sobrevivan al cambio generacional después de un número limitado de iteraciones.

Al igual que en el caso del algoritmo WBGA, este algoritmo lleva a cabo diferentes funciones como son: el control de convergencia, el registro de iteraciones y métricas, exportación y visualización de las soluciones óptimas y la interacción con el usuario, explicadas en la sección anterior.

CAPÍTULO 5

SOLUCIÓN AL PROBLEMA Y CASOS DE USO

Evaluar los resultados de un algoritmo genético supone contrastar la calidad de las soluciones que genera y la eficiencia del mismo en términos de su desempeño. Esta evaluación se puede llevar a cabo mediante diferentes criterios en función de la naturaleza del problema de optimización.

En este capítulo se analiza, en primer lugar, la calidad de las soluciones obtenidas por los algoritmos genéticos desarrollados en el capítulo anterior a través de casos de uso, para, posteriormente evaluar la convergencia, flexibilidad, robustez, escalabilidad y el consumo de recursos de los mismos. El análisis se realiza para cada uno de los algoritmos implementados.

5.1. Algoritmo genético basado en ponderaciones (WBGA)

En contextos reales, el análisis de la solución generada por un GA a través de casos de uso constituye una herramienta que permite probar y perfeccionar los GA y medir su rendimiento. Se trata de escenarios específicos diseñados con ese propósito. A continuación se presenta varios escenarios en los que se reflejan situaciones concretas cercanas a la realidad de la empresa. Para ello se establecen los valores iniciales de los parámetros, así como las ponderaciones de los diferentes objetivos en función de las necesidades de la empresa.

5.1.1. Evaluación de los resultados: casos de uso

Las condiciones iniciales tienen un efecto importante en la definición del problema, determinando, en mayor medida, cuál de los objetivos planteados debe tener un mayor peso. En este sentido, si, por ejemplo, en un momento determinado, los parámetros iniciales muestran una alta ocupación del área de almacenamiento de la campa que puede generar cuellos de botella en la producción (es decir, el espacio destinado para el almacenamiento de coches en la fábrica está casi lleno), la solución se enfocará en mover vehículos fuera del área lo antes posible, por lo que el objetivo prioritario será aquel que está relacionado con

la minimización de la capacidad de almacenamiento de la campa, evitando sobrecargas y asegurando un flujo constante. Si, por el contrario, el transporte marítimo desde los puertos es escaso o irregular y, por lo tanto, una salida diaria regular hacia los puertos puede generar costes logísticos de almacenamiento allí, la prioridad podría ser reducir dichos costes. En este caso, el algoritmo debe buscar soluciones que programen la entrega de los coches en fechas lo más cercanas posibles a la salida de los barcos, optimizando el uso del espacio y disminuyendo los costes derivados de los retrasos o tiempos de inactividad.

Este comportamiento, propio de los problemas con varios objetivos, pone de manifiesto la necesidad de ajustar el enfoque (las ponderación de la función de *fitness*) según las prioridades definidas. Dependiendo de las circunstancias del sistema, los profesionales en transporte pueden elegir qué objetivos se deben priorizar, si aumentar la eficiencia operativa o reducir los costes logísticos, por ejemplo.

A continuación, se analiza cómo estas configuraciones personalizadas pueden llevarse a cabo de manera efectiva, permitiendo que la herramienta se adapte a las necesidades específicas y a los criterios establecidos por los usuarios.

5.1.1.1. Ponderación de las funciones objetivo

La formulación del problema puede adaptarse para priorizar aquellos objetivos que sean más relevantes según el contexto o las necesidades específicas en cada momento. Esto permite ajustar el enfoque de la solución hacia metas que maximicen los objetivos de la empresa en escenarios particulares.

Actualmente, en el caso presentado, las indicaciones de los expertos y las necesidades más comunes en la fábrica de Vitoria establecen como objetivo principal la **minimización de los costes de almacenamiento en los puertos**. Esta decisión responde a:

- **Relevancia estratégica:** Reducir costes en los puertos tiene impacto financiero significativo y un efecto directo e inmediato en los márgenes operativos. La minimización de la ocupación y la regularidad en el transporte, aunque importantes, son objetivos secundarios que complementan al principal.
- **Limitaciones del sistema:** Algunas restricciones logísticas hacen que ciertos objetivos no sean igualmente alcanzables en todas las condiciones, por lo que es necesario priorizar.

Se ha establecido una ponderación específica (pesos de los diferentes objetivos en la función de *fitness*) para cada uno los objetivos, que refleja estas prioridades estratégicas y permite al algoritmo genético dar más valor a aquellas soluciones que se alinean con las mismas:

- **Minimización de costes de almacenamiento:** Peso 0.6
- **Minimización de la ocupación de la campa:** Peso 0.3
- **Regularidad en el transporte:** Peso 0.1

Esta ponderación puede ser ajustada según cambien las prioridades organizacionales o las condiciones del entorno. Como veremos más adelante, a través de las ponderaciones,

la herramienta garantiza la flexibilidad y adaptabilidad a nuevos objetivos o situaciones, asegurando que la solución es relevante y eficiente bajo las nuevas condiciones del entorno. Esto es especialmente relevante en un entorno industrial que está sujeto a cambios en demandas, regulaciones y capacidades logísticas.

5.1.1.2. Evaluación de los resultados y comparación con la solución real

El proceso de evaluación se realiza a continuación se divide en dos etapas. En la primera etapa, se revisan los resultados obtenidos con el algoritmo genético, examinando indicadores como el coste de almacenamiento, la utilización de la campa y la consistencia en la cantidad de vehículos transportados. Estos indicadores ayudan a decidir si el algoritmo cumple con los objetivos planteados y si se dirige hacia soluciones óptimas en términos logísticos y económicos. Para lograr resultados relevantes y comparables con la realidad se han establecido los parámetros iniciales del algoritmo basados en las condiciones reales observadas en febrero de 2024.

En la segunda etapa, se contrastan los resultados obtenidos con la información real del período de referencia. Así, se incluyen los datos reales de capacidad portuaria, circulación de vehículos, costes de almacenamiento y otras limitaciones específicas del periodo. Este contraste tiene como objetivo comprobar la eficiencia del modelo propuesto y examinar las posibles diferencias entre el resultado teórico y la realidad. Estas variaciones pueden ser de utilidad para localizar posibles áreas de mejora, tanto en el modelo como en los procedimientos operativos.

El escenario propuesto se detalla en el Apéndice 1 que incluye los valores iniciales de la producción diaria por destino, de la capacidad de los barcos y de los vehículos almacenados en los diferentes puertos.

Los resultados obtenidos con la implementación del algoritmo genético muestran una clara tendencia del proceso hacia soluciones que se acercan al mejor resultado dentro de los límites establecidos. Al detener el algoritmo después de 500 repeticiones sin un cambio notable, se alcanza una solución que cumple con todas las condiciones definidas. Durante las primeras 500 repeticiones, se aprecia una mejora significativa en los resultados, con el algoritmo superando claros óptimos locales y siguiendo hacia soluciones más eficientes. A partir de la repetición 2000, las mejoras son más lentas, mostrando la convergencia del algoritmo hacia una solución fija (ver figura 5.1).

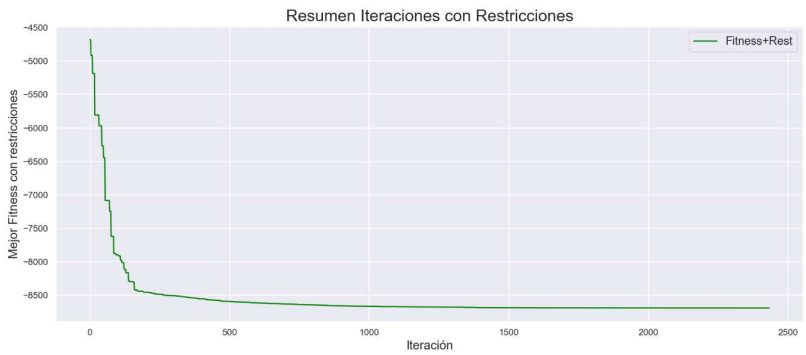

Figura 5.1: Resultado Algoritmo Genético

Si se analizan los resultados por objetivos, en lo que respecta la minimización de los costes de almacenamiento en los puertos (con ponderación 0.6), se observa que el número de vehículos almacenados es reducido y permanecen almacenados durante un tiempo breve.

Este resultado refleja una gestión más eficiente de los recursos portuarios que la llevada a cabo, reduciendo totalmente los costes. Los detalles concretos acerca de los puertos apoyan estos resultados. La ocupación de los almacenes de los puertos resultante es:

- Puerto Pasajes (buffer de 4 elementos): [521, 0, 0, 0]
- Puerto Barcelona (buffer de 22 elementos): [93, 0, 0, 0,.....]
- Puerto Valencia (buffer de 11 elementos): [0, 0, 0, 0,.....]
- Puerto Santander (buffer de 22 elementos): [26, 28, 5, 0„.....]

En relación a la minimización de la ocupación de la campa (con ponderación 0.3), los resultados muestran, según el criterio de los expertos del departamento de envíos, una situación muy buena, con niveles muy aceptables de ocupación (774) en relación a la ocupación habitual, asegurando una buena operatividad en el momento actual y dejando también margen suficiente para manejar posibles incrementos en el volumen de vehículos en los días siguientes, lo que garantiza la sostenibilidad de las operaciones logísticas a corto y medio plazo.

La función de regularidad es la función objetivo con una ponderación más baja (0.1), por lo que su impacto en la solución es menor en comparación con los objetivos anteriores. Sin embargo, esto no significa que su efecto sea despreciable. Por el contrario, garantiza que no haya grandes diferencias entre los volúmenes de vehículos manejados en distintos días, ayudando de forma secundaria, pero significativamente, a la estabilidad y previsibilidad del sistema logístico. Con un valor de regularidad de 253.31, el modelo, como veremos más adelante, logra mantener una estabilidad en las operaciones.

En relación a las restricciones del modelo, señalar que se cumplen todas ellas, lo que pone de manifiesto que la penalización asociada a su incumplimiento actúa de manera efectiva, guiando al algoritmo hacia el conjunto de soluciones factibles.

El *fitness* total del modelo alcanza los 257.53, lo que valida la efectividad del algoritmo en la optimización del sistema bajo las condiciones iniciales establecidas, tal y como se ver a continuación al compararlo con los resultados reales del periodo de referencia proporcionados por los expertos:

- **Minimización de la ocupación de la campa:** El valor real de la ocupación es menor, resultando en 524 coches almacenados frente a los 774 coches proporcionados por el algoritmo; no obstante, este último es inferior al 40 % de la capacidad total.
- **Gestión de puertos:** Los buffers de los puertos del periodo de referencia tienen la siguiente distribución:
  - **Puerto Pasajes:** [356, 297, 0, 0,]
  - **Puerto Barcelona:** [38, 42, 0, 0, 0,.....]
  - **Puerto Valencia:** [0, 0, 0, 0, 0, ...]

- **Puerto Santander:** [8, 12, 8, 0, 0,.....]

El valor real de gestión de puertos es significativamente mayor, alcanzando 376.74 frente a un valor nulo, lo que implica que el algoritmo proporciona la mayor mejora posible.

- **Función de regularidad:** La regularidad proporcionada por el algoritmo (253.31) es también superior a la solución de los expertos (334.57). Se observa una mejora próxima al 25 %.

- ***Fitness* total**: En el caso de la solución real del periodo de referencia el *itness* total tiene un valor de 416,77, frente a 257.53 proporcionado por el algoritmo, lo que supone una mejora cercana al 40 %.

Es conveniente señalar que durante el periodo de referencia surgieron problemas que no pueden ser considerados en la solución proporcionada por el algoritmo genético:

- 18 coches fueron afectados por problemas logísticos.
- 10 coches sufrieron daños durante las operaciones de aparcamiento o transporte.
- 8 coches fueron impactados por acciones correctoras en la producción.

Las soluciones generadas por los expertos, aunque menos eficientes en términos de la función de *fitness*, presentan ciertas ventajas derivadas de su capacidad para considerar factores externos que nuestro modelo no puede integrar directamente. Estos factores incluyen condiciones meteorológicas, vehículos averiados, rectificaciones logísticas y excepciones a las restricciones. Así, por ejemplo, en situaciones de capacidad limitada, los expertos pueden optimizar la disposición de los vehículos en la campa o recurrir al uso de campas auxiliares. Asimismo, en casos donde las condiciones meteorológicas impiden la salida de los vehículos desde los puertos debido a la imposibilidad de zarpar de los barcos, los expertos pueden implementar mecanismos de contingencia para mitigar los problemas, tales como la reorganización de rutas o el almacenamiento temporal en instalaciones alternativas.

