Informe Técnico - Technical Report
DPTOIA-IT
julio, 2024

Sistema inteligente para la optimización del proceso de empaquetado

Germán Francés Tostado

Departamento de Informática y Automática
Universidad de Salamanca

Resumen

En la sociedad globalizada actual el envío de artículos mediante mensajería está siendo cada vez más creciente, por lo que está ocurriendo un auge en técnicas de optimización para que las empresas puedan enviar más artículos con menos costes. Este trabajo consistirá en el estudio del problema de la mochila mediante la aplicación de técnicas de optimización combinatoria y aprendizaje automático, con el objetivo de superar las limitaciones de los métodos tradicionales.

Para ello, se evaluarán diversos algoritmos, incluyendo heurísticas clásicas y métodos basados en aprendizaje profundo. Estos métodos son puestos a prueba en un conjunto de escenarios simulados (casos de estudio) que reflejan diversas condiciones de empaquetado, con el fin de evaluar la adaptabilidad y robustez de las diferentes técnicas.

Con los resultados obtenidos del análisis, se ha implementado una plataforma web encargada de realizar la optimización de empaquetado de manera interactiva e ideada para operarios de almacén encargados de realizar la preparación de envíos. De esta manera, el sistema ayudará a los operarios que se encargan de la logística de los bultos, indicándoles cómo empaquetar los artículos, para reducir errores humanos y mejorando su eficiencia.

Palabras clave: Contenedores, Bultos, Optimización de empaquetado, Problema de la mochila, Heurísticas avanzadas

Abstract

In today's globalized society, the shipment of items through courier services is increasingly growing, leading to a surge in optimization techniques so that companies can ship more items at lower costs. This work involves the study of the knapsack problem through the application of combinatorial optimization techniques and machine learning, surpassing the limitations of traditional methods.

For this purpose, various algorithms, including classic heuristics and methods based on deep learning, will be evaluated. These methods are tested in a set of simulated scenarios that reflect various packing conditions, in order to assess the adaptability and robustness of the different techniques.

With the results obtained from the analysis, a web platform has been implemented that is responsible for carrying out packaging optimization interactively designed for warehouse operators in charge of preparing shipments. In this way, the system will help operators who are in charge of logistics of the packages, indicating how to pack to reduce human errors and improve their efficiency.

Keywords: Containers, Packages, Packaging Optimization, Knapsack Problem, Advanced Heuristics

Índice

Índice de figuras

Indice de tablas

1. Introducción

En los últimos años, las transacciones comerciales han experimentado una transición muy rápida a la era digital. Inicialmente todo el intercambio era completamente manual, donde el vendedor y los compradores tenían que encontrarse en algún lugar físico para llevar a cabo el negocio. En la actualidad, se ha producido un cambio hacia el uso de la tecnología de comercio electrónico (*E-commerce*), [24]. El vendedor y el comprador ya no tienen que reunirse en persona, sino que aprovechan una plataforma en línea para los intercambios comerciales y la exposición de los productos. Ha habido una transformación en el sistema comercial que comenzó desde el trueque, se desarrolló en mercados tradicionales, mercados modernos y hoy en día está en la era de los mercados digitales.

Las ventas por Internet han desempeñado un papel cada vez más importante en el comercio minorista. En 2022, el comercio electrónico representó casi el 19 % de las ventas minoristas en todo el mundo. Las previsiones indican que, en 2027, el segmento en línea representará cerca de una cuarta parte del total de las ventas minoristas mundiales (Figura 1).

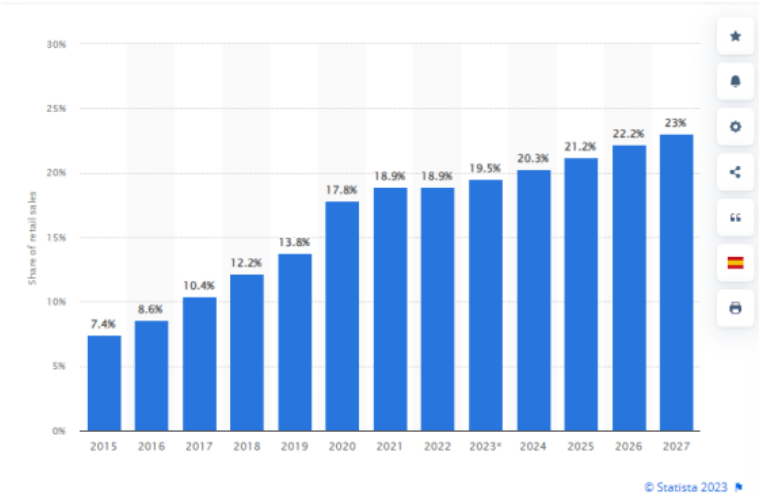

Figura 1: E-Commerce como porcentaje de las ventas minoristas totales en todo el mundo de 2015 a 2027 [5]

Al haber cada vez más compras de manera online, esto genera que cada vez más empresas tengan que enviar los productos vendidos a través de mensajería, desde los centros de operaciones hasta las casas de los clientes.

Los envíos estarán formados por artículos de muy diversos tipos y cada artículo tendrá asignado un embalaje contenedor, ya sea una caja o un sobre, habitualmente estas cajas tendrán unas medidas generales en lugar de ser un embalaje único para cada producto. Esto dependerá de cada empresa o del transportista encargado para cada una.

De este modo se genera una diversidad de bultos en movimiento desde el punto de venta hasta el punto del comprador. La problemática yace sobre si un cliente que

ha de recibir varios artículos, puede que reciba varios bultos diferentes con varios embalajes.

Este trabajo busca proponer una solución beneficiosa para sendas partes. Mediante la optimización en el proceso de empaquetado, el proveedor podrá enviar más artículos en un mismo bulto y el cliente recibirá el mínimo número necesario de bultos para su pedido.

La optimización del proceso de empaquetado representa un desafío crítico dentro del amplio campo de la logística y la gestión de la cadena de suministro. A medida que el comercio electrónico continúa expandiéndose y el volumen de envíos crece exponencialmente, las empresas buscan métodos más eficientes para organizar y enviar productos. En este contexto, el desarrollo de sistemas inteligentes capaces de optimizar el empaquetado no solo promete mejoras significativas en la eficiencia y la reducción de costes, sino que también tiene el potencial de impactar positivamente en la sostenibilidad ambiental al minimizar los desechos y maximizar el uso del espacio de transporte.

El trabajo se centrará en explorar y evaluar diversas técnicas de optimización aplicadas al empaquetado, con un interés particular en el modelado y solución del problema de la mochila tridimensional (*3D Knapsack Problem*). Este problema es un excelente representante de los desafíos prácticos enfrentados por las operaciones logísticas modernas, donde los objetos de diversas formas y tamaños deben ser organizados de manera óptima dentro de contenedores o cajas de envío, bajo restricciones de volumen y peso.

El objetivo principal de este estudio es analizar y comparar diferentes algoritmos y herramientas de empaquetado, desde heurísticas tradicionales hasta técnicas avanzadas de aprendizaje automático y algoritmos metaheurísticos. Se pretende identificar cuál de estas metodologías ofrece mejores resultados en términos de eficiencia de empaquetado, uso del espacio, tiempo de procesamiento y adaptabilidad a diferentes tipos de productos y requisitos logísticos.

Para alcanzar estos objetivos, se llevará a cabo un estudio comparativo de los algoritmos seleccionados, implementando cada uno en un conjunto diverso de escenarios de prueba para evaluar su rendimiento bajo condiciones controladas. Este enfoque no solo permitirá identificar las fortalezas y debilidades de cada técnica, sino que también facilitará el desarrollo de un marco integrado que podría ser utilizado por las empresas para mejorar sus operaciones de empaquetado en tiempo real.

Asimismo, se elegirá uno de los algoritmos estudiados para la implementación en una plataforma de ayuda a los empleados encargados de la empaquetación del pedido, de manera que podrán seguir en tiempo real el proceso de colocación óptimo estimado por el algoritmo a la hora de crear un envío.

En última instancia, este estudio pretende ser una aportación a la literatura existente dentro del tema de investigación al proporcionar una comprensión más profunda de las técnicas de optimización de empaquetado y ofrecer una guía práctica para su aplicación en entornos industriales. Generando múltiples beneficios en el ámbito de logística, este enfoque puede reducir el número de bultos a hacer se-

guimientos, disminuir el espacio ocupado en los transportes, y minimizar la huella medioambiental así como el desperdicio de materiales. Adicionalmente, este enfoque tiene el potencial de aumentar el margen de ganancia de los vendedores y reducir los costes a los clientes, contribuyendo a una operativa más eficiente y sostenible.

2. Objetivos

Como se ha mencionado en la introducción, el objetivo principal del trabajo es la exploración de diferentes técnicas de optimización aplicadas al proceso de empaquetado, mediante diferentes algoritmos y herramientas. Con este objetivo se busca poder mostrar una visión global del desarrollo actual y futuro de procesos automatizados en entornos logísticos. Partiendo de este objetivo principal, se han desarrollado diferentes subobjetivos con el fin de facilitar la realización del trabajo.

- En primer lugar se pretende identificar las diferentes herramientas y técnicas exploradas en la literatura actual referentes a la optimización del problema de la mochila tridimensional.
- Realizar un estudio sobre las diferentes herramientas y técnicas asociadas a la optimización.
- Establecer casos de estudio que reflejen diferentes desafíos de empaquetado
- Llevar a cabo una comparativa de los diferentes algoritmos en términos de eficiencia, porcentaje de espacio utilizado, tiempo de procesamiento, y adaptabilidad a diferentes tipos de productos.
- Desarrollar una plataforma que ayude a los operarios de almacén a llevar a cabo un empaquetado óptimo, mejorando la eficiencia operativa y reduciendo errores humanos.

3. Marco teórico

Para una mejor comprensión del trabajo realizado, se aclararán previamente una serie de conceptos recurrentes a lo largo de la memoria que no tienen por qué ser conocidos previamente por el lector. De esta manera, se establecerán algunos conceptos fundamentales para el trabajo.

3.1. NP-hard

Un problema se considera *NP-hard* cuando cumple con las características de dificultad computacional asociadas a los problemas de decisión *NP* (*No Deterministic Polynomial time*), pero no necesariamente tiene que ser un problema de decisión.

Los problemas *NP-hard* son de alta complejidad computacional. Resolverlos en un tiempo polinómico es un desafío considerable y, en muchos casos, se cree que es imposible dado que aun no se ha conseguido probar $P = NP$.

Un problema P es *NP-hard* si cualquier problema $Q \in NP$ puede ser reducido a P en tiempo polinómico. Esto significa que existe un algoritmo que transforme instancias del problema Q en instancias del problema P en tiempo polinómico, de manera que la solución al problema transformado de una solución al problema original, implicando que P es al menos tan difícil como cualquier problema en *NP*.

Existen muchos ejemplos de problemas *NP-hard*, algunos son:

- Problema del vendedor ambulante: Encontrar el recorrido más corto que pasa por un conjunto de ciudades y vuelve al punto de partida.
- Problema del SAT: Determinar si una fórmula booleana dada es satisfacible, es decir, si existe una asignación de valores de verdad (verdadero o falso) a las variables de la fórmula, que la evalúan como verdadera.
- Problema de la mochila: Decidir qué objetos se pueden meter en una mochila de capacidad limitada. Será el problema estudiado.

3.2. Problema de la mochila

También mencionado como *Knapsack problem* o *Bin packing problem* (BPP), es un problema de optimización combinatoria. Forma parte del conjunto de problemas conocidos como *NP-hard*. Un tipo de problema computacional para el que no se ha encontrado un algoritmo que pueda resolverlo de manera eficiente (en un tiempo razonable), y si se ha encontrado solución puede que sea verificable o no.

Según las definiciones en [9], un problema de optimización combinatoria puede ser un problema de minimización o maximización y consiste en los siguientes tres elementos:

1. Un conjunto D_Π de instancias;

2. Para cada instancia $I \in D_\Pi$, un conjunto finito $S_\Pi(I)$ de soluciones candidatas para I;

3. Una función m_Π que asigna a cada instancia $I \in D_\Pi$ y cada solución candidata $\sigma \in S_\Pi(I)$ un número racional positivo $m_\Pi(I, \sigma)$, llamado el *valor de solución* para σ.

Si Π es un problema de minimización, entonces una *solución óptima* para una instancia $I \in D_\Pi$ es una solución candidata $\sigma^* \in S_\Pi(I)$ tal que, para todo $\sigma \in S_\Pi(I)$, $m_\Pi(I, \sigma^*) \leq m_\Pi(I, \sigma)$. Usaremos $OPT_\Pi(I)$ para denotar el valor $m_\Pi(I, \sigma^*)$ de una solución óptima I (soltando el subíndice Π cuando el problema esté claro en el contexto).

Un algoritmo A es un *algoritmo de aproximación* para Π, dado cualquier instancia $I \in D_\Pi$, si encuentra una solución $\sigma \in S_\Pi(I)$. El valor $m_\Pi(I, \sigma)$ de la solución candidata σ encontrado por A aplicado a I se denota como $A(I)$ y si $A(I) = OPT(I)$ para todo $I \in D_\Pi$ entonces A se llama un algoritmo de optimización para Π.

Como en este caso de estudio el problema es *NP-hard*, se sabe que un algoritmo de optimización en tiempo polinomial no va a ser posible a menos que $P = NP$. La meta entonces será encontrar un algoritmo de aproximación A que reduzca el tiempo polinomial y que tenga la propiedad de, para todas las instancias I, $A(I)$ estén "cerca" a $OPT(I)$. Este tipo de resultado será el correspondiente los del problema de la mochila.

De esta manera el problema de la mochila quedará definido como: Dado un conjunto finito $U = \{u_1, \ldots, u_n\}$ de "objetos" y un tamaño racional $s(u) \in [0, 1]$ para cada objeto $u \in U$, se encuentre una partición U de conjuntos disjuntos $U_1, U_2, \ldots, U_k$, de tal manera que la suma de los tamaños de los objetos en cada subconjunto sea lo más pequeña posible. Queremos empaquetar los objetos de U en la menor cantidad de subconjuntos (o contenedores) posible. Este problema es *NP-hard*, por lo que hay pocas esperanzas de encontrar un algoritmo de optimización en tiempo pseudo-polinomial, sin embargo, hay varios algoritmos de aproximación simples que vale la pena considerar.

Uno de estos es conocido como el algoritmo *First Fit*. Se empezará con una secuencia infinita de contenedores $B_1, B_2, \ldots$, todos los cuales están vacíos. El algoritmo luego coloca los objetos uno a la vez, en orden de índice creciente. Hace esto de acuerdo con la siguiente regla simple: siempre coloca el objeto siguiente en el contenedor indexado más bajo, en el que la suma de los tamaños de los objetos ya colocados en ese contenedor, más el tamaño del objeto actual no exceda 1-$s(u_i)$. En otras palabras, u_i siempre se coloca en el primer contenedor en el que cabrá sin exceder la capacidad del contenedor. La Figura 2 muestra un ejemplo, donde cada objeto está representado por un rectángulo cuya altura es proporcional a su tamaño.

Utilizando esta base, se analizarán unos pocos de las diferentes metodologías y algoritmos propuestos a lo largo de los últimos años con el fin de resolver el problema

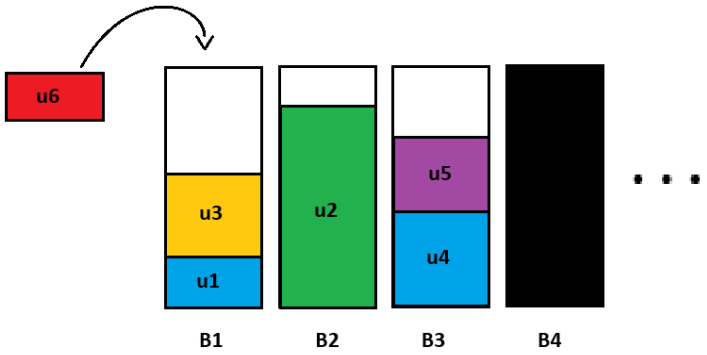

Figura 2: Ejemplo *First Fit*, u6 se colocará en el primer contenedor en que entre

de la mochila. Como distinción a mayores, el problema de la mochila se puede dividir en varios tipos en función de sus dimensiones.

1. En el problema con una dimensión, se clasifica teniendo en cuenta una sola variable relacionada con los objetos, como podría ser el peso, esto se conoce como el problema de la mochila simple.

2. En el problema con dos dimensiones, se clasifica teniendo en cuenta dos atributos de las clases, siendo conocido como el problema de la mochila de múltiple elección.

3. En concreto para este trabajo interesa la versión de tres dimensiones, siendo ésta la más compleja, ya que consiste en encontrar la mejor manera de llenar una mochila en el mundo real con objetos de diferentes tamaños y valores, dado un límite de capacidad en la mochila.

El objetivo es maximizar el valor total de los objetos seleccionados sin exceder la capacidad máxima de la mochila. Cada objeto tiene un peso y un valor asociados, y la restricción principal es que no se pueden fraccionar los objetos; es decir, se deben seleccionar o desechar por completo. Se considera el problema en tres dimensiones de manera ortogonal, es decir, los vértices de los objetos deberán estar paralelos a las aristas de los contenedores, y las rotaciones (si se dan) deberán ser de 90 grados. El desafío es encontrar una combinación de objetos que maximice el valor total sin sobrepasar el peso máximo permitido en la mochila. Puede haber casos donde el peso vaya asociado de un coste monetario, por lo que se puede establecer restricciones en base al máximo peso permitido y dividir el contenido en diferentes mochilas. Al ser la versión 3D del problema de la mochila, podremos tener diferentes mochilas y habrá que situar los objetos en el plano contenedor.

El estudio llevado a cabo se encargará de intentar conseguir siempre el mejor ajuste dadas las medidas físicas de los artículos y de los contenedores.

3.3. Heurísticas

En el contexto de este trabajo, el término *heurísticas* se referirá a métodos o estrategias simplificadas que se utilizan para resolver problemas complejos de manera más rápida que los métodos tradicionales, exactos o completos, pero sin garantizar necesariamente la obtención de la solución óptima. Las heurísticas son particularmente útiles en problemas de optimización como puede ser el mencionado problema de la mochila, donde la búsqueda de una solución óptima puede ser computacionalmente inviable debido a la gran cantidad de posibles configuraciones y restricciones. Estos casos, se verán más adelante.

3.4. Empaquetado Online - Offline

Los problemas de empaquetado se dividen en dos categorías dependiendo de si se conocen de antemano los elementos que se van a empaquetar. Específicamente, en la categoría *offline*, se conoce de antemano la información de todos los elementos que se van a empaquetar, mientras que en la categoría *online*, los elementos llegan uno por uno y el algoritmo o heurística solo tiene conocimiento del elemento actual a empaquetar en ese momento.

3.5. Aprendizaje automático

El aprendizaje automático es un campo de la Inteligencia Artificial que se enfoca en desarrollar algoritmos y modelos que permiten a las computadoras aprender y hacer predicciones o tomar decisiones basadas en datos. En lugar de ser programadas explícitamente para realizar una tarea específica, las máquinas utilizan técnicas estadísticas y matemáticas para identificar patrones y tendencias en grandes conjuntos de datos. A través de este proceso, las computadoras pueden mejorar su rendimiento en tareas específicas con el tiempo, ajustando sus modelos en respuesta a nuevos datos.

El aprendizaje automático se puede clasificar en aprendizaje supervisado, no supervisado y por refuerzo. En el aprendizaje supervisado, los modelos se construyen y entrenan con un conjunto de datos que incluyen entradas y salidas conocidas, permitiendo al algoritmo aprender una función que pueda predecir la salida correcta para nuevas entradas. Por otro lado, el aprendizaje no supervisado no utiliza datos etiquetados y está diseñado para identificar la estructura intrínseca de los datos, agrupando información similar o reduciendo la dimensionalidad de los datos. El aprendizaje por refuerzo se centra en aprender de la interacción con un entorno donde las decisiones secuenciales reciben recompensas o penalizaciones.

El proceso de aprendizaje implica varios pasos críticos que comienzan con la preparación y transformación de los datos crudos para el análisis. Esto puede incluir normalización, tratamiento de valores faltantes y codificación de variables. Los datos se dividen típicamente en conjuntos de entrenamiento y prueba, y a veces en un

conjunto de validación para ajustar los parámetros del modelo. El tipo de modelo apropiado se selecciona según la tarea específica, como clasificación o regresión, y se entrena utilizando el conjunto de datos de entrenamiento para ajustar los parámetros del modelo de manera que mejore la precisión de las predicciones o maximice la recompensa.

Una vez entrenado, el modelo se evalúa con el conjunto de prueba para determinar su eficacia en datos no vistos. Dependiendo de los resultados, el modelo puede requerir ajustes adicionales para optimizar su rendimiento antes de ser desplegado en un entorno de producción para realizar predicciones en tiempo real o facilitar la toma de decisiones basadas en datos nuevos. Esta metodología demuestra la capacidad del aprendizaje automático para adaptarse y mejorar continuamente, ofreciendo soluciones efectivas a problemas complejos de datos.

