UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

Profesor Patrocinante:

Rosa Medina

Informe de Tesis

Para optar al Grado de:

Magíster en Ingeniería Industrial

Metaheurística aplicada a un problema de *packing* con ruteo

Concepción, Mayo 2020

Matías Andrés Barros Vasquez

UNIVERSIDAD DE CONCEPCIÓN
Facultad de Ingeniería
Departamento de Ingeniería Industrial

Profesor Patrocinante:
Rosa Medina

Metaheurística aplicada a un problema de *packing* con ruteo

Matías Andrés Barros Vasquez

Informe de Tesis
Para optar al Grado de
Magíster en Ingeniería Industrial

Mayo 2020

Sumario

En el presente trabajo se estudió un problema de ruteo de vehículos con *packing* de tres dimensiones, con cajas no homogéneas, en un solo vehículo de carga, haciendo que los productos sean ordenados en relación al orden de visita a los clientes a los cuales pertenecen dichos productos, que en general, tienen dimensiones diferentes. Se considera un orden de entrega de arriba hacia abajo, de derecha a izquierda y de adelante hacia atrás, respecto a la puerta del contenedor. El objetivo de este estudio es cumplir la ruta óptima, entregando todos los productos a los clientes de dicha ruta, con un *layout* de carga que permita ordenar, de manera óptima, las cajas en el camión de carga según el orden de entrega mencionado anteriormente, utilizando una función objetivo, de tal forma, que da una mayor puntuación a los objetos que deben ser entregados últimos y que minimice la distancia entre la posición del objeto y el punto 0,0,0 del contenedor, considerando el factor de entrega.

Se propone utilizar la metaheurística *Harmony search*, para ordenar las cajas según el orden de visita de los clientes (problema de *packing*). Por otra parte se propone realizar un modelo matemático, para obtener la ruta óptima de entrega, la cual sirve como *input* de la metaheurística planteada para el problema de *packing*. Se realizaron 20 pruebas realistas (las cuales son con datos proporcionados por empresas locales de distribución de productos), para la calibración de los parámetros, con valores para los parámetros recomendados en la literatura y la comparación con estudios realizados anteriormente. También se consideraron las restricciones naturales del problema para el *layout* de las cajas. Para programar la metaheurística mencionada, se utilizó el lenguaje

de programación Python 3.7 en el IDE *Spider* y el solver comercial CPLEX 12.7.1 para calcular la ruta utilizada.
Finalmente, se representa el *layout* de la posición de las cajas según su orden de entrega y las soluciones entregadas por la metaheurística, resaltando las mejores soluciones encontradas. Se comparan los resultados obtenidos por la mejor solución de la metaheurística, con un estudio realizado anteriormente, que analiza un problema similar, pero solucionandolo mediante un modelo matemático.

En términos de programación, se lograron buenos resultados con la metaheurística trabajada, considerando la complejidad del problema. Se puede ver, un aproximado de la posición que deberían tener las cajas en cada iteración, además de conseguir un buen tiempo de ejecución para cada instancia. En este estudio no se consideraron factores como el peso de las cajas, perecibilidad, rotación, centro de gravedad, entre otros, por lo que si agregamos estos factores se podrían obtener soluciones más reales, aunque posiblemente en peores tiempos de ejecución.

Agradecimientos

Agradezco a mi familia por el constante apoyo y cariño que me entregaron durante toda mi etapa de formación profesional.
A mis amigos, por animarme día a día cumplir mis objetivos.
A mis profesores, por la paciencia y conocimientos brindados durante estos años de estudio y aprendizaje.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción y objetivos

1.1. Introducción

En las últimas décadas, la logística ha pasado de ser una simple actividad en las organizaciones a una fuente de ventaja competitiva generando valor, a través de actividades relacionadas con: disponibilidad de productos, niveles de inventario, rapidez en las entregas, estimaciones precisas de demanda, entre otras. Es en este contexto sobre el cual adquiere relevancia el transporte de productos, una actividad importante, en la mayoría de las economías y empresas del mundo.

En este estudio se considera la unión de dos tipos de problemas, el problema de empaquetamiento o *packing* (Ceschia et al. (2013)) para el *layout* de los objetos que deben ir en el camión de reparto y el problema de ruteo de vehículos para hacer una ruta óptima para entregar dichos objetos a los clientes que lo solicitan. Para determinar un *layout* óptimo de los objetos del camión se considera un problema de *packing* en 3 dimensiones o 3D-BPP (Iori (2016)), que consiste en llenar un espacio o vehículo de reparto, teniendo en consideración factores como: el peso, las dimensiones de los objetos a contener, la posible rotación de los objetos, su fragilidad, entre otros. Por otra parte el problema de ruteo de vehículos o VRP (*vehicle routing problem*), consiste en que uno o varios vehículos recorran una ruta de clientes con la menor distancia total posible partiendo y regresando a una sede de distribución. Ambos pertenecen a la clase *NP-hard* (Ceschia et al. (2013)) para instancias de tamaños reales, por lo que su solución óptima es prácticamente imposible de obtener en tiempos computacionales aceptables.

Las metaheurísticas son métodos creados para encontrar soluciones a problemas de optimización complejos, que otros métodos exactos no pueden resolver en tiempos computacionales aceptables. Se basa en la utilización de distintos procedimientos heurísticos, principalmente con relajación de restricciones de los problemas de optimización tratados. En este estudio en particular, se estudia la Metaheurística *Harmony search* (Yang (2009)), que se basa en el procedimiento natural de crear una melodía, considerando tres formas de crearlas, inventar una melodía totalmente nueva, modificar una melodía ya existente variando algunas notas o copiando una melodía ya existente. Esto tiene relación con la generación de las soluciones durante las iteraciones de la metaheurística. Este tipo de analogías a la naturaleza, se da en otras metaheurística utilizadas en la solución de los problemas tratados en este estudio como en Yassen et al(2015) o Liu et al. 2020. Se escogió esta metaheurística, a modo exploratorio, para visualizar las ventajas de esta, en comparación a un modelo matemático ya estudiado (Barros et al. (2018)).
En este estudio, se debe generar un *layout* de las cajas demandadas por los clientes, respetando la ruta que minimiza la distancia total recorrida por el camión de reparto, de manera que las cajas estén listas para la entrega, respetando el orden de entrega de arriba hacia abajo y de adelante hacia atrás, respecto a la puerta trasera del camión, y minimizando la función objetivo.

1.2. Objetivos

Objetivo General
Programar una metaheurística para resolver el problema de *packing* con ruteo en tiempos computacionales aceptables.
Objetivos Específicos

1. Estudiar el problema de *packing* con ruteo;
2. Utilizar la metaheurística *"Harmony search"* para el problema de *packing* con ruteo;
3. Comparar los tiempos de cómputo de trabajos anteriores, para las mismas instancias;
4. Representar el *layout* de las cajas en el camión de reparto según su orden de reparto.

La Hipótesis planteada para este estudio es: ”Se puede encontrar resultados factibles y en tiempos computacionales aceptables para el problema de packing con ruteo, utilizando la metaheurística *harmonic search*”.
Con los objetivos planteados se espera estudiar el uso de la metaheurística *Harmony search* en un problema de *packing* ruteo de vehículos, para los cuales es difíciles obtener una solución exacta actualmente. Comparando los resultados obtenidos con los trabajos anteriores para analizar la convergencia en cada situación(Barros et al. (2018)). Finalmente, presentar una representación gráfica del *layout* de los objetos, según su orden de reparto, para visualizar una posible aplicación en la vida real. Cabe mencionar que la metaheurística trabajada, encuentra soluciones solo para el problema de *packing*, el problema de ruteo de vehículos es solucionado en un modelo matemático exacto y entregando como *input* la solución de este problema a la metaheurística planteada. El presente documento se estructura en 6 capítulos, más las referencias. El Capítulo 2 presenta una descripción del problema abordado. El Capítulo 3, presenta una breve revisión de la literatura existente en ambos problemas por separado y juntos, además de la metaheurística, para tener una noción de cómo se aborda este problema. El Capítulo 4, describe en detalle la implementación de la metaheurística propuesta. El Capítulo 5, presenta los resultados computacionales obtenidos, para finalmente, en el Capítulo 6 presentar las conclusiones obtenidas del estudio.

Capítulo 2

Descripción del Problema

El empaquetamiento de productos o *packing*, es una fase importante en el diseño de un producto tanto, en su reparto y como en su venta, ya que el formato de este puede afectar el peso, las dimensiones y la frecuencia en que el producto se solicita. En general, una empresa de empaquetamiento y distribución de bebestibles y comestibles tiene más de un producto (o formato de producto) y más de un cliente, por lo que la forma de los empaques puede influir en el costo, tiempo de reparto, almacenamiento y distribución.
En Chile se transportan aproximadamente 645.000 toneladas al año de alimentos en camión (MTT (2018)). El reparto de distintos productos a diferentes clientes es un problema para las empresas, ya que muchas veces esta asignación de ruta y posición de cajas en el vehículo de reparto, se realiza por experiencia y/o comodidad de los actores de la cadena de suministro, generando costos de tiempo y dinero innecesarios.
Es importante estudiar los problemas de *packing* y ruteo de vehículos como conjunto, ya que tienen una estrecha relación en el mundo real. Dado a la estrecha relación que tienen el orden de entrega de las cajas con el cómo se entregan, se debe considerar la ruta de reparto a realizar al momento de cargarlas al camión, así minimizar el esfuerzo de descarga para los encargados de dicha labor, optimizar el tiempo de entrega de estos productos, mejorar la calidad de servicio. Además, tener en cuenta consideraciones físicas del problema, como centro de gravedad, perecibilidad, fragilidad, urgencia de entrega, ventanas de tiempo, entre otras muchas consideraciones que se pueden agregar al problema. Esta labor puede simplificarse al tener un camión con muchos productos iguales, pero al aumentar la especificación de los productos y las especificaciones de los clientes, se hará necesario tener claro donde se cargarán los objetos para su descarga, utilizando un estudio como el

presente, para su ubicación, además de muchas otras aplicaciones que se podrían desprender de la unión de estos dos problemas.

En el presente estudio se representa una empresa hipotética con más de 80 tipos de empaque diferentes, con alrededor de 9000 pedidos de productos al día, entonces, la tarea de logística y distribución es una de las que más tiempo toma dentro del proceso considerando las distintas ubicaciones de los clientes y las distintas dimensiones de las cajas de los productos. En este estudio se caracteriza un viaje con una cantidad de clientes determinados, a los cuales se les debe entregar una cierta cantidad de productos. Estos tienen dimensiones no homogéneas. Para este caso se dejan de lado factores como: peso, centro de gravedad, fragilidad, entre otros.

Por lo expuesto anteriormente, el problema planteado se puede considerar como de distribución de carga en contenedores, que es uno de los problemas NP-hard por lo que no se pueden encontrar soluciones óptimas en tiempos polinomiales, en instancias grandes los tiempos de ejecución son prácticamente infinitos. Existen distintos tipos de problemas de *packing* dependiendo de las dimensiones a las que se consideran los productos, objetos de dos o tres dimensiones, según la longitud del punto de origen de la caja hacia cada uno de sus vértices, considerando este como la esquina inferior izquierda de atrás de una caja y de los tipos de productos cargados en los contenedores, sus características y si estos son todos, o no, de formas homogéneas (cajas con cubicas o paralelepipedas con 6 caras lisas, sin considerar imperfecciones o baches). Para cada uno de éstos existen distintos modelos de programación matemática o heurísticas, en la literatura especializada, como en Grange et al. (2018), Zheng et al. (2015) o en perboli et al.(2012). Además, visitar distintos clientes, teniendo en cuenta que se debe iniciar y finalizar la ruta en la planta de distribución, se considera un problema de ruteo de vehículos (VRP) (Ceschia et al.(2013)) que consiste en un vehículo, en este caso un camión de reparto, que debe visitar a todos los clientes con la menor distancia recorrida total, sin formar *subtours* y sin que un cliente se visite más de una vez.

En conclusión, se considera un problema de *packing* en tres dimensiones, con cajas débilmente heterogéneas, por lo que las cajas tienen, en general, distintas dimensiones y considerando solo un camión de reparto, para múltiples clientes, junto a un problema de ruteo de vehículos (VRP).

Como función objetivo, se definió, minimizar la distancia entre la posición de las cajas y

el punto (0,0,0) del camión de reparto, que se considera como el último punto a revisar al entregar los productos, esto con una penalidad según el orden de reparto *lifo* y de derecha a izquierda, de arriba hacia abajo y de adelante hacia atrás, con respecto a la puerta trasera del camión. Los objetos que deben ser entregados primero, se les asigna una penalidad menor, intentando que estos queden a una mayor distancia del punto más alejado de reparto, comparando con los objetos que tienen que ser entregados últimos que tienen una mayor penalidad. Esta penalidad consiste en un vector de números consecutivos decrecientes, [N...1], con N = número de productos, considerando que el *input* de la metaheurística son las dimensiones de los productos en orden de entrega, desde el último a entregar hasta el primero, según las solicitudes de los clientes, cada uno tiene una mayor penalidad total en el orden de las cajas. En forma general, en la literatura (Iori (2016)), las dimensiones de cajas, ancho, largo y alto se definen como w_j, l_j y h_j, con $j \in N$, con N representando el número total de cajas utilizadas, y las dimensiones de los *pallets* o contenedores se representan W_i, L_i, H_i con $i \in M$, con M representando el número total de *pallets* a utilizar.

Capítulo 3
Revisión de la Literatura

3.1. Tipos de bin packing en la literatura

El *bin packing problem* o 1D-BPP es un problema de investigación de operaciones de clase *NP-hard*, por lo que no se conoce un algoritmo que entregue una solución óptima en tiempos computacionales polinomiales para instancias grandes o complejas. Por esto se han desarrollados varias heurísticas y metaheurísticas para encontrar soluciones a este tipo de problemas.
En Iori (2016), podemos ver que hay diversas clasificaciones para este tipo de problemas y para el método de solución, por ejemplo: peso de las carga o productos, centro de gravedad que afecta a la estabilidad del camión, posicionamiento o descarga de los productos cargados, tipo de producto, tipo de base donde se apilarán, etc.
Una extensión del problema de 1D-BPP se obtiene aumentando las dimensiones que se analizan, encontrando las siguientes clasificaciones, según las dimensiones del problema a tratar, considerando 4 puntos de vista:

- *Two-Dimensional Bin Packing Problem* (2D-BPP)
- *Two-Dimensional Strip Packing Problem* (2D-SPP)
- *Three-Dimensional Bin Packing Problem* (3D-BPP)

- *Three-Dimensional Strip Packing Problem* (3D-SPP)

Las clasificaciones anteriores se diferencian principalmente en las dimensiones que se analizan, para 2 o 3 dimensiones y la forma de los contenedores que se usarán, que en el caso de los problemas tipo *Strip* se considera una longitud del contenedor infinita, por lo que no se estudia la capacidad.
Se mencionan distintas variantes de problemas de 3 dimensiones, según las características de los contenedores y productos a almacenar, como se explica en la investigación bibliográfica de Zhao et al. (2016), dependiendo del objetivo del modelo matemático. Si el objetivo del problema es minimizar el total de contenedores a utilizar, encontramos los tipos de problemas presentados en Tabla 3.1, donde el significado de que los productos sean fuertemente o débilmente heterogéneos, deriva en la cantidad de cajas que tienen sus dimensiones iguales (son del mismo tamaño).