Si bien algunas de la situaciones señaladas podrían simularse cambiando los parámetros del modelo, el algoritmo puede carecer de la flexibilidad adaptativa que poseen los expertos al gestionar situaciones imprevistas, lo podría generar soluciones subóptimas o ineficientes en determinadas circunstancias. A pesar de esto, la capacidad del modelo para minimizar costes, maximizar la eficiencia en la ocupación y mantener la regularidad bajo condiciones de regularidad, lo posiciona como una herramienta valiosa para complementar la toma de decisiones.

En definitiva, las soluciones propuestas por el algoritmo representan un buen punto de partida desde una perspectiva matemática y globalmente pueden considerarse como alternativas que mejoran la logística actual. No obstante, habría que tener en cuenta que en escenarios reales en los que puede haber problemas o imprevistos como los señalados anteriormente, su implementación debería ir acompañada de un enfoque híbrido, donde las decisiones basadas en el modelo se ajusten mediante la experiencia y el conocimiento de los expertos para garantizar la operatividad en un entorno logístico dinámico.

5.1.2. Evaluación del Algoritmo

En esta sección se evalúan diferentes aspectos del algoritmos en relación a su desempeño y rendimiento, en base a diferentes criterios, abordando tanto su comportamiento técnico como los recursos necesarios para su ejecución.

5.1.2.1. Estabilidad: repetición del GA

Un algoritmo genético WBGA genera soluciones diferentes si se repite el GA porque utiliza operadores probabilísticos e inicialización aleatoria en cada ejecución:

- Inicialización Aleatoria: En cada ejecución, la población inicial de soluciones candidatas se genera aleatoriamente, lo que significa que el punto de partida siempre es distinto.
- Selección Estocástica: la selección de individuos para reproducción se realiza a través de un operador por Ruleta o *Wheel Selection*, lo que introduce variabilidad en cada iteración.
- Cruce y Mutación Aleatorios: Los operadores de cruce utilizados, porcentual y por segmentos, intercambian partes de dos individuos de manera aleatoria. La mutación gaussiana introduce pequeños cambios aleatorios en los individuos, lo que genera diversidad en cada ejecución.
- Exploración y Explotación: Los algoritmos exploran diferentes regiones del espacio de búsqueda en cada ejecución.

La Figura 5.2 visualiza los resultados proporcionados por el algoritmo en 4 ejecuciones diferentes.

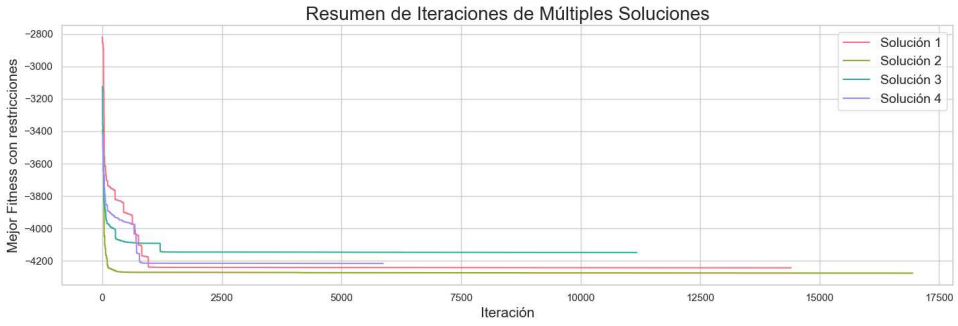

Figura 5.2: Resultados del Algoritmos WBGA en 4 ejecuciones

Como se observa en la Figura 5.2, el algoritmo proporciona soluciones con un valor del *fittnes* similar en las cuatro ejecuciones y con una convergencia similar. Esto demuestra que el algoritmo es estable y tiene baja variabilidad.

5.1.2.2. Robustez: cambio de variables de entrada y ponderación de objetivos

Para evaluar la robustez del algoritmo, realizaremos una serie de pruebas modificando los parámetros iniciales y las ponderaciones de las funciones objetivo. En primer lugar, se

llevará a cabo una modificación en la ponderación asignada a cada uno de los objetivos planteados en el modelo, lo que permitirá observar cómo el algoritmo ajusta sus soluciones frente a las nuevas prioridades. Posteriormente, se realizarán cambios en algunas de las variables iniciales para analizar cómo estas alteraciones afectan el comportamiento del algoritmo.

Estas pruebas tienen como propósito determinar si el algoritmo es capaz de adaptarse a las nuevas condiciones y si las soluciones generadas siguen siendo consistentes, óptimas y acordes con las restricciones establecidas. Este análisis permitirá validar su flexibilidad y robustez en contextos cambiantes, aumentando su efectividad en diferentes escenarios logísticos y operativos, algo que es esencial en aplicaciones reales donde los parámetros pueden variar frecuentemente.

- **Ponderaciones de las funciones objetivo:** Se modificaron las ponderaciones asignadas a las funciones objetivo, estableciendo una ponderación de $0{,}6$ para la ocupación de la campa, $0{,}3$ para la regularidad en el transporte y $0{,}1$ para los costes de almacenamiento. Bajo estas nuevas condiciones, el algoritmo genera, como es de esperar, una solución diferente a la inicial, pero muestra un proceso de convergencia similar, Además las restricciones establecidas siguen cumpliéndose. Esta nueva solución resultó válida, con una menor ocupación y una mejor regularidad, reflejando su capacidad de ajustarse a las prioridades redefinidas (ver figura 5.3)

Figura 5.3: Resumen de iteraciones con distinta ponderación

- **Cambio de los valores de entrada:** Las modificaciones en los valores iniciales pueden provocar que el algoritmo no cumpla con las restricciones si no están alineados con la realidad. Las restricciones están diseñadas y ajustadas en función de escenarios reales; por ejemplo, si se exige sacar 1000 coches al día, las restricciones serían imposibles de satisfacer, enfrentando al problema a un escenario irreal. Sin embargo, si los cambios en los parámetros tienen una lógica coherente y están vinculados a situaciones plausibles, el algoritmo genético se comportará según lo esperado y ofrece una solución factible y eficiente. En este caso hemos plasmado un escenario (ver Apéndice 2) donde tanto la ocupación de la campa como la producción diaria de vehículos presentan valores superiores a lo normal. Los resultados se muestran en la figura 5.4. En cuanto al cumplimiento de las restricciones, el modelo considera eficiente que se incumplan 3 de ellas:

    - Capacidad de los Transportistas en Barcelona y Valencia: La solución contempla enviar una mayor cantidad de coches que la cuota semanal correspondiente. Esto

no supone un problema, ya que los expertos suelen recurrir a tener semanas con más trafico que el establecido si es preciso, pudiéndolo compensar con menos envíos otra semana.

- Capacidad máxima campa destino Pasajes: El modelo considera óptimo sobrepasar la capacidad máxima de la zona M1 de la campa (destino Pasajes) para mejorar la gestión económica del almacenamiento en el puerto. Esta estrategia suele ser utilizada por los expertos y requiere utilizar campas adicionales.

Figura 5.4: Resumen de iteraciones con cambio de valores de entrada

Tras este análisis se puede afirmar que el GA ha demostrado un buen desempeño frente a variaciones en la configuración de parámetros que reflejan a cambios en las prioridades y/o las condiciones iniciales. Esto subraya su robustez al ser capaz de adaptarse a diferentes escenarios y mantener la eficacia en la búsqueda de soluciones. Adicionalmente, se demuestra que al repetir el algoritmo con diferentes condiciones iniciales, no siempre converge a una solución idéntica, pero sí a valores similares, lo que demuestra un comportamiento estable en términos generales.

5.1.2.3. Rendimiento del algoritmo genético

La evaluación del rendimiento del algoritmo implementado se realiza mediante los criterios siguientes:

- **Convergencia:** El algoritmo demuestra ser capaz de converger hacia soluciones óptimas dentro del espacio de búsqueda definido. Este comportamiento confirma que el GA es efectivo para resolver problemas de optimización con las características planteadas.
- **Velocidad de convergencia:** El número de iteraciones necesarias para alcanzar una solución aceptable, definida como aquella que no muestra mejoras significativas en 500 generaciones consecutivas, se sitúa en torno a unas 2500 generaciones. Este comportamiento refleja un equilibrio adecuado entre la exploración inicial del espacio de búsqueda y la posterior explotación de las soluciones con mayor aptitud.
- **Escalabilidad:** El algoritmo es capaz de manejar problemas más complejos manteniendo un buen rendimiento, ya que, a medida que aumenta el tamaño del problema, el tiempo de ejecución del algoritmo se incrementa de manera proporcional sin afectar de manera significativa a la calidad de las soluciones obtenidas. Además, el algoritmo

sería válido para incorporar más destinos si fuera necesario, siempre y cuando las nuevas restricciones estén alineadas con las condiciones reales y sean debidamente ajustadas.

- **Factibilidad:** el GA genera soluciones que cumplen las restricciones siempre que los valores de entrada se encuentren dentro de los rangos establecidos. Si los parámetros iniciales no están alineados con la realidad, es probable que alguna de las restricciones no se cumplan.
- **Recursos computacionales:**
  - *Tiempo de ejecución:* El tiempo medio requerido para resolver el problema oscila entre 20 y 30 minutos, dependiendo de la configuración y los valores iniciales.
  - *Equipo:* Se ha empleado un Portátil ASUS ROG Zephyrus G16 GU603VI.
  - *Uso de memoria:* El algoritmo utiliza aproximadamente un núcleo de procesador y 75.3 MB de memoria, lo cual es razonable para problemas de esta magnitud.
- **Visualización de resultados:** Tal y como se mostró en apartados anteriores, el comportamiento del algoritmo y los valores obtenidos se presentan mediante gráficos tales como las curvas de convergencia o los diagramas que ilustran la ocupación de los puertos y el cumplimiento de las restricciones. Estas herramientas visuales permiten analizar de manera efectiva el rendimiento del GA.

En conclusión, este algoritmo genético presenta un buen equilibrio entre eficiencia y adaptabilidad y permite abordar problemas logísticos de optimización con múltiples objetivos y restricciones como el planteado en este trabajo.

5.2. Algoritmo genético basado en frente de Pareto (NSGA-II)

Al igual que en el caso del algoritmo WBSA, el análisis de la solución generada por el algoritmo NSGA-II se realizará a través de casos de uso o escenarios en los que se reflejan situaciones concretas cercanas a la realidad de la empresa.

5.2.1. Obtención de los resultados: casos de uso

Al igual que en el caso anterior, el escenario propuesto incluye los valores iniciales de la producción diaria por destino, de la capacidad de los barcos y de los vehículos almacenados en los diferentes puertos basados en las condiciones reales observadas en febrero de 2024 que se detallan en el Apéndice 1.

El proceso de evaluación se divide en dos etapas. En la primera etapa, se obtienen los resultados proporcionados por el algoritmo genético y, posteriormente, en la segunda etapa, se contrastan los resultados obtenidos con la información real del período de referencia. Este contraste nos permite comprobar las mejoras en términos de eficiencia (dominancia) del algoritmo al examinar las posibles diferencias entre los resultados que proporciona y la realidad.

Los resultados obtenidos con la implementación del algoritmo genético después de 5000 iteraciones se muestran en la Figura 5.5. El conjunto de soluciones propuestas representa el

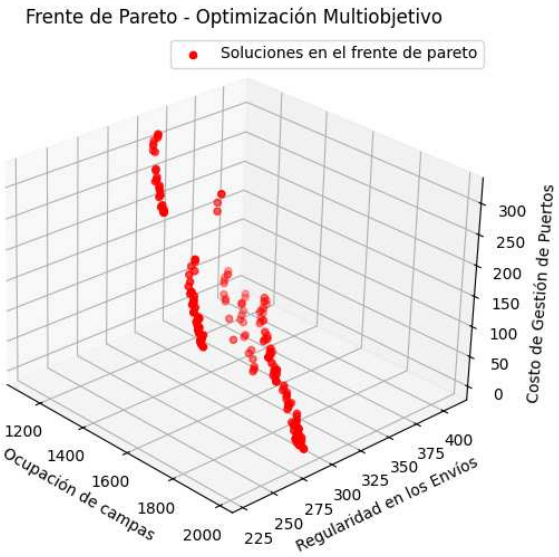

Figura 5.5: Resultado Algoritmo NSGA-II

frente de Pareto, aquellas soluciones no dominadas por ninguna otra, dadas las condiciones iniciales del problema. Se observa un frente extenso (200 soluciones) y uniforme lo que indica la diversidad y calidad de soluciones.

Teniendo en cuenta que hay tres objetivos, cuanto más cercana esté la solución al plano inferior, mejor responde a la minimización de los costes de los almacenes en los puertos. Conforme nos vamos alejando de dicho plano, los costes aumentan, pero se reduce la ocupación de la campa y/o aumenta la regularidad. De este conjunto de soluciones óptimas, es responsabilidad del experto elegir la que mejor se adecúe a las necesidades de la empresa.