3.6. Aprendizaje profundo

El aprendizaje profundo es una subdisciplina del aprendizaje automático. A diferencia de los algoritmos tradicionales de aprendizaje automático, que tienen una capacidad limitada para aprender sin importar cuántos datos se les proporcionen, los sistemas de aprendizaje profundo pueden mejorar su desempeño al tener acceso a una mayor cantidad de datos. Esto significa que, a medida que la máquina adquiere más datos, su capacidad para aprender y tomar decisiones también aumenta. Los sitemas de aprendizaje profundo logran esta capacidad a través de estructuras complejas conocidas como redes neuronales artificiales, inspiradas en las neuronas del cerebro humano.

Estas redes neuronales están compuestas por capas de nodos (o neuronas), interconectadas de manera que pueden transmitir señales de una capa a la siguiente. Cada capa transforma las entradas que recibe antes de pasarlas a la siguiente capa. Las redes neuronales profundas, incluyen múltiples capas intermedias entre las capas de entrada y salida, estas capas se denominan *capas ocultas*. Las capas ocultas son capaces de capturar relaciones complejas en los datos gracias a esta arquitectura de capas, lo que hace las redes neuronales profundas excepcionalmente buenas en tareas como reconocimiento de imágenes y el Procesamiento del Lenguaje Natural (PLN).

El entrenamiento de estas redes se realiza a través de un proceso conocido como propagación hacia atrás, que ajusta de manera iterativa los pesos de las conexiones entre las neuronas para minimizar la diferencia entre la salida predicha por la red y la salida real esperada. Este ajuste se realiza utilizando un algoritmo llamado Descenso de Gradiente, que busca optimizar una función de pérdida mediante el cálculo de gradientes de la función respecto a cada peso en la red. En la Figura 3 se expone un diagrama de ejemplo de la arquitectura conformada por una red neuronal profunda.

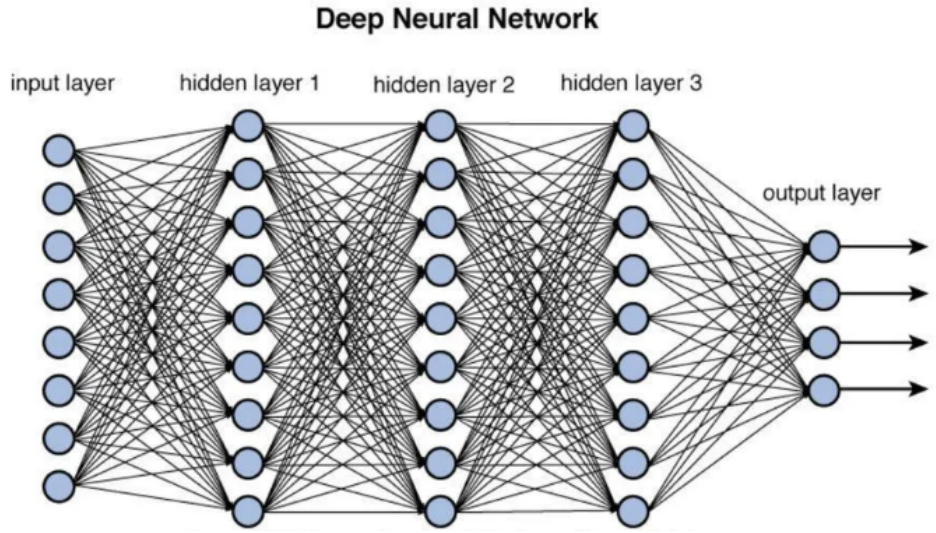

Figura 3: Arquitectura de una red neuronal profunda

3.7. Aprendizaje por refuerzo profundo

El aprendizaje por refuerzo profundo o Deep Reinforcement Learning (DRL), es una técnica avanzada que integra los principios del aprendizaje profundo con los del aprendizaje por refuerzo para crear sistemas que pueden aprender a tomar decisiones óptimas en entornos complejos. En esta metodología, un agente aprende a actuar en un entorno dinámico, donde cada acción que realiza es evaluada a través de recompensas o penalizaciones, que reflejan la efectividad de sus decisiones. El objetivo fundamental es maximizar la suma total de estas recompensas a lo largo del tiempo.

En el aprendizaje por refuerzo profundo, las redes neuronales, particularmente las Redes Neuronales Convolucionales (CNNs) y las Redes Neuronales Recurrentes (RNNs), se utilizan para aproximar la función de valor o la política de decisiones del agente. La función de valor estima qué tan buena es una determinada acción o estado en términos de recompensas futuras esperadas, mientras que la política de decisiones dirige al agente sobre qué acción tomar en un estado dado. Al combinar las capacidades de percepción y generalización del aprendizaje profundo con la estrategia de toma de decisiones basada en recompensas del aprendizaje por refuerzo, los agentes pueden aprender desde juegos complejos hasta tareas de navegación y control automático.

Uno de los desafíos clave en el aprendizaje por refuerzo profundo es el equilibrio entre la exploración de nuevas acciones y la explotación de las acciones que se sabe generan altas recompensas. Este balance se gestiona a través de estrategias como *epsilon-greedy*, donde el agente decide aleatoriamente explorar nuevas acciones con una pequeña probabilidad (*epsilon*), mientras que la mayoría del tiempo elige la acción que cree que maximizará las recompensas según su conocimiento actual.

4. Estado del arte

Para el desarrollo del estado del arte, se recurrirá a un Mapeo Sistemático de la Literatura sobre el tema de investigación. Para ello, se ha investigado la literatura relevante tanto de los sistemas inteligentes relacionados con procesos logísticos, y de almacén como la literatura pertinente sobre el problema de la mochila en especial, al proceso de empaquetado en cuatro dimensiones: longitud, altura, anchura y peso. Este proceso incluye la distribución equitativa de los bultos, optimizando el uso de espacio y el peso permitido, reduciendo costes y emisiones al medio ambiente.

Para llevar a cabo el mapeo de la literatura existente, se han utilizado diversas herramientas. En concreto, Parsifal se ha empleado como gestor de bibliografía para asegurar la trazabilidad del trabajo. Así se garantiza una descripción clara del proceso y los diferentes documentos y artículos revisados en cada fase. La metodología utilizada será la implementada por la herramienta, PICO. La metodología PICO ayudará a establecer los criterios de inclusión y exclusión de los estudios, así como preguntas de mapeo relevantes durante la revisión. De esta manera, se pretenderá tener una base sólida que permita evaluar el estado actual de la investigación en este tema.

Para delimitar la búsqueda de literatura, se considerarán únicamente los documentos y artículos publicados después de 2015, a fin de asegurar la inclusión de investigaciones recientes y relevantes en el campo. La búsqueda se realizará en bases de datos como Scopus, IEEE Xplore y Google Scholar, fuentes reconocidas por su amplitud y calidad de contenido académico. Los términos de búsqueda serán en inglés, aunque se aceptarán artículos tanto en español como en inglés. Los tipos de documentos considerados incluirán tanto revistas académicas como actas de congresos, garantizando una cobertura amplia de la literatura disponible sobre el problema de la mochila y su aplicación en procesos de logística.

4.1. Mapeo Sistemático de la Literatura

El proceso de mapeo seguido seguirá un enfoque estructurado y detallado que involucra una serie de pasos esenciales que garanticen la relevancia del estudio. Inicialmente se definirán las preguntas de mapeo que la revisión se propone responder, estableciendo así los objetivos del mapeo. Seguidamente, se describirán las fuentes de información que se utilizarán para la búsqueda. Posteriormente, se definirán las cadenas de búsqueda, incluyendo los términos más relevantes que facilitarán la recuperación de la literatura. Además, se establecerán criterios de selección, en particular, criterios de inclusión y exclusión que ayudarán a filtrar los estudios según su relevancia. Por último, se definirán criterios de calidad específicos que permitirán el filtrado y la evaluación de los resultados relevantes, asegurando que solo los estudios más relevantes y de mayor impacto en el tema sean considerados para el análisis final. Este procedimiento estructurado es fundamental para el desarrollo de una comprensión profunda en el campo de estudio.

4.1.1. Preguntas de Mapeo

Para tener un objetivo principal durante el proceso de la revisión, se han establecido las siguientes preguntas de mapeo, de esta manera para saber si se han cumplido los criterios establecidos al final de la revisión deberemos tener respuesta a las preguntas.

- ¿Qué tipo de algoritmos de optimización se han usado en la industria logística para mejorar el proceso de empaquetado?
- ¿Cómo de efectivas son estas técnicas de optimización comparadas con soluciones convencionales en los sistemas logísticos?
- ¿De qué manera estas soluciones influencian la eficiencia y producción en los negocios?
- ¿Qué desafíos y limitaciones están asociadas con la aplicación de soluciones inteligentes en las operaciones de almacén y logística del mundo real?

Estas preguntas de mapeo buscarán responder la pregunta de investigación principal que engloba el proyecto: ***¿Cuál es el estado actual del proceso de empaquetado en la industria logística y qué desafíos conlleva?***

4.1.2. Fuentes

Antes de establecer las cadenas de búsqueda de información se han definido las fuentes de información y sus respectivos motores de búsqueda. Se han seleccionado tres de las más populares e importantes, en especial Scopus, a la cual se ha podido acceder con los credenciales de la Universidad de Salamanca.

- SCOPUS [26]: es una base de datos bibliográfica administrada por Elsevier, que ofrece resúmenes y citas de artículos científicos en diversas disciplinas, facilitando el seguimiento y análisis de la producción científica.
- IEEE Xplore [13]: es una biblioteca digital del Instituto de Ingenieros Eléctricos y Electrónicos, especializada en ingeniería eléctrica, electrónica y tecnologías afines, proporcionando acceso a artículos de revistas, conferencias y normas técnicas.
- Google Scholar [10]: es un motor de búsqueda gratuito de Google que indexa literatura académica en múltiples formatos y disciplinas, incluyendo artículos de revistas, tesis y libros, y es accesible para una amplia audiencia.

4.1.3. Cadenas de búsqueda

Una vez establecidas las preguntas de investigación, se han definido palabras clave y sinónimos de ellas. Estas palabras clave serán las que ayudarán en la búsqueda en las diferentes fuentes de información.

En un principio se importaron las palabras claves PICO (*Population, Intervention, Comparison, Outcome, Context*, vistas en Figura 4), pero esto generó una cadena de búsqueda canónica demasiado grande y específica, por lo que los resultados arrojados eran nulos. De esta manera se ha decidido cribar los términos complejos por otros más sencillos.

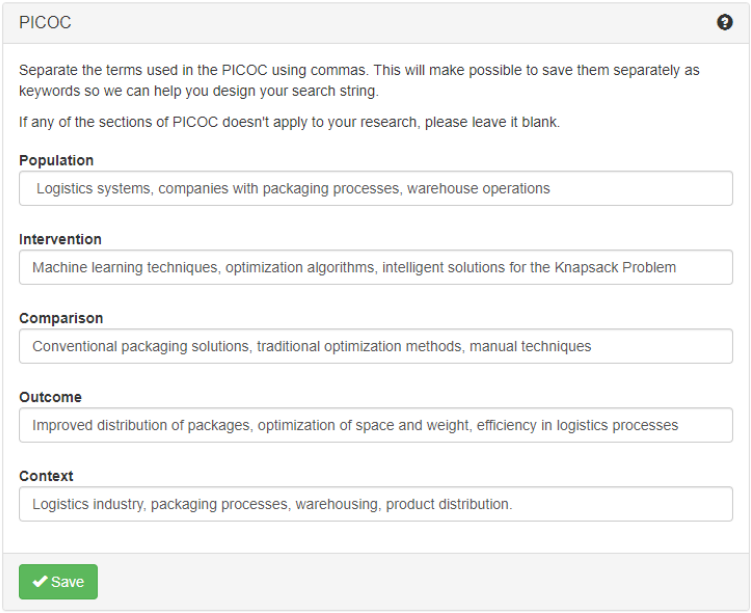

Figura 4: Términos PICOC

La cadena inicial probada ha sido:

("machine learning" OR "deep learning") AND ("bin packing optimization" OR "4d packing optimization" OR "knapsack problem" OR "4D knapsack problem" OR "box optimization" OR "box shipping" OR "capacity optimization") AND "optimization" OR "algorithm" OR "efficiency"

Arrojando **10.238** resultados en Scopus y **1.501.971** en IEEE Xplore, siendo unos resultados difícilmente manejables se ha ido editando y recortando palabras clave. Google Scholar siempre mostraba el mayor número de resultados dado el gran volumen de artículos que contiene, aunque no muchos sean de calidad, es por ello que se han tenido más en cuenta los resultados de Scopus e IEEE Xplore a la hora de crear la cadena de búsqueda.

Tras varias pruebas se ha llegado a una nueva cadena con resultados mucho menores:

(("machine learning" OR "deep learning") AND ("bin packing problem" OR "4d packing problem") OR ("knapsack problem" OR "4D knapsack problem") AND ("package"))

Arrojando **288** resultados en Scopus y **131** en IEEE Xplore. En un primer vistazo a estos resultados, el título indicaba que solo unos pocos se centraban en el problema de la mochila, mientras que el resto, aunque mencionaran el problema, lo trataban como base a partir de la cual poder realizar un estudio o método nuevo de relocalización de recursos en diferentes ámbitos (Redes, tareas computacionales en la nube, eficiencia energética, problema del viajante...)

Puesto que lo interesante en el trabajo es encontrar lo relacionado con la optimización de bultos en cajas preparadas para envíos, al final se redujo la cadena de búsqueda a sólo tres términos, pero que son los puntos clave para responder a las preguntas de mapeo.

Cadena final:

("optimization algorithms" AND "packaging" AND "bin packing problem")

Con esta cadena simplificada, y con los puntos clave a responder, se han encontrado **37** resultados en Scopus y **26** en IEEE Xplore. Para este último, se ha tenido que modificar la sintaxis de acuerdo a su propio motor de búsqueda tal que:

(("All Metadata":optimization algorithms) AND ("All Metadata":packaging) AND ("All Metadata":bin packing problem))

4.1.4. Criterios de inclusión y exclusión

Después de definir tanto las fuentes como las cadenas de búsqueda, es necesario definir algunos criterios que permitan decidir qué estudios desechar basándose en los metadatos, título y abstract. De esta manera se buscará acotar aún más el número de artículos.

Criterios de inclusión:

- Los artículos deben ser relevantes al tema de optimización del problema de la mochila, preferiblemente en contextos logísticos o de almacén
- Las investigaciones que tengan aprendizaje automático o estén enfocados a la optimización con sistemas inteligentes aplicados al problema
- Debe ser el problema de la mochila en al menos 3 dimensiones
- Fecha de publicación posterior a 2015
- Debe estar en inglés o español

- Publicado en fuentes fiables y revisadas

Criterios de exclusión:

- Artículos no relevantes al tema de optimización del problema de la mochila, o en contextos diferentes, a logísticos o de almacén
- Las investigaciones que no tengan aprendizaje automático o no estén enfocados a la optimización con sistemas inteligentes aplicados al problema
- Problema de la mochila en 1 o 2 dimensiones
- Fecha de publicación anterior a 2015
- No está en inglés o español
- Publicado en fuentes de dudosa credibilidad

4.1.5. Criterios de calidad

El último paso en esta primera fase de planificación en el mapeo, será definir cómo extraer la información de los artículos seleccionados, para ello, se han creado cuatro preguntas de calidad:

- ¿Están claros los objetivos de la investigación?
- ¿Proporciona el estudio resultados prácticos que respalden las conclusiones?
- ¿Compara el estudio, los resultados con soluciones tradicionales?
- ¿Utiliza el estudio alguna de las técnicas presentes en las palabras clave? (*Machine learning, deep learning, algoritmia*)

Estas preguntas tendrán tres posibles respuestas: Sí, Parcialmente, No.

Ponderarán 1, 0.5 y 0 respectivamente, creando una puntuación máxima de 4. Si los artículos extraídos superan la puntuación de corte establecida en 2.5, serán seleccionados como artículos válidos.

4.2. Fase de filtrado

Tras haber realizado las consultas pertinentes en cada portal, se guardarán los resultados en formato *bibtex* y se importarán a **Parsifal** (Figura 5). Para las consultas se aplicaron los diversos criterios expuestos anteriormente. Tanto Scopus como IEEE Xplore permitieron una exportación sencilla de los resultados al formato que admite Parsifal, pero para Google Scholar se ha tenido que usar una herramienta externa de ayuda llamada **Public or Perish**, debido a que Scholar no permite la

exportación a *bibtex* de manera nativa. Utilizando la herramienta, se ha ejecutado la misma consulta con el filtro de intervalo de años, y ha exportado los resultados a *bibtex*.

Una vez añadidos los resultados a Parsifal se obtuvo la siguiente lista de estudios importados con la que trabajar.

Figura 5: Estudios importados

De esta colección de 193 trabajos, se eliminaron los duplicados de las diferentes fuentes, para ello se utilizó la herramienta *Find Duplicates* que proporciona Parsifal, marcando uno a uno los posibles duplicados, tal y como se ve en la Figura 6.

Se eliminaron un total de 11 duplicados. Asimismo, como Google Scholar no permitía filtrar por idioma, en un primer vistazo se vieron títulos tanto en turco como en ruso, por lo que se eliminaron también.

Tras el filtrado inicial de los artículos mediante la metodología PRISMA [21] para la extracción de datos, se ha conseguido reducir el número de publicaciones enormemente. De los 193 originales, se quitaron los 14 mencionados anteriormente, 11 duplicados y 3 porque estaban en distintos idiomas, quedando así 179 artículos sobre los que se aplicarían los criterios de inclusión y exclusión.

Aplicando estos criterios, se descartaron un total de 149 registros, ya que o bien simplemente mencionaban el tema que interesa en citas, o no tenían nada que ver. También había un número alto de publicaciones que aunque abordaran el tema en una sección, no era el foco principal por lo que tampoco aportaban mucho conocimiento.

El número de registros tras los criterios de inclusión y exclusión quedaría entonces reducido a 30, sobre los cuales se aplicarían los criterios de calidad establecidos. Al aplicar los criterios de calidad, restarían 15 artículos finales.

La Figura 7 muestra el diagrama de flujo PRISMA que detalla el proceso seguido en las diferentes fases de la metodología y el resultado final de la extracción de información.

Por lo tanto, se pasó de 274 artículos encontrados inicialmente a una lista final de 15. La herramienta Parsifal proporcionó un par de ilustraciones que mostrarían la repartición de artículos según las fuentes, y cuántos fueron seleccionados de cada

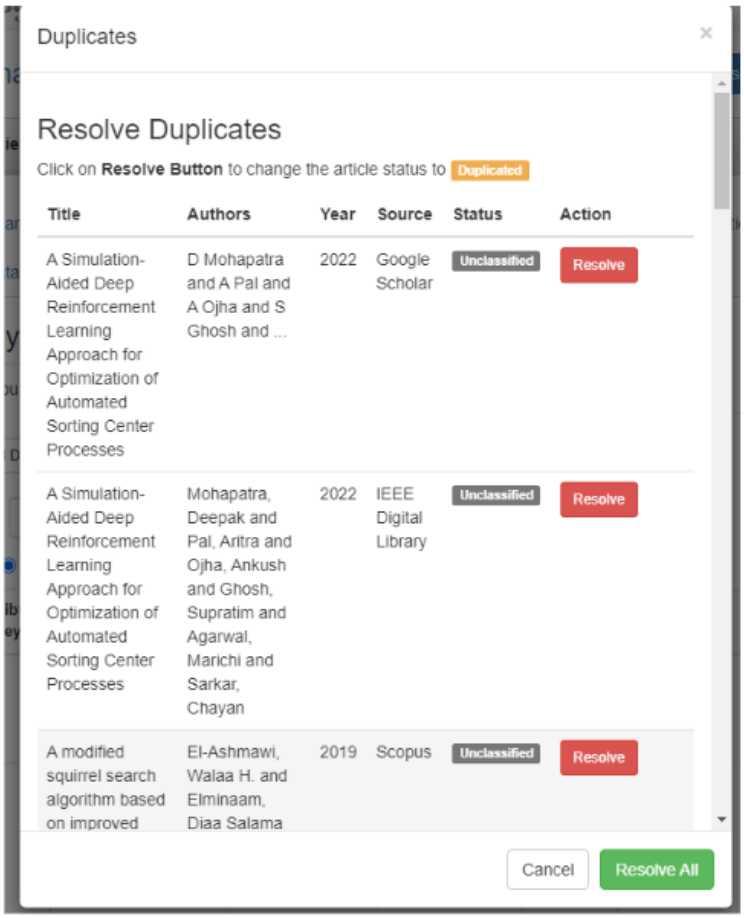

Figura 6: Estudios duplicados

una.

Tanto en la Figura 8 como la Figura 9 se puede observar que la mayor parte de los artículos fueron encontrados en Google Scholar, habiendo aceptado 7 de 24 en Scopus, 5 de 22 en IEEE Xplore y 18 de 133 en Google Scholar.

Como se mencionó anteriormente en 4.1.3, Google Scholar cuenta con mucha cantidad, pero en proporción de artículos aceptados, es la menor debido a que no cuenta con criterios demasiado estrictos de calidad como el resto. Estos treinta artículos se sometieron a las preguntas de calidad enunciadas en 4.1.5.Tras estos criterios, se aceptaron los 15 artículos finales.

En la Figura 10 se ilustra el número de artículos por año de publicación, indicando que la mayoría de artículos fueron publicados en 2022. Una posible causa de esto fue el aumento que ha habido en estos últimos años en el e-commerce tanto por las nuevas tecnologías como por el auge de comercio electrónico tras la pandemia, creando una necesidad de más envíos y por tanto, más paquetes. De esta manera las empresas buscaron formas de abaratar costes en los envíos agrupando el mayor número de bultos posibles en una sola caja, y esto se revela en el crecimiento de estudios científicos desde 2020 hasta 2022.