Tabla 3.1: Tipos de problemas que minimizan el número de contenedores (Zhao et al. (2016))

Problema	Contenedores	Productos
Single Stock-Size Cutting Stock Problem	Idénticos	Débilmente heterogéneos
Single Bin-Size Bin Packing Problem	Idénticos	Fuertemente heterogéneos
Multiple Stock-Size Cutting Stock Problem	Débilmente heterogéneos	Débilmente heterogéneos
Multiple Bin-Size Bin Packing Problem	Débilmente heterogéneos	Fuertemente heterogéneos
Residual Cutting Stock Problem	Fuertemente heterogéneos	Débilmente heterogéneos
Residual Cutting Stock Problem	Fuertemente heterogéneos	Débilmente heterogéneos
Residual Bin Packing Problem	Fuertemente heterogéneos	Fuertemente heterogéneos

Por otro lado si el la función objetivo es maximizar las cantidad de cajas del contenedor, para obtener el mayor valor en un contenedor, se observa en Tabla 3.2.
Estos problemas (Tabla 3.1 y la Tabla 3.2) se pueden resolver con distintos métodos, con heurísticas, metaheurísticas o con modelos matemáticos exactos y combinaciones de estas para lograr resultados aceptables, esto se ve en algunos de los estudios siguientes.

Tabla 3.2: Tipos de problemas que maximizan el valor del contenedor (Zhao et al. (2016))

Problema	Contenedores	Productos
Identical Item Packing Problem	Un contenedor	Idénticos
Single Large Object Placement Problem	Un contenedor	Débilmente heterogéneos
Single Knapsack Problem	Un contenedor	Fuertemente heterogéneos
Multiple Identical Large Object Placement Problem	Idénticos	Débilmente heterogéneos
Multiple Heterogeneous Large Object Placement Problem	Fuerte o débilmente heterogéneos	Débilmente heterogéneos
Multiple Identical Knapsack Problem	Idénticos	Fuertemente heterogéneos
Multiple Heterogeneous Knapsack Problem	Fuerte o débilmente heterogéneos	Fuertemente heterogéneos

3.2. Formulación de problemas de packing con ruteo en la literatura

En Ceschia et al. (2013), primero se explica el problema *three dimensional loading capacitated vehicle routing problem* (3L-CVRP), donde se representan los clientes en un grafo, con costos asociados a cada uno y si alguno tienen un trato especial; los productos que requiere cada cliente con sus dimensiones dadas, su peso y su fragilidad; y los vehículos que transportan los productos con sus capacidades y dimensiones; en este problema se busca minimizar el costo de transporte y satisfacer a los clientes teniendo ciertas restricciones de preferencia, aparte de las restricciones de logística, de *packing* como orientación de las cajas, fragilidad, política de *lifo* y área de soporte. Luego se especifica que pueden existir distintos tipos de problemas dependiendo de las características de los vehículos de reparto y de los productos a repartir y que dependiendo de la similitud entre los elementos, se ocupaban distintos métodos para la carga. El modelo también contempla la estabilidad y fragilidad de la carga, además de la secuencia de carga *lifo* y la introducción de una nueva restricción a las típicas de bin packing llamada "*Rechability*" que considera la distancia entre el personal encargado de cargar/descargar y el posicionamiento de la caja en la carga.

En Kuccukouglu et al. (2019) se estudia una nueva estrategia de logística, que busca reducir el costo de almacenamiento y acelerar la rapidez del flujo de los productos, considerando un problema de *packing* con ruteo, presentando un modelo matemático. Además, se propone una metaheurística híbrida, entre un *simulated anealing* y un *tabú search*, para apoyar al modelo matemático propuesto. El problema consiste en repartir una cierta cantidad de objetos provenientes de una cantidad de proveedores para repartir a una lista de clientes determinado, con restricciones de carga, considerando dimensiones bi dimensionales, con un solo centro de *cross-docking* para la distribución. Se programó los méto-

dos utilizados en el software CPLEX, se realizan 4 partes de experimentos con distintas cantidades de locaciones utilizadas (25,50 y 100 locaciones), cantidad de clientes, de proveedores y tamaños de los productos a repartir, para luego hacer pruebas con una mayor cantidad de variaciones, para la comparación del método utilizado. Finalmente se entrega una metodología eficiente para resolver este problema, mejor que las metaheurística utilizadas, además de proponer mejoras para la solución del problema.

3.3. Formulación de problemas de packing en la literatura

En Paquay et al. (2016) , se consideran las dimensiones y peso de las cajas, además de las dimensiones, peso y volumen del contenedor. El objetivo es minimizar el espacio no utilizado en el contenedor. Las variables del problema son binarias, por si una caja está al lado, sobre o detrás de otra, si la caja está a lo largo o a lo ancho, si la caja está en el contenedor y si el contenedor está siendo usado, además de las variables de localización de la caja desde la vista al fondo a la derecha y desde el frente a la izquierda. El modelo matemático busca minimizar el espacio no utilizado, que resta el espacio total del contenedor, si este está ocupado, menos la sumatoria del volumen de las cajas con restricciones de que una caja solo puede estar en un contenedor, que las dimensiones de las cajas en el contenedor no pueden superar las dimensiones máximas del contenedor donde están, restricciones de no superposición y de rotación, además de considerar las cajas que no se pueden rotar debido a su tamaño o sus especificaciones, haciendo variables especiales para ellas y adaptándolas a las restricciones anteriores. Este estudio también presenta un modelo matemático para los contenedores con cortes o no paralelepípedos, considerando el centro de gravedad de las cajas y del contenedor.

En Alonso et al. (2017), se definen las dimensiones de cada producto, su peso y la demanda de cada tipo de producto, las dimensiones y peso máximo soportado de los *pallets* y del contenedor donde se almacenarán los productos, considerando que todos los camiones son iguales, la idea es minimizar la cantidad de camiones usados. Se calcula la cantidad máxima de *pallet* posibles a utilizar, tanto a lo largo como a lo ancho considerando una cuadrícula en el espacio disponible del camión. Se explica que el camión tiene una carga máxima que depende de la distribución de los *pallets* en el camión, por lo que se divide

el camión en 3 secciones, divididas por los ejes y considerando el punto medio de cada sección, obteniendo finalmente la restricción de que la fuerza aplicada en equilibrio de cada eje no puede superar la máxima fuerza soportada por los ejes. Como características adicionales del problema se definen los centros de gravedad de la carga y la minimización del esfuerzo de carga y descarga.

En de Almeida and Figueiredo (2010) estudia un modelo matemático con una heurística, utilizando variables binarias, que indican si dos cajas están juntas, definiendo parámetros para las distintas posiciones que puede tener sin rotación, a la izquierda, a la derecha, atrás, al frente, abajo o encima, la función objetivo de este estudio es minimizar el número de *pallets* a utilizar.

En Perboli et al. (2012), se diferencian los parámetros clásicos del problema de *bin packing*, el valor obtenido de una caja, depende del *pallet* donde se cargue, definiendo un conjunto de productos, conjuntos de *pallets*, conjunto de tipos de *pallets*, valor de un *pallet*, el número máximo y mínimo de un tipo de *pallet*, el volumen de un *pallet*, el valor y el volumen de un producto y el número máximo de *pallets*, con restricciones como que un producto sólo puede ser puesto en un *pallet*, el volumen de los productos en un *pallet* no puede ser mayor al volumen máximo aceptado de un *pallet*, el número de *pallets* ocupados de cada tipo tiene que estar entre el mayor y menor número de pallets de cada tipo a utilizar y que la cantidad de *pallets* utilizados no puede superar la cantidad total de *pallets* en stock. La función objetivo del modelo matemático propuesto es maximizar las ganancias obtenidas, con restricciones propias del problema como carga, capacidad y uso de *pallets*.

Para la formulación matemática, del problema, se usan distintos métodos, como el encontrado en Liao and Hsu (2013), donde se definen las cotas inferiores del problema 1D-BPP, el cual dice que se puede extender fácilmente a un problema 3D-BPP, considerando el volumen de cada objeto, este límite se va mejorando hasta obtener m sub grupos de objetos en cada compartimiento, hasta que se obtenga una solución con tiempos de O(n). Este artículo extiende también el límite inferior a una versión más general, con un caso no orientado, donde las cajas se pueden rotar en 90 grados, para el caso de 3 dimensiones, considerando el *pallet* donde se depositarán las cajas.

En Adar and Epstein (2013), se definen restricciones de cardinalidad, correspondiendo a un límite superior de los elementos que el contenedor puede contener, también, un parámetro para determinar el costo en un *pallet*, definen una solución óptima fija para la entrada del problema con las restricciones de cardinalidad y un conjunto de cajas arbitrario, con estos datos crean lemas para resolver el problema de bin packing, considerando las posiciones de las cajas y la cantidad de *pallets* utilizados definiendo parámetros para luego, utilizarlos en un algoritmo de teoría de juego.

Podemos ver en Zheng et al. (2015), que se usa las distintas dimensiones de los contenedores y las cajas, el número total de cajas, el beneficio de cada caja, como parámetros y como variables de decisión la posición de las cajas según otras cajas y si están o no en el contenedor como variables de decisión. El modelo matemático propuesto intenta maximizar el largo, ancho y alto de las dimensiones de las cajas con respecto al contenedor, para ocupar el mayor espacio posible y además maximizar el beneficio obtenido por cada caja, tomando en cuenta restricciones de que las cajas no se superposicionen y asegura que las cajas dentro del contenedor no salgan de estas por alguna dirección.

En Bozejko et al. (2015), se define las dimensiones de un contenedor cuboide y un conjunto de cajas (P), de un conjunto de tipos (T) dados, con 6 posibles tipos de rotación (R), cada caja en el conjunto P, es de un tipo específico que define sus dimensiones y su posible rotación con la idea de maximizar el volumen del conjunto de las cajas individuales cargadas, para esto, se define una variable binaria que es 1 si la caja está en el contenedor y 0 en otro caso, utilizando restricciones de que cada caja tiene que estar completamente en un contenedor y de no superposición.

En Alonso et al. (2016), se considera la rotación de las cajas, con un parámetro r de rotación en las dimensiones, y la cantidad de productos necesarias cada día. El problema se divide en dos partes, primero la construcción de los *pallets* y luego la carga del contenedor, en la primera fase se considera las dimensiones del *pallet* y la cantidad de peso que resiste, se limita el total de la altura de los *pallets* cargados a la mitad de la altura total del camión, por lo que se pueden apilar dos *pallets*, uno encima del otro. Se definen dos tipos

de *pallets*, los que tienen el mismo tipo de productos y los que tienen distintos tipos de producto. Aparte, de la carga de los productos en el *pallets*, se consideran restricciones de orientación, de prioridad de los productos, la apilabilidad de los productos y de resistencia del *pallet*. Para la carga del camión, se consideran las dimensiones del camión, la capacidad de peso de este según la distribución en los ejes y el centro de gravedad de la carga. Para el orden de carga se considera la prioridad de los *pallets*, la estabilidad de la carga, la apilabilidad de los *pallets*.

En Grange et al.(2018) se estudia una extensión del problema de bin packing, donde se genera un overlap. Se utiliza un software genérico de programación, utilizando un modelo de programación lineal en una primera etapa, utilizando variables auxiliares para la solución del problema, luego se asignan valores a los objetos cargados según el volumen que ocupan, posicionando los objetos según la relación de estos números mientras se van cargando. Se prueban distintas instancias variando los distintos parámetros tanto para los objetos como para los contenedores. Se estudian 4 distintas heurísticas a utilizar para este problema, una heurística *greedy* inspirada principalmente en el *bin packing*, considerando los mejores y peores escenarios, haciendo variaciones entre ellos; una heurística *greedy* especializada donde se minimiza o maximiza la variación de los números asignados anteriormente; en la heurística "*Overload-and-Remove*", se consideran las decisiones de asignación anteriores para seleccionar la mejor oportunidad con una regla FIFO (*First In, First Out*); finalmente se prueba un algoritmo genético, donde se evalúan los números asignados a cada objeto intentando lograr una solución óptima, con cruces de tipo aleatorios entre las distintas configuraciones. Como se menciona en el estudio, se utilizaron heurísticas tanto *greedy* como no *greedy*, donde en ninguna se obtuvo una solución en tiempos no exponenciales, como se menciona este artículo, no estaba intentando solucionar el problema de *bin packing* sino para abrir nuevas posibilidades de solución de este.

En Hessler et al. (2018) se revisan distintas soluciones de *coverings formulations* y *columns-generation-based* para el problema de *vector packing* y *cutting* con distintas restricciones para cada uno (peso, longitud y valor), limitando la carga a solo dos dimensiones. Primero se propone un algoritmo unificado de *Branch and Price* en el cual, para procesar la generación de columnas, se estabiliza con un programa maestro. Se programa con C++ en

el IDE Visual Studio, donde se probaron 10 clases de 2-dimensional *Vector Packing Problem*, agrupadas en 40 instancias cada uno, generando un total de 400 instancias, además de una subclase del problema con 20 instancias dimensionales, con otras 40 instancias, en el total de 440 instancias, se agruparon los objetos con peso similar y se compararon las distintas instancias con distintos problemas planteados. Como conclusión se reconoce el problema de generación de columnas como una variante del problema de la mochila tridimensional y se postula que los algoritmos utilizados son capaces de resolver el problema de *Vector Packing Problem* en tiempos computacionales aceptables, pero que, las decisiones tomadas en el *branching* empeoran el proceso de resolución.

En Castro et al. (2019) se habla del *Balanced Fractional Bin Packing problem* (BFBPP) donde se asignan la misma cantidad de Objetos a todos los *bins* de los contenedores existentes, además considerando el *Balanced Multi stage Bin Packing problem*, donde se asignan la misma cantidad de objetos a los *bins* en todas las etapas que pasa el contenedor. El problema principal (BFBPP) consiste en relajar la restricción del problema de *packing* original de $x_{ij}E[0,1]$, donde x_{ij} es 1 si el objeto i se asigna al Bin j y considerando que el número mínimo de contenedores necesarios en $N_{frac} := (S/C)$, donde S es la suma de todos los tamaños de las cajas y C es la capacidad del contenedor. El *Balanced Multistage Bin Packing Problem*, se menciona como una variante del BFBPP donde se considera las mismas condiciones que el problema tratado anteriormente, pero esta vez asigna los objetos a los *bins* en distintas etapas, ocupando la misma proporción en las distintas fases de asignación, donde una etapa es la carga/descarga de los objetos en los distintos contenedores disponibles, considerando que se minimice la utilización de *bins* en cada una de ellas, utilizando un algoritmo *greedy* para agrupar los objetos. Como conclusión se prueba que, aunque el problema sea NP-hard, para las condiciones planteadas en el estudio, se pueden logran resultados factibles en tiempos computacionales aceptables.