En relación a las restricciones del modelo, señalar que se cumplen todas ellas, lo que pone de manifiesto que la penalización asociada a su incumplimiento actúa de manera efectiva, guiando al algoritmo hacia el conjunto de soluciones factibles.

5.2.1.1. Evaluación de los resultados y comparación con la solución real

A continuación, se contrastan los resultados obtenidos por el algoritmo con la solución real del período de referencia. Como se ha señalado anteriormente, los valores de las funciones objetivo para en dicho periodo son:

- Coste de almacenamiento en puertos: 376,74
- Ocupación de la campa: 524
- Regularidad: 334,57

Como se puede ver en la Figura 5.6 , las soluciones obtenidas por el algoritmo suponen una mejora en relación a la real o, lo que es lo mismo, la solución real está dominada por

las soluciones proporcionadas por el algoritmo y, por lo tanto, no se encuentra en el frente de Pareto.

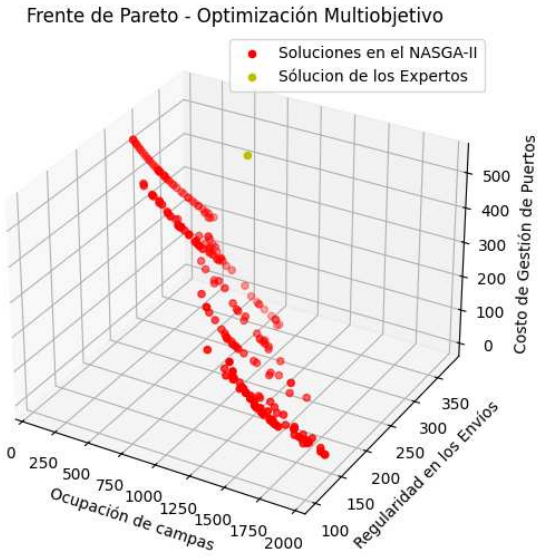

Figura 5.6: Resultado Algoritmo NSGA-II y resultado real

Nuevamente hay que señalar que las soluciones generadas por los expertos, aunque menos eficientes, pueden presentar ciertas ventajas derivadas de su capacidad para considerar factores externos que nuestro modelo no integra directamente. Sin embargo, habría que tener en cuenta que, en escenarios reales, en los que puede haber problemas o imprevistos, el hecho de que el algoritmo proporcione un conjunto amplio de soluciones, permite a los expertos elegir en cada momento aquella que mejor se ajuste al entorno.

5.2.2. Evaluación del Algoritmo

La evaluación del algoritmo NSGA-II se realiza en base a los mismos criterios que en el caso del algoritmo WBGA.

5.2.2.1. Estabilidad y robustez: cambio de variables de entrada

Para evaluar la estabilidad y robustez del algoritmo en contextos cambiantes, se modificarán los parámetros iniciales para analizar su impacto en el comportamiento del algoritmo. Como se ha señalado anteriormente, esta capacidad de adaptación es crucial en aplicaciones reales con parámetros dinámicos. El escenario elegido es el que se muestra en el Apéndice 2. Como se ha señalado en el caso del algoritmo WBGA, este escenario contempla una situación extrema en la que, tanto la ocupación de la campa como la producción diaria de vehículos presentan valores muy superiores a lo normal.

Los resultados se muestran en la Figura 5.7.

El análisis realizado confirma la estabilidad del algoritmo genético y su capacidad de ajustarse a cambios en las condiciones iniciales, produciendo soluciones eficientes.

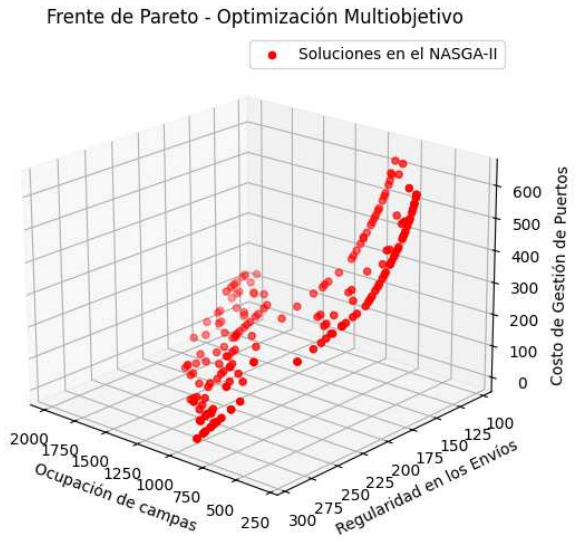

Figura 5.7: Resultado Algoritmo NSGA-II. Cambio en las condiciones iniciales

Esto permite afirmar que el GA ha demostrado un buen desempeño frente a variaciones en la configuración de parámetros y subraya su robustez al ser capaz de adaptarse a diferentes escenarios y mantener la eficacia en la búsqueda de soluciones. Adicionalmente, se demuestra que al repetir el algoritmo con diferentes condiciones iniciales, no siempre converge al mismo frente de Pareto, pero sí a valores similares, lo que demuestra un comportamiento estable en términos generales.

5.2.2.2. Rendimiento del algoritmo genético

La evaluación del rendimiento del algoritmo implementado se realiza mediante los criterios siguientes:

- **Escalabilidad:** El algoritmo es capaz de manejar problemas más complejos manteniendo un buen rendimiento, incorporando restricciones adicionales o más destinos si fuera necesario. En este sentido, un aumento del tamaño del problema, incrementa el tiempo de ejecución del algoritmo sin afectar de manera significativa a la calidad de las soluciones obtenidas.

- **Factibilidad:** el GA genera soluciones que cumplen las restricciones siempre, ya que las penalizaciones introducidas por el incumplimiento de las mismas, hacen que las soluciones que no las cumplen solo sobrevivan al cambio generacional después de un número limitado de iteraciones.

- **Recursos computacionales:**

  - *Tiempo de ejecución:* El tiempo medio requerido para resolver el problema depende del número de iteraciones que se establecen para la parada del proceso.

Si, por ejemplo se establecen 5000 generaciones el tiempo de ejecución es de unos 10 minutos. Si el número de generaciones aumenta, el tiempo aumenta proporcionalmente.

- *Equipo:* Se ha empleado un Portátil ASUS ROG Zephyrus G16 GU603VI.
- *Uso de memoria:* El algoritmo utiliza aproximadamente un núcleo de procesador y 54,6 MB de memoria, lo cual es razonable para problemas de esta magnitud.

- **Visualización de resultados:** Tal y como se mostró en apartados anteriores, el comportamiento del algoritmo y los valores obtenidos se presentan mediante gráficos que permiten visualizar el rendimiento del GA.

En conclusión, este algoritmo genético NSGA-II presenta un buen equilibrio entre eficiencia y adaptabilidad y permite abordar el problema logístico de optimización planteado.

5.3. Análisis comparativo de los algoritmos WBGA y NSGA-II

Como se ha demostrado en las secciones precedentes, tanto el algoritmo WBGA como el NSGA-II son herramientas adecuadas para resolver el problema planteado, pero tienen enfoques distintos. A continuación se realiza un análisis comparativo de los mismos, tanto en términos de la calidad de los resultados que proporcionan como en su comportamiento y desempeño.

En relación al rendimiento y desempeño de los dos algoritmos utilizados, podemos decir que, como se ha demostrado en las secciones precedentes, ambos son estables, robustos, escalables y con necesidades de recursos similares. En lo que al cumplimiento de las restricciones se refiere, el WBGA puede proporcionar, como se ha visto anteriormente, soluciones que no cumplan todas ellas cuando las condiciones iniciales son extremas, algo que no sucede con el NSGA-II, que demuestra ser más eficiente en estas condiciones.

En lo que a la calidad de los resultados obtenidos por ambos algoritmos se refiere, se puede realizar un análisis comparativo de los mismos partiendo de las mismas condiciones iniciales y deteniendo el proceso después del mismo número de iteraciones.

Una vez obtenidas las soluciones que presentan los dos algoritmos se comprueba si la solución de WBGA está en el frente de Pareto generado por NSGA-II:

- Si la solución del algoritmo WBGA está en el frente de Pareto del NSGA-II, significa que WBGA ha encontrado una solución no dominada por las soluciones propuestas por el NSGA-II, lo que indica que es una solución eficiente en términos de optimalidad.

- Si, por el contrario, la solución de WBGA no está en el frente de Pareto de NSGA-II, habrá que analizar si la solución del algoritmo WBGA está o no dominada por las soluciones del frente de Pareto:

    - Si la solución del algoritmo WBGA está dominada por las soluciones del frente de Pareto significa que NSGA-II ha encontrado soluciones dominantes, lo que sugiere que NSGA-II podría estar proporcionando soluciones de mejor calidad.

- Si la solución del algoritmo WBGA no está dominada por las soluciones del frente de Pareto es posible NSGA-II no haya explorado esa región, y no encontró una solución similar o porque las ponderaciones a favor de uno de los objetivos hacen que el WBGA haya encontrado una solución extrema en dicho objetivo, que NSGA-II ignoró en favor de soluciones más equilibradas.

La Figura 5.8 visualiza los resultados proporcionados por ambos algoritmos. Adicionalmente, en el Apéndice 3, se presentan dos soluciones concretas, una para cada tipo de algoritmo, que indican los vehículos transportados a cada destino durante una semana, el estado diario de la campa de almacenamiento, la ocupación de los almacenes en los puertos y el valor de las tres funciones objetivo para cada una de ellas.

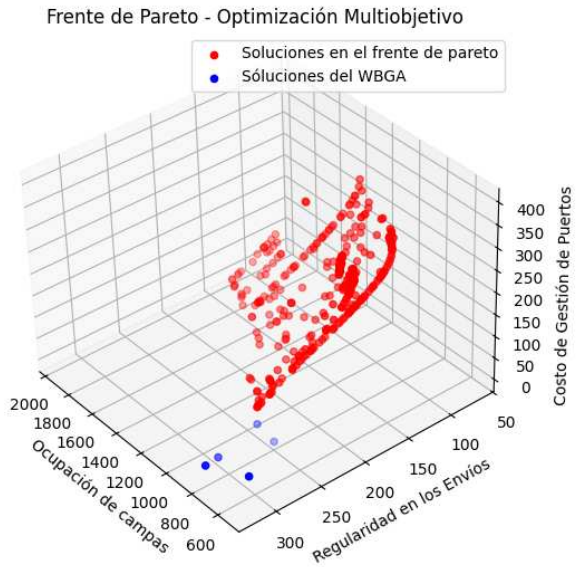

Figura 5.8: Resultados de los Altgorimos WBGA y NSGA-II

Como se puede ver en la Figura 5.8, las soluciones proporcionadas por el WBGA en 4 ejecuciones diferentes no están dominadas por las del NSGA-II, ya que, al emplear pesos, están explorando un espacio de búsqueda específico. Sin embargo, dentro del propio conjunto de soluciones del WBGA, sí se observa una relación de dominancia entre algunas de ellas, identificándose dos soluciones que han obtenido mejores resultados. Teniendo en cuenta esta comparativa, en la siguiente sección se realiza una propuesta de combinar ambos algoritmos.

5.4. Combinación de WBGA y NSGA-II

Teniendo en cuenta los resultados obtenidos en el análisis comparativo realizado en la sección anterior, una buena estrategia en la búsqueda de la solución al problema de optimización podría consistir en utilizar las soluciones generadas por el WBGA como población inicial para el NSGA-II. De este modo, se partiría de un conjunto de soluciones ya

dirigidas hacia una región específica del espacio de búsqueda, lo que permitiría al NSGA-II refinar y mejorar dichas soluciones dentro de ese área de interés. Esto facilitaría la obtención de alternativas viables que optimicen aún más los resultados proporcionados por el WBGA, explorando soluciones dominantes y diversificando las opciones disponibles.

Además, este enfoque híbrido aprovecha las fortalezas de ambos algoritmos: el WBGA, al asignar pesos a las funciones objetivo, enfoca la búsqueda en regiones específicas del espacio de soluciones, proporcionando un punto de partida dirigido; mientras que el NSGA-II, al ser un algoritmo basado en frentes de Pareto y crowding distance, puede mejorar la diversidad y calidad de las soluciones dentro de ese espacio.