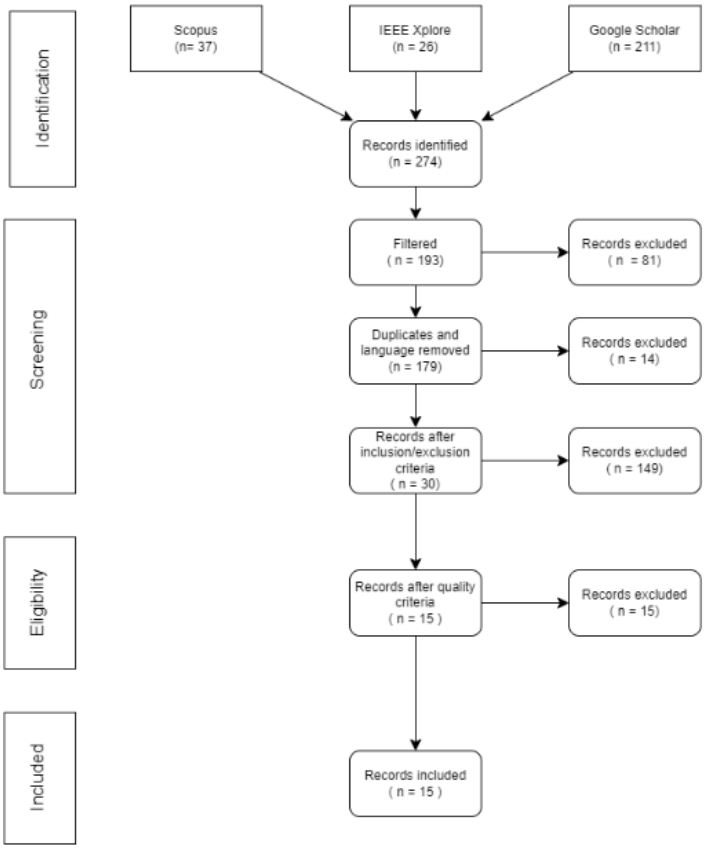

Figura 7: Diagrama de flujo PRISMA

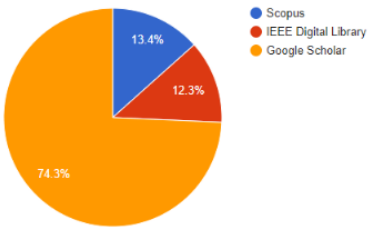

Figura 8: Repartición de artículos según fuente

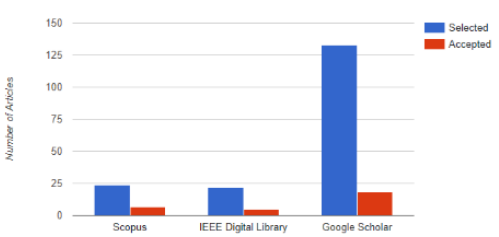

Figura 9: Artículos aceptados por fuente

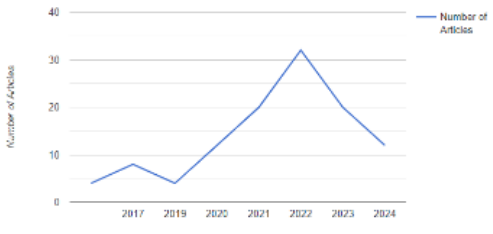

Figura 10: Artículos finales por año

4.3. Resultados

Una vez aplicados todos los criterios de aceptación de artículos, se procedió a revisar los artículos restantes en profundidad para evaluar el estado actual de la cuestión y poder responder a las preguntas de mapeo formuladas en primer lugar.

Como se ha ido observando, la investigación sobre el campo de optimización del proceso de empaquetado ha ganado importancia en los últimos años. El problema de la mochila, en particular, ha sido un pilar fundamental de estas publicaciones, ya que representa un desafío intrínseco en la maximización de recursos dentro de unos límites establecidos. En total durante esta revisión se analizaron 15 artículos estrechamente relacionados entre sí, pero cada uno con un enfoque diferente al tema principal.

Los artículos seleccionados exploran la implementación de diversas técnicas de optimización, desde algoritmos básicos y heurísticas hasta técnicas de aprendizaje profundo por refuerzo, con el fin de buscar la eficiencia en la asignación de espacio y recursos.

Por ejemplo, en [27] los autores proponen un marco que integra ambos enfoques, desarrollando tres heurísticas: Heurística de Física, Heurística de Empaquetado y Heurística de Desempaquetado, diseñadas para incorporar la experiencia de los empleados humanos encargados de empaquetar y las reglas de estabilidad física. Estas heurísticas se integran en un marco de aprendizaje por refuerzo profundo (Deep Reinforcment Learning, o DRL) para mejorar la estrategia de empaquetado y la utilización del espacio. La Heurística de Física asegura la estabilidad física de los bultos, evitando colocaciones inestables, mientras que las Heurísticas de Empaquetado y Desempaquetado aprovechan las estrategias humanas para un empaquetado eficiente y corrección de errores. Los experimentos mostraron que esta combinación de heurísticas y DRL logró un rendimiento superior en comparación con los métodos previos, resultando en un uso más eficiente del espacio y estrategias de empaquetado confiables.

En [1] se usan múltiples heurísticas y restricciones tales como la forma, el peso, orientación, centro de masa para obtener estabilidad, restricciones de apilado o soporte. También han utilizado *machine learning* DRL y han hecho un extenso *benchmarking* con 7 algoritmos diferentes del 3D-Packing Problem:

- *Heuristic block-loading algorithm based on multi-layer search* (HBMLS) [29].
- *Wall generation meta-heuristic algorithm* (WGMA) [4].
- *Container loading by tree search algorithm* (CLTRS) [8].
- *Variable neighborhood search* (VNS) [22].
- *Beam search algorithm* (BSG-VCS) [2].
- *Greedy random adaptive search* (GRASP) [19].

- *Hybrid genetic algorithm* (HGA) [3].

Además de los diferentes algoritmos, también se comprobó con numerosos bultos, calculando los niveles de entropía, etc. De esta manera han logrado un *benchmarking* extensísimo de todos los diferentes algoritmos. La contraparte del algoritmo propuesto por los autores es que aunque ofrezca soluciones eficientes, puede tener dificultades para pedidos extremadamente grandes y heterogéneos, es decir, con bultos muy diferentes entre sí. Al igual que los dos anteriores, [28] utiliza DRL estableciendo como valor de recompensa del sistema la experiencia de un trabajador humano, solo que en este caso el algoritmo lo ejecuta un brazo mecánico en combinación con un robot transportador. Además ha comparado sus resultados con otros algoritmos de empaquetado como *Random Policy*, *First Fit*, *Column Building*, *Floor Building* o *Extreme Point*.

[16] Va más allá, proponen un sistema automatizado de empaquetado basado en IoT, de manera que los componentes se comunican entre ellos. Mediante diferentes sensores se detectan los productos a empaquetar, sabiéndolos a priori, se seleccionan las cajas adecuadas y un brazo robótico insertará los objetos en las cajas. Para que el robot pueda empaquetarlo han tenido que desarrollar un algoritmo adaptativo que tiene en cuenta los objetos y sus tamaños para empaquetarlos de la mejor manera posible. Consiguiendo un sistema compuesto de una aplicación móvil para realizar pedidos, un robot de entrega y un robot manipulador. La aplicación móvil se conecta a un servidor que crea el pedido, el robot manipulador lo empaqueta y lo coloca en el robot de entrega para su partida al punto final.

Los artículos que también han realizado experimentos con un brazo robótico son [11], [31] y [32].

[31] Los objetos han de ser empaquetados inmediatamente según lleguen de la cinta, sin reajuste ni *buffering*, teniendo en cuenta la estabilidad del conjunto además se propone un método de aprendizaje por refuerzo de fácil implementación. [32] Es el único robot que en lugar de colocar desde una posición cenital, lo hace desde un lateral, de esta manera ha de tener en cuenta que según vaya colocando bultos, se va restringiendo su propio movimiento, ya que podría tirarlos. Por otra parte, [11] utiliza un sensor RGB-d. Este sensor combinará imágenes RGB con un sensor de profundidad (*Depth*) de manera que casarán píxel a píxel los datos obtenidos en cada cuadro de la cámara. Así se detectan en tiempo real los objetos y su posición relativa al resto, con esta información se calcula la posición y la orientación óptima dentro de la caja.

Tanto [7] como [12] se salen de los algoritmos preestablecidos que han utilizado la mayoría, en el primer caso utilizan Squirrel Search Algorithm para la optimización y lo comparan contra los clásicos, mientras que en el segundo caso realizan una implementación con un árbol de búsqueda ternario. Este último obtiene buenos resultados con 3 dimensiones, pero si aumentan, el rendimiento va decrementando.

Otros enfoques un poco más peculiares del mismo problema pueden ser los siguientes:

- En [34] donde investiga el BPP con formas deformables o con cambios de volumen dinámicos, como pueden ser los alimentos frescos, ya que a la hora de empaquetar se asume que no están rígidos completamente. Dado el tiempo necesario, el algoritmo propuesto mejora la utilización del espacio alrededor del 8-28 %, enseñando que aún hay gran mejora por hacer en el campo de optimización.

- En [23] proponen un método que cuenta cientos de objetos en un tiempo corto, probando que puede ser útil para entornos industriales donde hay una gran cantidad de objetos, pero de tipos limitados. Es decir, muchos objetos iguales. Definen los dos casos extremos, el *bin-packing*, donde el número de objetos no puede sobrepasar un valor, y el *bin-covering*, donde el número de objetos no puede ser inferior a un valor. Si no se da ninguno de los dos casos, fusionan una equivalencia de clases que provee una solución óptima reduciendo el tiempo de computación necesario.

- En [14] se aplica BPP a palés de cargo utilizados en aviones, demostrando así que este problema de optimización no solo es aplicable a cajas genéricas, sino que contempla un espectro mucho más amplio.

- En [33] se estudia la posibilidad de empaquetar objetos circulares dentro de otros contenedores circulares a larga escala.

- En [15] se realiza un estudio más general de los patrones geométricos, estudia tanto el *packing* como el hecho de cortar patrones como podría ser en la industria textil, donde el espacio no usado se contabiliza como pérdidas netas, al igual que en la industria logística si se desperdicia espacio en las cajas contenedoras.

Finalmente, [35] es el artículo más orientado a la concienciación y reducción de residuos, centrándose tanto en la optimización del empaquetado como en la optimización de las rutas encargadas de entregar los paquetes, haciendo un estudio del proceso completo.

Una vez revisados los artículos seleccionados, se puede observar la amplia variedad de técnicas y enfoques que se han utilizado para abordar el problema de optimización en el proceso de empaquetado. Tras haber analizado el conjunto, se puede responder a las preguntas de mapeo realizadas al principio del proceso en 4.1.1.

4.3.1. ¿Qué tipo de algoritmos de optimización se han usado en la industria logística para mejorar el proceso de empaquetado?

Los estudios incluyen una variedad de algoritmos y técnicas de optimización. Desde algoritmos heurísticos y metaheurísticos, hasta métodos de aprendizaje profundo por refuerzo (DRL). Además, otros enfoques incluyen sistemas basados en

IoT, métodos de clasificación de patrones geométricos, algoritmos para empaquetado de formas deformables, y técnicas especializadas para casos concretos como los palés de aviones para empaquetado en palés de cargo en aviones.

4.3.2. ¿Cómo de efectivas son estas técnicas de optimización comparadas con soluciones convencionales en los sistemas logísticos?

La mayoría de los estudios comparan las técnicas de optimización propuestas con métodos convencionales, como heurísticas de primera colocación (*First Fit*) y construcción de columnas (*Column Building*) al ser de los más básicos y extendidos. Los resultados indican una mejora significativa en la eficiencia del empaquetado con el uso de técnicas avanzadas. Por ejemplo, la combinación de heurísticas específicas con DRL logró mejorar el uso del espacio en comparación con métodos tradicionales. En otros casos más especializados, como el algoritmo *Ternary Search Tree* y el *Squirrel Search Algorithm*, demostraron una mejora significativa frente a algoritmos clásicos mientras que otros, mejoraban para muchos artículos homogéneos, pero empeoraban si eran pocos artículos heterogéneos.

4.3.3. ¿De qué manera estas soluciones influyen en la eficiencia y producción de los negocios?

Las soluciones inteligentes para el problema del empaquetado han demostrado influir positivamente en la eficiencia de los negocios. Por ejemplo, en [16], la integración de IoT y sistemas automatizados redujo significativamente los tiempos de manipulación y empaquetado. Otros estudios mostraron mejoras en la utilización del espacio [27],[1], [14] y una reducción de errores mediante sistemas robóticos y sensores [11]. Además, el empaquetado óptimo resultó en una disminución de los residuos y un mejor aprovechamiento de la capacidad de transporte [35].

4.3.4. ¿Qué desafíos y limitaciones están asociadas con la aplicación de soluciones inteligentes en las operaciones de almacén y logística del mundo real?

A pesar de los avances, existen desafíos clave para implementar soluciones inteligentes en escenarios logísticos reales. El algoritmo propuesto en [1] tiene dificultades con pedidos extremadamente grandes y heterogéneos, mientras que el enfoque de DRL [28] requiere un ajuste cuidadoso de los parámetros para evitar comportamientos subóptimos. Los sistemas robóticos enfrentan retos en la manipulación de objetos inestables o frágiles [32], y la integración de IoT puede ser costosa para algunas empresas [16]. Además, los enfoques basados en DRL requieren conjuntos de datos extensos para el entrenamiento, lo cual puede ser una limitación para su implementación a gran escala [27].

El problema de la mochila, aplicado al empaquetado en la industria logística, ha sido abordado desde múltiples perspectivas, proporcionando soluciones cada vez

más efectivas y eficientes. Sin embargo, las soluciones inteligentes presentan desafíos prácticos en su implementación a gran escala, como la heterogeneidad de pedidos, la inestabilidad de objetos y la necesidad de datos extensos para el entrenamiento. A pesar de estas limitaciones, los avances en algoritmos heurísticos, metaheurísticos y técnicas de aprendizaje automático continúan demostrando mejoras significativas en la eficiencia del empaquetado, influyendo positivamente en la productividad y sostenibilidad de las operaciones logísticas.

5. Métodos y algoritmos de empaquetado

Una vez establecido el marco teórico y el estado del arte actual sobre la optimización en procesos de empaquetado, podemos realizar experimentación con diferentes algoritmos y así poder evaluar las soluciones a casos específicos. Para ello, en primer lugar se escogerá una variedad representativa de los algoritmos de empaquetado, incluyendo heurísticas observadas en el estado del arte.

Posteriormente, se definirán las métricas de evaluación para poder determinar el desempeño de cada algoritmo bajo un conjunto de casos de prueba que representen diferentes escenarios y tipos de desafíos.

En último lugar, se analizarán los datos recopilados y se determinará que algoritmos ofrecen el mejor rendimiento en diferentes condiciones.

5.1. Conceptos básicos

Volviendo al problema definido en 3.2, podemos adaptar el problema de la mochila al contexto de un almacén para que los operarios puedan llevar a cabo un empaquetado óptimo, mejorando la eficiencia operativa: *Dado un conjunto de objetos de cierto tamaño y un conjunto de contenedores de cierto volumen, ¿es posible colocar todos los objetos en los contenedores?*

Como indica [18], muchos algoritmos de aproximación han sido propuestos para encontrar soluciones subóptimas en un tiempo de respuesta corto. Esta medida se toma en la mayoría de casos como un intercambio entre la calidad de la solución y el tiempo que se tarda en resolver. Si situamos un bulto de seis facetas rectangulares en el espacio, se tendrán tres facetas distintas dado que las caras opuestas serán idénticas. Cada una de las tres caras podrán ser rotadas ortogonalmente, dando así seis posiciones posibles en el espacio (Figura 11). Esta característica aumentará exponencialmente el número de posiciones posibles según entran más bultos al cálculo, ya que incrementan el orden y orientación entre ellas. Si se tuvieran en cuenta bultos con formas geométricas diferentes a un hexaedro, el cálculo se complicaría aún más.

Es por ello que buscar la mejor solución en este problema necesitaría explorar todas las soluciones posibles, llevando así una carga de cómputo alta. En los algoritmos propuestos en los diferentes estudios, se asume que no va a ser la solución óptima, sino que serán soluciones *suficientemente buenas* de manera que sean factibles para

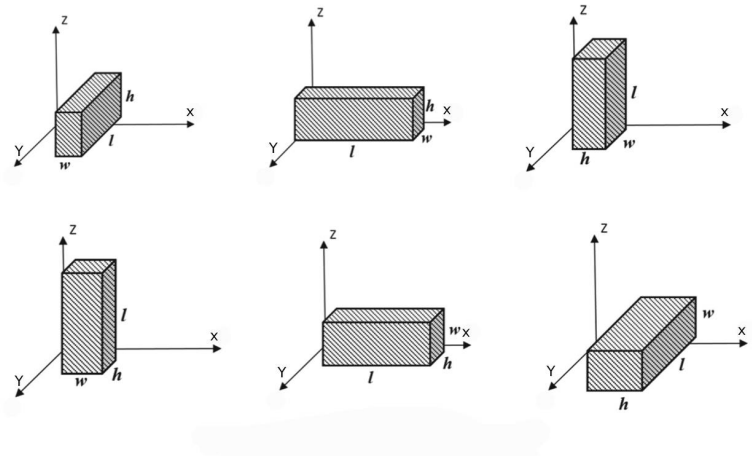

Figura 11: Combinaciones de colocación

los operarios (que no sea una colocación tediosa), factible para los objetos (no colocar un objeto más pesado encima de uno más ligero) y factible para la unidad de procesamiento encargada de hacer el cálculo en un tiempo razonable.

Las soluciones dependerán de la finalidad de la optimización, clasificándose en tres casos según el contenedor:

1. Contenedor único: El objetivo será empaquetar el mayor número de objetos posible dentro de un único contenedor, minimizando el espacio libre.

2. Múltiples contenedores: Se colocarían los objetos en un conjunto de contenedores con objetivos tales como maximizar la ocupación de los contenedores, minimizar el número de contenedores, optimizar costes de envío según el transportista.

3. Contenedores personalizados: En este caso, los contenedores tendrán una lista de restricciones que incluirán diferentes variables como puede ser el peso o casos más específicos como no juntar objetos inflamables.

Por ejemplo, para el segundo caso, si suponemos que el transportista establece una tarifa sobre la base del peso, puede ser interesante establecer restricciones en el optimizado tal que se maximice el espacio utilizado en un contenedor ajustándose a un peso límite, siendo más barato hacer dos envíos ligeros respecto a un envío pesado (Figura 12).

Siguiendo la línea del tercer caso, las diferentes técnicas de empaquetado establecerán ciertas restricciones o limitaciones a tener en cuenta para que el cálculo sea aplicable a la realidad. Algunos ejemplos pueden ser:

- **Anti-Solapamiento**: La restricción más básica, en el cálculo de la posición de

[1]Rubulis , M. (2023) 3D Bin Packing: The Tetris of Logistics, Optioryx. Available at: https://blog.optioryx.com/3d-bin-packing.

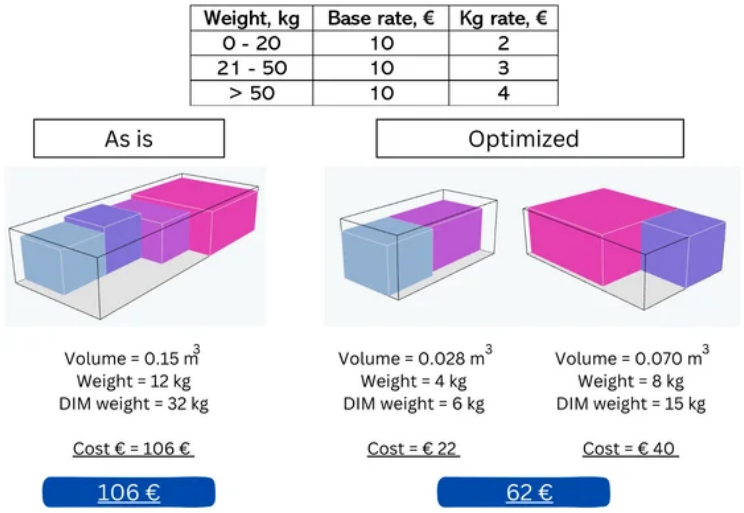

Figura 12: Optimización según el coste por peso[1]

los bultos, un bulto no podrá solaparse con otro. Una restricción físicamente imposible, pero que al ser un cálculo abstracto podría pasarse por alto.