3.4. Métodos de solución en la literatura

En la presente sección, se consideran los métodos de solución existentes en la literatura para los distintos problemas expuestos:

En Ceschia et al. (2013) , para lograr la solución se realizó una búsqueda local que tiene como fin generar una ruta de los clientes, ajustar las órdenes de acuerdo al orden de carga y seleccionar una heurística que determine la distribución actual de la carga en el vehículo, esto para cada vehículo y para cada tipo de objeto. Se determina la función de costo, que es la suma de los pesos de la función objetivo y la distancia factible, además se definen dos tipos de infactibilidad: el exceso de peso y el volumen descargado, haciendo alusión al peso de la carga y de los objetos no cargados. Luego se genera una solución inicial aleatoria en clientes y vehículos, y se definen tres relaciones entre los productos vecinos:

- Cambiar al cliente y la estrategia: consiste en eliminar un cliente de la ruta y situarlo en otra posición lo que hará que el bloque de ese cliente se mueva de una posición antigua a una nueva, agregando seis nuevos atributos (cliente, ruta nueva, ruta antigua, posición nueva, posición antigua y estrategia de carga), para finalmente asignar la estrategia a una nueva ruta y si el cliente no se mueve solo se cambia la estrategia.
- Intercambiar clientes: consiste en la identificación de dos clientes, rutas y posiciones de los productos, los cuales se intercambian manteniendo fija su posición y orientación.
- Mover y rotar bloques: selecciona un bloque de productos y se insertan en una nueva posición y otra ruta con atributos del tipo de caja las secuencia de carga, las posiciones nuevas y antiguas y la cantidad de cajas que se mueven. El bloque se divide en dos uno que se queda en la ruta antigua y el otro se asigna a una nueva ruta.

Para la implementación de la heurística de carga se desarrolla una extención de los algoritmos de tres dimensiones "*bottom left algorithm*" y "*touching perimeter algorithm*". Para cada objeto se escoge una posición entre todas las posibles posiciones dependiendo del vehículo donde se cargue y se hacen pruebas para ver la factibilidad del objeto. Otra estrategia utilizada es la de maximizar el área del contenedor tocada por los objetos cargados basado en el "*touching perimeter algorithm*". Las metaheurísticas utilizadas son un "*sequential solving aproach*" alternado a un "*simulated annealing algorithm*" y una "*large-neighbohood search*", el algoritmo de control es ilustrado al estilo de "*generalizad*

local search machines". Se presenta un grafo donde los nodos son las estrategias de carga y los arcos representan la transición de las estrategias. Cada componente parte con la mejor solución del proceso anterior y se obtienen soluciones de acuerdo a un proceso de recocido simulado, y si estas soluciones tienen errores de infactibilidad se crea un proceso que las repara y las vuelve factibles, además se crea un intensificador para cuando el algoritmo no mejore la función de costo. Finalmente se logran resultados de acuerdo a los metros cúbicos utilizados y los kilómetros recorridos por los vehículos.

En de Almeida and Figueiredo (2010), trata una heurística que usa el algoritmo CPBOX y el BOXCP, los cuales logran buenos resultados computacionales, además modifican el problema, agregando restricciones adicionales, y lo llaman 3D-Bicriteria agregando nuevas reglas para la heurística. Finalmente analizan los resultados obtenidos en tiempo y en espacio ocupado, y definen distintos tipos de cajas de acuerdo al tamaño recomendado y a la forma de cada caja.

La solución propuesta en Perboli et al. (2012), se enfoca en un modelo estocástico, se considera un conjunto de posibles escenarios con una variable aleatoria de beneficio de cada *pallet*, en cada escenario posible, que es una variable independiente e idénticamente distribuida, es escalada con una constante, que se le asigna una probabilidad y se logra una variable junto con el beneficio de cada *pallet*, que luego se cambia en el modelo matemático propuesto, obteniendo una función objetivo con la esperanza de la variable aleatoria multiplicado por la cantidad de cajas en un *pallet*.

En Zheng et al. (2015), se utiliza un algoritmo genético co-evolutivo multiobjetivo, que toma como entrada los datos del problema de programación lineal, resuelto con un modelo matemático y los parámetros del algoritmo genético (tamaño de la generación, la cantidad de población, el tamaño de la población, el rango elitista, el rango de cruzamiento y el rango de mutación), obteniendo como resultado una solución óptima de Pareto. En el algoritmo propuesto se presentan dos claves aleatorias para la codificación, la primera es para obtener la secuencia de empaquetado de las cajas (BPS) y la segunda es para obtener los tipos capas de las cajas utilizadas (BLT). El tipo de caja y las capas de esta se obtienen de acuerdo al BPS y BLT del cromosoma del AG que encuentra los máximos posibles

espacios vacíos de acuerdo a las coordenadas x, y y z, comparando desde el fondo abajo a la izquierda. Finalmente se experimenta con tres tipos de cajas con distintas dimensiones en un contenedor especificado, con los parámetros del AG dados, se obtienen resultados óptimos del modelo matemático y para el algoritmo genético co-evolutivo multiobjetivo se obtienen mejores resultados que en un AG simple.

Para el problema propuesto por Bozejko et al. (2015), proponen un algoritmo genético que presenta un cromosoma de dos partes, la primera representa el orden recibido por el procedimiento de las cajas y la segunda el número de rotaciones para cada caja, ambas partes del cromosoma tienen una longitud igual a la cantidad de cajas. El procedimiento de las cajas consiste que un cromosoma, que es la representación codificada de la solución considerada, una lista de elementos que contienen las coordenadas del sistema donde se puede ubicar una caja, las dimensiones del contenedor y la lista de las dimensiones de las cajas a almacenar y las cajas ya puestas en el contenedor. El procedimiento consiste en la posibilidad de insertar una caja en un espacio de la lista de espacios disponibles considerando su rotación. Debido a que es posible que exista más de un espacio disponible, para reducir espacio computacional se utiliza una implementación de hilos paralelos. Para encontrar soluciones codificadas para el cromosoma, que permitan una búsqueda eficiente del espacio de soluciones se usó un algoritmo de recocido simulado. Finalmente, se evalúa el algoritmo con cajas de 3 a 20 tipos diferentes y contenedores, teniendo mayores tiempos para mayor cantidad de cajas, obteniendo resultados prometedores para la cantidad de cajas que se evaluaron.

En el estudio de Paquay et al. (2016), se desarrolló un modelo matemático con las variables y parámetros de un problema de *bin packing* tradicional (dimensiones de los productos, contenedores, peso) y se evaluaron en el programa CPLEX y a medida que se fueron dando los resultados se escribió un código en java para preparar los datos y analizarlos. Se experimentó con tres series con distintos tipos de forma de contenedores, todos idénticos con forma paralelepípedo, todos idénticos con forma no paralelepípedo y con contenedores paralelepípedos o no, y se realizaron 5 instancias con cada contenedor y aumentando el número de cajas obteniendo tiempos bajos con pocas cajas y tiempos altos con hartas cajas a cargar, este proceso repetido para las tres series del problema que definieron.

En Alonso et al. (2017), se desarrolla un modelo de programación lineal entera incluyendo las bases presentadas para los *pallets*, enfocado en dos casos: *pallets* simples y *pallets* dobles. Para los *pallets* simples, se minimiza el número de contenedores utilizados, con restricciones de peso, espacio y utilización del *pallet*. Para los *pallets* dobles se agregan restricciones de altura y peso, además de que los *pallets* de arriba tienen que corresponder con sus *pallets* de abajo correspondiente. Finalmente se presentan soluciones con los dos tipos de *pallets* con distintos números de productos e instancias de las cuales 643 de 666 fueron resueltas en un tiempo de 5 min.

En Grange et al.(2018) se estudia una intencional del problema de bin packing, llamado VM packing, para el cual se utiliza un software genérico de programación, utilizando un modelo de programación lineal en una primera etapa, utilizando variables auxiliares para la solución del problema, con valores a los objetos cargados según el volumen que ocupan, posicionando los objetos según la relación de estos números mientras se van cargando. Se estudian 4 distintas heurísticas a utilizar para este problema, una heurística *greedy*, una heurística *greedy* especializada, la heurística "*Overload-and-Remove*", finalmente se prueba un algoritmo genético. Como se menciona en el estudio, se utilizaron heurísticas tanto *greedy* como no *greedy*, donde en ninguna se obtuvo una solución en tiempos no exponenciales.

En el estudio de Castro et al. (2019) se postula el *Balanced Fractional Bin Packing problem* (BFBPP) donde se asignan la misma cantidad de Objetos a todos los *bins* dentro los contenedores existentes, además considerando el *Balanced Multistage Bin Packing problem*, donde se asignan la misma cantidad de objetos a los *bins* en todas las etapas que pasa el contenedor. El problema principal (BFBPP) consiste en relajar la restricción del problema de *packing* original. En el *Balanced Multistage Bin Packing Problem*, se considera las mismas condiciones que el problema tratado anteriormente, pero esta vez asigna los objetos a los *bins* en distintas etapas, ocupando la misma proporción en las distintas fases de asignación, considerando que se minimice la utilización de *bins* en cada una de ellas, utilizando un algoritmo *greedy* para agrupar los objetos.
La Tabla 3.3 es un resumen de los artículos presentes en la literatura, estudiados en este

trabajo, con el método de solución, tipo de problema tratado en cada trabajo de los distintos autores, el máximo de cajas utilizadas en una instancias y el limite de tiempo que en cada estudio se consideró para obtener la solución.

Tabla 3.3: Métodos de solución en la literatura

Publicación	Método de solución	Tipo de problema	Máximo de cajas utilizadas	Limite de tiempo
de Almeida and Figueiredo (2010)	Heurística	3D-BPP	200	10000 s.
Perboli et al. (2012)	Exacto	3D-BPP	-	-
Ceschia et al. (2013)	Metaheurística	Bin packing con ruteo.	27	Entre 300 s y 10000 s.
Zheng et al. (2015)	Metaheurística	2D/3D-BPP	100	-
Bozejko et al. (2015)	Metaheurística	3D-BPP	5127	-
Paquay et al. (2016)	Exacto	3D-BPP	27	3600 s.
Alonso et al. (2017)	Exacto	3D-BPP	142	300 s.
Grange et al.(2018)	Heurística	VM packing	11	90 s.
Castro et al. (2019)	Heurística	BFBPP	9	-

3.5. Metaheurística *Harmony Search*

La Metaheurística Harmony search se propone en el estudio de Yang (2009) donde se plantea un algoritmo basado en la situación natural de la creación de una armonía, representando 3 situaciones diferentes, hacer una melodía a partir de una ya existente, variando una cantidad aleatoria de notas, hacer una melodía idéntica a otra ya compuesta o hacer una melodía totalmente nueva con notas totalmente aleatorias. Estos casos se ven reflejados en el pseudocódigo de la Figura 3.1.

Primero se determina la función objetivo que se utilizará, para definir qué se quiere obtener en el modelo, luego se genera una solución inicial con números de una solución no óptima, llamada *Initial Harmonic*. Se define el rango de ajuste para seleccionar qué etapa se usará: r_{accept} para escoger una solución ya generada y r_{pa} para hacer una melodía a partir de una ya existente. La iteración comienza definiendo una condición de detención de la heurística, en cada iteración, se considera la mejor solución encontrada hasta el momento, para luego, mediante la selección de números aleatorios se define en cuál de los tres sucesos va a variar la melodía en proceso: el uso de una harmonía que ya existe, ajustando algunas notas a la harmónica tratada, o generando una harmonía totalmente nueva con “notas” aleatorias, en cualquiera de los tres casos, elegir la mejor solución encontrada

Harmony Search

Begin

Función Objetico $f(x)$, $x = (x_1, x_2, ..., x_n)^T$
Generar Ïnitial Harmonics"(un vector de números reales)
Difinir el ratio de ajuste (r_{pa}),el limite y la cantidad que puede variar
Definir velocidad de aceptación de la memoria(r_{accept})
while (t<Máximo de iteraciones):
    Generar nuevas armónicas aceptando los mejores soluciones.
    Ajuste el tono para obtener nuevos armónicas (soluciones)
    if (rand $>$ (r_{accept})), Escoger una harmonía existente
    else if(rand $>$ (r_{pa})), Ajustar el tono al azar dentro de los límites
    else Generar nuevas armónicas a través de aleatorizaciones
    end if
    Aceptar la nueva harmónica (soluciones) si es mejor
end while
Encuentra la mejor solución actual

End

Figura 3.1: Metaheurística *Harmony Search* Yang (2009)

hasta el momento, respecto a las iteraciones realizadas. Una vez cumplido el criterio de detención, se selecciona la mejor solución encontrada. Se menciona en el estudio, que se recomienda utilizar para r_{accept} un valor dentro del rango de [0.7,0.95] y para r_{pa} un valor dentro del rango [0.1,0.5].

3.6. Estudios que utilizan la Metaheurística *Harmony Search*

En Yassen et al. (2015) se estudió el problema de *vehicle routing problem* con ventanas de tiempo, utilizando la metaheurística *harmony search*. Este problema consiste en minimizar el costo de transporte, con una cantidad de vehículos disponibles, satisfaciendo las necesidades de los clientes, con la restricción de que solo se puede entregar productos en una/s cierta/s ventana/s de tiempo. Para resolver este problema se utiliza el algoritmo básico de la *harmony search algorithm*, definiendo los parámetros de inicialización de la metaheurística. Luego, se define un híbrido entre el *HSA* básico y distintos métodos de solución, utilizando: *hill climbing algorithm*, *simulated annealing algorithm*, *great deluge algorithm*, *record-to-record traver algorithm*, *tabu search algorithm* y *page hinkley test algorithm*, todos algoritmos de local search para mejorar el algoritmo, dando una solución inicial calculada por el *HSA*. Se comparan los métodos de solución y se llega a la conclusión de que los métodos de búsqueda local mejoran la exploración del *HSA* siempre que se mantenga el equilibrio entre las combinaciones, dependiendo de los componentes principales de la *HSA* híbrida: los parámetros, el tipo de algoritmo de *local search* utilizado y la configuración del *local search*.

En Liu et al. (2020) se propone un algoritmo para mejorar la metaheurística *harmony search*, proporcionando un factor de comparación global del algoritmo, con un problema de ruteo de vehículos. Para encontrar la *initial harmony* y la ruta inicial de reparto, se utiliza un código de una harmonía natural, proponiendo que cada iteración se calcule por separado utilizando una nueva estrategia propuesta que consiste en probar distintos métodos para mejorar la diversidad de las soluciones encontradas, respecto a la solución inicial. La metaheurística se trabajó utilizando el software Visual C++ y se diseñan experimentos variando los distintos parámetros de esta, logrando resultado mejores que otros métodos estudiados, tanto en velocidad de convergencia como en eficiencia.