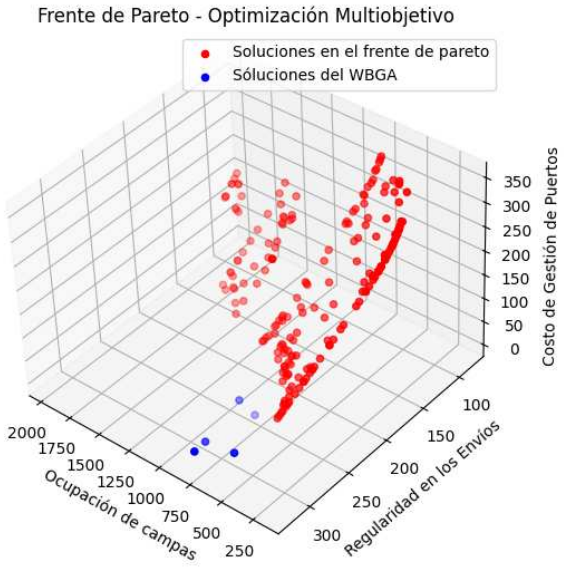

Figura 5.9: Resultados de los Algoritmos WBGA y NSGA combinados

De este modo, el NSGA-II no solo refinaría las soluciones del WBGA, sino que también exploraría posibles configuraciones que el WBGA podría haber pasado por alto debido a su enfoque basado en pesos. Esto permitiría obtener soluciones más equilibradas entre los diferentes objetivos, asegurando que no se favorezca excesivamente una función objetivo en detrimento de otra.

Asimismo, al reducir la aleatoriedad en la población inicial del NSGA-II y comenzar con soluciones ya preprocesadas, se podría mejorar la velocidad de convergencia. Además, este enfoque permitiría transformar una solución parcial de un escenario extremo en un conjunto de soluciones viables, ya que el NSGA-II sería capaz de refinar estas soluciones iniciales y generar alternativas que no solo sean viables, sino también eficientes en términos de las restricciones y objetivos establecidos.

CAPÍTULO 6

CONCLUSIONES Y TRABAJO FUTURO

6.1. Conclusiones principales

A lo largo de este TFM, se han desarrollado propuestas de soluciones para optimizar el proceso logístico vinculado al almacenamiento de vehículos en puertos y campas de la fábrica de Mercedes Benz en Vitoria-Gasteiz, basadas en algoritmos genéticos multiobjetivo. Para ello, se ha propuesto un modelo de optimización con tres objetivos, tales como la ocupación de la campa de la fábrica, los costes de almacenamiento en los puertos y la regularidad en la cantidad de vehículos transportados, objetivos que son interdependientes y que pueden entrar en conflicto. A través de un enfoque multifase, se han implementado dos algoritmos genéticos con enfoques diferentes, el WBGA y el NSGA-II, para iterar sobre una población de soluciones, buscando la mejor o mejores de ellas para cada posible escenario y teniendo en cuenta las prioridades que se establecen desde el Departamento de Logística de Envíos de la empresa.

Se trata de un problema complejo, multiobjetivo, no lineal, con variables discretas y restricciones que presentan alta interdependencia. Este tipo de problemas presenta comúnmente retos importantes debido a que los diferentes objetivos de optimización pueden en general ser contradictorios y difíciles de cuantificar. Por este motivo, son especialmente difíciles de abordar con métodos de optimización tradicionales, lo que justifica el uso de algoritmos genéticos. Estas técnicas están diseñadas para explorar grandes espacios de búsqueda, superar óptimos locales y encontrar soluciones cercanas al óptimo global.

Como se ha demostrado en este trabajo, los resultados obtenidos con la implementación de los algoritmos WBGA y NSGA-II, tanto en lo que la calidad de las soluciones se refiere como al desempeño de dichos algoritmos, son buenos. En este sentido, se puede concluir que los algoritmos genéticos son una herramienta adecuada para proporcionar buenas soluciones para el problema planteado que, en combinación con el juicio de expertos, pueden derivar en estrategias óptimas para la toma de decisiones.

El análisis realizado se ha completado con una comparativa de las soluciones propuestas por ambos algoritmos. En dicho análisis se pone de manifiesto que el algoritmo WBGA

puede proporcionar soluciones que, no estando en el frente de Pareto del NSGA-II, no están dominadas por las soluciones de dicho frente. Esto puede explicarse por el hecho de los pesos asignados a la función de *fittnes* llevan al algoritmo WBGA hacia soluciones extremas con un valor mejor respecto al objetivo con mayor ponderación, que NSGA-II ignoró en favor de soluciones más equilibradas. Una propuesta en la que se combine ambos algoritmos utilizando las soluciones generadas por el WBGA como población inicial para el NSGA-II facilitaría la obtención de alternativas viables que optimicen aún más los resultados proporcionados por el WBGA, explorando soluciones dominantes y diversificando las opciones disponibles.

La implementación y desarrollo de estos problemas permite ajustar los conocimientos de la materia y aplicar conceptos teóricos a escenarios reales. A través de la aplicación de algoritmos genéticos, se puede lograr una mejora significativa en la optimización de los objetivos planteados.

6.2. Limitaciones del análisis

Es importante señalar que el modelo que se ha desarrollado a lo largo de este trabajo es una representación simplificada de la realidad, creado para abordar el problema de optimización de una manera sencilla y fácil de entender, pero que no tiene en cuenta todos los elementos y variables complejas que podrían influir en el problema en un entorno real. En este sentido, en su forma actual, debe considerarse como una primera tentativa de generar soluciones aproximadas, ya que esta simplificación genera limitaciones en lo que a la validez de los resultados se refiere, al no tener en cuenta todos los factores que pueden afectar el problema y que podrían tener un efecto significativo en los resultados. No obstante, puede considerarse un punto de partida ya que, a partir del mismo, con el equipo correcto, el conocimiento necesario y el tiempo adecuado, sería factible crear una herramienta más fuerte y precisa.

En este sentido, tal y como se ha señalado anteriormente, en próximas actuaciones del proyecto, sería clave incluir elementos adicionales, como pueden ser la variabilidad de los costes, las limitaciones logísticas en tiempo real, las condiciones climáticas o la sostenibilidad medioambiental, entre otros, que pueden influir en el funcionamiento del sistema.

En la actualidad, en la toma de decisiones en la empresa, no existe ninguna estrategia basada en un análisis matemático riguroso o un enfoque de optimización formal. Las decisiones, en la mayoría de los casos, se basan en la experiencia y el juicio de los expertos, sin el apoyo de herramientas de análisis cuantitativo. En este contexto, el modelo propuesto podría ser útil, ya que podría complementar el proceso de toma de decisiones mediante una evaluación más sistemática y basada en datos, ofreciendo soluciones que podrían ser más efectivas y fundamentadas.

Si bien el análisis desarrollado en este trabajo no suple la necesidad de un análisis más profundo y detallado, puede considerarse como un punto de partida para mejorar la toma de decisiones, proporcionando una visión más clara de las posibles opciones y su impacto. A medida que el modelo se mejore y se incluyan adecuadamente más elementos y restricciones realistas, su capacidad para abordar la optimización de los procesos de una manera más eficiente será mayor.

6.3. Trabajo futuro

Aunque los algoritmos genéticos desarrollados han demostrado ser un buen instrumento para resolver el problema planteado, existen diversas áreas en las que se podrían realizar mejoras para optimizar su desempeño y capacidad de uso. A continuación, se presentan algunas propuestas:

- **Mejoras en el operador de cruce:** Implementar operadores de cruce más avanzados que, además de intercambiar segmentos de cromosomas entre padres, puedan modificar directamente los valores de los genes en función de los datos, permitiendo explorar de manera más eficiente el espacio de búsqueda.
- **Adaptación dinámica de la tasa de mutación:**
  - Incrementar el porcentaje de mutación a medida que el algoritmo no logre mejoras significativas durante un número determinado de iteraciones. Esto permitiría introducir más diversidad en la población y evitar estancamientos en óptimos locales.
  - Reducir la desviación típica de la distribución normal si no se observan mejoras, ajustándola gradualmente para que las alteraciones sean más precisas y dirigidas hacia zonas óptimas del espacio de búsqueda.
- **Interfaz de usuario (UI):** Diseñar una interfaz intuitiva y accesible que permita a los expertos introducir los datos necesarios, ajustar parámetros y observar los resultados del algoritmo sin necesidad de soporte técnico especializado. Esta interfaz podría incluir opciones como:
  - Carga y modificación de parámetros iniciales y restricciones.
  - Visualización gráfica de los resultados, como la ocupación de los puertos y la regularidad de los envíos.
  - Exportación de los datos procesados a un formato amigable para un análisis más detallado.
- **Optimización computacional:** Reducir el tiempo de ejecución y el uso de memoria mediante técnicas como la paralelización de procesos o el uso de bibliotecas especializadas para computación de alto rendimiento.

Estas mejoras, de ser implementadas, podrían aumentar considerablemente la efectividad y aplicabilidad del modelo, adaptándolo a una gama más amplia de escenarios logísticos y haciéndolo más útil y accesible para los expertos en la materia.

Bibliografía

[1] Ronald H. Ballou. *Logística. Administración de la cadena de suministros.* México: Prentice Hall, Pearson Education, 2004. Ver página 1.

[2] Richard J. Trudeau. *Introduction to graph theory.* Dover Pub., New York, 1993. Ver página 8.

[3] J.J. Ruiz Ortiz. Optimización de redes logísticas. *Introducción a la Programación Matemática*, Universidad Complutense de Madrid, 2023. Ver páginas 8, 18.

[4] A. Osyczka. *Multicriteria optimization for engineering design.* In J. S. Gero, editor, Design Optimization, Academic Press, 1985. Ver página 8.

[5] Singh P. Tomar V, Bansal M. Metaheuristic algorithms for optimization: A brief review. *Engineering Proceedings*, 59(1), 2023. Ver páginas 10, 14.

[6] Carlos A.Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for solving multi-objetive problems. 2ª edición.* Springer, 2002. Ver página 10.

[7] Kalyan Deb. *Multiobjective Optimization Using Evolutionary Algorithms. Wiley, New York.* 01 2001. Ver páginas 10, 13.

[8] D.A. Coley. *An Introduction to Genetic Algorithms for Scientists and Engineers.* World Scientific Publishing Co., 1999. Ver página 10.

[9] D.B. Fogel Z. Michalewicz. *How to Solve it: Modern Heuristics.* Springer, 2002. Ver página 10.

[10] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor., 1975. Ver página 10.

[11] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley Longman Publishing Co., Inc., 1989. Ver páginas 10, 14.

[12] T.V.N. Narques et all. Use of real coded genetic algorithm as a pre-dimensioning tool for prestressed concrete beams. *Buildings*, 13, 2023. Ver página 11.

[13] Alhijawi B. & Awajan A. Genetic algorithms: theory, genetic operators, solutions, and applications. *Evolutionary Intelligence*, 2024. Ver página 11.

[14] Khalid Jebari. Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, 3:333–344, 12 2013. Ver página 12.

[15] Nitasha Soni and Dr. Tapas Kumar. Study of various crossover operators in genetic algorithms. In *Khosravy, M., Gupta, N., Patel, N. (eds) Frontiers in Nature-Inspired Industrial Optimization.*, 2014. Ver página 12.

[16] Nitasha Soni and Tapas Kumar. Study of various mutation operators in genetic algorithms. In *Khosravy, M., Gupta, N., Patel, N. (eds) Frontiers in Nature-Inspired Industrial Optimization.*, 2014. Ver página 12.

[17] Alice E. Smith Abdullah Konak, David W. Coit. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91, 2006. Ver páginas 13, 14.

[18] Kumanan Somasundaram and Prasannavenkatesan Shanmugam. A genetic algorithm for optimization of supply chain logistics network. *International Journal of Applied Management and Technology*, 3, 2005. Ver página 14.

[19] The Logistic World. Optimización de la cadena de suministro con algoritmos genéticos: Casos de estudio en méxico. *https://thelogisticsworld.com/logistica-y-distribucion/optimizacion-de-la-cadena-de-suministro-con-algoritmos-geneticos-casos-de-estudio-en-Mexico/*, 2024. Accessed: 2024-11-27. Ver página 14.

[20] Schaffer JD. Multiple objective optimization with vector evaluated genetic algorithms. *In: Proceedings of the international conference on genetic algorithm and their applications*, 1985. Ver página 14.

[21] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2), 2002. Ver página 14.

[22] Félix-Antoine Fortin, François-Michel De Rainville, M.A. Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research, Machine Learning Open Source Software*, 13, 2012. Ver página 43.

[23] Ankita Golchha and Shahana Gajala Qureshi. Non-dominated sorting genetic algorithm-ii – a succinct survey. *International Journal of Computer Science and Information Technologies*, 6(1), 2015. Ver página 43.

[24] César Peñuela and Mauricio Granada Echeverri. Optimización multiobjetivo usando un algoritmo genético y un operador elitista basado en un ordenamiento no-dominado (nsga-ii). *Scientia Et Technica*, 2007. Ver página 43.