- **Limite de peso**: Cada contenedor puede tener un peso límite que no ha de ser excedido, ya sea por razones físicas (el contenedor no puede soportarlo) o por razones logísticas como la dificultad de transportarlo o el aumento de coste visto anteriormente.
- **Estabilidad del contenedor**: Cada bulto tendrá que ser colocado teniendo en cuenta la estabilidad del contenedor. El peso tendrá que ser distribuido equitativamente sobre la superficie. Si un lado pesa más que otro podría ser peligroso tanto para los artículos como para el transporte.
- **Puntos de soporte**: Cada artículo tendrá unas restricciones de apoyo de tal manera que el artículo sea estable a lo largo del tiempo. Según [1], esta restricción se cumple si se da alguna de las siguientes condiciones:
  - Al menos el 40 % de la base de un objeto y sus cuatro vértices son soportados por otros objetos o,
  - un mínimo del 50 % de la base es soportada o,
  - un 75 % o más de la base del objeto y 2 vértices son soportados.
- **Número de contenedores disponibles**: El número de contenedores disponibles también será una restricción aplicable al problema. De esta manera puede que haya objetos que no entren en el espacio dado y se queden fuera del envío. Habitualmente el número de contenedores disponibles tenderá a infinito para la simplificación del cálculo.
- **Recolocación**: Otra restricción habitual en los algoritmos y heurísticas comúnmente utilizados es la recolocación, o mejor dicho, la no-recolocación. Una vez que un objeto haya sido colocado en el contenedor, no se moverá más tarde.

5.2. Algoritmos seleccionados

En la literatura actual se puede encontrar diversidad de métodos tanto algorítmicos como heurísticos. Cabe destacar que en 4.3 se vio que los algoritmos más sencillos de implementar y comunes pueden ser *First-fit, Column-Building o Layer-building*. Estos algoritmos serán la base de la que partirán la mayoría de los casos de estudio, por lo que se hace una introducción para poder desarrollarlos en los algoritmos estudiados.

First-fit es el algoritmo más básico de este problema, su definición y un ejemplo ha sido visto en 3.2. A modo de recordatorio, simplemente escogerá un objeto y lo colocará en el primer contenedor que tenga espacio disponible suficiente, si no hay espacio suficiente, pasará al siguiente contenedor.

Column-Building es una técnica que busca maximizar la utilización del espacio vertical mediante la construcción de columnas, apilando los objetos. La definición viene dada por [17].

Dado un contenedor con dimensiones $L \times W \times H$ y una lista de objetos ordenados SI, el algoritmo expuesto se utiliza para insertar los objetos de SI en contenedores utilizando el concepto de Puntos Extremos (EP). Se inicializa EP con el punto $(0, 0, 0)$. Los Espacios Residuales (RS), que son todos los espacios residuales, se inicializan con (L, W, H), que es el RS de $(0, 0, 0)$ en los ejes X, Y, Z respectivamente. Un espacio vacío alrededor de un EP es denominado espacio residual (RS). La distancia entre un EP y el lado del contenedor a lo largo del eje es el RS del EP en ese mismo eje. Cada objeto empaquetado y su posición dentro del contenedor se guardan en un conjunto denominado PI.

Para cada objeto $i \in SI$, el procedimiento *PonerArtículoEnContenedor* examina los EP desde el principio hasta el final para identificar un EP, ep, para determinar dónde colocar el objeto i dentro del contenedor. Un objeto puede colocarse en un EP si el emplazamiento del mismo no se superpone con otros objetos ya empaquetados dentro del contenedor. Dado que un objeto puede ser colocado en seis orientaciones diferentes (Figura 11), se utiliza el siguiente proceso para seleccionar la orientación del objeto.

Supongamos que (L_{ep}, W_{ep}, H_{ep}) es el espacio residual de ep. Para seleccionar la orientación del objeto i, se escoge el valor mínimo del conjunto $\{(L_{ep} - l_i), (L_{ep} - w_i), (L_{ep} - h_i), (W_{ep} - l_i), (W_{ep} - w_i), (W_{ep} - h_i), (H_{ep} - l_i), (H_{ep} - w_i), (H_{ep} - h_i)\}$. Si por ejemplo $(H_{ep} - l_i)$ es el valor mínimo mencionado, significa que la longitud del objeto debe estar a lo largo del eje Z. Entonces, el valor mínimo del conjunto $\{(L_{ep} - w_i), (L_{ep} - h_i), (W_{ep} - w_i), (W_{ep} - h_i)\}$ tiene la rotación del objeto i en los ejes X e Y.

Dadas las posiciones de los objetos ya empaquetados en el contenedor, el nuevo objeto i y ep, el procedimiento *ActualizarEP* genera nuevos EP y los inserta en los EP. El RS de los EP se actualiza por el procedimiento *ActualizarEspacioResidual*. Los objetos se agregan al contenedor mientras haya espacio disponible. Todos los patrones de carga factibles con un costo reducido negativo se agregan al conjunto

all-negativePacking.

Entrada: SI, (L, W, H)

Salida: *all-negativePacking*

1. Inicializar el contenedor con dimensiones $L \times W \times H$
2. $EPs \leftarrow \{(0, 0, 0)\}$
3. $RSs \leftarrow \{(L, W, H)\}$
4. $PI \leftarrow \emptyset$
5. Para cada artículo $i \in SI$ hacer:
   a) Si *PutItemInBin*(i, RSs, EPs) entonces
      1) UpdateEPs(PI, i, EPs)
      2) UpdateResidualSpacel(i, EPs)
      3) Remover el objeto i de SI y añadirlo a PI
   b) Si la suma del costo reducido de los objetos en PI es negativa entonces
      1) Añadir PI a *all-negativePacking*
   c) Si no hay suficiente capacidad residual para empaquetar más objetos entonces
      1) Inicializar otro contenedor y regresar al paso 1
6. Devolver *all-negativePacking*

La técnica contraria sería **Layer-Building**, centrado en la construcción de capas horizontales. Cada capa se construye para cubrir la mayor área posible dentro del contenedor, minimizando los huecos y la altura del contenedor. Una definición más formal de este algoritmo es la vista en [25] tal que:

S0: $k = 1$, $\pi_1 = 1$.

S1: Si $\pi_k > m$ entonces establece $t = k - 1$ — stop.

S2: Calcula el indice máximo $j(\leq m)$ tal que $\sum_{i=\pi_k}^{j} w_i \leq 1$.
Establece $r := \pi_k$, $s := j$.

S3: Sea $I_k = \{i : \pi_k \leq i \leq j\}$. Utiliza el algoritmo de empaquetado en dos dimensiones para colocar (empaquetar) los rectángulos $l_i \times w_i$, $i \in I_k$, dentro del cuadrado unitario.

S4: Si no todas las piezas de I_k están colocadas entonces establece $s := j$, $j := \left[\frac{r+j}{2}\right]$, si $j > r$ entonces vuelve a S3, sino ve a S6.

S5: Si todas las piezas de I_k están colocadas entonces establece $r := j$, $j := \left[\frac{i+s+1}{2}\right]$, Si $j < s$ entonces vuelve a S3.

S6: Se obtiene una nueva capa: establece $H_k := h_{\pi_k}$, $k := k + 1$, $\pi_k := r + 1$, vuelve a S1.

En el paso 3 del algoritmo, los objetos se colocan respecto a un preorden asumido. Es esencial que las piezas restantes no empaquetadas tengan alturas menores que las alturas ya empaquetadas. El algoritmo de empaquetado en dos dimensiones (A) utilizado en el paso 3, puede ser un algoritmo de empaquetado por sí solo o uno bidimensional más general. Para calcular el coste computacional del algoritmo, denotamos con $\alpha(A)$ el coste computacional del algoritmo A para colocar m objetos en una tira de longitud 1 (límite de peor caso del algoritmo A). La estrategia de bisección definida por los pasos 3, 4 y 5 produce un factor multiplicativo de a lo sumo $\ln(m)$ de esfuerzo computacional. El número de capas es como máximo m. Por lo tanto, el coste total del algoritmo está acotado por $O(m \cdot \ln(m) \cdot \alpha(A))$. Si el algoritmo A es polinómico, entonces el algoritmo *LayerBuilding* también lo es.

Esta última técnica suele ser la base sobre la que parten el resto de técnicas más especializadas, ya que cuenta con una implementación relativamente sencilla que además cumple una de las restricciones más comunes: el reparto de peso equitativo en el contenedor al ir por capas.

A continuación se verán seis diferentes técnicas de empaquetado con su implementación y casos de estudio.

5.2.1. 3D Best-Fit con pivote

Este algoritmo se centrará en colocar cada objeto de manera que ocupe el mínimo espacio disponible en el contenedor que mejor se adapte, minimizando así el volumen desperdiciado. El funcionamiento según [6] es el siguiente:

1. Dirección de empaquetado
    - El algoritmo decide inicialmente una dirección de empaquetado, puede ser la dimensión de ancho (x), altura (y) o profundidad (z) del contenedor.
2. Selección de pivote
    - Se elige un punto de pivote dentro del contenedor. Esta coordenada (x,y,z) será donde se intentará colocar el objeto.
    - El punto de pivote inicial en un contenedor vacío será (0,0,0), correspondiente a la esquina trasera inferior izquierda (por convenio).
3. Colocación y Rotación del Objeto
    - El objeto se intenta colocar en el punto pivote seleccionado. Si no cabe en la orientación inicial, se rota.
    - Se prueba cada rotación hasta encontrar una que permita que el objeto encaje o hasta que se agoten las rotaciones (Recordemos que había 6 posibles, Figura 11).

- Si no es posible colocar el objeto después de todas las rotaciones, se pasa al siguiente objeto. El objeto que no se pudo empaquetar se añade a una lista de objetos pendientes.

4. Siguiente objeto

   - Se actualiza el pivote según la nueva distribución del espacio dentro del contenedor
   - Se repiten los pasos del algoritmo hasta que no haya más objetos iniciales. Cuando estén colocados todos los objetos iniciales, se intentarán colocar los objetos de la lista de pendientes.

Al haber partido de la base de Layer-building, este algoritmo no tiene en cuenta la restricción de la gravedad. Es decir, construirá la primera capa en la base del contenedor y la siguiente capa tendrá como base el punto más alto de la primera capa. Si se representa visualmente podremos ver que hay objetos con la apariencia de que estén flotando, puesto que las coordenadas tienen como base el punto más alto de la capa anterior. La solución es la implementación de la restricción de puntos de soporte (Figura 13).

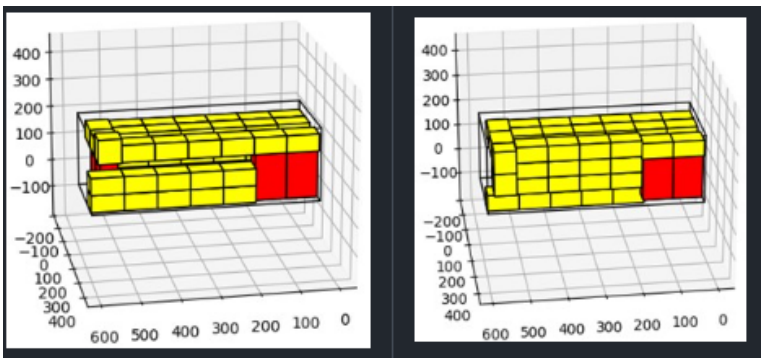

Figura 13: Comparativa aplicando la restricción de puntos de soporte

5.2.2. Largest Area First-Fit

El algoritmo Largest Area First-Fit (LAFF), propuesto en [20], se centra en colocar los objetos con mayor área de superficie primero, minimizando la altura que ocupan, desde el fondo del contenedor (Figura 14).

1. Preparación inicial

   - Se deben definir las dimensiones de ancho y profundidad del contenedor, basándose en las dimensiones más largas de los objetos disponibles, asumiendo altura ilimitada.

2. Selección de Objetos

- Se seleccionan los objetos que tengan mayor superficie para ser colocados primero.
- De los objetos seleccionados, se da prioridad al que tenga menor altura para maximizar la utilización del espacio y minimizar la altura. Se establece esta altura como "tope de nivel".

3. Métodos de colocación

   - **Primer método**: Los objetos seleccionados se colocan de manera que su superficie más grande esté paralela al fondo del contenedor, comenzando desde el punto más bajo posible y con altura máxima igual al tope de nivel.
   - **Segundo método**: Se intentará ocupar el espacio restante alrededor del objeto ocupado por el primer método (si no ha ocupado todo), continuando sin superar la altura del tope de nivel.

4. Uso del espacio restante

   - Después de colocar un objeto, se calculará la diferencia entre la superficie del objeto y la superficie del contenedor para saber el espacio disponible en el nivel actual.
   - Si hay algún objeto que entre en el nuevo espacio disponible del nivel actual, se colocará el que más espacio ocupe. Se repite este proceso hasta que no haya más objetos o ningún objeto entre en esos espacios.
   - Si ningún objeto entra en el nivel actual, el tope de nivel pasará a ser el punto base para la siguiente iteración, comenzando con el primer método de colocación.

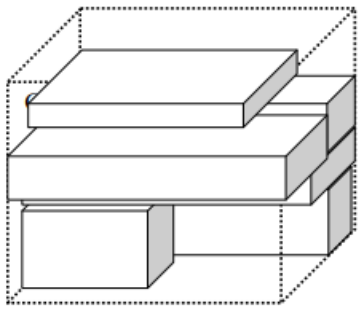

Figura 14: Posible solución del algoritmo LAFF

5.2.3. Guillotine

El algoritmo Guillotine orientado para espacios 3D, se centrará en la división del contenedor en espacios más pequeños usando cortes rectangulares similares a los de una guillotina.

1. Preparación inicial

    - Se seleccionará el contenedor inicial según una lista ordenada de contenedores disponibles.
    - Se inicializará un hexaedro (cuboideo) que ocupe todo el espacio, representando el espacio libre inicial.

2. Inserción y colocación

    - Se selecciona un objeto de la lista de objetos que se desean empaquetar en el contenedor.
    - Se evalúan todos los espacios libres y se determina la mejor posición y orientación para el objeto, de manera que se minimice el espacio no utilizado.

3. División del contenedor

    - Una vez colocado el objeto, el espacio restante se divide. Las divisiones pueden ser horizontales o verticales, creando nuevos espacios libres y actualizando las listas de espacios disponibles.
    - El *corte* se realiza de manera perpendicular a una de las dimensiones del objeto colocado.

4. Recursividad

    - El proceso se repite para los sub-contenedores resultantes después del corte. Cada sub-contenedor se trata como un nuevo contenedor para empaquetar el siguiente objeto seleccionado.
    - El algoritmo continúa empaquetando objetos hasta que todos hayan sido colocados en el contenedor inicial o hasta que no sea posible colocar más objetos.

5.2.4. Shelf + Guillotine

El algoritmo Shelf es un algoritmo de división de espacio en *estantes*. Dada su naturaleza, es un algoritmo muy eficiente para entornos de dos dimensiones. Para poder utilizarlo en tres dimensiones, se ha combinado con el algoritmo Guillotine visto en el punto anterior. De esta manera el funcionamiento sería el siguiente:

1. Preparación inicial

    - Se determinan el ancho y profundidad del contenedor. Al inicio no habrá altura máxima, permitiendo flexibilidad en el manejo de la altura a medida que se vayan agregando los objetos.

- Los objetos se organizarán en *estantes* que serán niveles dentro del contenedor, de manera similar al algoritmo ya visto LayerBuilding. Cada estante tiene una altura que comienza en la base del contenedor y se extenderá hasta la altura del objeto más alto colocado en ese nivel.

2. Selección y colocación de objetos

   - Los objetos se colocarán en los estantes basándose en su tamaño y forma, evaluando las diferentes orientaciones hasta buscar la óptima.
   - Una vez que se coloca en un estante, el espacio restante alrededor del objeto se divide utilizando el método Guillotine, haciendo cortes a lo largo del eje más largo del espacio libre restante.

3. Heurísticas de colocación

   - La elección de dónde colocar cada nuevo objeto se guiará por Best-Fit de manera similar al algoritmo visto en 5.2.1. De esta manera se decidirá si colocar un objeto en el estante actual o empezar un nuevo estante, dependiendo del espacio disponible y la altura del objeto.

4. Recursividad

   - El proceso se repite iterativamente colocando objetos uno por uno, hasta que todos estén empaquetados o hasta que no se puedan encontrar configuraciones viables para los objetos restantes.

Al combinar dos algoritmos, esta versión aprovecha las fortalezas de ambos: la eficiencia en el manejo de espacio vertical del algoritmo Shelf y la flexibilidad en el uso del espacio residual del algoritmo Guillotine. Adaptándose así a diferentes formas y tamaños de objetos.

5.2.5. Empaquetado 3D con DRL

Un enfoque diferente al problema de la mochila se estudia en [30] mediante técnicas de aprendizaje automático, en concreto aprendizaje profundo por refuerzo (DRL). Hay varios factores que hacen que DRL sea una opción adecuada para una tarea tan compleja como puede ser este problema, en concreto:

- DRL es efectivo en ambientes con dinámicas complejas y cambiantes donde las decisiones deben optimizarse sobre la marcha, como puede ser el empaquetado *online* donde a priori no se sabe el siguiente objeto a colocar. El algoritmo aprenderá de las diferentes interacciones con el entorno e irá ajustando sus estrategias para mejorar continuamente el rendimiento.
- El problema no solo implica colocar los objetos de manera eficiente, sino también respetar las diferentes restricciones de orden y estabilidad necesarias. Un sistema con aprendizaje automático puede integrar estas restricciones en su

proceso de aprendizaje, ajustando automáticamente las decisiones para cumplir los requisitos.

- DRL permite una flexibilidad superior a las heurísticas tradicionales, teniendo en cuenta diferentes tipos de contenedores o criterios de empaquetado específicos.

La implementación del modelo DRL incorpora técnicas de predicción y optimización para manejar las complejidades del problema, para ello se utiliza un modelo *actor-crítico*, donde el *actor* sugiere acciones de empaquetado según el estado actual y el *crítico* evalúa estas acciones, proporcionando *feedback* sobre la eficacia. De esta manera se tendrá un aprendizaje equilibrado entre la exploración de nuevas estrategias, y la explotación de aquellas que ya son conocidas.

Para esta evaluación, antes de tomar decisiones, el modelo predice máscaras de viabilidad que determinarán si las posiciones potenciales para los objetos son factibles. Esto reducirá considerablemente el espacio de búsqueda de acciones y permite un entrenamiento más rápido y focalizado. A su vez, el modelo empleará técnicas de optimización en tiempo real para evaluar y reordenar las secuencias de acciones, permitiendo al sistema probar diferentes configuraciones rápidamente y seleccionar la mejor según el estado actual del sistema.

De manera más detallada, el algoritmo propuesto combinará las técnicas de aprendizaje automático con las heurísticas especializadas para cumplir el empaquetado y sus restricciones.

1. Preparación inicial
    - El contenedor empieza vacío y el algoritmo recibe el conjunto inicial de objetos a empaquetar, junto con sus dimensiones y posibles restricciones (por ejemplo, algunos objetos necesitan evitar ciertas orientaciones o posiciones).
    - Se emplea una arquitectura *actor-crítico* donde el actor propone posiciones y orientaciones para colocar los objetos, y el crítico evalúa estas acciones según su viabilidad y eficacia.
2. Máscaras de Viabilidad
    - Antes de tomar una decisión de empaquetado, el modelo utiliza la red neuronal para predecir máscaras de viabilidad. Estas máscaras indican qué áreas del contenedor son adecuadas para colocar cada objeto, teniendo en cuenta las dimensiones del objeto y las restricciones del contenedor.
    - Las acciones sugeridas por el *actor* son evaluadas con estas máscaras para asegurar que solo se consideren movimientos viables, reduciendo así el espacio de búsqueda y aumentando la eficacia del proceso de decisión.
3. Decisión y colocación de objetos

- Basándose en el estado actual del contenedor y la información de las máscaras, el *actor* selecciona la mejor posición y orientación para cada objeto.
- El *crítico* evaluará la acción tomando en cuenta cómo esta acción afectará al futuro del espacio, la estabilidad del contendor y el resto de restricciones.

4. Aprendizaje con refuerzo

   - Después de cada acción, el sistema recibe una recompensa o penalización basada en varios factores como el volumen del espacio no utilizado, la estabilidad del empaquetado o el cumplimiento de las restricciones.
   - Mediante este refuerzo el modelo se ajustará continuamente mejorando las decisiones según las recompensas y penalizaciones acumuladas.

5. Optimización

   - El modelo podrá experimentar con diferentes permutaciones y secuencias de empaquetado en tiempo de ejecución para explorar diversas configuraciones y seleccionar la que mejor uso haga del espacio y que cumpla mejor las restricciones.

5.2.6. Heurísticas especializadas

En lugar de buscar unos resultados lo más óptimos posible, esta última implementación será una selección de heurísticas encadenadas que simularán un enfoque humano, de esta manera se buscará una solución suficientemente buena. De esta manera se tendrá una ejecución mucho más rápida que, por ejemplo, el caso anterior, y permitirá definir muchas más restricciones mediante nuevas heurísticas.