Capítulo 4
Metodología

4.1. Métodos utilizados

En este estudio se descompuso el problema de *packing* con ruteo, en los dos problemas que lo componen, utilizando un método exacto, en específico un modelo matemático, para resolver el problema de ruteo de vehículos, cuya solución sirvió para desarrollar el *input* de los parámetros utilizados en la metaheurística *harmony search*, para resolver el problema de *packing*. Se busca obtener la ruta óptima del modelo matemático y ordenar las dimensiones de las cajas a cargar en el camión de reparto, según el orden de entrega a los clientes, esto teniendo en cuenta el orden *lifo* de carga y descarga.
Para este caso es mejor abordar el problema separado, principalmente por la dificultad de programación que se tiene al abordar ambos problemas juntos en una metaheurística, lo que podría afectar a los tiempos de cómputo y al posicionamiento de las cajas. Para el caso de este estudio, enfocada en minimizar la distancia entre el punto de origen (0,0,0) y la posición de la caja, considerando el orden de entrega de la misma.

4.1.1. Ruta de entrega de los objetos

Para establecer el orden de carga de las cajas en el contenedor, dependiendo del orden en que estas sean entregadas al cliente, se resuelve un problema de ruteo de vehículos (VRP por sus siglas en inglés), con el fin de generar la ruta de entrega de los objetos, para esto se desarrolla una programación de un método exacto.

Se considera la cantidad de objetos que se cargaban en el contenedor, para ser repartidos en los clientes seleccionados previamente, utilizando solo un camión de reparto de ciertas características.

A continuación, se presentan los conjuntos y parámetros de la formulación propuesta, para conformar la ruta de entrega de productos, considerando el conjunto N, de clientes a visitar y con *n* el número total de clientes. Por otra parte, se tiene la distancia entre los clientes y la planta distribuidora, en metros en c_{ij}.

Conjuntos y Parámetros:

N : Conjunto de clientes.
n : Número de clientes.
c_{ij} : Distancia entre el cliente i y el cliente j. $\forall i, j \in N$

Variables:

x_{ij} : Variable binaria, que es 1 cuando se recorre el arco (i, j) y 0, en otro caso. $\forall i, j \in N$.
u_i : Variable entera que indica el orden de visita a los clientes i. $\forall i \in N$-$\{1\}$.

Función objetivo:

$$Minimize \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij} \tag{4.1}$$

La función objetivo 4.1 intenta minimizar la distancia recorrida por el camión de reparto.

Restricciones:

$$sujeto\ a:$$

$$\sum_{j \in N} x_{1j} = 1, \forall j \in N, j \neq 1 \tag{4.2}$$

$$\sum_{j \in N} x_{i1} = 1, \forall i \in N, i \neq 1 \tag{4.3}$$

$$u_i - u_j + |N| * x_{ij} = |N| - 1, \forall i, j \in N - \{1\} \tag{4.4}$$

$$u_1 = 0 \tag{4.5}$$

$$u_i \leq |N - 1|, \forall i \in N \tag{4.6}$$

Las expresiones 4.2 y 4.3 establecen, que todos los clientes sean visitados, la restricción 4.4 es para evitar sub tours en la ruta de reparto, la restricción 4.5 permite que el camión de reparto inicie el recorrido en la empresa de distribución y la restricción 4.6 establece el orden de visita no supere la cantidad de clientes.

$$x_{ij} \in \{0, 1\}, \forall i, j \in N \tag{4.7}$$

$$u_i \in \mathbb{N}, \forall i \in N \cup \{1\} \tag{4.8}$$

Finalmente, las restricciones 4.7 y 4.8, se refieren a la no negatividad de la variable u_i y que las variables x_{ij} son binarias.

4.2. Harmony Search

En este estudio, se consideró como una harmonía, una matriz con las posiciones de la esquina inferior izquierda trasera de cada caja a cargar en el camión de reparto, considerando un espacio de tres dimensiones(x,y,z), obteniendo una matriz de (Cantidad de cajas)x3, teniendo en cuenta que los objetos cargados al contenedor tienen una forma cúbi-

ca o paralelepípeda regular (con caras totalmente lisas y paralelas o perpendiculares, a los planos correspondientes).
Se consideró una función objetivo de minimización de la suma de las distancias entre la coordenada (0,0,0), definida como la esquina inferior izquierda trasera del camión, con una visión desde la parte posterior, hasta la esquina inferior izquierda trasera de cada caja que se tendrá que descargar, con un *input* previo del orden en que los clientes serán visitados, según una ruta definida como óptima.

Se definió una solución inicial, en este caso la *Initial Harmonics*, como un *layout* de las cajas, que formaba una solución factible, determinando la posición de las cajas (Figura 5.1) de una forma manual, que se consideró poco óptima, dado a las condiciones de la función objetivo, posicionando virtualmente una por una, las cajas en el contenedor teniendo en cuenta las dimensiones del resto de las cajas ya cargadas. Esta instancia es la que da el inicio al pseudocódigo y sirve como comparación inicial para las siguientes soluciones.
Se utilizó un tiempo máximo para terminar las iteraciones de la metaheurística, empezan-

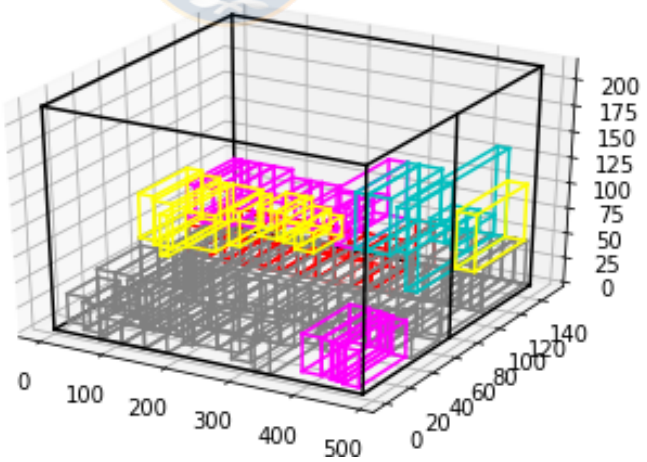

Figura 4.1: *Layout* Harmonía Inicial

do las iteraciones generando una nueva harmonía, para la primera iteración, será la *Initial Harmonic*, luego se evalúa el parámetro aleatorio *rand* para determinar cómo continua la iteración. Si *rand* es mayor que r_{accept}, pasa a copiar una harmonía que ya se haya evalua-

do anteriormente, buscada aleatoriamente, en este caso, utilizar una matriz de posiciones ya utilizada. Luego si el parámetro *rand* es mayor que r_{pa}, solo se cambian algunas notas de la harmónica correspondiente, ósea, se cambian algunos valores aleatorios de las localizaciones de las cajas. Finalmente, si no ocurre ninguna de las condiciones anteriores, se genera una harmonía totalmente nueva con notas aleatorias, generando una matriz con todas las posiciones de las cajas aleatorias.

Una vez terminado las condiciones de movimiento de las cajas, se termina la iteración y se compara con la última mejor solución encontrada, en el caso de la primera iteración se compara con la *Initial Harmonics*. En el caso que sea una mejor solución, se selecciona como la mejor solución encontrada. En cualquier caso, la solución entra de nuevo a las iteraciones hasta que se cumpla la condición de término, la cual, una vez que se cumpla, selecciona la mejor solución encontrada hasta el momento, lo que quiere decir las posiciones de las cajas que logran una menor distancia entre el punto (0,0,0) y las cajas a descargar, considerando descarga tipo *lifo*.

Las soluciones mencionadas anteriormente, para ser seleccionadas por la metaheurística, deben ser factibles, lo que implica que los objetos cargados no estén flotando en el contenedor, a lo que se considera como afectadas por la gravedad, y que no presenten *overlap* con otras cajas que se cargarán o que se hayan cargado con anterioridad, además de considerar el orden en que estas serán entregadas de una forma *lifo*. Una consideración importante dentro de este estudio es que los objetos cargados se posicionen dentro del camión de reparto, las posiciones de los objetos se determinan calculando la distancia del punto de origen (0,0,0), así las cajas se posicionan en un espacio (x, y, z) positivos, con límites según las dimensiones del camión de reparto, es de recalcar que una caja se puede cargar en la posición del límite del camión menos las dimensiones de esta. Las funciones de *Overlap* y *Gravedad* se inician luego de obtener una solución en cada iteración, pero antes del cálculo de la función objetivo, en cambio las consideraciones del orden a entregar las cajas, es una condición para la entrada de la información a la metaheurística, igualmente que las dimensiones del camión de entrega, dentro de las cuales, se deben mantener dichos objetos, las funciones mencionadas anteriormente están programadas para no salirse de los márgenes del camión de reparto.

4.2.1. Overlap

Una de las consideraciones para obtener soluciones factibles, a las condiciones físicas del problema es el *Overlap*, el cual significa que las cajas que sean cargadas, no se traslapen uno de otra con cajas que ya han sido cargadas con anterioridad. Esta función se realizó considerando las dimensiones de los objetos cargados y la posición en que estos se cargaban, obteniendo los siguientes tipos de sobreposición considerando un espacio de 3 dimensiones:

- Un vértice del objeto dentro de otro objeto;
- Una arista, o dos vértices, dentro de otro objeto;
- Una cara, o cuatro vértices, dentro de otro objeto;
- Un objeto totalmente dentro de otro u ocho vértices;
- Dos aristas de un objeto atravesando otro objeto, sin ningún vértice dentro de él;
- Cuatro aristas de un objeto atravesando otro objeto, sin ningún vértice dentro de él.

4.2.2. Gravedad

Otra consideración para la obtención de soluciones factibles, debido a las condiciones del problema, es la gravedad, que significa que los objetos cargados no estén suspendidos en el aire sin que exista un objeto que los soporte o en la base del camión de reparto en el plano X,Y (no se consideró en este caso el centro de gravedad de los objetos que estén sobre otros). Para esta consideración se toma en cuenta la superficie del contenedor, tanto como la de un posible objeto al cual se le pueda cargar otro en su parte superior, considerando los espacios donde tendrían que posicionarse, ya sea que no exista un objeto que los soporte. Se diferencia los objetos que después de una iteración de la heurística, estaban suspendidos en el aire, considerando la base y las dimensiones para posicionar los objetos en el camión de reparto y comparando la posición respecto a otras cajas que si estaban posicionadas en un lugar donde se podían sostener, ya sea en la base del camión de reparto o sobre otro objeto cargado, diferenciando ambas situaciones con una variable binaria, con 1 si estaban sobre otro objeto en la base del camión de reparto y 0 si estaban

en el aire. Además de estas consideraciones, se considera la posición de las cajas respecto a otras, si está esta por sobre la cara superior de otra caja o debajo, en el caso que no existan otras cajas para comparar, las cuales se reposicionaron conforme a lo explicado anteriormente. Las cajas que estaba en proceso se guardan en un vector.

4.2.3. Mover caja

La función Mover caja corresponde al movimiento que se realiza al tener una infactibilidad en las funciones anteriores, moviendo una caja previamente cargada en una de las direcciones posibles (x, y o z) aleatoriamente positivo, dejando esta en uno de los costados de la caja con la que presenta la infactibilidad, considerando los límites del camión, ya establecidos.

4.3. Modelo matemático utilizado en estudios anteriores (Barros et al.(2018))

A continuación se presentan los conjuntos y parámetros de la formulación propuesta en Barros (2018), considerando el conjunto N, de clientes a visitar, y el conjunto K, el de productos a repartir, con *n* y *k* siendo, el número total de clientes y productos respectivamente. Por otra parte se tiene la distancia entre los clientes y la planta distribuidora, en metros en c_{ij}, las dimensiones de los productos (w_{ki},l_{ki},h_{ki}) y las dimensiones del camión que los transportará (W_{max}, L_{max},H_{max}), considerando que las cajas se retiran: desde arriba hacia abajo, de derecha a izquierda y de adelante hacia atrás, respecto a la puerta del contenedor.
Conjuntos y Parámetros:

N	:	Conjunto de clientes.
K	:	Conjunto de productos.
n	:	Número de clientes.
k	:	Número de productos.
c_{ij}	:	Distancia entre el cliente i y el cliente j. $\forall i, j \in N$
w_{ki}	:	Ancho del producto k del cliente i. $\forall k \in K$, $\forall i \in N$
l_{ki}	:	Largo del producto k del cliente i. $\forall k \in K$, $\forall i \in N$
h_{ki}	:	Alto del producto k del cliente i. $\forall k \in K$, $\forall i \in N$

W_{max} : Ancho del camión.
L_{max} : Largo del camión.
H_{max} : Alto del camión.

Variables:

x_{ij} : Variable binaria, que es 1 cuando se recorre el arco (i, j) y 0, en otro caso. $\forall i, j \in N$.
u_i : Variable entera que indica el orden de visita a los clientes i. $\forall i \in N$-$\{1\}$.
W_k : Ubicación en el eje x de la esquina inferior izquierda de un producto. $\forall k \in K$.
L_k : Posición en el eje y de la esquina inferior izquierda de un producto. $\forall k \in K$.
H_k : Posición en el eje z de la esquina inferior izquierda de un producto. $\forall k \in K$.
a_{kg} : Variable binaria que es 1 si el producto g esta a la izquierda del producto k, o 0, en otro caso. $\forall k, g \in K$.
b_{kg} : Variable binaria que es 1 si el producto g esta a la derecha del producto k, o 0, en otro caso. $\forall k, g \in K$.
c_{kg} : Variable binaria que es 1 si el producto g esta al frente del producto k, o 0, en otro caso. $\forall k, g \in K$.
d_{kg} : Variable binaria que es 1 si el producto g esta atrás del producto k, o 0, en otro caso. $\forall k, g \in K$.
e_{kg} : Variable binaria que es 1 si el producto g esta encima del producto k, o 0, en otro caso. $\forall k, g \in K$.
f_{kg} : Variable binaria que es 1 si el producto g esta abajo del producto k, o 0, en otro caso. $\forall k, g \in K$.

Función objetivo:

$$\text{Minimize} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij} \tag{4.9}$$

La función objetivo 4.9 intenta minimizar la distancia recorrida por el camión de reparto.