[25] M. Shahriari. Multi-objective optimization of discrete time–cost tradeoff problem in project networks using non-dominated sorting genetic algorithm. *Journal of Industrial Engineering International*, 12(2), 2016. Ver página 45.

Apéndices

Apéndice 1: Ejemplo real de planificación semanal

- Producción diaria de vehículos por destino: Los datos muestran la cantidad de vehículos fabricados diariamente y que se tienen que enviar a cada destino (tabla 1).

Día	CP	CB	CV	CS	CF	CE	CSW
1	520	100	15	15	13	20	64
2	391	17	24	16	60	50	86
3	381	70	8	12	35	25	28
4	362	31	12	8	86	16	15
5	448	48	16	34	45	45	33
6	428	99	10	10	23	23	23
7	504	39	19	26	47	47	31

Tabla 1: Producción diaria de vehículos por destino

- Capacidad diaria de transporte marítimo: La capacidad de los barcos disponibles por día para transportar vehículos hacia diferentes destinos se refleja en la tabla 2.

Día	BarcosP	BarcosB	BarcosV	BarcosS
1	0	0	0	0
2	0	100	0	70
3	0	0	0	0
4	1560	0	100	0
5	0	0	0	50
6	0	250	0	0
7	850	0	50	0
8	0	0	0	0
9	750	0	0	0
10	0	200	220	0
11	0	0	0	150
12	930	500	0	220
13	1560	0	100	0

Tabla 2: Capacidad de los barcos disponibles para trasporte marítimo

Se establecen los datos para 13 días, en lugar de 7, para poder evaluar también los 6 días posteriores para la minimización de los costes de almacenamiento.

- Almacenamiento en los puertos (buffer): los datos relacionados con la cantidad de vehículos almacenados en los diferentes puertos se detallan en la tabla 3.

Posición	**bufferPas**	**bufferBar**	**bufferVal**	**bufferSan**
1	195	153	8	15
2	21	11	29	11
3	19	4	24	4
4	0	100	0	0
5	-	91	57	5
6	-	88	6	8
7	-	22	31	0
8	-	112	4	12
9	-	0	0	0
10	-	0	63	0
11	-	79	0	17
12	-	28	-	8
13	-	24	-	24
14	-	99	-	30
15	-	40	-	9
16	-	78	-	16
17	-	0	-	32
18	-	0	-	45
19	-	0	-	0
20	-	0	-	0
21	-	0	-	0
22	-	0	-	0
Días	**4**	**22**	**11**	**22**

Tabla 3: Cantidad de coches almacenados en puertos (buffer)

Apéndice 2: Cambio de los valores de entrada

Producción diaria de vehículos por destino con una cantidad de vehículos superior a la normal: Los datos muestran la cantidad de vehículos para realizar la prueba de cambio de los valores de entrada (ver tabla).

Día	CP	CB	CV	CS	CF	CE	CSW
1	520	150	55	45	33	50	94
2	451	17	24	16	60	50	86
3	424	70	8	12	35	25	28
4	412	31	12	8	86	16	20
5	478	48	16	34	45	45	33
6	528	99	10	10	23	23	23
7	504	39	19	26	47	47	31

Apéndice 3

Día	Pasajes	Barcelona	Valencia	Santander	España	Francia	Suiza
Día 1	0	43	0	13	0	0	0
Día 2	445	42	7	15	23	13	49
Día 3	316	65	25	12	21	45	44
Día 4	351	63	25	11	28	36	34
Día 5	361	53	8	16	27	85	39
Día 6	474	45	17	28	61	38	44
Día 7	521	93	22	26	66	45	39

Tabla 4: Envíos a destinos: solución WBGA

Posición	bufferPas	bufferBar	bufferVal	bufferSan
1	521	93	0	26
2	0	0	0	28
3	0	0	0	5
4	0	0	0	0
5	-	0	0	0
6	-	0	0	0
7	-	0	0	0
8	-	0	0	0
9	-	0	0	0
10	-	0	0	0
11	-	0	0	0
12	-	0	-	0
13	-	0	-	0
14	-	0	-	0
15	-	0	-	0
16	-	0	-	0
17	-	0	-	0
18	-	0	-	0
19	-	0	-	0
20	-	0	-	0
21	-	0	-	0
22	-	0	-	0
Días	**4**	**22**	**11**	**22**

Tabla 5: Almacenamiento en puertos: solución WBGA

Día	1	2	3	4	5	6	7
Stock M1	520	466	531	542	659	713	696
Stock M2	171	275	241	212	205	160	78

Tabla 6: Stock de M1 y M2: solución WBGA

Día	Pasajes	Barcelona	Valencia	Santander	España	Francia	Suiza
1	2	64	14	11	19	0	0
2	427	53	14	14	33	13	44
3	281	53	14	13	29	31	43
4	347	36	15	13	30	36	36
5	309	60	14	19	30	56	43
6	368	68	14	25	35	69	41
7	251	70	19	26	50	57	42

Tabla 7: Envíos a destinos: solución NSGA-II

Posición	bufferPas	bufferBar	bufferVal	bufferSan
1	251	70	0	26
2	0	0	0	25
3	0	0	0	0
4	0	0	0	0
5	-	0	0	0
6	-	0	0	0
7	-	0	0	0
8	-	0	0	0
9	-	0	0	0
10	-	0	0	0
11	-	0	0	0
12	-	0	-	0
13	-	0	-	0
14	-	0	-	0
15	-	0	-	0
16	-	0	-	0
17	-	0	-	0
18	-	0	-	0
19	-	0	-	0
20	-	0	-	0
21	-	0	-	0
22	-	0	-	0
Días	**4**	**22**	**11**	**22**

Tabla 8: Almacenamiento en puertos: solución NSGA-II

Día	1	2	3	4	5	6	7
Stock M1	518	482	582	597	766	926	1049
Stock M2	119	201	196	198	197	133	78

Tabla 9: Stock de M1 y M2: solución NSGA-II

Algoritmo	Ocupación	Regularidad	Gestión de puertos
WBGA	7748	253.31	0
NASGA-II	1127	191.64	1.54

Tabla 10: Valores de las funciones objetivo: soluciones WBGA y NSGA-II

Apéndice 4

CÓDIGO ALGORITMO WBGA

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
import time

dim=6                           #número de dimensiones del problema
itera=200000                #número máximo de iteraciones
rep=200              #número máximo de iteraciones sin cambios
tamanoPoblacion=110       #tamaño de la población
pc=0.65                          #porcentaje de cruzamiento
ub=[600,174,72,36,97,110,93]      #límite superior
lb=0                            #límite inferior
nc=10                          #número de hijos descendientes (Ha de ser par!)
nm=round(0.1*tamanoPoblacion)  #número de descendientes mutados
mu=0.1                          #tasa de mutación
beta=6                     #parámetro de selección
restr_prio=130                  #parametro de premio por cumplir las restriciones
destinos=["Pasajes","Barcelona","Valencia","Santander","España","Francia","Suiza"]
destinos_regex = '|'.join(destinos)
bufferPas = [195, 21, 19, 0]
bufferBar = [153, 11, 4, 100, 91,88,22,112,0,0,79,28, 24,99, 40,78,0,0,0,0,0,0]
bufferVal = [8, 29, 24, 0, 57,6,31,4,0,63,0]
bufferSan = [15, 11, 4, 0, 5,8,0,12,0,0,17,8, 24, 30, 9,16,32,45,0,0,0,0]
barcosPas = [0, 0, 0, 1560, 0, 850, 0, 0, 750, 0, 0, 930, 1560]
barcosBar = [0, 100, 0, 0, 0, 250, 0, 0, 0, 200, 0, 500, 0]
barcosVal = [0, 0, 0, 100, 0, 0, 50, 0, 0, 220, 0, 0, 100]
barcosSan  = [0, 70, 0, 0, 50, 0, 0, 0, 0, 0, 150, 220, 0]
CP =  [520, 391, 381, 362, 478, 528, 504]
CB =  [100, 17,  70,  31,  48,  99,  39 ]
CV =  [15,  24,  8,   12,  16,  10,  19 ]
CS =  [15,  16,  12,  8,   34,  10,  26 ]
CF =  [13,  60,  35,  86,  45,  23,  47 ]
CE =  [20,  50,  25,  16,  45,  23,  47 ]
CSW = [64,  86,  28,  15,  33,  23,  31 ]



########FUNCIONES DE OPTIMIZACION
# Función de ocupación del campamento
def OcupacionCampa(p, CP):
    return sum(CP) - sum(p)
```

```
# Función de ocupación completa
def Ocupational_function(p, b, v, s, e, f, sw, CP, CB, CV, CS, CE, CF, CSW):
    return abs(OcupacionCampa(p, CP)) + abs(OcupacionCampa(b, CB)) +
    abs(OcupacionCampa(v, CV)) + abs(OcupacionCampa(s, CS)) +
    abs(OcupacionCampa(e, CE)) + abs(OcupacionCampa(f, CF)) +
    abs(OcupacionCampa(sw, CSW))

def printOcupational_function(p, b, v, s, e, f, sw, CP, CB, CV, CS, CE, CF, CSW):
    print(f"  Ocupación pasajes: {OcupacionCampa(p, CP)}")
    print(f"  Ocupación barcelona: {OcupacionCampa(b, CB)}")
    print(f"  Ocupación valencia: {OcupacionCampa(v, CV)}")
    print(f"  Ocupación santander: {OcupacionCampa(s, CS)}")
    print(f"  Ocupación Nacional: {OcupacionCampa(e, CE)}")
    print(f"  Ocupación Francia: {OcupacionCampa(f, CF)}")
    print(f"  Ocupación swiza: {OcupacionCampa(sw, CSW)}")

# Función de regularidad
def Regular_function(values):
    total_variance = 0
    for val_list in values:
        mean = sum(val_list) / len(val_list)
        variance = math.sqrt(sum((val - mean) ** 2 for val in val_list) / len(val_li
        total_variance += variance
    return total_variance
# Función de gestión de puertos
def gestion_puertos(buffer, cochesEntrantes, barcos, costo):
    costoTotal = 0
    for i in range(len(barcos)):
        # Guardar el último elemento de la lista
        ultimo_elemento = buffer[-1]
        # Eliminar el último elemento
        buffer.pop()
        # Sumar el número al último elemento de la lista
        buffer[-1] += ultimo_elemento
        # Insertar el número de coches entrantes en la primera posición de la lista
        if i < len(cochesEntrantes):
            buffer.insert(0, cochesEntrantes[i])
        else:
            buffer.insert(0, 0)
        # Sumar los costos
        costoTotal += buffer[-1] * costo
        if barcos[i] > 0:
            buffer = restar_en_cascada(buffer, barcos[i])
    return costoTotal
def printgestion_puertos(buffer, cochesEntrantes, barcos, costo):
    costoTotal = 0
    for i in range(dim+1):
```

```
        # Guardar el último elemento de la lista
        ultimo_elemento = buffer[-1]
        # Eliminar el último elemento
        buffer.pop()
        # Sumar el número al último elemento de la lista
        buffer[-1] += ultimo_elemento
        # Insertar el número de coches entrantes en la primera posición de la lista
        if i < len(cochesEntrantes):
            buffer.insert(0, cochesEntrantes[i])
        else:
            buffer.insert(0, 0)
        # Sumar los costos
        costoTotal += buffer[-1] * costo
        if barcos[i] > 0:
            buffer = restar_en_cascada(buffer, barcos[i])
    return buffer
# Resta en cascada de la lista
def restar_en_cascada(lista, numero):
    for i in range(len(lista) - 1, -1, -1):  # Iterar sobre los elementos de la lista en orde
        if numero >= lista[i]:  # Si el número es mayor o igual que el elemento actual de la
            numero -= lista[i]  # Restar el elemento actual de la lista del número #
            lista[i] = 0  # Establecer el elemento actual de la lista a cero
        else:  # Si el número es menor que el elemento actual de la lista
            lista[i] -= numero  # Restar el número del elemento actual de la lista
            numero = 0  # Establecer el número a cero
            break  # Salir del bucle
    return lista

# Función de fitness
def fitness(pob, dim):
    fitness_values = []
    for index, row in pob.iterrows():
        # Extraer los valores de cada país (7 columnas consecutivas para cada país)
        p = [row[f'Pasajes{i}'] for i in range(dim+1)]
        b = [row[f'Barcelona{i}'] for i in range(dim+1)]
        v = [row[f'Valencia{i}'] for i in range(dim+1)]
        s = [row[f'Santander{i}'] for i in range(dim+1)]
        e = [row[f'España{i}'] for i in range(dim+1)]
        f = [row[f'Francia{i}'] for i in range(dim+1)]
        sw = [row[f'Suiza{i}'] for i in range(dim+1)]