A un alto nivel, el algoritmo funcionará de la siguiente manera:

1. Colocar los objetos más grandes según volumen primero
2. Colocar los objetos verticalmente en el lateral del contenedor
3. Colocar un objeto al lado del anterior si cabe
4. En un primer momento se intentará posicionar los objetos por filas, primero colocándolos a lo ancho de la caja, cuando no haya más espacio, creará una nueva fila a lo largo. Sin embargo, a veces se logra un mejor resultado colocando los objetos primero a lo largo de la caja. De esta manera, el algoritmo intentará empaquetar usando ambas maneras, transponiendo los anchos y longitudes según sea necesario
5. Si se necesita más de un contenedor para colocar todos los objetos, se intentará que los contenedores sean de peso similar, repartiendo los objetos y recolocando

Durante el proceso de empaquetado se tendrán diversas variables y heurísticas adicionales para la definición de restricciones, entre ellas tendremos:

1. Rotación
   - **Keep-flat**: Para objetos que deben ser enviados horizontalmente o que deban mantener un lado hacia arriba
   - **Best-fit**: Permitirá todas las rotaciones con la finalidad de que entre en la mejor posición
   - **No-rotation**: Restringe la rotación completamente de un objeto
2. Ordenación

   El algoritmo será *online* de manera que colocará los objetos de manera secuencial, independientemente de lo que venga después. Para ello es importante que el orden de los objetos sea el adecuado antes de comenzar a colocar.
   - Se intentarán colocar los objetos más grandes o pesados al fondo del contenedor, dejando los más pequeños y ligeros por encima. De esta manera se evita que se rompa mercancía
   - Si un objeto de la lista es demasiado grande para un espacio, se omitirá temporalmente y se pasará al siguiente en la lista
   - Por defecto se intentarán usar los contenedores más pequeños para la ordenación, ya que se asume que es el que menor coste conllevará en el transporte
   - Si hay dos tipos de contenedores que pueden almacenar el mismo número de objetos diferentes, se elegirá el contenedor que almacene los objetos más grandes
3. Distribución de peso

   Puede haber casos que para un pedido concreto se necesiten varias cajas, una estará completamente ocupada y otra prácticamente vacía. Con la finalidad de distribuir el peso se hará una segunda vuelta al algoritmo una vez se haya hecho la colocación inicial. En caso de que haya varios contenedores:
   - La segunda vuelta recolocará objetos desde los contenedores más pesados a los contenedores más ligeros
   - Para la mayoría de casos los beneficios son mayores que el tiempo adicional de cálculo, con la excepción de pedidos demasiado grandes

5.3. Métricas de evaluación

Para el estudio del problema de la mochila en tres dimensiones será necesario definir una serie de métricas de evaluación y crear un conjunto de casos de estudio para asegurar que los métodos no solo sean teóricamente válidos, sino que sean soluciones plausibles.

Las métricas de evaluación serán necesarias para cuantificar el rendimiento de los algoritmos, proporcionando así una medida objetiva para comparar las diferentes estrategias y heurísticas. Cada métrica abordará una faceta diferente del problema, desde la maximización del espacio hasta la estabilidad física y operatividad logística.

- **Eficiencia del volumen**: Esta métrica se centra en medir el porcentaje del volumen V total del contenedor que es efectivamente utilizado por los n objetos empaquetados. Es fundamental porque proporciona una medida directa del aprovechamiento del espacio, un recurso limitado y a menudo costoso en los contextos logísticos o de transporte. Una mayor eficiencia de volumen se traduce directamente en costes reducidos y mayor eficiencia operativa, haciendo esta métrica esencial para evaluar la capacidad de un algoritmo para maximizar el espacio disponible. Siendo V_i el volumen de cada objeto, matemáticamente se expresará la eficiencia del volumen como:

$$\text{Eficiencia del Volumen} = \frac{\sum_{i=1}^{n} V_i}{V_{contenedor}} \times 100\,\%$$

- **Balance de peso**: Se evaluará qué tan uniformemente se distribuye el peso entre los contenedores, cuando proceda. Un balance ideal implicará que todos los contenedores tendrán un peso similar, minimizando así riesgo de desestabilización durante el transporte. Conocer la estabilidad física aporta un factor de seguridad durante el transporte de los bultos, por lo que esta métrica es especialmente relevante en industrias donde la distribución desigual de peso puede conducir a ineficiencias logísticas o incluso accidentes. Para conseguir esta métrica se realiza el siguiente procedimiento:

  1. Se divide la base del contenedor en cuatro áreas cardinales:
     - *Area_1*: Superior izquierda
     - *Area_2*: Superior derecha
     - *Area_3*: Inferior izquierda
     - *Area_4*: Inferior derecha

     Cada área se representa como un conjunto de rangos de coordenadas x e y.
  2. Para cada objeto en el contenedor, se determinan sus coordenadas iniciales (x_{st}, y_{st}) y finales (x_{ed}, y_{ed})
  3. Se generan conjuntos de rangos x e y que representan las posiciones ocupadas por el objeto
  4. Para cada objeto, se determina en qué área cardinal se encuentra
  5. Si el objeto está completamente contenido en un área, su peso total se agrega a esa área
  6. Si el objeto se solapa con más de una área, el peso se distribuye proporcionalmente entre las áreas involucradas según la fracción del objeto que cae en cada área

7. Finalmente, se suma el peso de todas las áreas y se calcula el porcentaje de peso de cada área en relación con el peso total

$$\text{Balance de peso} = \text{Peso } \textit{Area\_1}, \text{Peso } \textit{Area\_2}, \text{Peso } \textit{Area\_3}, \text{Peso } \textit{Area\_4}$$

Debido a la complejidad asociada a esta métrica, no se podrá evaluar en todos los algoritmos, pero el valor que aporta es demasiado interesante para desecharlo.

- **Tiempo del algoritmo**: El tiempo total requerido para colocar todos los objetos en los contenedores. Será una medida vital para los casos donde la optimización se realiza *in-situ* como la carga de mercancías en tiempo real. Un tiempo de cálculo rápido puede traducirse en una mayor eficiencia operativa y costes reducidos en las operaciones logísticas. Se evaluará en segundos o milisegundos.

$$T_{algoritmo}$$

- **Tasa de éxito**: Porcentaje de objetos que se han colocado con éxito en los contenedores cumpliendo las restricciones. Esta métrica evalúa la eficacia del algoritmo en cumplir con los requisitos del problema. Un alto porcentaje de éxito indica que el algoritmo es competente en manejar las restricciones impuestas y en encontrar soluciones viables.

$$\text{Tasa de Éxito} = \frac{\text{Número de objetos empaquetados}}{\text{Número total de objetos}} \times 100\,\%$$

- **Número de contenedores utilizados**: Si se utilizan varios contenedores, se evaluará la eficiencia del algoritmo en minimizar el número de contenedores necesarios. Menos contenedores implican menores gastos y una logística más sencilla, además esta métrica también refleja la eficiencia del algoritmo en términos de espacio y recursos.

$$N_{contenedores}$$

Cada una de estas métricas tiene en cuenta aspectos diferentes pero complementarios del problema de la mochila. La eficiencia del volumen y el balance de peso se centran en la utilización óptima del espacio y la distribución del peso, respectivamente. El tiempo de ejecución del algoritmo y la tasa de éxito evalúan la eficacia y eficiencia computacional del algoritmo, mientras que el número de contenedores utilizados aborda la eficiencia logística y posibles costes asociados al transporte. Juntas, estas métricas proporcionan una evaluación integral del rendimiento de los algoritmos, asegurando que no solo sean teóricamente válidos, sino también prácticos y aplicables en escenarios del mundo real.

5.4. Casos de estudio

Se considerarán diferentes combinaciones y configuraciones de objetos y contenedores para probar la adaptabilidad de los algoritmos. Se probarán desde escenarios básicos que involucren pocos objetos hasta situaciones más complejas con múltiples contenedores, pesos variados y restricciones de orientación.

1. Caso básico
   - **Contenedor**: 100x50x50 cm
   - **Objetos**:
     - Objeto A: 10x10x10 cm, 10 unidades
     - Objeto B: 20x20x15 cm, 5 unidades
     - Objeto C: 30x30x30 cm, 4 unidades
   - **Algoritmos**: 3D Best-Fit, LAFF, Guillotine, Shelf + Guillotine, Empaquetado 3D con DRL, Heurísticas especializadas
2. Caso alta densidad
   - **Contenedor**: 120x120x100 cm
   - **Objetos**:
     - Objeto A: 60x60x60 cm, 1 unidad
     - Objeto B: 30x30x30 cm, 10 unidades
     - Objeto C: 15x15x15 cm, 20 unidades
   - **Algoritmos**: 3D Best-Fit, LAFF, Guillotine, Shelf + Guillotine, Empaquetado 3D con DRL, Heurísticas especializadas
3. Caso peso diverso
   - **Contenedores**:
     - Contenedor 1: 200x100x100 cm, unidades infinitas.
   - **Objetos**:
     - Objeto A: 50x50x50 cm, 50 kg, 2 unidades
     - Objeto B: 25x25x25 cm, 30 kg, 10 unidades
     - Objeto C: 10x10x10 cm, 5 kg, 30 unidades
   - **Algoritmos**: 3D Best-Fit, Heurísticas especializadas
4. Caso multiples contenedores y rotación
   - **Contenedores**:
     - Contenedor 1: 80x80x80 cm
     - Contenedor 2: 120x100x100 cm
   - **Objetos**:

- Objeto A: 40x40x40 cm, puede rotar, 4 unidades
- Objeto B: 70x70x10 cm, NO puede rotar, 4 unidades

- **Algoritmos**: 3D Best-Fit, Guillotine, Shelf + Guillotine, Heurísticas especializadas

5.5. Resultados

Se presentarán los resultados de las ejecuciones de los algoritmos en los diferentes casos, no todas las métricas han sido posibles en todos los algoritmos por la naturaleza intrínseca de cada uno.

5.5.1. Caso básico

En las Figuras 15 y 16 se tendrá el resultado del algoritmo 3D Best-fit para el caso básico. En la Figura 15 se muestra una buena utilización del espacio con todos los objetos de diferentes tamaños, aparentemente empaquetados de manera eficiente. Sin embargo, la Figura 16 muestra una utilización del espacio baja, puesto que solo tiene un objeto más. Las estadísticas muestran que en el primer contenedor se tiene una utilización del espacio del **48.4 %** mientras que para el segundo contenedor este porcentaje se ve reducido hasta **10.8 %**.

Estos datos sugieren que el algoritmo no logró optimizar completamente el espacio disponible en los contenedores, sin embargo, dadas las dimensiones de los objetos, el hecho de que el único bulto del segundo contenedor sea del tipo más grande no es casualidad, puesto que el algoritmo tomó la decisión de repartir el peso entre los dos contenedores. Aunque uno tenga muchos bultos, tener los 4 *Objeto C* en un solo contenedor podría ser contraproducente en términos de transporte. De cualquier modo, cabe margen de mejora en las heurísticas, quizás se podría haber puesto otro *Objeto C* en el segundo contenedor.

Los tiempos de ejecución para empaquetar son bastante eficientes, puesto que el primer caso tomó 0.03669 segundos, y el segundo caso tomó 0.5479 segundos. La demora en el segundo contenedor puede deberse a las técnicas de recolocación necesarias para repartir el peso en los contenedores.

Este análisis sugiere que aunque el algoritmo 3D Best-fit puede manejar rápidamente el empaquetado de múltiples objetos, hay un margen significativo para mejorar la eficiencia del volumen en este conjunto de objetos.

El algoritmo LAFF ha utilizado también dos contenedores, esta vez el contenedor número 2 tiene: 1 *Objeto C* y 2 *Objeto B*, el resto estarán en el contenedor 1. Estos datos indican una utilización del espacio del 15.6 % para el contenedor 2, y del 43.6 % para el contenedor 1.

La distribución de los objetos en los contenedores indica que el algoritmo está priorizando el empaquetado de objetos grandes primero, pueden darse casos donde este enfoque no es efectivo para maximizar la utilización del espacio, especialmente

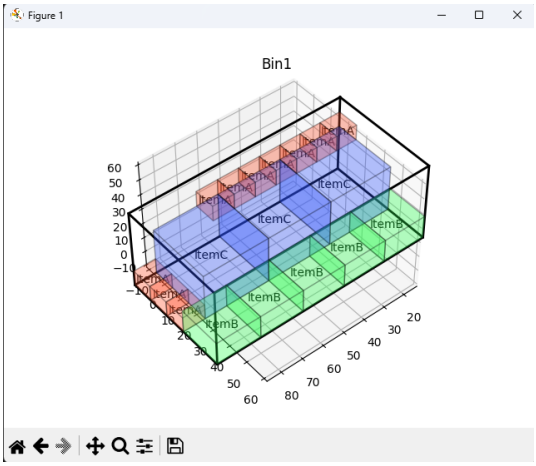

Figura 15: 3D Best-Fit básico 1

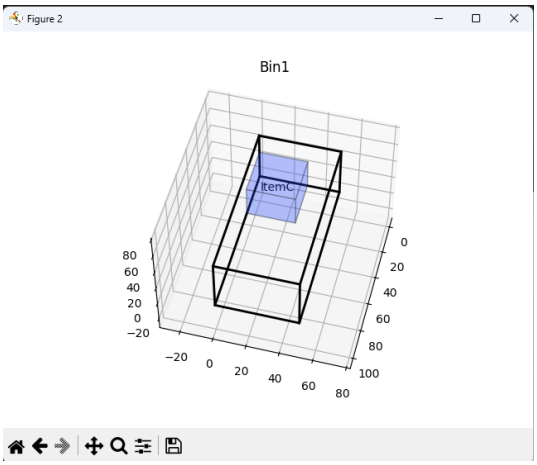

Figura 16: 3D Best-Fit básico 2

si los objetos grandes bloquean la colocación eficiente de objetos más pequeños o medianos. Comparado con el algoritmo 3D Best-fit, LAFF parece tener problemas similares en términos de baja utilización del espacio. Ambos algoritmos no alcanzan una eficiencia del espacio superior al 50 % en este caso de prueba.

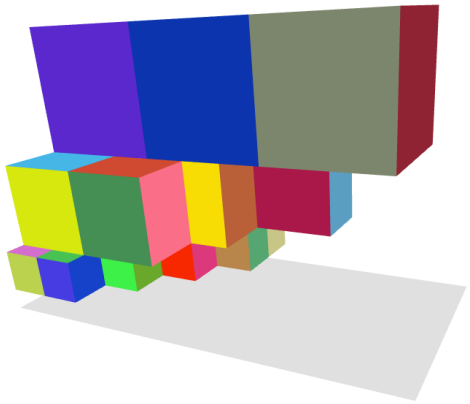

Figura 17: Guillotine básico

La Figura 17 será el resultado del algoritmo Guillotine. Este algoritmo sufre de ineficiencia tanto temporal como práctica. La ejecución de este caso de prueba se ha demorado alrededor de **245 segundos**. Siendo un tiempo de cálculo excesivamente alto comparado con el resto. La ineficiencia práctica viene dada por la inestabilidad observada en el modelo. Se puede ver que hay objetos grandes sobresaliendo de bases más pequeñas, en el mundo real esto sería impracticable. Además, ha necesitado un contenedor de 100x50x60, que es un poco mas grande del contenedor establecido, por lo que técnicamente no es válido.

El algoritmo ha priorizado la colocación de los objetos pequeños primero y ha ido generando secciones con los cortes de guillotina. El problema de esta prioridad es que no resultará en una colocación estable.

En comparación con los algoritmos 3D Best-fit y LAFF, el algoritmo Guillotine parece requerir una revisión considerable para mejorar su viabilidad en entornos operativos que demanden rapidez y eficiencia, así como en aplicaciones donde la estabilidad física no puede ser comprometida.

Al contrario que el algoritmo Guillotine, la combinación de Shelf + Guillotine es una mejora sustancial. En la Figura 18 se observa el resultado de la ejecución.

El funcionamiento del algoritmo se puede notar, ya que los objetos están organizados en filas o "estanterías" como se vio anteriormente. La colocación de los objetos grandes primero y luego llenando los espacios con objetos más pequeños es una estrategia que mejora la utilización del espacio, minimizando el espacio vertical. La mejora más significativa ha sido el tiempo de ejecución, puesto que este caso ha sido inferior a un segundo, mientras que el algoritmo Guillotine tomaba 245 segundos por sí solo.

Dada la rapidez y eficiencia de esta técnica, puede ser una buena opción para entornos donde el tiempo y el espacio son críticos.

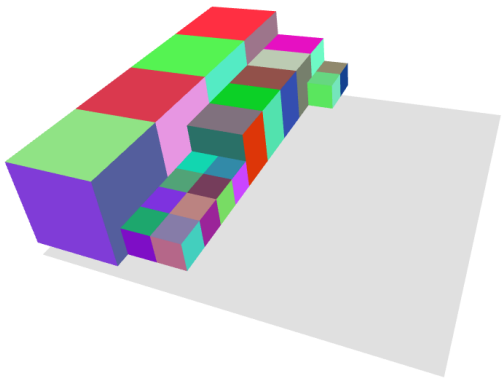

Figura 18: Shelf + Guillotine básico

El algoritmo de aprendizaje automático puede optimizar continuamente su estrategia de empaquetado a través del entrenamiento, por lo que es razonable esperar que logre una utilización del espacio superior al resto de métodos tradicionales o heurísticos. Para un caso básico como es este, puede llegar al 57 % de utilización de espacio, dependiendo de la ejecución. Respecto al tiempo de ejecución, se puede dividir en dos fases:

- Fase de entrenamiento: El tiempo de entrenamiento inicial del modelo puede ser considerable. Un entrenamiento completo con el conjunto de datos proporcionado por los autores, puede llegar a tardar un día entero.
- Fase de ejecución: Una vez que el modelo está entrenado, el tiempo de ejecución para realizar el cálculo es de unos segundos, dependiendo de la complejidad del problema.

Hay que tener en cuenta que las estimaciones que realiza el modelo dependen en

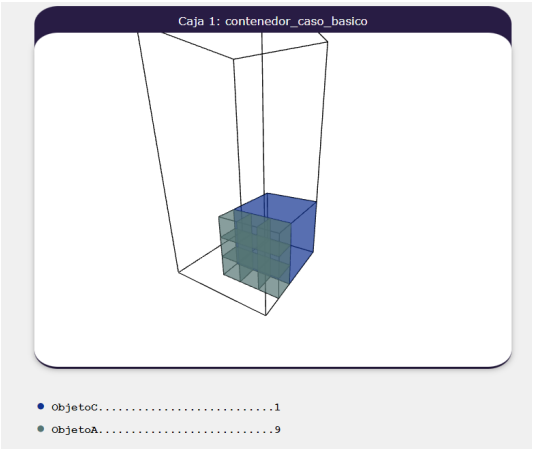

Figura 19: Heurísticas básico 1

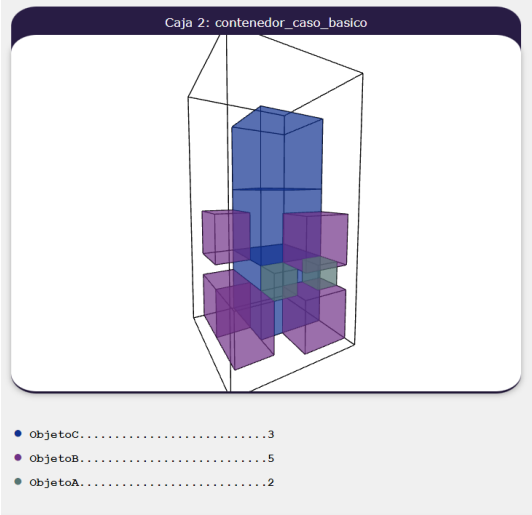

Figura 20: Heurísticas básico 2

gran parte del conjunto de entrenamiento, por lo que las ejecuciones entre diferentes problemas pueden llegar a variar mucho.

Por último, al igual que el algoritmo 3D Best-Fit, el algoritmo de heurísticas especializadas divide el caso en dos contenedores (Figuras 19 y 20). El primer contenedor tendrá 9 *Objeto A* y 1 *Objeto C*. Mientras que el segundo contenedor tendrá 3 *Objeto C* apilados y combina 5 *Objeto B* y 2 *Objeto A*. Esta división puede venir dada de que al no tener los casos de prueba un peso específico, el algoritmo los ha tomado todos como del mismo peso, dividiendo el peso en dos cajas con el mismo peso.

Las configuraciones en ambos contenedores indican un esfuerzo consciente para ajustar los objetos de la manera más eficiente posible. Otra diferencia notable al resto de algoritmos es la decisión de haber rotado los contenedores para una empaquetación vertical, especialmente efectiva en la Figura 20, logrando mayor densidad de empaquetado.

Como se ha comentado anteriormente, este algoritmo proviene del básico *Layer-building* por lo que se puede ver en el modelo que se está violando la restricción de la gravedad, provocando que en la visualización algunos objetos "vuelen".

En este caso básico se han evaluado todos los algoritmos para una lista de objetos estándar definida en 5.4, de manera que se alcanzan las siguientes conclusiones, resumidas en la Tabla 1:

- 3D Best-fit: Este algoritmo intenta colocar cada nuevo objeto en la posición que deja el menor espacio residual, buscando optimizar la utilización del espacio dentro del contenedor. En las visualizaciones observadas, el algoritmo mostró una configuración razonablemente buena en términos de colocación de objetos, aunque la utilización del espacio no fue óptima. Con una utilización reportada del 48.4 %, queda claro que, aunque el algoritmo es efectivo en ciertos aspectos, aún hay margen para mejorar la densidad del empaquetado y la estabilidad

de la carga.

- Largest Area Fit First (LAFF): Este algoritmo prioriza los objetos más grandes primero, intentando colocarlos en la base del contenedor para luego llenar los espacios con objetos más pequeños. En el caso básico, se observó que el LAFF logró una utilización de espacio del 43.6 % en uno de los contenedores, mientras que otro contenedor alcanzó solo un 15.6 % de utilización. Estos resultados sugieren que, aunque el algoritmo puede ser efectivo para ciertas configuraciones de objetos, puede no ser consistente en todos los escenarios, especialmente cuando la diversidad de tamaños y formas es considerable.