Restricciones:

Sujeto a:

$$\sum_{j \in N} x_{1j} = 1, \forall j \in N, j \neq 1 \tag{4.10}$$

$$\sum_{j \in N} x_{i1} = 1, \forall i \in N, i \neq 1 \tag{4.11}$$

$$u_i - u_j + |N| * x_{ij} = |N| - 1, \forall i, j \in N - \{1\} \tag{4.12}$$

$$u_1 = 0 \tag{4.13}$$

$$u_i \leq |N - 1|, \forall i \in N \tag{4.14}$$

Las restricciones de la 4.10 y 4.11 establecen, que todos los clientes sean visitados, la restricción 4.12 es para evitar sub tours en la ruta de reparto, la restricción 4.13 permite que el camión de reparto inicie el recorrido en la empresa de distribución y la restricción 4.6 establece el orden de visita no supere la cantidad de clientes.

$$W_g + w_{gj} \leq W_k + W_{max}(1 - a_{kg}), \forall k, g \in K, k \neq g, \forall j \in N \tag{4.15}$$

$$W_k + w_{ki} \leq W_g + W_{max}(1 - b_{kg}), \forall k, g \in K, k \neq g, \forall i \in N \tag{4.16}$$

$$L_g + l_{gj} \leq L_k + L_{max}(1 - c_{kg}), \forall k, g \in K, k \neq g, \forall j \in N \tag{4.17}$$

$$L_k + l_{ki} \leq L_g + L_{max}(1 - d_{kg}), \forall k, g \in K, k \neq g, \forall i \in N \tag{4.18}$$

$$H_g + h_{gj} \leq H_k + H_{max}(1 - e_{kg}), \forall k, g \in K, k \neq g, \forall j \in N \tag{4.19}$$

$$H_k + h_{ki} \leq H_g + H_{max}(1 - f_{kg}), \forall k, g \in K, k \neq g, \forall i \in N \tag{4.20}$$

$$a_{kg} + b_{kg} + c_{kg} + d_{kg} + e_{kg} + f_{kg} \geq 1, \forall k, g \in K, k < g \tag{4.21}$$

Las restricciones 4.15 a 4.20 determinan la posición de la esquina inferior izquierda de una caja, con respecto a las otras cajas, que podrían estar arriba, abajo, adelante, atrás, a la derecha o a la izquierda de la posición de cualquier caja. La restricción 4.21 evita la superposición de las cajas.

$$a_{kg} + a_{gk} \leq 1, \forall k, g \in K, k \neq g \tag{4.22}$$
$$b_{kg} + b_{gk} \leq 1, \forall k, g \in K, k \neq g \tag{4.23}$$
$$c_{kg} + c_{gk} \leq 1, \forall k, g \in K, k \neq g \tag{4.24}$$
$$d_{kg} + d_{gk} \leq 1, \forall k, g \in K, k \neq g \tag{4.25}$$
$$e_{kg} + e_{gk} \leq 1, \forall k, g \in K, k \neq g \tag{4.26}$$
$$f_{kg} + f_{gk} \leq 1, \forall k, g \in K, k \neq g \tag{4.27}$$
$$a_{kg} + b_{kg} \leq 1, \forall k, g \in K, k \neq g \tag{4.28}$$
$$c_{kg} + d_{kg} \leq 1, \forall k, g \in K, k \neq g \tag{4.29}$$
$$e_{kg} + f_{kg} \leq 1, \forall k, g \in K, k \neq g \tag{4.30}$$

Las restricciones desde 4.22 a la 4.27 permiten que solo una caja este a un lado de otra caja. Las restricciones 4.28, 4.29 y 4.30 son para que las cajas no estén a la izquierda o a la derecha, alfrente o atrás, arriba o abajo con respecto a otra caja, al mismo tiempo.

$$W_k + w_{ki} \leq W_{max} \forall k \in K, \forall i \in N \tag{4.31}$$
$$L_k + l_{ki} \leq L_{max} \forall k \in K, \forall i \in N \tag{4.32}$$
$$H_k + h_{ki} \leq H_{max} \forall k \in K, \forall i \in N \tag{4.33}$$
$$\sum_{k \in K} H_k \leq \sum_{k=1}^{K} L_k \tag{4.34}$$
$$\sum_{k \in K} H_k \leq \sum_{k=1}^{K} W_k \tag{4.35}$$

Las restricciones 4.31, 4.32 y 4.33 establecen que las cajas no superen el límite del contenedor en sus 3 dimensiones, alto, largo y ancho, las restricciones 4.34 y 4.35 son para que no existan cajas elevadas sin alguna debajo de ella.

$$u_i - u_j \leq (|N| - 1)(1 - a_k g), \forall k, g \in K, k \neq g, \forall i, j \in N, i < j \tag{4.36}$$

$$2 * (u_i - u_j) \leq (|N| - 1)(1 - c_k g), \forall k, g \in K, k \neq g, \forall i, j \in N, i < j \quad (4.37)$$

$$3 * (u_i - u_j) \leq (|N| - 1)(1 - e_k g), \forall k, g \in K, k \neq g, \forall i, j \in N, i < j \quad (4.38)$$

$$u_i - u_j \leq (|N| - 1)(1 - b_k g), \forall k, g \in K, k \neq g, \forall i, j \in N, i < j \quad (4.39)$$

$$u_i - u_j \leq (|N| - 1)(1 - d_k g), \forall k, g \in K, k \neq g, \forall i, j \in N, i < j \quad (4.40)$$

$$u_i - u_j \leq (|N| - 1)(1 - f_k g), \forall k, g \in K, k \neq g, \forall i, j \in N, i < j \quad (4.41)$$

Las restricciones de la 4.36 a la 4.41 establecen el ordenan de las cajas en el orden de visita de las cajas. Los números adicionales de las restricciones 4.37 y 4.38 son para que el orden de reparto de las cajas se adecue al especificado, primero de arriba hacia abajo, luego de derecha a izquierda y finalmente de adelante hacia atrás.

$$x_{ij}, a_{kg}, b_{kg}, c_{kg}, d_{kg}, e_{kg}, f_{kg} \in \{0, 1\}, \forall k, g \in K, \forall i, j \in N \quad (4.42)$$

$$W_k, L_k, H_k, u_i \in \mathbb{N}, \forall k \in K, \forall i \in N \cup \{1\} \quad (4.43)$$

Finalmente las restricciones 4.42 y 4.43, se refieren a la no negatividad de las variables W_k, L_k, H_k, u_i y que las variables $x_{ij}, a_{kg}, b_{kg}, c_{kg}, d_{kg}, e_{kg}, f_{kg}$ son binarias.
La principal diferencia entre el método explicado anteriormente y la metaheurística realizada, es que, en el método exacto se evalúa una cierta cantidad de productos, con una cierta cantidad de clientes, para lo cual, el modelo entrega una solución óptima a la ruta a utilizar por el camión de reparto, asignando en el proceso la posición que deberían tener las cajas para está ruta. Por otro lado, la metaheurística realizada, busca obtener soluciones óptimas para el problema de packing, teniendo como *input* para la armonía inicial, la ruta óptima obtenida previamente en un modelo matemático.

4.4. Explicación pseudocódigo *overlap*

Al inicio del pseudocódigo, en línea 1, se define la cantidad de cajas a revisar, que es el número total de cajas que se cargarán al camión, considerando solo la que está en evaluación (caja i), para luego en la línea 2 definir la dimensión k que se trabajará considerando 3 dimensiones, para este caso X, Y y Z. En línea 3 se define la matriz que contiene el punto en el plano de k de la caja i, luego en línea 4 se definen las dimensiones físicas que tiene la caja i en el plano k. En línea 5 se define la caja a comparar en cada iteración u que tiene las misma cantidad de datos que i y en línea 6 se definen otras dimensiones para considerar t y r que igualmente consideran las 3 dimensiones mencionadas.
En línea 7 empieza un ciclo *for* para todas las dimensiones de k considerando que k sea distinto de t y r en las líneas 10 y 12 respectivamente. En línea 8 se inicia un ciclo *for* para i considerando todas las cajas, la cual tiene que ser distinto de u en 13, evitando que una caja se compare a si misma. En las líneas 9 y 11 empiezan el ciclo *for* las variables t y r respectivamente. En línea 14 se indica que mientas no se evalúen todas las condiciones, no se terminará la iteración haciendo que esta sea una consideración para factibilizar el problema, terminando al final de todas las consideraciones en la línea 48.
Desde la línea 15 empiezan las consideraciones para evaluar las distintas condiciones planteadas donde una caja puede estar *overlap* de otra, calculando las dimensiones de i en todos sus planos y comparándola con otra caja u en todos sus planos. En la línea 16 se encuentra la primera restricción considerada, en este caso, que una caja no tenga dos aristas dentro de otra, vista desde un punto de vista que la caja i esté sobre la caja u y en la línea 17 lo contrario. En la línea 18 se considera la restricción de que una caja no tenga 4 vértices dentro de otra. En la línea 19 la restricción de que una caja no esté totalmente dentro de otra, desde la línea 20 a la línea 25, que una caja no tenga 4 vértices dentro de otra, o una cara completamente dentro de otra vista desde los puntos de vista de las 6 caras de un objeto cuboide o paralelepipedo. Desde la línea 26 a la 37 se presentan la restricción de que 2 vértices de una caja i, no estén dentro de otra caja u, desde los distintos puntos de vista y dimensiones. En la línea 38 hasta la 45 las restricciones de que una caja i no tenga un solo vértice dentro de otra caja u, desde todos los puntos de vista con 8 vértices en un cubo o paralelepipedo. En la línea 46 y 47, la restricción de que una caja i no esté en las mismas dimensiones que otra caja u. En la línea 48, al terminar de evaluar todas

Pseudocódigo overlap

begin:

1. Caja a revisar: i:=1..N, N=Cantidad de Cajas a cargar
2. Dimensiones: k:=0,1,2
3. Punto del plano k donde se encuentra la caja i: $HA_{ik}[i][k]$
4. Dimensión k de la caja i como: $dimensiones[i][k]$
5. Caja a comparar: u
6. Otras dimensiones diferente a la revisada: t,r:=0,1,2
7. **for:** k en 0,1,2:
8. **for:** i en 1..N:
9. **for:** t en 0,1,2:
10. **if:** $t! = k$
11. **for:** r en 0,1,2:
12. **si:** $r! = t$ y $r! = k$
13. **for:** u en 1..N:
14. **while:** true:
15. Calcular espacio que ocupa la caja i en los distintos planos con $H_{ik}[i][k] + Dimensiones[i][k]$
    Comparar respecto a la caja u para todas las dimensiones
16. **if:**($HA_{ik}[i][k] <= HA_{ik}[u][k] < HA_{ik}[u][k] + Dimensiones[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[u][t] < HA_{ik}[i][t] < HA_{ik}[i][t] + Dimensiones[i][t] < HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[u][r] < HA_{ik}[i][r] < HA_{ik}[u][r] + Dimensiones[u][r] < HA_{ik}[i][r] + Dimensiones[i][r]$):
    pasar a función MoverCaja
17. **elif:**($HA_{ik}[i][k] <= HA_{ik}[u][k] < HA_{ik}[u][k] + Dimensiones[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[u][r] < HA_{ik}[i][r] < HA_{ik}[i][r] + Dimensiones[i][r] < HA_{ik}[u][r] + Dimensiones[u][r]$
    pasar a función MoverCaja
18. **elif:** ($HA_{ik}[i][k] <= HA_{ik}[u][k] and HA_{ik}[u][t] <= HA_{ik}[i][t]$ and $HA_{ik}[i][r] < HA_{ik}[u][r]$ and $HA_{ik}[i][t] + Dimensiones[i][t] < HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[u][r] + Dimensiones[u][r] < HA_{ik}[i][r] + Dimensiones[i][r]$ and $HA_{ik}[i][k] + Dimensiones[i][k] < HA_{ik}[u][k] + Dimensiones[u][k]$):
    pasar a función MoverCaja

19. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$):
    pasar a función MoverCaja
20. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[u][k] + Dimensiones[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
21. **elif:**($HA_{ik}[i][k] < HA_{ik}[u][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
22. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[u][t] + Dimensiones[u][t] < HA_{ik}[i][t] + Dimensiones[i][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
23. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[i][r] < HA_{ik}[u][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
24. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[i][r] + Dimensiones[i][r]$):
    pasar a función MoverCaja
25. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[i][t] < HA_{ik}[u][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja

26. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[u][t] + Dimensiones[u][t] < HA_{ik}[i][t] + Dimensiones[i][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[u][r] + Dimensiones[u][r] < HA_{ik}[i][r] + Dimensiones[i][r]$):
    pasar a función MoverCaja
27. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[u][k] + Dimensiones[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[u][t] < HA_{ik}[i][t] < HA_{ik}[u][t] + Dimensiones[u][t] <= HA_{ik}[i][t] + Dimensiones[i][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
28. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[u][k] + Dimensiones[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[u][r] + Dimensiones[u][r] < HA_{ik}[i][r] + Dimensiones[i][r]$):
    pasar a función MoverCaja
29. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[u][t] + Dimensiones[u][t] < HA_{ik}[i][t] + Dimensiones[i][t]$ and $HA_{ik}[i][r] < HA_{ik}[u][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
30. **elif:**($HA_{ik}[i][k] < HA_{ik}[u][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][t] < HA_{ik}[i][t] < HA_{ik}[u][t] + Dimensiones[u][t] <= HA_{ik}[i][t] + Dimensiones[i][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
31. **elif:**($HA_{ik}[i][k] <= HA_{ik}[u][k] < HA_{ik}[i][k] + Dimensiones[i][k] < HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[u][r] + Dimensiones[u][r] < HA_{ik}[i][r] + Dimensiones[i][r]$):
    pasar a función MoverCaja

32. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[u][k] + Dimensiones[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[i][r] < HA_{ik}[u][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
33. **elif:**($HA_{ik}[i][k] < HA_{ik}[u][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[i][r] < HA_{ik}[u][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
34. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[i][t] < HA_{ik}[u][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[i][r] < HA_{ik}[u][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
35. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[i][t] < HA_{ik}[u][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[u][r] + Dimensiones[u][r] < HA_{ik}[i][r] + Dimensiones[i][r]$):
    pasar a función MoverCaja
36. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[u][k] + Dimensiones[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[i][t] < HA_{ik}[u][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
37. **elif:**($HA_{ik}[i][k] < HA_{ik}[u][k] < HA_{ik}[i][k] + Dimensiones[i][k] <= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[i][t] < HA_{ik}[u][t] < HA_{ik}[i][t] + Dimensiones[i][t] <= HA_{ik}[u][t] + Dimensiones[u][t]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
38. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[u][k] + Dimensiones[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] < HA_{ik}[u][r] + Dimensiones[u][r] < HA_{ik}[i][r] + Dimensiones[i][r]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[u][t] + Dimensiones[u][t] < HA_{ik}[i][t] + Dimensiones[i][t]$):
    pasar a función MoverCaja