        # Calcular la función de ocupación
        ocupation_value = Ocupational_function(p,b,v,s,e,f,sw,CP,CB,CV,CS,CE,CF,CSW)
```

```
        # Calcular la función de regularidad para cada país por separado
        regularity_value = Regular_function([p]) + Regular_function([b]) +
                           Regular_function([v]) + Regular_function([s]) +
                           Regular_function([e]) + Regular_function([f]) +
                           Regular_function([sw])

        gestion_puer = gestion_puertos(bufferPas, p, barcosPas, 0.77) +
                       gestion_puertos(bufferBar, b, barcosBar, 1.5) +
                       gestion_puertos(bufferVal, v, barcosVal, 1.38) +
                       gestion_puertos(bufferSan, s, barcosSan, 1.5)
        # Combinar los valores para obtener el fitness total
        # Puedes ajustar los pesos
        fitness_value = 0.3*occupation_value + 0.1*regularity_value+0.6*gestion_puer
        fitness_values.append(fitness_value)

    return fitness_values

########
def fitness_display(pob, dim):
    print("\nIndividuo:")

    # Extraer valores
    p = [pob[f'Mejor_Posicion_De_Pasajes{i}'] for i in range(dim+1)]
    b = [pob[f'Mejor_Posicion_De_Barcelona{i}'] for i in range(dim+1)]
    v = [pob[f'Mejor_Posicion_De_Valencia{i}'] for i in range(dim+1)]
    s = [pob[f'Mejor_Posicion_De_Santander{i}'] for i in range(dim+1)]
    e = [pob[f'Mejor_Posicion_De_España{i}'] for i in range(dim+1)]
    f = [pob[f'Mejor_Posicion_De_Francia{i}'] for i in range(dim+1)]
    sw = [pob[f'Mejor_Posicion_De_Suiza{i}'] for i in range(dim+1)]

    # Cálculos de funciones
    printOcupational_function(p, b, v, s, e, f, sw, CP, CB, CV, CS, CE, CF, CSW)
    occupation_value = Ocupational_function(p,b,v,s,e,f,sw,CP,CB,CV,CS,CE,CF,CSW)
    regularity_value = sum(Regular_function([x]) for x in [p, b, v, s, e, f, sw])
    gestion_puer = gestion_puertos(bufferPas, p, barcosPas, 0.77) +
                   gestion_puertos(bufferBar, b, barcosBar, 1.5) +
                   gestion_puertos(bufferVal, v, barcosVal, 1.38) +
                   gestion_puertos(bufferSan, s, barcosSan, 1.5)
    print(f"  Puerto pas: {printgestion_puertos(bufferPas, p, barcosPas, 0.77)}")
    print(f"  Puerto bar: {printgestion_puertos(bufferBar, b, barcosBar, 1.5)}")
    print(f"  Puerto val: {printgestion_puertos(bufferVal, v, barcosVal, 1.38)}")
    print(f"  Puerto san: {printgestion_puertos(bufferSan, s, barcosSan, 1.5)}")
    # Fitness total
    fitness_value = 0.6 * occupation_value + 0.3 * regularity_value + 0.1 * gestion_

    # Salida
    print(f"  Valor de ocupación: {occupation_value:.2f}")
```

```
    print(f"  Valor de regularidad: {regularity_value:.2f}")
    print(f"  Valor de gestión de puertos: {gestion_puer:.2f}")
    print(f"  Fitness total: {fitness_value:.2f}")


## RESTRICIONES
# Función para calcular el stock de coches entrantes y coches en campa
def Stock(cochesEntrantes, cochesCampa, day):
    res = 0
    for i in range(0, day + 1):
        res = cochesCampa[i] - cochesEntrantes[i] + res
    return res

# Función para calcular el stock para Francia y Suiza
def StockFySW(cochesEntrantes, cochesCampa, day):
    res = 0
    for i in range(day + 1):
        if i - 1 < 0:
            res = cochesEntrantes[i] + res
        else:
            res = cochesEntrantes[i] - cochesCampa[i - 1] + res
    return res

# Función para calcular el out para los tres últimos días
def OutThree(cochesEntrantes, cochesCampa, day):
    res = 0
    for i in range(day + 1):
        if i - 3 < 0:
            res = cochesEntrantes[i] + res
        else:
            res = cochesEntrantes[i] - cochesCampa[i - 3] + res
    return res

# Restricciones
def rerstricciones(pob, dim):
    restr_list = []
    for index, row in pob.iterrows():
        p = [row[f'Pasajes{i}'] for i in range(dim+1)]
        b = [row[f'Barcelona{i}'] for i in range(dim+1)]
        v = [row[f'Valencia{i}'] for i in range(dim+1)]
        s = [row[f'Santander{i}'] for i in range(dim+1)]
        e = [row[f'España{i}'] for i in range(dim+1)]
        f = [row[f'Francia{i}'] for i in range(dim+1)]
        sw = [row[f'Suiza{i}'] for i in range(dim+1)]
        restr_impor = 0
        # Restricciones de sumas
        restr_impor += int(sum(p) <= 3584)
```

```
        restr_impor += int(sum(b) <= 457)
        restr_impor += int(sum(sw) <= 280)
        restr_impor += int(sum(s) <= 126)
        restr_impor += int(sum(v) <= 121)

        # Restricciones para cada día
        for i in range(7):
            restr_impor += int(b[i] + s[i] + v[i] + e[i] + f[i] + sw[i] <= 400)
            restr_impor += int(Stock(p, CP, i) >= 0)
            restr_impor += int(Stock(b, CB, i) >= 0)
            restr_impor += int(Stock(s, CS, i) >= 0)
            restr_impor += int(Stock(v, CV, i) >= 0)
            restr_impor += int(Stock(e, CE, i) >= 0)
            restr_impor += int(Stock(p, CP, i) <= 1350)
            restr_impor += int(Stock(b, CB, i)+Stock(s, CS, i)+Stock(v, CV, i)+
                        Stock(e, CE, i)+ StockFySW(f, CF, i)+StockFySW(sw, CSW, i) <=
            restr_impor += int(StockFySW(f, CF, i) <= 0)
            restr_impor += int(StockFySW(sw, CSW, i) <= 0)
        restr_list.append(restr_impor)

    return restr_list

##
def restricciones_display(pob, dim):

    # Extraer los valores de cada país (7 columnas consecutivas para cada país)
    p = [pob[f'Mejor_Posicion_De_Pasajes{i}'] for i in range(dim + 1)]
    b = [pob[f'Mejor_Posicion_De_Barcelona{i}'] for i in range(dim + 1)]
    v = [pob[f'Mejor_Posicion_De_Valencia{i}'] for i in range(dim + 1)]
    s = [pob[f'Mejor_Posicion_De_Santander{i}'] for i in range(dim + 1)]
    e = [pob[f'Mejor_Posicion_De_España{i}'] for i in range(dim + 1)]
    f = [pob[f'Mejor_Posicion_De_Francia{i}'] for i in range(dim + 1)]
    sw = [pob[f'Mejor_Posicion_De_Suiza{i}'] for i in range(dim + 1)]

    # Restricciones de sumas
    print(" Restricciones de suma:")
    print(f" Suma de pasajes (<= 3584): {'Cumple' if sum(p) <= 3584 else 'No cumple'
    print(f" Suma de Barcelona (<= 427): {'Cumple' if sum(b) <= 427 else 'No cumple'
    print(f" Suma de Suiza (<= 280): {'Cumple' if sum(sw) <= 280 else 'No cumple'}")
    print(f" Suma de Santander (<= 126): {'Cumple' if sum(s) <= 126 else 'No cumple'
    print(f" Suma de Valencia (<= 119): {'Cumple' if sum(v) <= 119 else 'No cumple'}

    # Restricciones para cada día
    print("  Restricciones diarias:")
    for i in range(7):
    print(f" Día {i + 1}:")
    print(f" Suma diaria (<= 400): {'Cumple' if b[i] +      s[i] + v[i] + e[i] + f[i
```

```
        + sw[i] <= 400 else 'No cumple'}")
    print(f" Stock de Pasajes (>= 0): {'Cumple' if Stock(p, CP, i) >= 0
            else 'No cumple'}")
    print(f" Stock de Barcelona (>= 0): {'Cumple' if Stock(b, CB, i) >= 0
            else 'No cumple'}")
    print(f" Stock de Santander (>= 0): {'Cumple' if Stock(s, CS, i) >= 0
            else 'No cumple'}")
    print(f" Stock de Valencia (>= 0): {'Cumple' if     Stock(v, CV, i) >= 0
            else 'No cumple'}")
    print(f" Stock de España (>= 0): {'Cumple' if Stock(e, CE, i) >= 0
            else 'No cumple'}")
    print(f" Stock de Francia (<= 0): {'Cumple' if StockFySW(f, CF, i) <= 0
            else 'No cumple'}")
    print(f" Stock de Suiza (<= 0): {'Cumple' if StockFySW(sw, CSW, i) <= 0
            else 'No cumple'}")
    print(f" Capacidad maxima campa pas (<= 1350): {'Cumple' if Stock(p, CP, i) <= 1350
            else 'No cumple'}")
    print(f" Capacidad maxima campa resto (<= 643): {'Cumple' if (Stock(b, CB, i)+
          Stock(s, CS, i)+Stock(v, CV, i)+Stock(e, CE, i)+ StockFySW(f, CF, i)+
          StockFySW(sw, CSW, i)) <= 643 else 'No cumple'}")

##CREACION DE POBLACION
##
#CREACIÓN POBLACIÓN
def creacion_poblacion(tamanoPoblacion, dim,paises, lb, ub, destinos):
    pob=pd.DataFrame(index=range(tamanoPoblacion))
    pob['Individuo']=range(0, tamanoPoblacion)
    #Añado una columna con el número de cada individuo


for l in range(paises):
 for i in range(dim+1):
    if ((l == 5 or l == 6) and i == 0):
pob[destinos[l]+str(i)]=0
    else:
pob[destinos[l]+str(i)]= np.random.randint(lb,ub[l],[len(pob),1]).flatten().tolist()
#Relleno las columnas de posicion (hay tantas como dim)

 Columnas_pop= pd.DataFrame(pob.columns.to_list()).rename(columns={0:'Nombres_Columnas'})
 #Creo un df con los nombres de las columnas que hay en pob
 Columnas_pop_posicion= Columnas_pop[Columnas_pop['Nombres_Columnas'].str.contains(destinos_r
 ['Nombres_Columnas'].unique().tolist()
 #Creo una lista con los nombres de las columnas Posicion que hay
 #En este caso ['Posicion_0', 'Posicion_1']


    #Evaluación en la función objetivo
```

```
    pob['Fitness'] = fitness(pob, dim)
    #En la función fitness también se premia el hecho de cumplir las restricciones
    pob['Restricciones'] = rerstricciones(pob,  dim)
    pob['Fitness+Rest'] = pob['Fitness'] - restr_prio * pob['Restricciones']

    print(pob)
    print(Columnas_pop_posicion)

    return pob
##

##ELECCION
#SELECCIÓN el algoritmo ya habia sido implementado
def seleccion(P):
    # Generamos dos números aleatorios, r1 y r2
    # Para probar los casos extremos, fijar r1 en 1 y r2 en 0 respectivamente
    r1 = random.random()  # Elige un número al azar entre 0 y 1
    r2 = random.random()  # Elige otro número al azar entre 0 y 1

    # Creamos una copia de los datos de P  P_it
    P_it = P.copy()

    # Agregamos dos nuevas columnas a P_it: 'Mayor_r1' y 'Mayor_r2'
    # Las columnas tendrán valores de 1 si el valor es mayor que r1 o r2.
    # En caso contrario, serán 0
    P_it['Mayor_r1'] = np.where((P['Fitness_Aptitud_Norm_cumsum'] > r1), 1, 0)
    P_it['Mayor_r2'] = np.where((P['Fitness_Aptitud_Norm_cumsum'] > r2), 1, 0)

    # Encontramos el valor mínimo en P_it
    # donde 'Mayor_r1' es el valor máximo de 'Mayor_r1'
    min_aptitud_umbral_r1 = P_it[P_it['Mayor_r1'] ==
    P_it['Mayor_r1'].max()]['Fitness_Aptitud_Norm_cumsum'].min()

    # Seleccionamos los registros en P_it es el valor mínimo encontrado
    # Esto representa el primer padre
    padre_1 = P_it[P_it['Fitness_Aptitud_Norm_cumsum'] == min_aptitud_umbral_r1]

    # Buscamos el individuo correspondiente en el conjunto de datos
    # basado en el primer padre seleccionado
    padre_1_pob = pob[pob['Individuo'] == padre_1['Individuo'].unique().tolist()[0]]