- Guillotine: Este algoritmo, al aplicarlo solo, mostró una eficiencia relativamente baja en la utilización del espacio y produjo estructuras que podrían ser inestables en aplicaciones prácticas. Aunque es eficaz en la optimización del material y en realizar cortes precisos, su tiempo de ejecución de 245 segundos es demasiado alto, lo que limita su aplicabilidad en entornos que requieren rapidez.

- Shelf + Guillotine: La combinación de estos dos algoritmos ofreció resultados significativamente mejores, logrando una utilización más eficiente del espacio mediante la estrategia de estanterías, que permitió un mejor aprovechamiento tanto del espacio horizontal como vertical. Además, el tiempo de ejecución se redujo drásticamente a menos de un segundo, destacando su eficiencia y potencial para aplicaciones en tiempo real.

- Aprendizaje Profundo por Refuerzo (DRL): DRL puede ofrecer una optimización continua y una adaptabilidad superior, con un potencial de utilización del espacio de hasta 57 % según la ejecución. Sin embargo, requiere tiempo y recursos considerables para el entrenamiento, aunque su ejecución en tiempo real es eficaz una vez que el modelo está adecuadamente entrenado.

- Heurísticas Especializadas: Este método utilizó un enfoque de rotación y empaquetado vertical, empleando múltiples contenedores para adaptarse a la variedad de tamaños de los objetos. Los resultados indican una alta adaptabilidad y una eficiente utilización del espacio, mostrando cómo la combinación de diferentes técnicas puede superar las limitaciones de los algoritmos individuales.

Algoritmos	**Nº de contenedores**	**Eficiencia**	**Tiempo (s)**	**Tasa de éxito**	**Balance de peso**
3D Best-Fit	2	48.4 % , 10.8 %	0.5479 + 0.03669	100 %	39,02 − 26,26 − 20,43−14,3, 86,67− 0,0 − 13,33 − 0,0
LAFF	2	43.6 %, 15.6 %	0.7534	100 %	-
Guillotine	1	46.66 %	245	0 % [1]	-
Guillotine+Shelf	1	56 %	<0.00	100 %	-

Empaquetado 3D DRL	1	56 %	2.34 + Entrenamiento	100 %	-
Heurísticas especializadas	2	14.4 %, 44.8 %	0.3	100 %	-

Tabla 1: Tabla comparativa Caso Básico

5.5.2. Caso alta densidad

En el caso de alta densidad, la Figura 21 muestra los resultados del algoritmo 3D Best-fit. Ha conseguido empaquetar todos los objetos correctamente. El tiempo de ejecución de este algoritmo fue de 0.063 segundos, mostrando un rendimiento eficaz, puesto que la distribución de los objetos ocupa toda la base del contenedor, de manera que no pese más un lateral de la caja que otro.

[1]Ha necesitado un contenedor más grande de lo estipulado

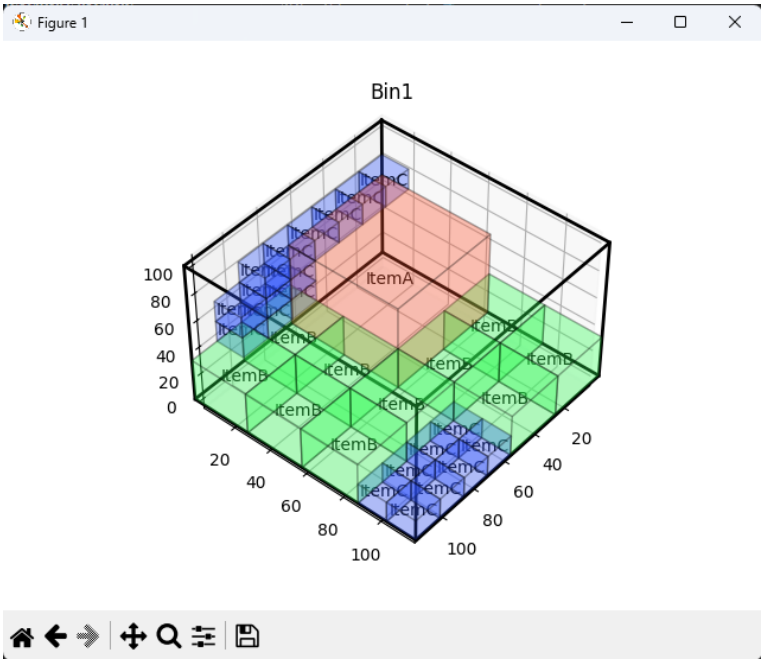

Figura 21: 3D Best-fit alta densidad

El algoritmo LAFF también empaquetará todos los objetos del caso de estudio, logrando un porcentaje de utilización del 38.4 % al igual que 3D Best-fit. En este caso la repartición de los objetos no varía de manera significativa dado que hay más cantidad de los objetos que ocupan menos área.

El algoritmo Guillotine vuelve a dejar bastante que desear. La ejecución fue cortada prematuramente al no encontrar solución completa durante 20 minutos. La solución parcial encontrada es la mostrada en la Figura 22 (Vista superior) y en la Figura 23 (Vista inferior). En la vista inferior se observa que no todos los objetos pequeños han sido empaquetados y que además han sido puestos abajo del todo, generando inestabilidad en el sistema real.

De igual manera que el caso anterior, el algoritmo Guillotine acompañado de Shelf, mejora sustancialmente el rendimiento, ejecutándose en un tiempo inferior a 1 segundo y con una solución bastante plausible (Figura 24), prácticamente igual a la creada por 3D Best-fit.

En los escenarios de alta densidad, donde el objetivo es empaquetar el mayor número de unidades en el menor espacio posible, DRL ajustará dinámicamente las estrategias según las funciones de recompensas vistas en su funcionamiento. En este caso utiliza un solo contenedor con un porcentaje de utilización del 38.4 % que es el máximo posible con todos los objetos, en un tiempo de decisión de 0.002 segundos.

De nuevo, en este caso la heurística especializada (Figura 25) opta por la rotación del contenedor de manera vertical, logrando así una mayor repartición del peso sobre un punto, mientras que se optimiza el espacio vertical y se forma una base sólida para los objetos dentro del contenedor, con los objetos más pesados y grandes a tocando el suelo. Se realiza la comparación de los algoritmos en la Tabla 2.

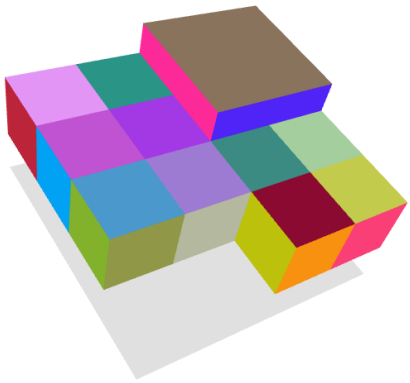

Figura 22: Guillotine alta densidad

Figura 23: Guillotine alta densidad (Vista inferior)

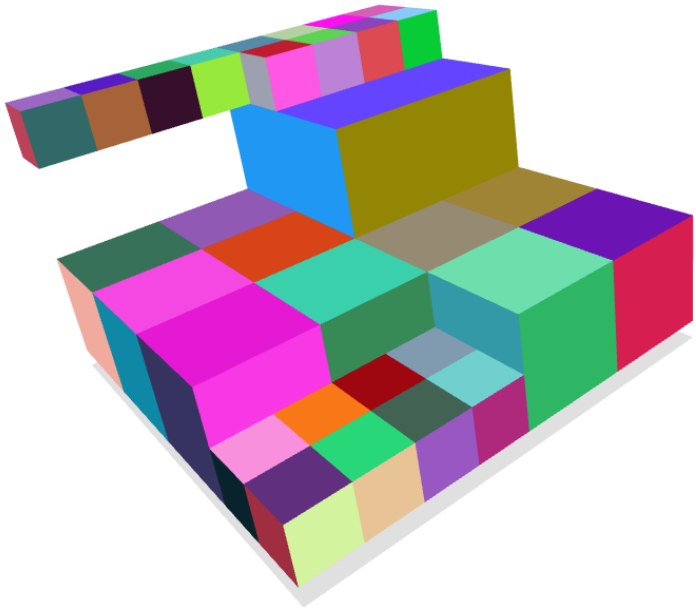

Figura 24: Shelf+Guillotine alta densidad

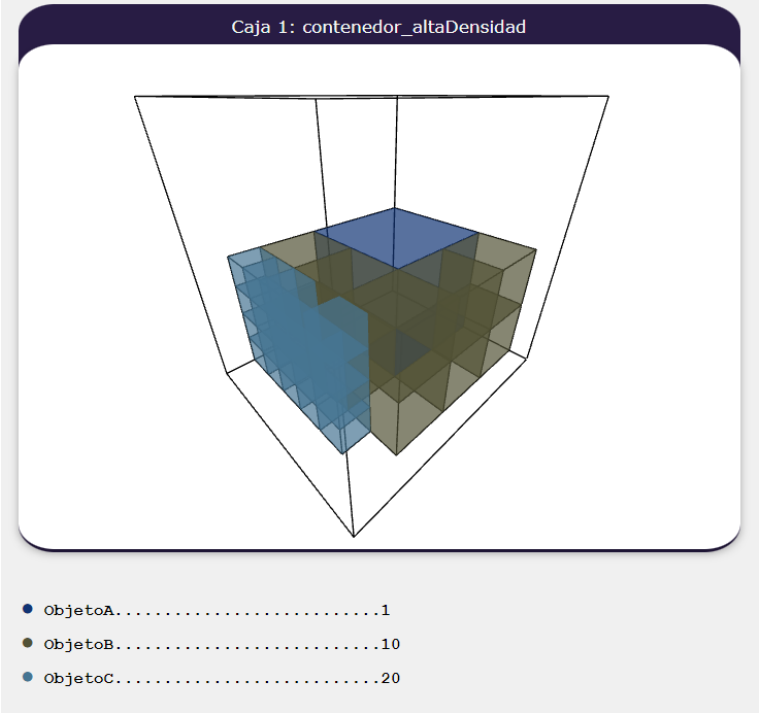

Figura 25: Heuristicas alta densidad

Algoritmos	**Nº de contenedores**	**Eficiencia**	**Tiempo (s)**	**Tasa de éxito**	**Balance de peso**
3D Best-Fit	1	38.44 %	0.0632	100 %	23,23 – 22,79 – 28,49 – 25,49
LAFF	1	38.44 %	0.0351	100 %	-
Guillotine	1	35.6 %	>1200	61.2 %	-
Guillotine+Shelf	1	38.44 %	<0.00	100 %	-
Empaquetado 3D DRL	1	38.44 %	0.002	100 %	-
Heurísticas especializadas	1	38.44 %	0.0535	100 %	-

Tabla 2: Tabla comparativa Caso Alta Densidad

5.5.3. Caso peso diverso

El caso peso diverso buscará una correcta repartición del peso en más de un contenedor para evitar un solo bulto muy pesado.

En las Figuras 26, 27, 28 y 29 vemos la resolución propuesta por el algoritmo 3D Best-fit, en este caso muestran una distribución equilibrada del peso entre cuatro contenedores, cada uno con capacidad máxima de 150 kg. El contenedor 1 alcanza una utilización de espacio del 13.48 % y lleva objetos más ligeros y compactos, mientras que el contenedor 2 logra una utilización del 3.91 %, conteniendo menos objetos,

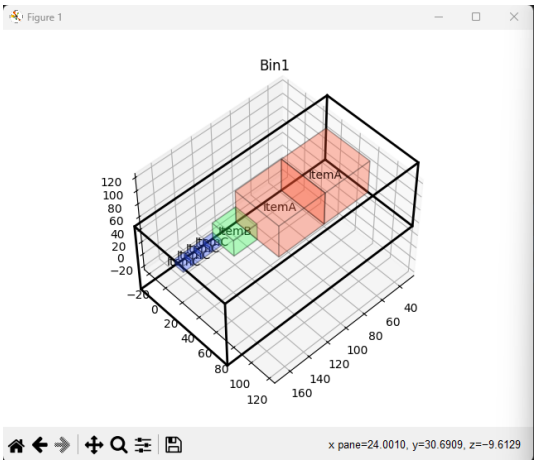

Figura 26: 3D Best-fit peso diverso 1

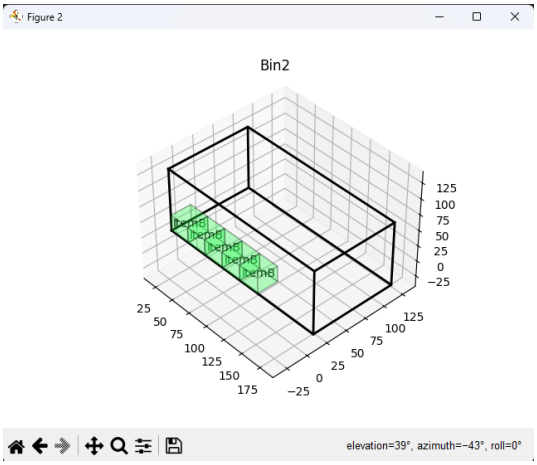

Figura 27: 3D Best-fit peso diverso 2

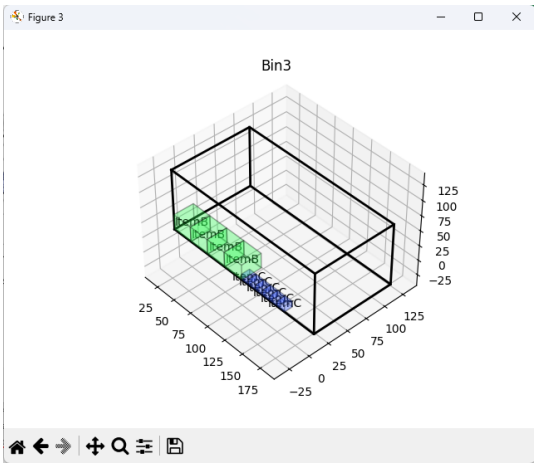

Figura 28: 3D Best-fit peso diverso 3

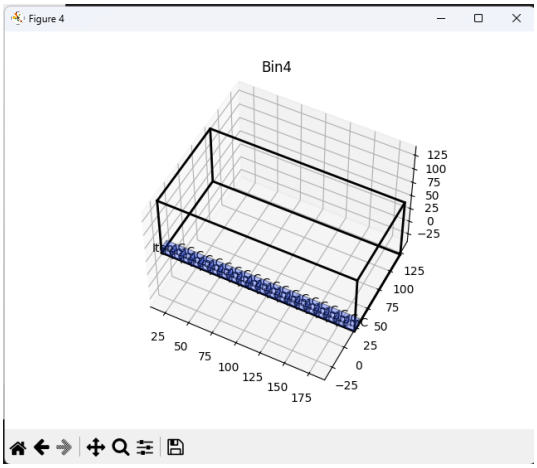

Figura 29: 3D Best-fit peso diverso 4

pero de mayor peso cada uno. El contenedor 3 logra una utilización de 3.43 % y el contenedor 4 una utilización del 1.00 %, siendo el menor de todos, ya que contiene muchos *Objetos C* hasta llegar al peso máximo y completar el empaquetado.

Aunque la ocupación del contenedor no sea muy alta, el objetivo de esta prueba es otro, pues este patrón indica que la distribución de peso es efectiva y equilibrada. Los contenedores 1, 2 y 3 tendrían un peso cada uno de 150 kg, que es el máximo soportado por el contenedor, mientras que el último contenedor son 100 kg, ya que no hay más objetos.

También cabe indicar la rapidez del algoritmo, 0.042, 0.207, 0.298 y 0.40 segundos respectivamente según el contenedor, llegando a un tiempo total de 0.947 segundos, siendo un punto fuerte, indica que el algoritmo es eficiente en tiempo, aunque no cumple la densidad de empaquetado, cumple la restricción de peso, siendo la importante.

LAAF, Guillotine, Shelf+Guillotine ni Empaquetado 3D con DRL permiten la restricción de división de peso en contenedores iguales. Los tres primeros ni lo tienen

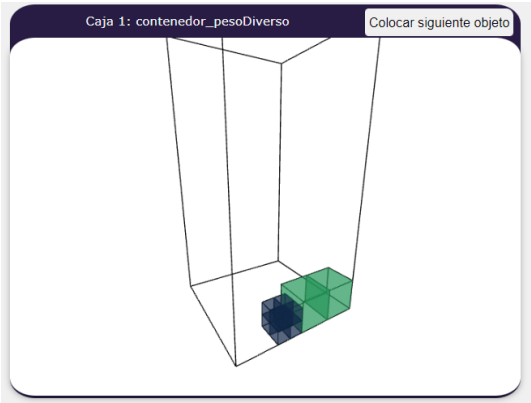

Figura 30: Heurísticas peso diverso 1

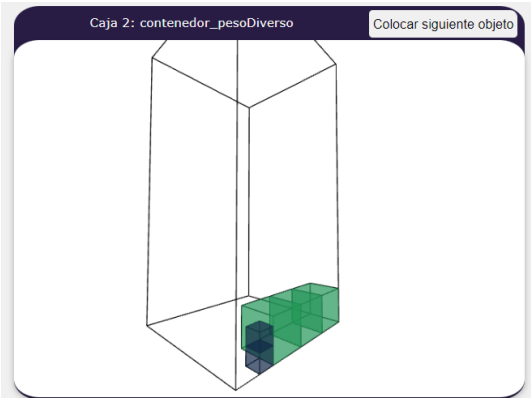

Figura 31: Heurísticas peso diverso 2

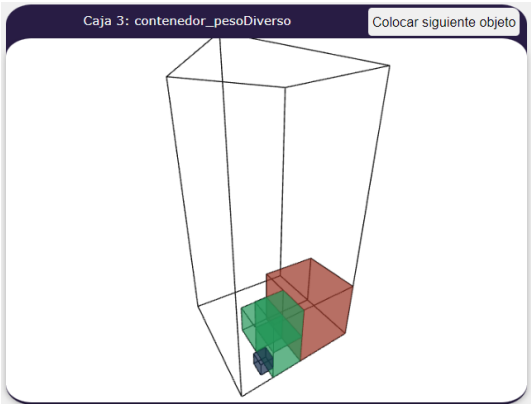

Figura 32: Heurísticas peso diverso 3

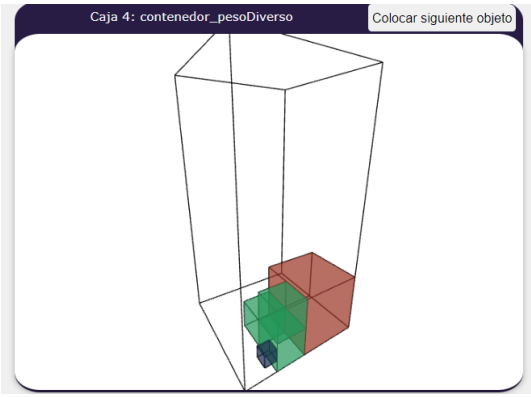

Figura 33: Heurísticas peso diverso 4

en cuenta, mientras que el último solo establece que el peso total de los objetos no sea superior al peso máximo soportado por el contenedor.

El algoritmo de heurísticas especializadas también podrá hacer caso a la restricción de peso, y también opta por repartir los objetos en 4 contenedores diferentes. En este caso, el contenedor 1 y el contenedor 2 (Figuras 30 y 31) pesarán 100 kg cada uno mientras que los contenedores 3 y 4 (Figuras 32 y 33) pesarán 145 kg. En general el peso estará más repartido, pero no ha conseguido empaquetar todos los objetos, ha priorizado la repartición de peso frente a completar el pedido y tener una mayor estabilidad.

Por tanto, se hace la comparativa resultante en la Tabla 3.

Algoritmos	**Nº de contenedores**	**Eficiencia**	**Tiempo (s)**	**Tasa de éxito**	**Pesos según contenedores**
3D Best-Fit	4	13.48 %, 3.91 %, 3.43 %, 1.0 %	0.947	100 %	150, 150, 150, 100

Heurísticas especializadas	4	0.019 %, 0.024 %, 0.086 %, 0.086 %	0.63	59 %	100, 100, 145, 145

Tabla 3: Tabla comparativa Caso Peso Diverso

5.5.4. Caso múltiples contenedores y rotación

En este caso, lo que se pretende evaluar es la adaptabilidad de los algoritmos a que los objetos puedan rotar, y si es necesario, utilizar múltiples contenedores o adaptarse al mejor. Se ofrecen dos tipos de contenedores, el primero será más pequeño y ajustado, obligando a una posición horizontal, mientras que el segundo permitirá diferentes posiciones y combinaciones.

Para el algoritmo 3D Best-fit se observarán diferentes posibilidades.

En primer lugar (Figura 34) escogerá el contenedor más ajustado, colocando todos los objetos de manera horizontal. De esta manera se consigue una utilización del espacio del 88.28 % en un tiempo de 0.0085 segundos, siendo así una solución muy buena. Además al estar prácticamente completo el contenedor, la estabilidad del bulto será muy alta.