39. **elif:**($HA_{ik}[i][k] < HA_{ik}[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ $<= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] <$ $HA_{ik}[u][r] + Dimensiones[u][r] < HA_{ik}[i][r] + Dimensiones[i][r]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[u][t] + Dimensiones[u][t] <$ $HA_{ik}[i][t] + Dimensiones[i][t]$):
    pasar a función MoverCaja
40. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[u][k] + Dimensiones[u][k]$ $< HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[i][r] < HA_{ik}[u][r] <$ $HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[u][t] + Dimensiones[u][t] <$ $HA_{ik}[i][t] + Dimensiones[i][t]$):
    pasar a función MoverCaja
41. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[u][k] + Dimensiones[u][k]$ $< HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] <$ $HA_{ik}[u][r] + Dimensiones[u][r] < HA_{ik}[i][r] + Dimensiones[i][r]$ and $HA_{ik}[i][t] < HA_{ik}[u][t] < HA_{ik}[i][t] + Dimensiones[i][t] <=$ $HA_{ik}[u][t] + Dimensiones[u][t]$):
    pasar a función MoverCaja
42. **elif:**($HA_{ik}[i][k] < HA_{ik}[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ $<= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[i][r] < HA_{ik}[u][r] <$ $HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$ and $HA_{ik}[u][t] <= HA_{ik}[i][t] < HA_{ik}[u][t] + Dimensiones[u][t] <$ $HA_{ik}[i][t] + Dimensiones[i][t]$):
    pasar a función MoverCaja
43. **elif:**($HA_{ik}[i][k] < HA_{ik}[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ $<= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[u][r] <= HA_{ik}[i][r] <$ $HA_{ik}[u][r] + Dimensiones[u][r] < HA_{ik}[i][r] + Dimensiones[i][r]$ and $HA_{ik}[i][t] < HA_{ik}[u][t] < HA_{ik}[i][t] + Dimensiones[i][t] <=$ $HA_{ik}[u][t] + Dimensiones[u][t]$):
    pasar a función MoverCaja
44. **elif:**($HA_{ik}[u][k] <= HA_{ik}[i][k] < HA_{ik}[u][k] + Dimensiones[u][k]$ $< HA_{ik}[i][k] + Dimensiones[i][k]$ and $HA_{ik}[i][r] < HA_{ik}[u][r] <$ $HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$ and $HA_{ik}[i][t] < HA_{ik}[u][t] < HA_{ik}[i][t] + Dimensiones[i][t] <=$ $HA_{ik}[u][t] + Dimensiones[u][t]$):
    pasar a función MoverCaja

45. **elif:**($HA_{ik}[i][k] < HA_{ik}[u][k] < HA_{ik}[i][k] + Dimensiones[i][k]$ $<= HA_{ik}[u][k] + Dimensiones[u][k]$ and $HA_{ik}[i][r] < HA_{ik}[u][r] <$ $HA_{ik}[i][r] + Dimensiones[i][r] <= HA_{ik}[u][r] + Dimensiones[u][r]$ and $HA_{ik}[i][t] < HA_{ik}[u][t] < HA_{ik}[i][t] + Dimensiones[i][t] <=$ $HA_{ik}[u][t] + Dimensiones[u][t]$):
    pasar a función MoverCaja
46. **elif:**($HA_{ik}[i][k] + Dimensiones[i][k]$ $== HA_{ik}[u][k] + Dimensiones[u][k]$and $HA_{ik}[i][t]$ $+Dimensiones[i][t] == HA_{ik}[u][t] + Dimensiones[u][t]$and HA$_{ik}[i][r] + Dimensiones[i][r] == HA_{ik}[u][r] + Dimensiones[u][r]$):
    pasar a función MoverCaja
47. **elif:**($HA_{ik}[i][k] == HA_{ik}[u][k]$ and $HA_{ik}[i][r] == HA_{ik}[u][r]$ and $HA_{ik}[i][t] == HA_{ik}[u][t]$):
    pasar a función MoverCaja
    else:
48. break

end

las condiciones, se termina el algoritmo y terminan las evaluaciones para seguir con los ciclos *for* mencionados. Cabe mencionar que todas las condiciones mencionadas, si no se cumplen, pasan a la función movercaja, la cual mueve la caja respecto a su posición actual.

4.5. Explicación pseudocódigo gravedad

En la línea 1 se define la posición de la caja i en el camión de reparto, en la línea 2 se crea una matriz MA vacía para guardar información. En la línea 3 se identifican las cajas que están en la base del camión, con eje Z=0 y se les asigna un valor 1. En la línea 4 se identifican las cajas que no están volando y se les asigna un valor 0 en la matriz MA, quedando la matriz MA con valores 1 para las cajas que no están volando y 0 si las cajas están volando, con la cantidad de productos que se están evaluando. En la línea 5 se agrega un vector llamado compatibles, vacío para agregar cajas que están volando.
En la línea 6 se evalúan solo las cajas que están volando, comparando en la línea 7 con otra caja u, la cual no tiene que estar volando y ser distinta a la caja i, según la línea 8. En la línea 9 se evalúa que la caja i esté parcialmente sobre otra, según la proyección de sus planos X e Y, si no es (línea 10), la caja u se agrega al vector compatible en la línea 11. En la línea 12, se consideran las cajas que están volando, solo si el vector compatible no esté vació, en la línea 13 para todas las cajas u en el vector compatible, se evalúa si la caja i está parcialmente sobre u, si es así, la caja i cambia su estado a no estar flotando, teniendo un valor 1 en la matriz MA, si no es así, continua el código en la línea 14 donde se remueve la caja u de matriz compatible.
En la línea 16 se agrega la matriz vacía vuelan, en la línea 17 se evalúan las cajas i en la cantidad de productos, solo para las cajas que se encuentran suspendidas en la línea 18 donde se agregan a la matriz vuelan el número de la caja i y la posición en Z de la caja i. En la línea 20 se evalúan todas las cajas en la matriz vuelan, donde i es igual a el primer valor de la matriz, vaciando la matriz compatibles, luego en la línea 21, se evalúan todas las cajas u en los productos evaluados, en la línea 22, solo si la caja u es distinta a la caja i y la caja u no está flotando. En la línea 21 solo si la caja u esta completamente debajo de la caja i, la caja u se agrega a la lista de compatibles. En la línea 24 se evalúa la caja u en la matriz compatible, agregando la matriz borrar, donde si la caja u no está parcialmente sobre la caja i, se agrega u a la matriz borrar, para que en la línea 21 quitar las cajas que están en la matriz borrar de la matriz compatibles.
En la línea 27, si compatibles es una matriz vacía, se define una cota máxima que es igual a -1 y una caja máxima que es igual a 0, para que nada pueda sobrepasar dichos limites, después de que todas las cajas hayan pasado por la heurística, para luego en la línea 28,

Pseudocódigo gravedad

begin:

1. Definir posición de caja i en el camión de reparto, para las dimensiónk con i en 1..N con N=cantidad de productos y k en 0,1,2 para las distintas dimensiones
2. Definir matriz MA vacía
3. Identificar las cajas que están en la base del camión(dimensión Z=0), con valor 1
4. identificar el total de cajas que no están volando, las cuales tienen valor 0 en la matriz MA
5. Vector compatibles vacío para agregar cajas que están volando
6. **if** MA[i]=0:
7. **for** u en (Cantidad de Productos):
8. **if** MA[u]=1 y i!=u:
9. **if** $HA_{ik}[u][1] <= HA_{ik}[i][1]$ and $HA_{ik}[u][1] + Dimensiones[u][1] >= HA_{ik}[i][1] + Dimensiones[i][1]$ and $HA_{ik}[u][0] <= HA_{ik}[i][0]$ and $HA_{ik}[u][0] + Dimensiones[u][0] >= HA_{ik}[i][0] + Dimensiones[i][0]$:
   MA[i]=1
10. **else:**
11. agregar a compatible[u]
12. **if** MA[i]=0 y len(compatible)!=0:
13. **for** u en compatibles:
14. **if** $HA_{ik}[u][0] + Dimensiones[u][0] > HA_{ik}[i][0]$ and $HA_{ik}[u][0] < HA_{ik}[i][0] + Dimensiones[i][0]$ and $HA_{ik}[u][1] + Dimensiones[u][1] > HA_{ik}[i][1]$ and $HA_{ik}[u][1] < HA_{ik}[i][1] + Dimensiones[i][1]$:
    MA[i]=1
15. **else**:
    remover de compatibles[u]
16. Agregar matriz vacía vuelan
17. **for** i en (Cantidad de Productos):
18. **if** MA[i]=0:
19. Agregar a matriz vuelan $(i, HA_{ik}[i][2])$
20. **for** $w$en (vuelan):

i=w[0]

compatibles = vacío

21. **for** u en (productos):
22. **if** $i! = u$ and MA[u] == 1:
23. **if**$HA_{ik}[u][2] + Dimensiones[u][2] < HA_{ik}[i][2]$:
agregar u a compatibles
24. **for** u in compatibles:
agregar *borrar* matriz vacía
25. **if not** $HA_{ik}[u][0] + Dimensiones[u][0] > HA_{ik}[i][0]$
and $HA_{ik}[u][0] < HA_{ik}[i][0] + Dimensiones[i][0]$
and $HA_{ik}[u][1] + Dimensiones[u][1] > HA_{ik}[i][1]$
and $HA_{ik}[u][1] < HA_{ik}[i][1] + Dimensiones[i][1]$:
agregar u a matriz *"borrar"*
26. compatibles es igual a una lista de cajas en compatibles menos cajas en borrar
27. **if** compatibles=[]:
definir cotaMax = -1
definir cajaMax = 0
28. **for** u en compatibles:
29. **if** $HA_{ik}[u][2] + Dimensiones[u][2] >= cotaMax$:
cotaMax es igual a la posición de la
caja u mas las dimensiones de la caja u en el eje z
cajaMax es u
30. **if** cotaMax != -1:
La posición de la caja i en el eje z es igual a la posición de la caja cajaMax
mas su dimensión en el eje z
La caja no flota por lo que MA[i] = 1
31. **else**:
la caja i se posiciona en la base del camión
La caja no flota, por lo que MA[i] = 1
32.**end**

evaluar todas las cajas u que están en compatibles, si la altura de la caja i es mayor a la cota máxima definida anteriormente, se define la cota máxima como la posición de la caja u mas la dimensión de dicha caja y pasa a ser la nueva caja máxima. En la línea 30 si la cota máxima es distinto a -1, la posición de la caja i es igual a la de la caja máxima mas su dimensión, pasando a ser una caja que no flota teniendo el valor de 1 en la matriz MA. Finalmente en la línea 31, si no se cumple la condición de la línea 27, la caja i pasa a estar en la base del camión, con un valor en la matriz MA de 1. Una vez terminados todos los ciclos *for* termina la heurística en la línea 32.

4.6. Explicación pseudocódigo mover caja

En la línea 1 se utilizan los parámetros de la función anterior, overlap, HA_{ik}, k, i, r, tyu, se definen las dimensiones a utilizar en la línea 2, para las cajas correspondientes. En la línea 3 se define las dimensiones del camión, las cuales no podrán ser sobrepasadas, luego se seleccionan 3 variables vacías A, B y C, en la línea 4. En la línea 5 mientras no se revisen todas las condiciones, no se terminará el algoritmo, luego en la línea 6, se asignan aleatoriamente a A, B y C un valor de 0 o 1, para definir en que dimensión se moverá la caja, si A es 1 se mueve en la dirección X, si B es 1 se mueve en la dirección Y y si C es 1 se mueve en la dirección Z, este movimiento se debe hacer en solo una dirección por lo que esta asignación se termina solo si el valor de la suma de las variables es igual a 1. En la línea 8 se mueven las cajas según lo obtenido anteriormente, en la dirección asignada, según la dimensión de la caja u en que se encuentra el overlap. Finalmente en las líneas 9, 10 y 11, se revisa si alguno de estos movimientos hizo que una de las cajas se moviera fuera de los limites del camión de reparto, para en el caso de que así sea, se mueve aleatoriamente la caja en una de las direcciones restantes y volver a la caja a la posición original de la dimensión revisada.

Pseudocódigo mover caja

begin:

1. Tomar los parámetros desde la función *overlap* HA_{ik}, k, i, r, tyu
2. Dimensión k de la caja i como: $dimensiones[i][k]$
3. MX: valor máximo para las dimensiones del camión de carga, en dimensiones X, Y y Z
4. Seleccionar 3 variables vacías A, B, C
5. **while:** true
6. Asignar aleatoriamente 0 o 1 a las variables A,B y C
7. **if:** A+B+C=1:
   break
8. Mover la caja seleccionada a uno de los lados de la caja, considerando
   que si el valor de la variable A es 1 se mueve en la dimensión k,
   si el valor B es 1 se mueve en la dirección t, y finalmente
   si el valor de C es 1 se mueve en la dimensión r
9. **if** $MX[k] - Dimensiones[i][k] < HA_{ik}[i][k]$:
   Asignar la posición de la caja al valor máximo que puede llegar dentro del camión de reparto para la dimensión k
   Asignar aleatoriamente 0 o 1 a la variable WA
   La posición de la caja en las dimensiones restantes queda como el mínimo de una de estas, dependiendo de la variable aleatoria WA
10. **if** $MX[t] - Dimensiones[i][t] < HA_{ik}[i][t]$:
    Asignar la posición de la caja al valor máximo que puede llegar dentro del camión de reparto para la dimensión t
    Asignar aleatoriamente 0 o 1 a la variable WA
    La posición de la caja en las dimensiones restantes queda como el mínimo de una de estas, dependiendo de la variable aleatoria WA
11. **if** $MX[r] - Dimensiones[i][r] < HA_{ik}[i][r]$:
    Asignar la posición de la caja al valor máximo que puede llegar dentro del camión de reparto para la dimensión r
    Asignar aleatoriamente 0 o 1 a la variable WA
    La posición de la caja en las dimensiones restantes queda como el mínimo de una de estas, dependiendo de la variable aleatoria WA

12.**end**

Capítulo 5
Resultados computacionales

En este capítulo se presentan los resultados obtenidos al resolver el problema planteado mediante la propuesta metodológica, implementado en el IDE Spyder versión 3.3.3, utilizando el lenguaje de programación Python 3.7, en un computador HP con un procesador Intel Inside core i7, con 8 GB de ram y 2.2 GHz.
La ruta de entrega de los objetos cargados en el contenedor se obtuvo con el solver comercial CPLEX versión 12.7.1 en el mismo equipo.

Por otra parte, el modelo matemático de las instancias a comparar, también se calculó en el solver comercial CPLEX versión 12.7.1 en el mismo equipo.

5.1. Obtención de la instancia de prueba

Para obtener la ruta óptima, se replicó un pedido de entrega de cajas a una cierta cantidad de clientes (5), según un pedido real de la empresa donde se obtuvieron los datos para este estudio, considerando la distancia real entre los clientes seleccionados, para el modelo matemático.
La instancia utilizada para el problema de *packing* es una caracterización de un camión con dimensiones de 485 cm. de largo, 150 cm. de ancho y 215 cm. de alto, con 101 cajas cargadas a este camión, de los clientes mencionados anteriormente, estas son débilmente heterogéneas cuboides o paralelepipedas.

Para este estudio se seleccionó una única ruta de reparto con 5 clientes y 101 cajas, esta ruta se obtuvo mediante la solución del problema matemático de un VRP, utilizando una matriz de distancia real entre clientes reales, iniciando la ruta en el centro de distribución, donde el primer cliente visitado tiene 11 productos asignados, el segundo 7, el tercero 17, el cuarto 56 y el último 10. La ruta de entrega es de CD-C1-C2-C3-C4-C5-CD.