    # Encontramos el valor mínimo en P_it donde  es el valor máximo de 'Mayor_r2'
    min_aptitud_umbral_r2 = P_it[P_it['Mayor_r2'] ==
    P_it['Mayor_r2'].max()]['Fitness_Aptitud_Norm_cumsum'].min()

    # Seleccionamos los registros en P_it  es el valor mínimo encontrado
    # Esto representan el segundo padre
```

```
    padre_2 = P_it[P_it['Fitness_Aptitud_Norm_cumsum'] == min_aptitud_umbral_r2]

    # Buscamos el individuo basado en el segundo padre seleccionado
    padre_2_pob = pob[pob['Individuo'] == padre_2['Individuo'].unique().tolist()[0]]

    columnas_a_eliminar = ['Individuo','Fitness', 'Restricciones', 'Fitness+Rest']
    padre_1_pob = padre_1_pob.drop(columns=columnas_a_eliminar, errors='ignore')
    padre_2_pob = padre_2_pob.drop(columns=columnas_a_eliminar, errors='ignore')

    return padre_1_pob, padre_2_pob  # Devolvemos los padres seleccionados

#CRUCE
def cruce_porcent(padre_1_pob, padre_2_pob, porcentaje_cruzamiento):

    padre_1_pob = padre_1_pob.reset_index(drop=True)
    padre_2_pob = padre_2_pob.reset_index(drop=True)
    # Calcular el número de columnas a cruzar basado en el porcentaje de cruzamiento
    num_columnas = int(porcentaje_cruzamiento * padre_1_pob.shape[1])
    # Seleccionar las columnas que serán cruzadas
    columnas_a_cruzar = padre_1_pob.columns[:num_columnas].tolist()

    # Inicializar los DataFrames de los hijos como copias de padre_1_pob
    hijo_1 = padre_1_pob.copy()
    hijo_2 = padre_2_pob.copy()

    for i, columna in enumerate(columnas_a_cruzar):

        hijo_1[columna] = padre_2_pob[columna]
        hijo_2[columna] = padre_1_pob[columna]

    # Convertir los hijos a listas de posiciones
    hijo_1_posiciones = hijo_1.values.flatten().tolist()
    hijo_2_posiciones = hijo_2.values.flatten().tolist()

    return hijo_1_posiciones, hijo_2_posiciones


def cruce_segmentos(padre_1_pob, padre_2_pob ):
    posiciones_padre_1 = padre_1_pob.values.flatten().tolist()
    posiciones_padre_2 = padre_2_pob.values.flatten().tolist()

    hijo_1 = []
    hijo_2 = []

    segmento = 7
    for i in range(0, len(posiciones_padre_1), segmento):
        if (i // segmento) % 2 == 0:
```

```
            hijo_1.extend(posiciones_padre_1[i:i+segmento])
            hijo_2.extend(posiciones_padre_2[i:i+segmento])
        else:
            hijo_1.extend(posiciones_padre_2[i:i+segmento])
            hijo_2.extend(posiciones_padre_1[i:i+segmento])

    return hijo_1, hijo_2

#MUTACIÓN
def mutacion(xa, mu, sigma, lb, ub):
    for i, col in enumerate(xa.columns):
        if random.random() < mu:
            mutation_value = np.random.normal(0, sigma[i % len(ub)])
            xa[col] = xa[col]+mutation_value
            xa[col] = xa[col].round().astype(int)
            limites(xa, col, lb, ub[i % len(ub)])

def limites(df, columna, lb, ub):
    df[columna] = np.where(df[columna] < lb, lb, df[columna])
    df[columna] = np.where(df[columna] > ub, ub, df[columna])

def flatter(lst): #flatten para las listas
    ret = []
    for elem in lst:
        if isinstance(elem, list):
            ret.extend(flatter(elem))
        else:
            ret.append(elem)
    return ret

##Algoritmo
##inicializacion de variables de control de iteraciones
start_time = time.time()
meyor= 100000
control=0
###########CREACIÓN POBLACIÓN#################################################

pob=creacion_poblacion(tamanoPoblacion, dim,len(destinos), lb, ub,destinos)

pob=pob.sort_values('Fitness+Rest')
pob_res_max=np.transpose(pob.sort_values('Fitness').iloc[0])
Costs=pd.DataFrame(pob[[ 'Individuo','Fitness+Rest']])

#Almacena la solución mejor y peor
xMejor=np.transpose(pd.DataFrame(pob.iloc[0]))
xPeor=pob.iloc[-1]['Fitness+Rest']
```

```
Columnas_pop=
pd.DataFrame(pob.columns.to_list()).rename(columns={0:'Nombres_Columnas'})
# Creo un df con los nombres de las columnas que hay en pob
Columnas_pop_posicion=
Columnas_pop[Columnas_pop['Nombres_Columnas'].str.contains(destinos_regex)]['Nombres_Columnas
# Creo una lista con los nombres de las columnas Posicion que hay
# En este caso ['Posicion_0', 'Posicion_1']

##

resumen_iteraciones=pd.DataFrame(columns=flatter(['Numero_iteracion','Mejor_Fitness',
'Valor_fitnes_Más_restriciones',['Mejor_Posicion_De_'+ str(pos)
for pos in variables]]))

resumen_iteraciones=pd.DataFrame(columns=flatter(['Numero_iteracion', 'Mejor_Fitness',
'Valor_fitnes_Más_restriciones','Restriciones',['Mejor_Posicion_De_'+ str(pos)
for pos in Columnas_pop_posicion]]))

#BUCLE PRINCIPAL#################################################################################

for i in range(0,itera):
    P=Costs.copy()
    # Tengo el numero de individuo y Fitness
    P['Fitness_Aptitud']=np.exp(-beta*Costs['Fitness+Rest']/xPeor)
    # Añado la columna aptitud
    P['Fitness_Aptitud_Norm']=P['Fitness_Aptitud']/P['Fitness_Aptitud'].sum()
    # Añado la columna Aptitud normalizada
    P['Fitness_Aptitud_Norm_cumsum']=np.cumsum(P['Fitness_Aptitud_Norm'])
    # Añado la columna suma acumulada de la aptitud normalizada


    popc=pd.DataFrame(columns=pob.columns)
    #las columnas son individuo, posiciones y fitness  for j in range(0,round(nc/2)):

###########SELECCIÓN: Para elegir a los padres#####################################################
    padre_1_pob, padre_2_pob = seleccion(P)

###########CRUCE: Creamos los hijos################################################################
    hijo_1, hijo_2 = (cruce_segmentos(padre_1_pob, padre_2_pob) if j % 2 == 0
    else cruce_porcent(padre_1_pob, padre_2_pob, pc) )



    #Nº de individuo
    n_individuo_hijo_1=(i+1)*len(pob)+(j)*2
    n_individuo_hijo_2=(i+1)*len(pob)+(j)*2+1
```

```
    #Añadimos las filas
    popc_it=pd.DataFrame(columns=pob.columns)

    popc_it.loc[0]=flatter([n_individuo_hijo_1, hijo_1, np.nan,np.nan,np.nan])
    popc_it.loc[1]=flatter([n_individuo_hijo_2, hijo_2, np.nan,np.nan, np.nan])


    # for columna in Columnas_pop_posicion:
    # limites(popc_it, columna, lb, ub)


    #Fitness
    popc_it['Fitness']= fitness(popc_it, dim)
    popc_it['Restricciones']=rerstricciones(popc_it, dim)
    popc_it['Fitness+Rest']= popc_it['Fitness'] - restr_prio * popc_it['Restriccione
        if not popc.empty and not popc_it.empty:
            popc = pd.concat([popc, popc_it], axis=0)
        elif not popc.empty:
            popc = popc.copy()
        elif not popc_it.empty:
            popc = popc_it.copy()
        else:
            popc = pd.DataFrame()


    ########################ETAPA DE MUTACIÓN#############################################
    popm = pd.DataFrame(index=range(nm))
    # Crear un DataFrame vacío para los mutantes
    popm['Individuo'] = range((i + 1) * tamanoPoblacion +
    nc, (i + 1) * tamanoPoblacion + nc + nm)
    for l in range(len(destinos)):
        for n in range(dim):
            popm[destinos[l] + str(n)] = np.nan

    sigma = [0.05 * uub for uub in ub]  # Reducir sigma
    for k in range(nm):
        z = random.randint(0, tamanoPoblacion - 1)
        p = pob.iloc[z]
        p = pd.DataFrame(p).T
        xa = p[Columnas_pop_posicion].copy()

        mutacion(xa, mu, sigma, lb, ub)

        for col in Columnas_pop_posicion:
            popm.loc[k, col] = xa.iloc[0][col]

        # Rellenar la columna Fitness y restricciones
```

```
    popm['Fitness'] = fitness(popm, dim)
    popm['Restricciones'] = rerstricciones(popm, dim)
    popm['Fitness+Rest'] = popm['Fitness'] - restr_prio * popm['Restricciones']


  #########FIN ETAPA DE MUTACION#########################################################################

    pob = pd.concat([df for df in [pob, popc,popm ]
    if not df.isnull().all().all()], axis=0)
    # Ordena la población
    pob=pob.sort_values('Fitness+Rest')
    # Elegimos los mejores teniendo en cuenta las restriciones
    # Guardamos al mejor sin restricciones para mostraselo al usuario
    # Lo hacemos por si quiere cambier el criterio
    pob_res=np.transpose(pob.sort_values('Fitness').iloc[0])
    if pob_res['Fitness']<pob_res_max['Fitness']:
      pob_res_max=pob_res

    # Queremos que ordene segun el fitnes y las restriciones
    Costs=pd.DataFrame(pob[[ 'Individuo','Fitness+Rest']])

    # Cogemos los  primeros de la pob ordenada y de Costs (Fitness+Rest ordenado)
    pob=pob.iloc[0:tamanoPoblacion]
    Costs=Costs.iloc[0:tamanoPoblacion]

    # Almacena la solución mejor y peor
    xMejor=np.transpose(pd.DataFrame(pob.iloc[0]))
    xPeor=pob.iloc[-1]['Fitness+Rest']

    # control de iteraciones repetidas
    if xMejor['Fitness+Rest'].values[0]< meyor:
      meyor=xMejor['Fitness+Rest'].values[0]
      print(meyor)
      control=0
    else:
      control +=1
    #imprimo
    #print('Iteración:' , i)

    resumen_iteraciones.loc[i]=flatter([i, xMejor['Fitness'].values[0],
    xMejor['Fitness+Rest'].values[0],xMejor['Restriciones'],
    [xMejor['Posicion_'+str(pos)].values[0] for pos in range(0, dim)]])

    resumen_iteraciones.loc[i]=flatter([i, xMejor['Fitness'].values[0],
    xMejor['Fitness+Rest'].values[0],xMejor['Restricciones'].values[0],
    [xMejor[pos].values[0] for pos in Columnas_pop_posicion]])
## En caso que no se haya mejorado en rep interaciones el algoritmo para
```

```
    if control==rep:
      break;
print(flatter([i, xMejor['Fitness'].values[0],xMejor['Fitness+Rest'].values[0],
xMejor['Restricciones'].values[0],[xMejor[pos].values[0] for pos in Columnas_pop_pos
elapsed_time = (time.time() - start_time)/60
print(f"Tiempo transcurrido: {elapsed_time:.6f} minutos")
print(f"Iteraciones Ralizadas: {i} Iteraciones")
ultimo_valor = resumen_iteraciones.iloc[i]
fitness_display(ultimo_valor, dim)
restricciones_display(ultimo_valor, dim)

#########Gráfico Resumen ITeraciones-Fitness#########################################
sns.set()
fig, ax=plt.subplots(figsize=(18,5))
sns.lineplot(x = 'Numero_iteracion', y = 'Mejor_Fitness', data=resumen_iteraciones,
color='Blue', label='Fitness')
plt.legend(fontsize=14)
ax.set_xlabel('Iteración', fontsize=15)
ax.set_ylabel('Mejor Fitness', fontsize=15)
ax.set_title('Resumen Iteraciones sin Restricciones', fontsize=20)
fig, axs=plt.subplots(figsize=(18,5))
sns.lineplot(x = 'Numero_iteracion', y = 'Valor_fitnes_Más_restriciones',
data=resumen_iteraciones, color='Green', label='Fitness+Rest')
plt.legend(fontsize=14)
axs.set_xlabel('Iteración', fontsize=15)
axs.set_ylabel('Mejor Fitness con restricciones', fontsize=15)
axs.set_title('Resumen Iteraciones con Restricciones', fontsize=20)