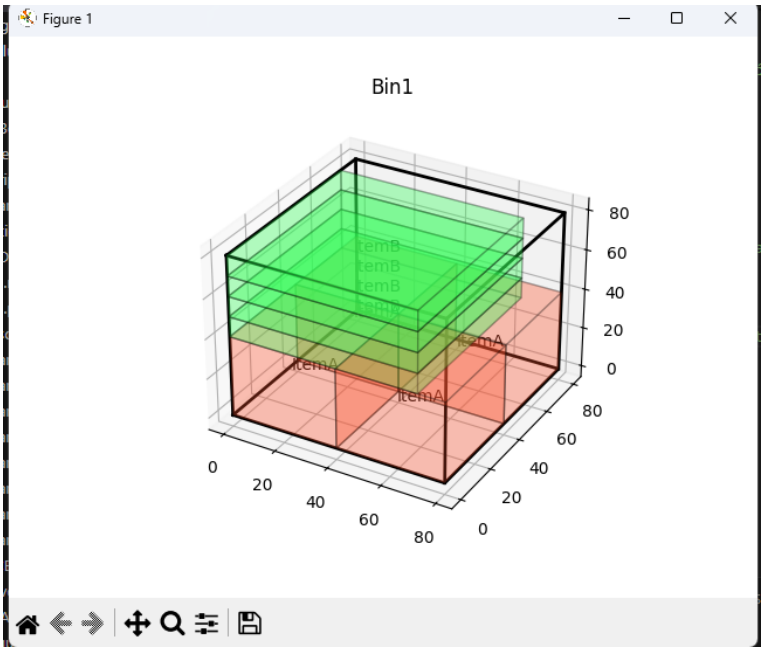

Figura 34: 3D Best-fit rotación

En la Figura 35 se tendrá el contenedor 2, que será más holgado. En este caso al tener más espacio disponible, se ha decidido rotar los *Objetos B*. En este caso la utilización del espacio desciende hasta 37.67 % en un tiempo de 0.01 segundos, es un poco más lento y menos estable, pero cumple la restricción de que el *Objeto*

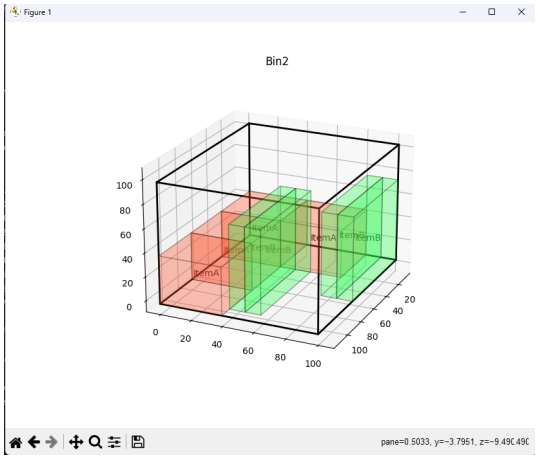

Figura 35: 3D Best-fit rotación 2

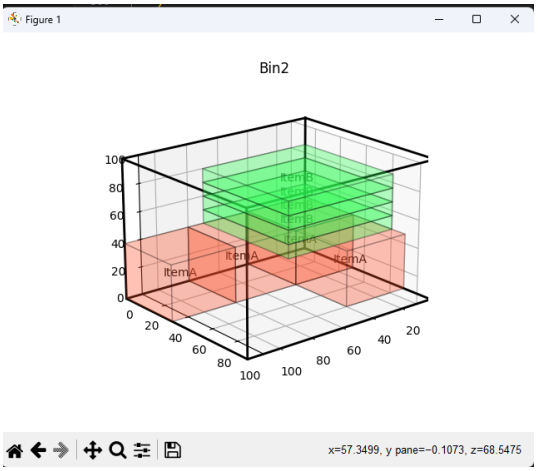

Figura 36: 3D Best-fit no rotación

B debe estar mirando hacia arriba, una restricción de lo más común. También se han repartido los pesos a lo largo de la caja de manera que los cuadrantes estén equilibrados respecto al peso.

Por motivos de comparación, se ha añadido la figura del contenedor 2 eliminando la restricción de que el *Objeto B* deba estar mirando hacia arriba (Figura 36).

LAFF no permitirá añadir rotaciones a los objetos, pero empaquetará todos los objetos en el primer contenedor de manera similar a 3D Best-fit solo que priorizando los *Objetos B*, ya que tienen una mayor superficie.

Guillotine (Figura 37) rotará el objeto sobre el primer contenedor para lograr una estructura estable y realizada con los cortes de guillotina característicos del algoritmo. El hecho de poder rotar el objeto permite que la ejecución del algoritmo sea mucho más rápida que en otros casos vistos anteriormente.

De esta manera, el algoritmo Shelf + Guillotine no será una mejora tan pronunciada como en otros casos. En las Figuras 38 y 39 se puede notar claramente el funcionamiento del algoritmo Shelf, ya que se intuyen las "estanterías" y la separación por alturas con las que trabaja internamente.

Por otra parte, la heurística especializada hará uso de los dos tipos de contenedores (Figuras 40 y 41). En ambos respetará la restricción de que el *Objeto B* ha de estar mirando hacia arriba, y se repartirán los objetos para tener un volumen utilizado y un peso similar en las dos cajas.

Se resume la comparativa en la Tabla 4:

Algoritmos	**Nº de contenedores**	**Eficiencia**	**Tiempo (s)**	**Tasa de éxito**	**Balance de peso**
3D Best-Fit	1	37.67 % [1]	0.01	100 %	29,84 – 13,38 – 48,91 – 7,88
Guillotine	1	37.6 %	0.006	100 % [2]	-

Guillotine+Shelf	1	37.6 %	0.01	100 %	-
Heurísticas especializadas	2	41.2 % (Contenedor 1), 20.08 % (Contenedor 2)	0.65	100 %	-

Tabla 4: Tabla comparativa Caso Multiples contenedores

6. Sistema propuesto

Tras el análisis de comportamiento de los diferentes algoritmos, se ha seleccionado el algoritmo arbitrario de heurísticas especializadas por la versatilidad que ofrece en los diferentes casos y la gestión que realiza frente a las restricciones impuestas, asimismo, la facilidad que ofrece respecto a la implementación de futuras heurísticas y la integración con una plataforma web ha sido un punto decisivo a la hora de su selección.

Se creará una plataforma capaz de llevar la teoría expuesta a una aplicación del mundo real. Para ello, se propone un sistema integral de optimización tanto para embalaje como para paletización, con el objetivo de maximizar la eficiencia y reducir los costes de materia prima y transporte. La plataforma utilizará el algoritmo de heurísticas especializadas visto en esta investigación para optimizar el espacio en cajas y contenedores, buscando ahorrar recursos humanos, económicos y materiales en el transporte y almacenaje de productos.

Con la puesta en marcha de este sistema, se mejorarán los tiempos empleados en la preparación de los pedidos y el proceso de empaquetado, ahorrando costes y mejorando la cadena de valor de la propia empresa. El algoritmo optimizador calculará la disposición de los elementos que tienen que ser colocados en el contenedor, de manera que se maximice el espacio ocupado y se minimice el coste del envío cuanto sea posible. A su vez, se tendrán en cuenta numerosas restricciones, por ejemplo, peso, máximo número de elementos iguales en cada una de las cajas (como podría ser objetos inflamables) etc.

Además se contará con un sistema por pasos que indicará al operario los movimientos a realizar de uno en uno para el proceso de empaquetado. De esta manera, se ahorrarán sobrecostes en envíos y fallos humanos en los procesos logísticos. Si se reduce la dependencia de procesos manuales propensos a errores, saldrán más ventajas competitivas sobre el mercado global, al ofrecer envíos más rápidos y costes de

[1]Escogiendo el caso donde *Objeto B* NO puede rotar
[2]Fue exitoso, pero un *Objeto B* tuvo que ser rotado horizontalmente

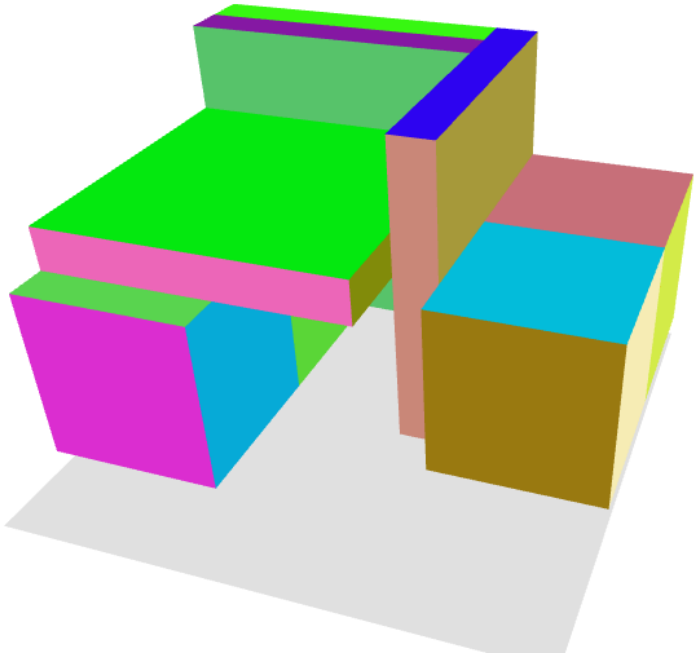

Figura 37: Guillotine rotación

transporte más bajos. Además de la contribución positiva a la reducción del impacto ambiental al gestionar de manera más eficiente los residuos.

El sistema propuesto está diseñado para ser utilizado por operarios de almacén, quienes configurarán el algoritmo con el tamaño de los productos y contenedores, e interpretarán los resultados.

6.1. Componentes del sistema

El sistema tendrá varios componentes que interactúan entre sí para las diferentes acciones:

- **Back-end**: Será el componente encargado de manejar las interacciones del usuario con el sistema, conectando la interfaz de usuario *Front-end* con el algoritmo de optimización y la base de datos. Está diseñado en NodeJS y el framework Express, ampliamente conocidos por la eficiencia que proponen en el desarrollo web y la capacidad para manejar operaciones API para las comunicaciones.
- **Algoritmo optimizador**: Será el algoritmo estudiado con las heurísticas especializadas. Estará implementado en Slim PHP para la integración de una API RESTful facilitando la comunicación y paso de datos con el *back-end*.
- **Base de datos**: Se contará con una base da datos para almacenar las dimensiones de los productos y contenedores, además se guardará la configuración de los pedidos con el fin de facilitar la automatización del algoritmo.

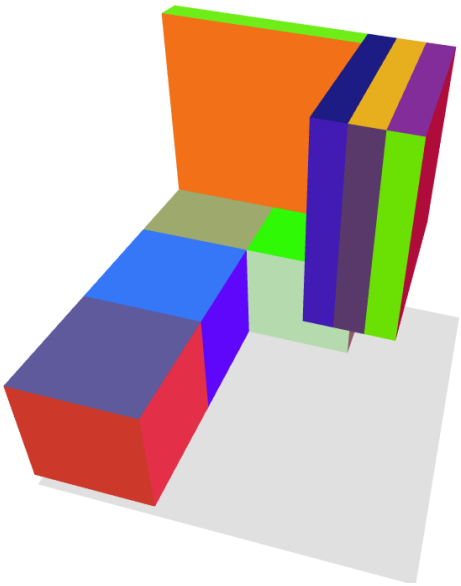

Figura 38: Shelf + Guillotine rotación

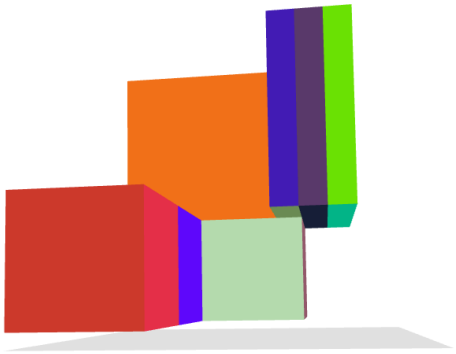

Figura 39: Shelf + Guillotine rotación 2

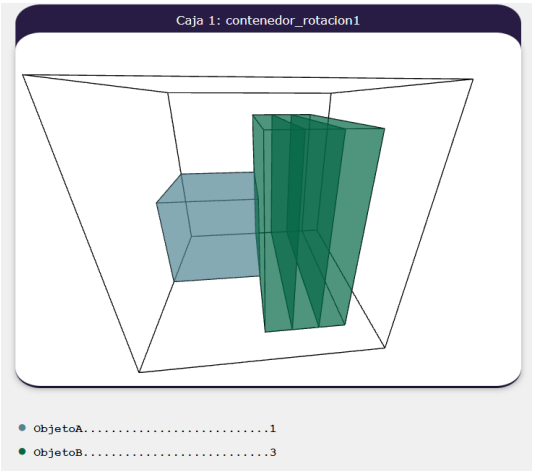

Figura 40: Heurísticas rotación

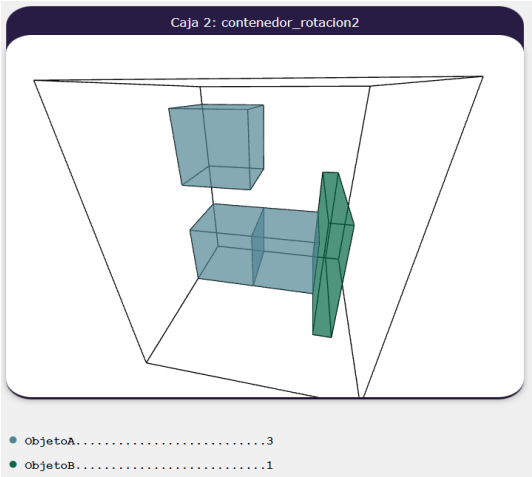

Figura 41: Heurísticas rotación 2

- **Front-end**: Será la parte encargada de la interfaz de usuario, es desarrollada en JavaScript y contará con *Three.js* para la representación 3D de cada uno de los pedidos y los bultos colocados.

Una porción del código referente al componente Algoritmo Optimizador se muestra a continuación, será la creación de los diferentes contenedores y bultos según los datos recibidos mediante peticiones API. Se crearán además listas de colores para la representación gráfica de cada bulto.

```
function boxPacker($boxes, $objects) {
    $packer = new Packer();

    for($i=0; $i<count($boxes); $i++) {
        $packer->addBox(new TestBox($boxes[$i]["referencia"],
    $boxes[$i]["anchura"], $boxes[$i]["altura"], $boxes[$i]["
    longitud"], 10, $boxes[$i]["anchura"]-4, $boxes[$i]["altura"]-4,
     $boxes[$i]["longitud"]-4, $boxes[$i]["pesoMax"]));
    }

    $colorArray = [];
    for($i=0; $i<count($objects); $i++) {
        $colorArray[$i] = rand(0, 999999);
        $colorArray[$i] = str_pad($colorArray[$i], 6, "0",
    STR_PAD_LEFT);
        $colorArray[$i] = '#' . $colorArray[$i];

        $item = new TestItem($i, $objects[$i]["anchura"], $objects[
    $i]["longitud"], $objects[$i]["altura"], $objects[$i]["peso"],
    true);

        $packer->addItem($item, $objects[$i]["cantidad"]);
    }


    try {
        $packedBoxes = $packer->pack();
    } catch (Exception $e) {
        return array("recognizeObjects" => -1, "recognizeBoxes" =>
    -1);
    }
```

El método *pack()* de cada contenedor será el siguiente, empaquetará los objetos en los contenedores usando las heurísticas para la mejor solución:

```
public function pack(): PackedBoxList
{
    $this->logger->log(LogLevel::INFO, '[PACKING STARTED]');

    $packedBoxes = $this->doBasicPacking();

    // If we have multiple boxes, try and optimise/even-out weight
    distribution
    if (!$this->beStrictAboutItemOrdering && $packedBoxes->count()
    > 1 && $packedBoxes->count() <= $this->maxBoxesToBalanceWeight)
    {
```

```
9         $redistributor = new WeightRedistributor($this->boxes,
    $this->packedBoxSorter, $this->boxQuantitiesAvailable);
10        $redistributor->setLogger($this->logger);
11        $packedBoxes = $redistributor->redistributeWeight(
    $packedBoxes);
12     }
13
14     $this->logger->log(LogLevel::INFO, "[PACKING COMPLETED], {
    $packedBoxes->count()} boxes");
15
16     return $packedBoxes;
17 }
```

Cada contenedor individualmente hará el empaquetado básico en el método *DoBasicPacking()*, que realizará el empaquetado de los bultos mediante una serie de iteraciones para encontrar la mejor forma de ordenarnos en las cajas disponibles. Opcionalmente se puede forzar el uso de una sola caja:

```
1  public function doBasicPacking(bool $enforceSingleBox = false):
       PackedBoxList
2      {
3          $packedBoxes = new PackedBoxList($this->packedBoxSorter);
4
5          // Keep going until everything packed
6          while ($this->items->count()) {
7              $packedBoxesIteration = [];
8
9              // Loop through boxes starting with smallest, see what
       happens
10             foreach ($this->getBoxList($enforceSingleBox) as $box)
       {
11                 $volumePacker = new VolumePacker($box, $this->items
       );
12                 $volumePacker->setLogger($this->logger);
13                 $volumePacker->beStrictAboutItemOrdering($this->
       beStrictAboutItemOrdering);
14                 $packedBox = $volumePacker->pack();
15                 if ($packedBox->getItems()->count()) {
16                     $packedBoxesIteration[] = $packedBox;
17
18                     // Have we found a single box that contains
       everything?
19                     if ($packedBox->getItems()->count() === $this->
       items->count()) {
20                         $this->logger->log(LogLevel::DEBUG, "Single
        box found for remaining {$this->items->count()} items");
21                         break;
22                     }
23                 }
24             }
25
26             if (count($packedBoxesIteration) > 0) {
27                 // Find best box of iteration, and remove packed
       items from unpacked list
28                 usort($packedBoxesIteration, [$this->
       packedBoxSorter, 'compare']);
```

```
29                  $bestBox = $packedBoxesIteration[0];
30
31                  $this->items->removePackedItems($bestBox->getItems
    ());
32
33                  $packedBoxes->insert($bestBox);
34                  $this->boxQuantitiesAvailable[$bestBox->getBox()] =
     $this->boxQuantitiesAvailable[$bestBox->getBox()] - 1;
35              } elseif (!$enforceSingleBox) {
36                  throw new NoBoxesAvailableException("No boxes could
     be found for item '{$this->items->top()->getDescription()}'",
    $this->items->top());
37              } else {
38                  $this->logger->log(LogLevel::INFO, "{$this->items->
    count()} unpackable items found");
39                  break;
40              }
41          }
42
43          return $packedBoxes;
44      }
```

6.2. Vistas del sistema Web

La plataforma web desarrollada se ha denominado BoxPacker y contará con diferentes vistas según las acciones a realizar por el usuario.

6.2.1. Vista Principal

Será la vista inicial de la plataforma, presentará un buscador por fechas de realización de *pedidos*. Si no se introduce ninguna fecha, por defecto estará ordenado de más antiguo a más reciente, mostrará una lista con todos los pedidos (Figura 42).

En la parte inferior se contará con dos botones:

1. **Seleccionar**: Estará resaltado en gris, ya que es la pantalla actual de búsqueda de pedidos.

2. **Añadir**: Será la siguiente vista, permitirá añadir nuevos pedidos o contenedores.

6.2.2. Vista Añadir

Será una vista *puente*, desde ella se podrá acceder a añadir pedidos o añadir nuevas cajas/contenedores (Figura 43).

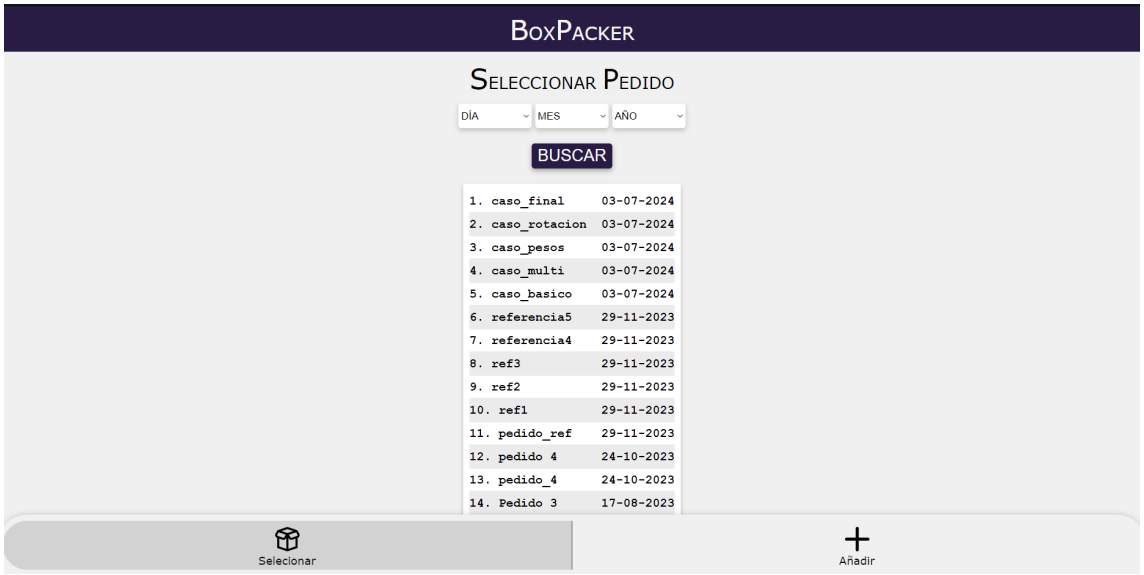

Figura 42: Vista Principal

6.2.3. Vista Añadir Caja

En esta vista se podrán añadir las cajas o contenedores que se guardarán en el sistema. A la hora de realizar un pedido, se seleccionará el tipo de caja a usar, para ello, ha de estar definida previamente aquí.