	CD	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Cliente 5
CD	0	24200	24150	25000	23000	24200
Cliente 1	23400	0	3500	4200	4500	3500
Cliente 2	23300	3200	0	55	350	600
Cliente 3	23000	4000	55	0	400	450
Cliente 4	24000	3900	450	400	0	400
Cliente 5	23500	4100	550	450	400	0

Tabla 5.1: Distancias entre clientes y CD en metros

5.2. Solución inicial

La solución inicial se obtuvo tomando en cuenta las consideraciones del problema, gravedad, overlap y limites del camión, haciendo una solución trivial, de forma manual y que sirva como cota máxima de las soluciones encontradas, con una función objetivo de 1449721 cm., este se mantiene fijo para cada instancia realizada, para tomarlo como un punto de comparación inicial igual para cada iteración, considerando el orden de entrega inverso al determinado en este estudio, para el orden de carga de las cajas, como se ve en la Figura 5.1.

5.3. Parametrización de la Metaheurística

Para este estudio se consideraron 8 horas de tiempo de cómputo considerado el doble del tiempo medio entre las iteraciones obtenidas en la solución del modelo matemático. Para el *layout* de las cajas en el camión de reparto, se realizaron 20 combinaciones de parámetros dentro de los rangos recomendados en la literatura, con $r_{pa} = [0{,}1; 0, 5]$y

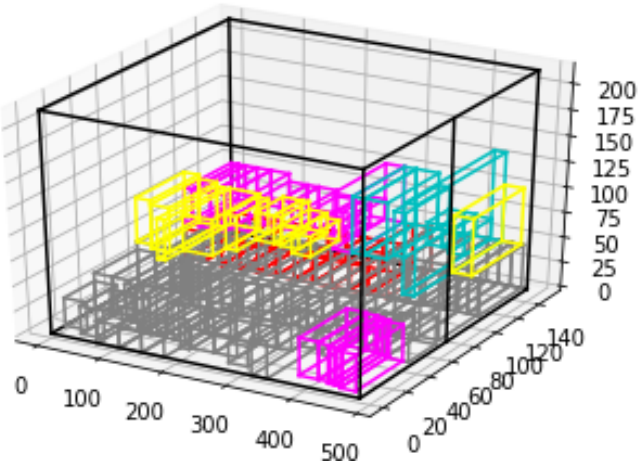

Figura 5.1: *Layout* Harmonía Inicial

$r_{accept} = [0{,}7; 0{,}95]$.

En la Tabla 5.2 se muestran las soluciones obtenidas con la variación de los parámetros. En la primera columna se muestra el parámetro r_{pa} que se consideró, en la segunda se muestra el parámetro r_{accept} utilizado, en la tercera se muestra la mejor función objetivo encontrada en este tiempo, en centímetros, considerando esta función objetivo como la mencionada en el capítulo anterior y en la cuarta columna se muestra el tiempo en que se encontraron las mejores soluciones factibles en segundos, cabe mencionar que todas estas pruebas se realizaron bajo la misma semilla de aleatoriedad, por lo que son comparables, al ser evaluados con la misma secuencia de número aleatorios.

Tabla 5.2: Pruebas.

r_{pa}	r_{accept}	Función Objetivo cm.	Tiempo s.
$0,1$	$0,7$	1236512	8451
$0,1$	$0,8$	1141456	26558
$0,1$	$0,9$	1188430	28801
$0,1$	$0,95$	1188711	16625
$0,2$	$0,7$	1101593	7909
$0,2$	$0,8$	1171486	28070

r_{pa}	r_{accept}	Función Objetivo cm.	Tiempo s.
$0,2$	$0,9$	1159351	25021
$0,2$	$0,95$	1182989	3831
$0,3$	$0,7$	1181799	25935
$0,3$	$0,8$	1180139	10224
$0,3$	$0,9$	1116959	17360
$0,3$	$0,95$	1179308	19017
$0,4$	$0,7$	1201389	7825
$0,4$	$0,8$	1164736	24517
$0,4$	$0,9$	1184625	27629
$0,4$	$0,95$	1205995	4609
$0,5$	$0,7$	1192109	7059
$0,5$	$0,8$	1216178	540
$0,5$	$0,9$	1171280	24362
$0,5$	$0,95$	1127328	20597

A continuación se muestran las imágenes de los resultados de dichas pruebas, con un gráfico de convergencia (Figura 5.2) y el *layout* final donde se tendrán que poner los objetos cargados(Figura 5.3).

En la Figura 5.2 se observa la función objetivo (eje vertical) y la progresión que esta tiene en el tiempo (eje horizontal) para $r_{pa} = 0{,}2$ y $r_{accept} = 0{,}7$, en este caso la mejor solución se encuentra en 7909 segundos, con 1101593 cm, corresponde a la mejor solución encontrada, con la combinación de los parámetros recomendados en la literatura. Analizando el gráfico se observa la variación de las soluciones en el área inferior, encontrando un total de 29 óptimos locales, la Figura 5.3, visualiza el *layout* que deberían tener los objetos, para la mejor solución encontrada teniendo un orden de carga de rojo, gris, magenta, celeste y finalmente amarillo, con la regla de descargar de arriba hacia abajo, de derecha a izquierda y de adelante hacia atrás según el orden de entrega, en este caso se ve una distribución cargada hacia la pared posterior del camión de reparto, esto es esperable de la función objetivo utilizada, dado al valor que se les da a las cajas según su posición, priorizando que estas se encuentren cargadas hacia el punto de origen caracterizado como la esquina inferior izquierda trasera del camión de reparto, desde un punto de vista de la

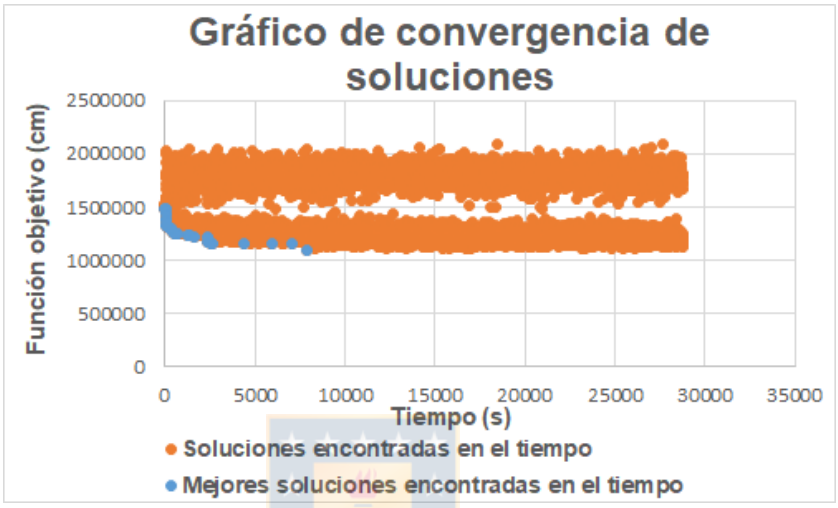

Figura 5.2: Gráfico con $r_{pa} = 0{,}2$, $r_{accept} = 0{,}7$

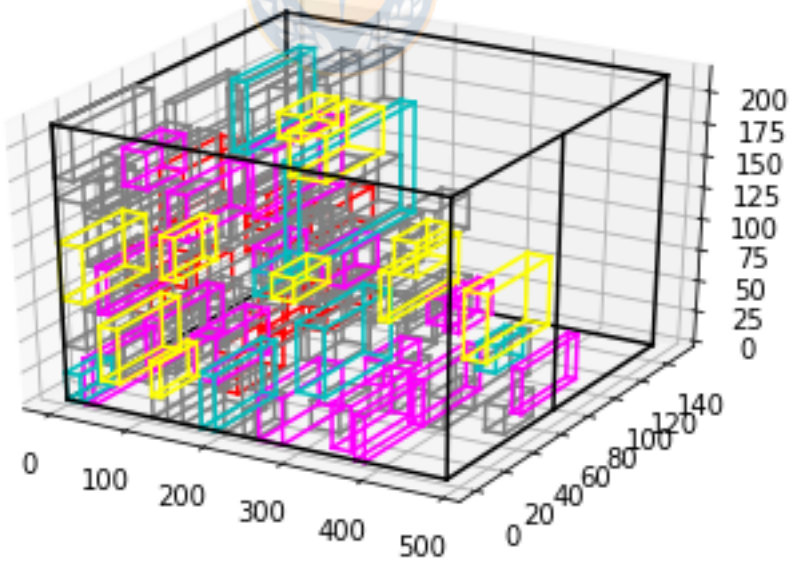

Figura 5.3: *Layout* con $r_{pa} = 0{,}2$, $r_{accept} = 0{,}7$

puerta de descarga. Además podemos hablar sobre el *layout* representado de las mejores soluciones encontradas, que se observa qué, este quizás no representa un orden de entrega ideal para la descarga de los objetos cargados, pero dado a las características del problema, se considera que una solución ideal será encontrada en tiempos computacionales mucho mayores, por las condiciones del problema establecidas.

5.4. Pruebas con instancias de estudios anteriores

En el estudio realizado con un modelo matemático, se realizaron 3 instancias con el modelo matemático propuesto: con 40 cajas, 101 cajas y 122 cajas para 4, 5 y 6 clientes respectivamente. Se realizaron 3 instancias con la misma cantidad de cajas en el camión de reparto, con las mismas cajas cargadas. Analizando los resultados, se obtuvieron soluciones en tiempos computacionales menores, ya que en dicho estudio, se obtuvieron las soluciones óptimas en 40 segundos, 3969 segundos y 39000 segundos, para 40, 101 y 122 cajas respectivamente. Las principales diferencias de estos dos estudios recae en dos puntos principales: la obtención de las soluciones y posicionamiento de las cajas. El primer estudio, al ser un modelo matemático, tiende a tener un mayor tiempo de cómputo para una mayor cantidad de cajas a cargar en el camión, obteniendo una solución óptima en el proceso, por otro lado, con la metaheurística utilizada, se obtienen soluciones de una manera mas rápida, pero para esta cantidad de cajas, logra soluciones óptimas locales, las que no se podrían decir que son soluciones óptimas globales. Por el tipo de problema, para una mayor cantidad de cajas, con un modelo matemático, sería imposible lograr soluciones, el aumento exponencial del tiempo de cómputo del estudio anterior, refleja en parte, dicho acontecimiento.

Para las instancias utilizadas se obtuvieron los resultados de la tabla 5.3 evaluando la función objetivo para los resultados del posicionamiento de las cajas obtenidos del modelo matemático y de la metaheurística utilizada, para la metaheurística se utilizaron los mismos valores para r_{pa} y r_{accept} (0,2 y 0,7 respectivamente) utilizados en la mejor prueba realizada y las mismas dimensiones para las cajas.

	Metaheurística		Modelo Matemático	
Cantidad de Cajas	Función Objetivo cm.	Tiempo s.	Función Objetivo cm.	Tiempo s.
40	124653	28407	209419	40
101	1101593	23818	1124875	3969
122	1816471	7909	1891035	39000

Tabla 5.3: Instancias realizadas

Se observa las distintas mejores soluciones encontradas para las instancias, estas son considerablemente distintas dado a la cantidad de cajas que se cargan y que la función objetivo utilizada depende de esta cantidad. El modelo matemático no encuentra mejores soluciones que la metaheurística, ya que, la función objetivo del modelo es diferente a la utilizada en este estudio. El modelo matemático encuentra la solución óptima para el problema conjunto de *packing* con ruteo, a diferencia de la metaheurística que encuentra la solución únicamente para el problema de *packing* considerando como *input* la información de la ruta óptima. La comparación se da para evaluar el uso y tiempo de respuesta de ambos problemas, que aunque obtengan una función objetivo distinta, son de naturaleza similar, tomando en cuenta la posición de las cajas para ambos estudios, utilizando métodos de solución distintos.

Capítulo 6
Conclusiones

En este trabajo se abordó el problema de ruteo de vehículos con *packing* de tres dimensiones, con cajas no homogéneas y un contenedor. Existen pocas investigaciones en la literatura de la unión de estos dos problemas y en lo revisado no se utiliza el método que se utilizó en este estudio (la metaheurística *harmony search* para resolver el problema de *packing* y ruteo de vehículos en conjunto).

Para solucionar el problema de ruteo de vehículos con *packing* de tres dimensiones, se realizó un código en Python basado en la metaheurística *harmony search* a modo exploratorio dado al uso de esta en la literatura. Esta clase de métodos, las metaheurísticas, son utilizadas para solucionar los problemas de *packing* y ruteo por separado, en la literatura. El modelo intenta ordenar los objetos cargados dentro del camión de reparto según cómo estos serán entregados por la ruta óptima entregada por el VRP, como *input* del código realizado, considerando que las cajas se entregan con prioridad de arriba hacia abajo, de derecha a izquierda y de adelante hacia atrás. El código de la metaheurística fue programado en el IDE Spyder 3.3.3 en el lenguaje Python 3.7, teniendo en consideración el tamaño de las instancias y las condiciones naturales del problema. Las pruebas se obtienen en tiempos relativamente pequeños una de otra, con soluciones factibles dentro de lo esperado.

Por otra parte, se hicieron comparaciones con un estudio anterior, donde se realizó un modelo matemático para resolver el conjunto del problema de *packing* con ruteo, con las mismas condiciones del presente estudio (mismas cajas, camión y clientes). Las principales comparaciones entre estos dos métodos, son el tiempo de ejecución para instancias cada vez más grandes y al posicionamiento de las cajas, con respecto a las condiciones de

factibilidad planteada. También, se considera la posibilidad de agregar mas características al estudio mediante un método metaheurístico, sin obtener necesariamente un mayor tiempo de cómputo. Por otra parte, la programación del código de la metaheurística, da una mayor libertad al incluir ciertas restricciones que naturalmente se presentarían en el problema, además de poder considerar el movimiento de las cajas y si estas presentan algún tipo de infactiblidad planteada en este problema. Se evaluaron los resultados del posicionamiento de las cajas para ambos métodos, en la función objetivo utilizada para la metaheurística, obteniendo mejores resultados en el procesamiento de la metaheurística que en el modelo matemático. Esto se debe a la diferencia entre los objetivos de ambos estudios, demostrando que la metaheurística si da buenos resultados, mejorando los obtenidos por el modelo matemático, refiriéndose a la posición de las cajas.

Como trabajo futuro, se propone refinar la metaheurística utilizada programando nuevas funciones, basadas en los métodos utilizados en estudios encontrados en la literatura y mejorar la programación de las condiciones del problema, cómo son overlap y gravedad, ya que en este estudio se sentaron precedentes para una visión de como abordar estas consideraciones. Además, unir el problema de *packing* y ruteo completamente, para que la ruta utilizada no sea un *input*, sino que se haga en conjunto dentro del mismo código. También, incluir otros factores al problema, como la fragilidad de las cajas, la rotación, con distintos tipos de vehículos, dado a las distintas empresas de reparto que están asociadas a la empresa de *packing*, demanda variable, la perecibilidad de un producto, ventanas de tiempo de entrega, prioridad de la demanda, entre otros; intentando hacer el modelo más realista y a su vez con una posible aplicación real, con la creación de un software o aplicación para una futura automatización de procesos, para este caso, de la carga del camión de reparto.