#########################################################################################
resumen_iteraciones.loc[i]=flatter(['Mejor resultado sin restricciones',
pob_res['Fitness'],pob_res['Fitness+Rest'],pob_res['Restricciones'],[pob_res[pos]
for pos in Columnas_pop_posicion]])

resumen_iteraciones.loc[i+1]=flatter(['Mejor resultado con restricciones',
xMejor['Fitness'].values[0],xMejor['Fitness+Rest'].values[0],
xMejor['Restricciones'].values[0], [xMejor[pos].values[0]
for pos in Columnas_pop_posicion]])

resumen_iteraciones.to_excel('resumen_iteraciones.xlsx', index=False)

# Mostrar mensaje de confirmación
print("El DataFrame ha sido exportado a 'resumen_iteraciones.xlsx'")
plt.show()
```

Apéndice 5

CÓDIGO ALGORITMO NSGA-II

```
from deap import base, creator, tools, algorithms
import random
import numpy as np
import matplotlib.pyplot as plt
import json
import math
import time
# Parámetros del problema
NUM_DIAS = 7
NUM_DESTINOS = 7
NUM_VARIABLES = NUM_DIAS * NUM_DESTINOS  # 49 variables
tamanoPoblacion = 200  # Tamaño de la población
dim=6                           #número de dimensiones del problema
itera=20000                 #número máximo de iteraciones
rep=200                 #número máximo de iteraciones sin cambios
ub=[600,174,72,36,97,110,93]      #límite superior
lb=0                            #límite inferior
nc=10                          #número de hijos descendientes (Ha de ser par!)
nm=round(0.1*tamanoPoblacion)  #número de descendientes mutados
mu=0.1                          #tasa de mutación
beta=6                      #parámetro de selección
NGEN = 5000  # Número de generaciones
CXPB, MUTPB = 0.65, 0.3  # Probabilidades de cruce y mutación
restr_prio=130                  #parametro de premio por cumplir las restriciones
destinos=["Pasajes","Barcelona","Valencia","Santander","España","Francia","Suiza"]
destinos_regex = '|'.join(destinos)
bufferPas = [195, 21, 19, 0]
bufferBar = [153, 11, 4, 100, 91,88,22,112,0,0,79,28, 24,99, 40,78,0,0,0,0,0,0]
bufferVal = [8, 29, 24, 0, 57,6,31,4,0,63,0]
bufferSan = [15, 11, 4, 0, 5,8,0,12,0,0,17,8, 24, 30, 9,16,32,45,0,0,0,0]
barcosPas = [0, 0, 0, 1560, 0, 850, 0, 0, 750, 0, 0, 930, 1560]
barcosBar = [0, 100, 0, 0, 0, 250, 0, 0, 0, 200, 0, 500, 0]
barcosVal = [0, 0, 0, 100, 0, 0, 50, 0, 0, 220, 0, 0, 100]
barcosSan  = [0, 70, 0, 0, 50, 0, 0, 0, 0, 0, 150, 220, 0]
CP = [520, 391, 381, 362, 478, 528, 504]
CB = [100, 17, 70, 31, 48, 99, 39]
CV = [15, 24, 8, 12, 16, 10, 19]
CS = [15, 16, 12, 8, 34, 10, 26]
CF = [13, 60, 35, 86, 45, 23, 47]
CE = [20, 50, 25, 16, 45, 23, 47]
CSW = [64, 86, 28, 15, 33, 23, 31]
```

```
# Función de ocupación del campamento
def OcupacionCampa(p, CP):
    return sum(CP) - sum(p)
# Función de ocupación completa
def Ocupational_function(p, b, v, s, e, f, sw, CP, CB, CV, CS, CE, CF, CSW):
    return abs(OcupacionCampa(p, CP)) + abs(OcupacionCampa(b, CB)) +
    abs(OcupacionCampa(v, CV)) + abs(OcupacionCampa(s, CS)) +
    abs(OcupacionCampa(e, CE)) + abs(OcupacionCampa(f, CF)) +
    abs(OcupacionCampa(sw, CSW))
def restar_en_cascada(lista, numero):
    for i in range(len(lista) - 1, -1, -1):
    # Iterar elementos de la lista en
    orden inverso
    if numero >= lista[i]:
    # Si el número es > o = que el elemento actual de la lista numero -= lista[i]
    # Restar el elemento actual de la lista del númeroli sta[i] = 0
            # Establecer el elemento actual de la lista a cero
else:
            # Si el número es menor que el elemento actual de la lista
            lista[i] -= numero  # Restar el número del elemento actual de la lista
            numero = 0  # Establecer el número a cero
            break  # Salir del bucle
    return lista
def Regular_function(values):
    total_variance = 0
    for val_list in values:
        mean = sum(val_list) / len(val_list)
        variance = math.sqrt(sum((val - mean) ** 2 for val in val_list) / len(val_li
        total_variance += variance
    return total_variance
# Función de gestión de puertos
def gestion_puertos(buffer, cochesEntrantes, barcos, costo):
    costoTotal = 0
    for i in range(len(barcos)):
        # Guardar el último elemento de la lista
        ultimo_elemento = buffer[-1]
        # Eliminar el último elemento
        buffer.pop()
        # Sumar el número al último elemento de la lista
        buffer[-1] += ultimo_elemento
        # Insertar el número de coches entrantes en la primera posición de la lista
        if i < len(cochesEntrantes):
            buffer.insert(0, cochesEntrantes[i])
        else:
            buffer.insert(0, 0)
        # Sumar los costos
        costoTotal += buffer[-1] * costo
```

```
        if barcos[i] > 0:
            buffer = restar_en_cascada(buffer, barcos[i])
    return costoTotal



# Función para generar un individuo respetando restricciones
def generar_individuo():
    individuo = []
    for dia in range(NUM_DESTINOS):
     for destino in range(NUM_DIAS):
        if dia in [5, 6] and destino == 0: # Francia y Suiza en Día 1 siempre 0
            individuo.append(0)
        else:
            individuo.append(random.randint(lb, ub[dia]))  # Rango permitido
    return individuo
def rerstricciones(individual):
    restr_list = []
    # Las mismas operaciones, pero adaptadas al individuo (no al DataFrame)
    p = individual[0:7]
    b = individual[7:14]
    v = individual[14:21]
    s = individual[21:28]
    e = individual[28:35]
    f = individual[35:42]
    sw = individual[42:49]

    restr_impor = 0
    # Restricciones de sumas
    restr_impor += int(sum(p) <= 3584)
    restr_impor += int(sum(b) <= 457)
    restr_impor += int(sum(sw) <= 280)
    restr_impor += int(sum(s) <= 126)
    restr_impor += int(sum(v) <= 121)

    # Restricciones para cada día
    for i in range(7):
        restr_impor += int(b[i] + s[i] + v[i] + e[i] + f[i] + sw[i] <= 400)
        restr_impor += int(Stock(p, CP, i) >= 0)
        restr_impor += int(Stock(b, CB, i) >= 0)
        restr_impor += int(Stock(s, CS, i) >= 0)
        restr_impor += int(Stock(v, CV, i) >= 0)
        restr_impor += int(Stock(e, CE, i) >= 0)
        restr_impor += int(Stock(p, CP, i) <= 1350)
        restr_impor += int(Stock(b, CB, i) + Stock(s, CS, i) + Stock(v, CV, i)
        + Stock(e, CE, i) + Stock(f, CF, i) + Stock(sw, CSW, i) <= 643)
```

```
        restr_impor += int(StockFySW(f, CF, i) <= 0)
        restr_impor += int(StockFySW(sw, CSW, i) <= 0)

    return restr_impor



def fitness(individual):
    # Extraer los valores del individuo en grupos de 7 (uno por destino)
    p = individual[0:7]   # Pasajes
    b = individual[7:14]  # Barcelona
    v = individual[14:21] # Valencia
    s = individual[21:28] # Santander
    e = individual[28:35] # España
    f = individual[35:42] # Francia
    sw = individual[42:49]# Suiza

    # Calcular las funciones objetivo
    occupation_value = Ocupational_function(p,b,v,s,e,f,sw,CP,CB,CV,CS,CE,CF,CSW)
    regularity_value = Regular_function([p, b, v, s, e, f, sw])
    gestion_value = gestion_puertos(bufferPas, p, barcosPas, 0.77) + \
                    gestion_puertos(bufferBar, b, barcosBar, 1.5) + \
                    gestion_puertos(bufferVal, v, barcosVal, 1.38) + \
                    gestion_puertos(bufferSan, s, barcosSan, 1.5)
    # Verificar las restricciones
    restriction_penalty = rerstricciones(individual)

    # Sumar la penalización por restricciones (si alguna restricción se viola)
    if rerstricciones(individual)>=75:
      return (occupation_value , regularity_value , gestion_value )
    else:
      return 1000000-restriction_penalty,1000000-restriction_penalty,
      1000000-restriction_penalty
    # Devolver los valores de los objetivos + penalización (debe ser minimización)

## RESTRICIONES
# Función para calcular el stock de coches entrantes y coches en campa
def Stock(cochesEntrantes, cochesCampa, day):
    res = 0
    for i in range(0, day + 1):
        res = cochesCampa[i] - cochesEntrantes[i] + res
    return res

# Función para calcular el stock para Francia y Suiza
def StockFySW(cochesEntrantes, cochesCampa, day):
    res = 0
    for i in range(day + 1):
```

```
        if i - 1 < 0:
            res = cochesEntrantes[i] + res
        else:
            res = cochesEntrantes[i] - cochesCampa[i - 1] + res
    return res



def mutacion_deap(individual, mu, sigma, lb, ub):
    for i in range(len(individual)):
        if random.random() < mu:
        # Probabilidad de mutación
            mutation_value = np.random.normal(0, sigma[i % len(sigma)])
            # Mutación gaussiana
            individual[i] += mutation_value
            # Aplicar mutación
            individual[i] = int(max(lb, min(ub[i % len(ub)], individual[i])))
            # Asegurar límites
    return individual,
def cxSegmentoFijo(ind1, ind2, segment_size=7):
    """
    Cruce por segmentos fijos: intercambia bloques de tamaño `segment_size`
    entre dos individuos con una probabilidad del 50% por segmento.
    """
    num_segments = len(ind1) // segment_size
    # Número de segmentos en el individuo

    for i in range(num_segments):
        if random.random() < 0.5:
        # Probabilidad del 50% de intercambiar cada segmento
            start = i * segment_size
            end = start + segment_size
            ind1[start:end], ind2[start:end] = ind2[start:end], ind1[start:end]
            # Intercambio de segmentos

    return ind1, ind2  # DEAP requiere que se retornen los individuos modificados
# Definir el toolbox
start_time = time.time()
creator.create("FitnessMin", base.Fitness, weights=(-1, -1, -1))
# Minimizar una función objetivo
creator.create("Individual", list, fitness=creator.FitnessMin)
toolbox = base.Toolbox()
toolbox.register("individual", tools.initIterate, creator.Individual,
generar_individuo)

toolbox.register("population", tools.initRepeat, list, toolbox.individual)
# Registrar la función de evaluación en DEAP
```

```
toolbox.register("evaluate", fitness)

# Selección por NSGA-II (multiobjetivo)
toolbox.register("select", tools.selNSGA2)

# Cruce en dos puntos
toolbox.register("mate", tools.cxTwoPoint)
# Cruce de segmmento fijo
toolbox.register("mate", cxSegmentoFijo)
# Registrar mutación en DEAP
toolbox.register("mutate", mutacion_deap, mu=0.2,
sigma = [0.05 * uub for uub in ub], lb=0, ub=[600,174,72,36,97,110,93])

# Crear la población inicial
poblacion = toolbox.population(n=tamanoPoblacion)
#poblacion.append(creator.Individual([0,445,316,351,361,474,521,43,42,65,63,
53,45,93,0,7,25,25,8,17,22,13,15,12,11,16,28,26,0,23,21,28,27,61,66,0,13,45,
36,85,38,45,0,49,44,34,39,44,39]))
# Evaluar la población inicial
fitnesses = list(map(toolbox.evaluate, poblacion))

for ind, fit in zip(poblacion, fitnesses):
    ind.fitness.values = fit

# Ejecutar el algoritmo evolutivo
algorithms.eaMuPlusLambda(poblacion, toolbox,
                          mu=tamanoPoblacion, lambda_=tamanoPoblacion * 2,
                          cxpb=CXPB, mutpb=MUTPB, ngen=NGEN, verbose=True)

# Extraer los individuos del frente de Pareto
pareto_front = tools.sortNondominated(poblacion, len(poblacion), first_front_only=Tru
objectives = np.array([ind.fitness.values for ind in pareto_front])



# Guardar como JSON
elapsed_time = (time.time() - start_time)/60
print(f"Tiempo transcurrido: {elapsed_time:.6f} minutos")
with open("pareto_front.json", "w") as f:
    json.dump(pareto_front, f, indent=4)

with open("pareto_front_objetives.json", "w") as f:
    json.dump(objectives.tolist(), f)
```