Las cajas tendrán un nombre de referencia, las dimensiones y el peso máximo soportado (Figura 44). También se dispondrá de un visor 3D para estimar el tamaño de la caja, y un botón de eliminación (Figura 45).

6.2.4. Vista Realizar Pedido

Esta será la vista donde se introducirán los bultos a optimizar. Para ello se guardará un identificador único *Referencia de pedido*, se seleccionarán las cajas que se tienen disponibles para el empaquetado. Se tiene un desplegable con todos los tipos de caja registrados en el sistema y un bloque numérico que indicará cuántas cajas hay disponibles para hacer la optimización.

Posteriormente, se indicarán uno a uno los productos diferentes a empaquetar, indicando el identificador de producto, sus dimensiones y la cantidad de objetos iguales que habrá (Figura 46).

6.2.5. Vista Solucionador

Una vez se seleccione un pedido en la Vista Inicial, se redimirá aquí, será la vista principal de la plataforma, mostrará un visor 3D del pedido seleccionado, el cual se podrá rotar, desplazar, aumentar y disminuir de tamaño con acciones del ratón.

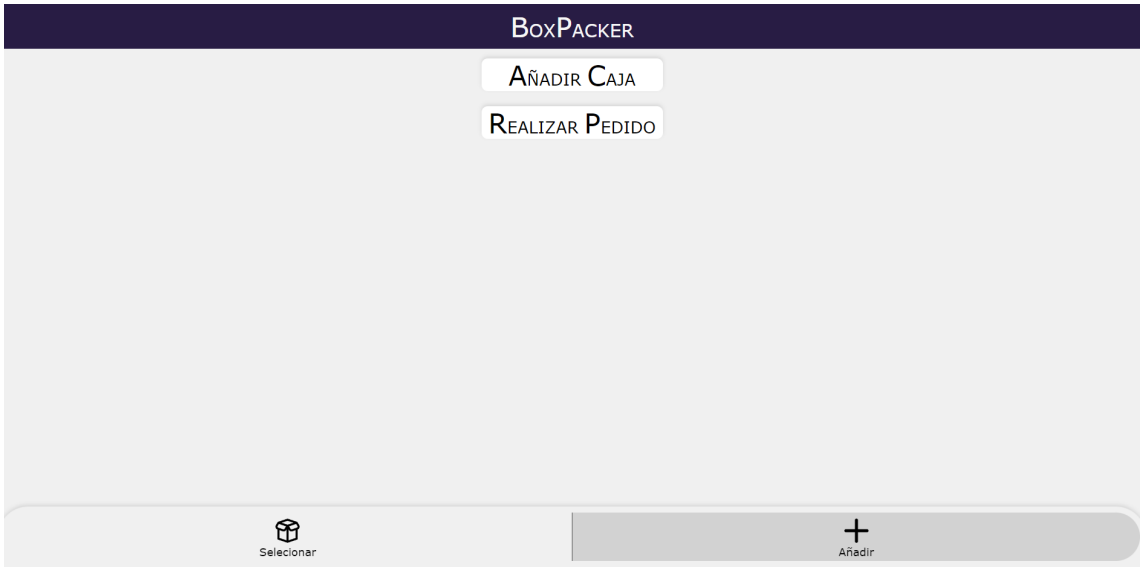

Figura 43: Vista Añadir

Por defecto será la función *guía*, que indicará al operario paso a paso qué bultos colocar en la caja, el último bulto colocado en cada paso parpadeará para una mejor comprensión. Debajo del visor se tendrá una lista con los objetos ya empaquetados y sus identificadores (Figura 47).

Figura 44: Vista Añadir Caja

Figura 45: Vista 3D Cajas

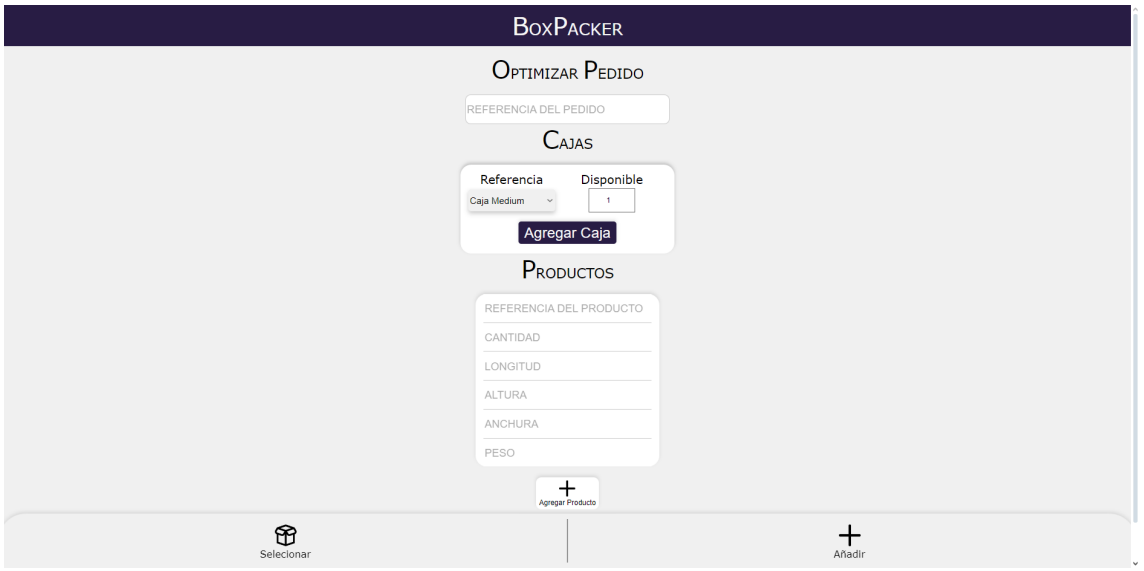

Figura 46: Vista Añadir Pedido

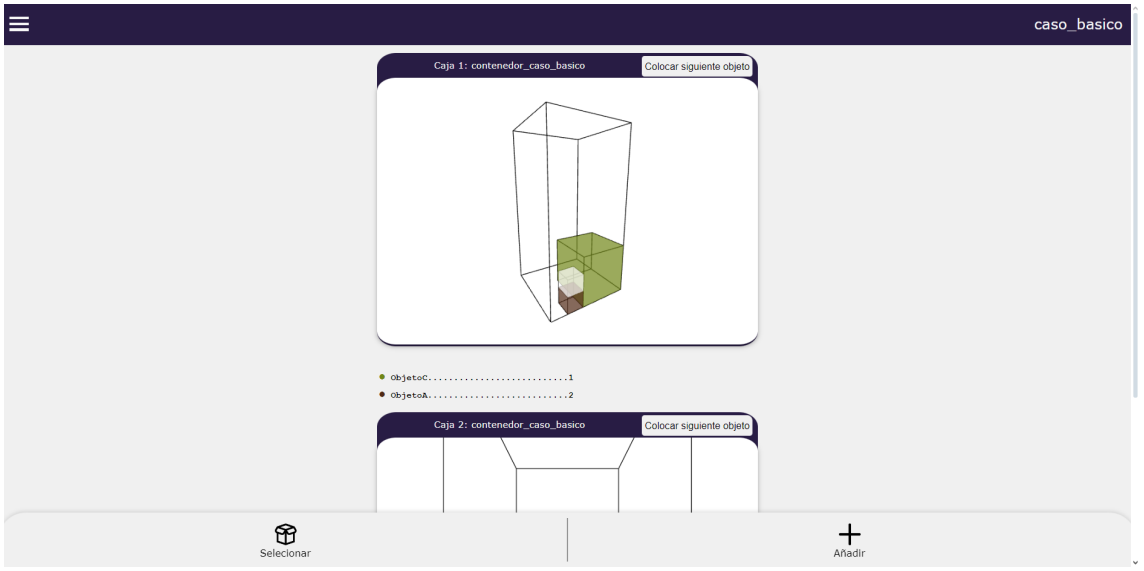

Figura 47: Vista Solución con pasos

Si se desea, arriba a la izquierda habrá un menú para acceder directamente a la versión resuelta, sin necesidad de avanzar paso a paso (Figura 48).

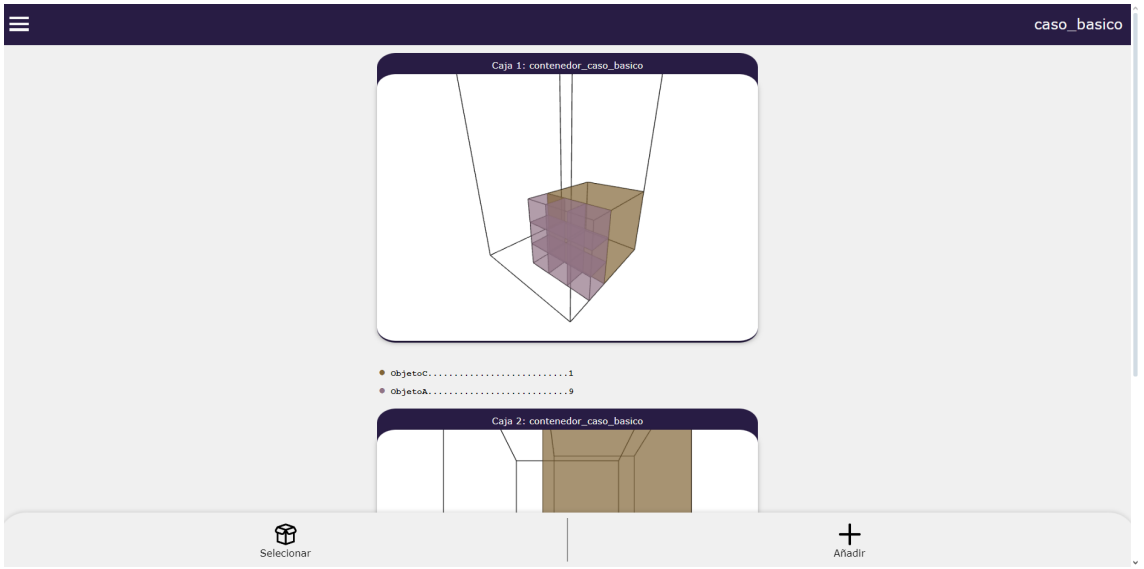

Figura 48: Vista Resolución sin pasos

7. Conclusiones y líneas de trabajo futuras

A lo largo de este proyecto, se ha investigado el problema de optimización del empaquetado para el entorno logístico, centrándose en la mejora de la eficiencia y la reducción de costes mediante uso de diversas técnicas y algoritmos avanzados. Se han analizado diferentes enfoques, desde heurísticas tradicionales hasta algoritmos

de aprendizaje automático por refuerzo profundo. Uno de los principales logros de esta investigación ha sido demostrar la aplicabilidad y efectividad de las diversas técnicas en los escenarios de empaquetado, siendo estos unos entornos complejos.

Las diferentes técnicas no solo mejoraron la eficiencia en el uso del espacio, sino que también proporcionaron soluciones más adaptativas según el caso, y escalables comparadas con los métodos convencionales.

Respecto a estas soluciones adaptativas, se destaca la importancia de desarrollar algoritmos que puedan adaptarse a las necesidades que puedan surgir en la industria, como variaciones en tamaño y forma de los paquetes, lo cual es esencial para las operaciones logísticas modernas.

A pesar de los estudios llevados a cabo, puede haber área de mejora en partes del trabajo. Primero, la integración de algoritmos de aprendizaje automático en entornos industriales aún presenta desafíos significativos. La implementación práctica de estos algoritmos requiere una infraestructura para el entrenamiento de las redes neuronales y la capacidad de manejar grandes volúmenes de datos en tiempo real, lo cual puede ser costoso y complejo de implementar en muchas empresas. Asimismo, la adaptabilidad de estos algoritmos a diferentes tipos de productos y escenarios es algo que necesita más investigación y desarrollo.

Otro aspecto a mejorar es la consideración de factores ambientales y de sostenibilidad en los algoritmos de empaquetado. Si bien algunos estudios han comenzado a abordar la reducción de residuos y la optimización del uso del espacio, hay un margen considerable para mejorar en la integración de prácticas sostenibles. Por ejemplo, se podrían desarrollar algoritmos que no solo optimicen el espacio, sino que también minimicen el uso de materiales de embalaje, creando contenedores a medida, de manera que reduzcan la huella de carbono asociada con el transporte de mercancías.

Como líneas futuras, podría estudiarse la integración de tecnologías modernas como Internet de las cosas (IoT) mediante sensores que podrían proporcionar datos en tiempo real sobre el estado de los productos y el entorno de empaquetado, lo que permitiría ajustes dinámicos y optimizaciones continuas, relocalizando los recursos humanos en las áreas que más ayuda necesiten. También se pueden explorar nuevos desarrollos de herramientas de simulación más avanzadas, que permitan un análisis más profundo de las estrategias de empaquetado, proporcionando así una plataforma para probar los diferentes algoritmos antes de su implementación real, unificando todo en un mismo proyecto.

Referencias

[1] Anan Ananno and Luis Ribeiro. A multi-heuristic algorithm for multi-container 3-d bin packing problem optimization using real world constraints. *IEEE Access*, PP, 03 2024. [*Citado en págs. 18, 21 y 24.*]

[2] Ignacio Araya, Keitel Guerrero, and Eduardo Nuñez. Vcs: A new heuristic function for selecting boxes in the single container loading problem. *Computers & Operations Research*, 82, 01 2017. [*Citado en pág. 18.*]

[3] Andreas Bortfeldt and Hermann Gehring. Hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, 131:143–161, 02 2001. [*Citado en pág. 19.*]

[4] E.F. da Silva, A.A.S. Leão, F.M.B. Toledo, and T. Wauters. A matheuristic framework for the three-dimensional single large object placement problem with practical constraints. *Computers & Operations Research*, 124:105058, 2020. [*Citado en pág. 18.*]

[5] D.C. Daniela Coppola. Global e-commerce share of retail sales 2027. 2023. [*Citado en págs. v y 1.*]

[6] Erick Dube, Leon Kanavathy, Leon K@i, and Owave Za. Optimizing three-dimensional bin packing through simulation. 01 2006. [*Citado en pág. 27.*]

[7] Walaa H. El-Ashmawi and Diaa Salama Abd Elminaam. A modified squirrel search algorithm based on improved best fit heuristic and operator strategy for bin packing problem. *Applied Soft Computing*, 82:105565, 2019. [*Citado en pág. 19.*]

[8] Tobias Fanslau and Andreas Bortfeldt. A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, 22:222–235, 05 2010. [*Citado en pág. 18.*]

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Mathematical Sciences Series. Freeman, 1979. [*Citado en pág. 4.*]

[10] Google Scholar. Google Scholar. `https://scholar.google.es/`. Accessed: 10-05-2024. [*Citado en pág. 11.*]

[11] Young-Dae Hong, Young-Joo Kim, and Ki-Baek Lee. Smart pack: Online autonomous object-packing system using rgb-d sensor data. *Sensors*, 20(16), 2020. [*Citado en págs. 19 y 21.*]

[12] Ying Huang, Ling Lai, Wei Li, and Hui Wang. A differential evolution algorithm with ternary search tree for solving the three-dimensional packing problem. *Information Sciences*, 606:440–452, 2022. [*Citado en pág. 19.*]

[13] IEEE Xplore. IEEE Xplore. `https://ieeexplore.ieee.org/Xplore/home.jsp`. Accessed: 10-05-2024. [*Citado en pág. 11.*]

[14] Wiphawi Kaeothep and Sarayuth Nonsiri. Optimization of uld load planning using milp: Mixed integer linear programming. In *2022 7th International Conference on Business and Industrial Research (ICBIR)*, pages 155–160, 2022. [*Citado en págs. 20 y 21.*]

[15] Kalvin M. Kroth. *Cutting and Packaging Optimization*, pages 45–67. Springer Nature Switzerland, Cham, 2024. [*Citado en pág. 20.*]

[16] Tzuu-Hseng Li, Chin-Yin Liu, Ping-Huan Kuo, Nien-Chu Fang, Cheng-Hui Li, Ching-Wen Cheng, Cheng-Ying Hsieh, Li-Fan Wu, Jie-Jhong Liang, and Chih-Yen Chen. A three-dimensional adaptive pso-based packing algorithm for an iot-based automated e-fulfillment packaging system. *IEEE Access*, PP:1–1, 05 2017. [*Citado en págs. 19 y 21.*]

[17] Batoul Mahvash, Anjali Awasthi, and Satyaveer Chauhan. A column generation-based heuristic for the three-dimensional bin packing problem with rotation. *Journal of the Operational Research Society*, 69, 03 2017. [*Citado en pág. 25.*]

[18] Feng Mao, Edgar Blanco, Mingang Fu, Rohit Jain, Anurag Gupta, Sebastien Mancel, Rong Yuan, Stephen Guo, Sai Kumar, and Yayang Tian. Small boxes big data: A deep learning approach to optimize variable sized bin packing. In *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 80–89, 2017. [*Citado en pág. 22.*]

[19] A. Moura and J.F. Oliveira. A grasp approach to the container-loading problem. *IEEE Intelligent Systems*, 20(4):50–57, 2005. [*Citado en pág. 18.*]

[20] M.Zahid Gürbüz, Selim Akyokuş, İbrahim Emiroğlu, Aysun Güran. An efficient algorithm for 3d rectangular box packing. 09 2009. [*Citado en pág. 28.*]

[21] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, Roger Chou, Julie Glanville, Jeremy M Grimshaw, Asbjørn Hróbjartsson, Manoj M Lalu, Tianjing Li, Elizabeth W Loder, Evan Mayo-Wilson, Steve McDonald, Luke A McGuinness, Lesley A Stewart, James Thomas, Andrea C Tricco, Vivian A Welch, Penny Whiting, and David Moher. The prisma 2020 statement: an updated guideline for reporting systematic reviews. *BMJ*, 372, 2021. [*Citado en pág. 15.*]

[22] F. Parreño, R. Alvarez-Valdes, J. F. Oliveira, and J. M. Tamarit. Neighborhood structures for the container loading problem: a vns implementation. *Journal of Heuristics*, 16(1):1–22, Feb 2010. [*Citado en pág. 18.*]

[23] Sabino Roselli, Fredrik Hagebring, Sarmad Riazi, Martin Fabian, and Knut Åkesson. On the use of equivalence classes for optimal and suboptimal bin

packing and bin covering. *IEEE Transactions on Automation Science and Engineering*, 18(1):369–381, 2021. [*Citado en pág. 20.*]

[24] George Saridakis, Yanqing Lai, Anne-Marie Mohammed, and Jared M. Hansen. Industry characteristics, stages of e-commerce communications, and entrepreneurs and smes revenue growth. *Technological Forecasting and Social Change*, 128:56–66, 2018. [*Citado en pág. 1.*]

[25] Guntram Scheithauer. A three-dimensional bin packing algorithm. *Elektronische Informationsverarbeitung und Kybernetik*, 27:263–271, 01 1991. [*Citado en pág. 26.*]

[26] Scopus. Scopus. `https://www.scopus.com`, 2024. Accessed: 10-05-2024. [*Citado en pág. 11.*]

[27] Shuo Yang, Shuai Song, Shilei Chu, Ran Song, Jiyu Cheng, Yibin Li, and Wei Zhang. Heuristics integrated deep reinforcement learning for online 3d bin packing. *IEEE Transactions on Automation Science and Engineering*, 21(1):939–950, 2024. [*Citado en págs. 18 y 21.*]

[28] Zifei Yang, Shuo Yang, Shuai Song, Wei Zhang, Ran Song, Jiyu Cheng, and Yibin Li. Packerbot: Variable-sized product packing with heuristic deep reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5002–5008, 2021. [*Citado en págs. 19 y 21.*]

[29] Defu Zhang, Yu Peng, and Stephen Leung. A heuristic block-loading algorithm based on multi-layer search for the container loading problem. *Computers & OR*, 39:2267–2276, 10 2012. [*Citado en pág. 18.*]

[30] Hang Zhao, Qijin She, Chenyang Zhu, Yin Yang, and Kai Xu. Online 3d bin packing with constrained deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1):741–749, May 2021. [*Citado en pág. 31.*]

[31] Hang Zhao, Qijin She, Chenyang Zhu, Yin Yang, and Kai Xu. Online 3d bin packing with constrained deep reinforcement learning, 2022. [*Citado en pág. 19.*]

[32] Hang Zhao, Chenyang Zhu, Xin Xu, Hui Huang, and Kai Xu. Learning practically feasible policies for online 3d bin packing. *Science China Information Sciences*, 65(1):112105, Dec 2021. [*Citado en págs. 19 y 21.*]

[33] Jianrong Zhou, Kun He, Jiongzhi Zheng, and Chu-Min Li. Geometric batch optimization for the packing equal circles in a circle problem on large scale, 2023. [*Citado en pág. 20.*]

[34] Qiruyi Zuo, Liu Xinglu, Lu Xu, Li Xiao, Chengyin Xu, Jianfeng Liu, and Victor W. K. Chan. The three-dimensional bin packing problem for deformable items. pages 0911–0918, 12 2022. [*Citado en pág. 20.*]

[35] Volkan Çetin, Başak Gök, and Hadi Gökçen. Information system proposal to improve warehouse operations in a production system. *Bilişim Teknolojileri Dergisi*, 15(4):413–426, 2022. [*Citado en págs. 20 y 21.*]