Capítulo 7

Referencias

Yang, X. S. (2009). Harmony search as a metaheuristic algorithm. In Music-inspired harmony search algorithm (pp. 1-14). Springer, Berlin, Heidelberg.

de Almeida, A., & Figueiredo, M. B. (2010). A particular approach for the Three-dimensional Packing Problem with additional constraints. Computers & Operations Research, 37(11), 1968-1976.

Perboli, G., Tadei, R., & Baldi, M. M. (2012). The stochastic generalized bin packing problem. Discrete Applied Mathematics, 160(7-8), 1291-1297.

Adar, R., & Epstein, L. (2013). Selfish bin packing with cardinality constraints. Theoretical Computer Science, 495, 66-80.

Liao, C. S., & Hsu, C. H. (2013). New lower bounds for the three-dimensional orthogonal bin packing problem. European Journal of Operational Research, 225(2), 244-252.

Ceschia, S., Schaerf, A., & Stützle, T. (2013). Local search techniques for a routing-packing problem. Computers & industrial engineering, 66(4), 1138-1149.

Zheng, J. N., Chien, C. F., & Gen, M. (2015). Multi-objective multi-population biased random-key genetic algorithm for the 3-D container loading problem. Computers & Industrial Engineering, 89, 80-87.

Yassen, E. T., Ayob, M., Nazri, M. Z. A., & Sabar, N. R. (2015). Meta-harmony search algorithm for the vehicle routing problem with time windows. Information Sciences, 325, 140-158.,

Bożejko, W., Kacprzak, Ł., & Wodecki, M. (2015, August). Parallel packing procedure for three dimensional bin packing problem. In 2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR) (pp. 1122-1126). IEEE.

Iori, M. (2016). An annotated bibliography of combined routing and loading problems. Yugoslav Journal of Operations Research, 23(3).

Alonso, M. T., Alvarez-Valdes, R., Parreño, F., & Tamarit, J. M. (2016). Algorithms for pallet building and truck loading in an interdepot transportation problem. Mathematical Problems in Engineering, 2016.

Paquay, C., Schyns, M., & Limbourg, S. (2016). A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. International Transactions in Operational Research, 23(1-2), 187-213.

Zhao, X., Bennell, J. A., Bektaş, T., & Dowsland, K. (2016). A comparative review of 3D container loading algorithms. International Transactions in Operational Research, 23(1-2), 287-320.

Alonso, M. T., Alvarez-Valdes, R., Iori, M., Parreño, F., & Tamarit, J. M. (2017). Mathematical models for multicontainer loading problems. Omega, 66, 106-117.

Barros Vásquez, M., & Medina, Rosa. (2018). Modelo matemático de PLE para optimización del problema de bin packing con ruteo.

Heßler, K., Gschwind, T., & Irnich, S. (2018). Stabilized branch-and-price algorithms for vector packing problems. European Journal of Operational Research, 271(2), 401-419.

Grange, A., Kacem, I., & Martin, S. (2018). Algorithms for the bin packing problem with overlapping items. Computers & Industrial Engineering, 115, 331-341.

ANÁLISIS ECONÓMICO DEL TRANSPORTE DE CARGA NACIONAL (2018), MTT - Ministerio de Transportes y Telecomunicaciones de Chile, www.subtrans.cl /upload/estudios/ CargaNacional-IF.pdf, 28/Marzo/2018

Castro-Silva, D., & Gourdin, E. (2019). A study on load-balanced variants of the bin packing problem. Discrete Applied Mathematics, 264, 4-14.

Küçükoğlu, İ., & Öztürk, N. (2019). A hybrid meta-heuristic algorithm for vehicle routing and packing problem with cross-docking. Journal of Intelligent Manufacturing, 30(8), 2927-2943.

Liu, L., Huo, J., Xue, F., & Dai, Y. (2020). Harmony Search Method with Global Sharing Factor Based on Natural Number Coding for Vehicle Routing Problem. Information, 11(2), 86.

Capítulo 8

Anexos: Código Utilizado en Lenguaje de Programación Python

```
import numpy as np
import random
from time import time
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
random.seed(0)
np.random.seed(0)
def dibujar(HA_ik,Productos,Dimensiones):
ProductosCliente1=10
ProductosCliente2=66
ProductosCliente3=83
ProductosCliente4=90
ProductosCliente5=101
ProductosCliente6=122
fig = plt.figure()
x0=HA_ik[:, [0]]
y0=HA_ik[:, [1]]
z0=HA_ik[:, [2]]
x1=HA_ik[:, [0]]+Dimensiones[:,[0]]
y1=HA_ik[:, [1]]
```

```
z1=HA_ik[:, [2]]
x2=HA_ik[:, [0]]+Dimensiones[:,[0]]
y2=HA_ik[:, [1]]+Dimensiones[:,[1]]
z2=HA_ik[:, [2]]
x3=HA_ik[:, [0]]
y3=HA_ik[:, [1]]+Dimensiones[:,[1]]
z3=HA_ik[:, [2]]
x4=HA_ik[:, [0]]
y4=HA_ik[:, [1]]
z4=HA_ik[:, [2]]+Dimensiones[:,[2]]
x5=HA_ik[:, [0]]+Dimensiones[:,[0]]
y5=HA_ik[:, [1]]
z5=HA_ik[:, [2]]+Dimensiones[:,[2]]
x6=HA_ik[:, [0]]+Dimensiones[:,[0]]
y6=HA_ik[:, [1]]+Dimensiones[:,[1]]
z6=HA_ik[:, [2]]+Dimensiones[:,[2]]
x7=HA_ik[:, [0]]
y7=HA_ik[:, [1]]+Dimensiones[:,[1]]
z7=HA_ik[:, [2]]+Dimensiones[:,[2]]
ax1 = fig.add_subplot(111, projection='3d')
ax1.plot((0,0),(0,150),(0,0),color="black")
ax1.plot((485,485),(0,150),(0,0),color="black")
ax1.plot((0,485),(150,150),(0,0),color="black")
ax1.plot((485,485),(150,150),(0,215),color="black")
ax1.plot((0,0),(150,150),(0,215),color="black")
ax1.plot((0,485),(150,150),(215,215),color="black")
ax1.plot((0,0),(0,150),(215,215),color="black")
for i in range(Productos):
if i<=ProductosCliente1: ax1.plot((x0[i][0],x1[i][0]),(y0[i][0],y1[i][0]),(z0[i][0],z1[i][0]),
color=red")
ax1.plot((x1[i][0],x2[i][0]),(y1[i][0],y2[i][0]),(z1[i][0],z2[i][0]), color=red")
ax1.plot((x2[i][0],x3[i][0]),(y2[i][0],y3[i][0]),(z2[i][0],z3[i][0]), color=red")
```

```
ax1.plot((x0[i][0],x4[i][0]),(y0[i][0],y4[i][0]),(z0[i][0],z4[i][0]), color=red”)
ax1.plot((x4[i][0],x5[i][0]),(y4[i][0],y5[i][0]),(z4[i][0],z5[i][0]), color=red”)
ax1.plot((x5[i][0],x6[i][0]),(y5[i][0],y6[i][0]),(z5[i][0],z6[i][0]), color=red”)
ax1.plot((x6[i][0],x7[i][0]),(y6[i][0],y7[i][0]),(z6[i][0],z7[i][0]), color=red”)
ax1.plot((x4[i][0],x7[i][0]),(y4[i][0],y7[i][0]),(z4[i][0],z7[i][0]), color=red”)
ax1.plot((x0[i][0],x3[i][0]),(y0[i][0],y3[i][0]),(z0[i][0],z3[i][0]), color=red”)
ax1.plot((x2[i][0],x6[i][0]),(y2[i][0],y6[i][0]),(z2[i][0],z6[i][0]), color=red”)
ax1.plot((x3[i][0],x7[i][0]),(y3[i][0],y7[i][0]),(z3[i][0],z7[i][0]), color=red”)
ax1.plot((x1[i][0],x5[i][0]),(y1[i][0],y5[i][0]),(z1[i][0],z5[i][0]), color=red”)
elif i>ProductosCliente1 and i<=ProductosCliente2:
ax1.plot((x0[i][0],x1[i][0]),(y0[i][0],y1[i][0]),(z0[i][0],z1[i][0]), color=”grey”)
ax1.plot((x1[i][0],x2[i][0]),(y1[i][0],y2[i][0]),(z1[i][0],z2[i][0]), color=”grey”)
ax1.plot((x2[i][0],x3[i][0]),(y2[i][0],y3[i][0]),(z2[i][0],z3[i][0]), color=”grey”)
ax1.plot((x0[i][0],x4[i][0]),(y0[i][0],y4[i][0]),(z0[i][0],z4[i][0]), color=”grey”)
ax1.plot((x4[i][0],x5[i][0]),(y4[i][0],y5[i][0]),(z4[i][0],z5[i][0]), color=”grey”)
ax1.plot((x5[i][0],x6[i][0]),(y5[i][0],y6[i][0]),(z5[i][0],z6[i][0]), color=”grey”)
ax1.plot((x6[i][0],x7[i][0]),(y6[i][0],y7[i][0]),(z6[i][0],z7[i][0]), color=”grey”)
ax1.plot((x4[i][0],x7[i][0]),(y4[i][0],y7[i][0]),(z4[i][0],z7[i][0]), color=”grey”)
ax1.plot((x0[i][0],x3[i][0]),(y0[i][0],y3[i][0]),(z0[i][0],z3[i][0]), color=”grey”)
ax1.plot((x2[i][0],x6[i][0]),(y2[i][0],y6[i][0]),(z2[i][0],z6[i][0]), color=”grey”)
ax1.plot((x3[i][0],x7[i][0]),(y3[i][0],y7[i][0]),(z3[i][0],z7[i][0]), color=”grey”)
ax1.plot((x1[i][0],x5[i][0]),(y1[i][0],y5[i][0]),(z1[i][0],z5[i][0]), color=”grey”)
elif i>ProductosCliente2 and i<=ProductosCliente3:
ax1.plot((x0[i][0],x1[i][0]),(y0[i][0],y1[i][0]),(z0[i][0],z1[i][0]), color=”magenta”)
ax1.plot((x1[i][0],x2[i][0]),(y1[i][0],y2[i][0]),(z1[i][0],z2[i][0]), color=”magenta”)
ax1.plot((x2[i][0],x3[i][0]),(y2[i][0],y3[i][0]),(z2[i][0],z3[i][0]), color=”magenta”)
ax1.plot((x0[i][0],x4[i][0]),(y0[i][0],y4[i][0]),(z0[i][0],z4[i][0]), color=”magenta”)
ax1.plot((x4[i][0],x5[i][0]),(y4[i][0],y5[i][0]),(z4[i][0],z5[i][0]), color=”magenta”)
ax1.plot((x5[i][0],x6[i][0]),(y5[i][0],y6[i][0]),(z5[i][0],z6[i][0]), color=”magenta”)
ax1.plot((x6[i][0],x7[i][0]),(y6[i][0],y7[i][0]),(z6[i][0],z7[i][0]), color=”magenta”)
ax1.plot((x4[i][0],x7[i][0]),(y4[i][0],y7[i][0]),(z4[i][0],z7[i][0]), color=”magenta”)
ax1.plot((x0[i][0],x3[i][0]),(y0[i][0],y3[i][0]),(z0[i][0],z3[i][0]), color=”magenta”)
```

ax1.plot((x2[i][0],x6[i][0]),(y2[i][0],y6[i][0]),(z2[i][0],z6[i][0]), color=”magenta”)
ax1.plot((x3[i][0],x7[i][0]),(y3[i][0],y7[i][0]),(z3[i][0],z7[i][0]), color=”magenta”)
ax1.plot((x1[i][0],x5[i][0]),(y1[i][0],y5[i][0]),(z1[i][0],z5[i][0]), color=”magenta”)
elif i>ProductosCliente3 and i¡=ProductosCliente4:
ax1.plot((x0[i][0],x1[i][0]),(y0[i][0],y1[i][0]),(z0[i][0],z1[i][0]), color=”c”)
ax1.plot((x1[i][0],x2[i][0]),(y1[i][0],y2[i][0]),(z1[i][0],z2[i][0]), color=”c”)
ax1.plot((x2[i][0],x3[i][0]),(y2[i][0],y3[i][0]),(z2[i][0],z3[i][0]), color=”c”)
ax1.plot((x0[i][0],x4[i][0]),(y0[i][0],y4[i][0]),(z0[i][0],z4[i][0]), color=”c”)
ax1.plot((x4[i][0],x5[i][0]),(y4[i][0],y5[i][0]),(z4[i][0],z5[i][0]), color=”c”)
ax1.plot((x5[i][0],x6[i][0]),(y5[i][0],y6[i][0]),(z5[i][0],z6[i][0]), color=”c”)
ax1.plot((x6[i][0],x7[i][0]),(y6[i][0],y7[i][0]),(z6[i][0],z7[i][0]), color=”c”)
ax1.plot((x4[i][0],x7[i][0]),(y4[i][0],y7[i][0]),(z4[i][0],z7[i][0]), color=”c”)
ax1.plot((x0[i][0],x3[i][0]),(y0[i][0],y3[i][0]),(z0[i][0],z3[i][0]), color=”c”)
ax1.plot((x2[i][0],x6[i][0]),(y2[i][0],y6[i][0]),(z2[i][0],z6[i][0]), color=”c”)
ax1.plot((x3[i][0],x7[i][0]),(y3[i][0],y7[i][0]),(z3[i][0],z7[i][0]), color=”c”)
ax1.plot((x1[i][0],x5[i][0]),(y1[i][0],y5[i][0]),(z1[i][0],z5[i][0]), color=”c”)
elif i>ProductosCliente4 and i¡=ProductosCliente5:
ax1.plot((x0[i][0],x1[i][0]),(y0[i][0],y1[i][0]),(z0[i][0],z1[i][0]), color=”$yellow$”)
ax1.plot((x1[i][0],x2[i][0]),(y1[i][0],y2[i][0]),(z1[i][0],z2[i][0]), color=”$yellow$”)
ax1.plot((x2[i][0],x3[i][0]),(y2[i][0],y3[i][0]),(z2[i][0],z3[i][0]), color=”$yellow$”)
ax1.plot((x0[i][0],x4[i][0]),(y0[i][0],y4[i][0]),(z0[i][0],z4[i][0]), color=”$yellow$”)
ax1.plot((x4[i][0],x5[i][0]),(y4[i][0],y5[i][0]),(z4[i][0],z5[i][0]), color=”$yellow$”)
ax1.plot((x5[i][0],x6[i][0]),(y5[i][0],y6[i][0]),(z5[i][0],z6[i][0]), color=”$yellow$”)
ax1.plot((x6[i][0],x7[i][0]),(y6[i][0],y7[i][0]),(z6[i][0],z7[i][0]), color=”$yellow$”)
ax1.plot((x4[i][0],x7[i][0]),(y4[i][0],y7[i][0]),(z4[i][0],z7[i][0]), color=”$yellow$”)
ax1.plot((x0[i][0],x3[i][0]),(y0[i][0],y3[i][0]),(z0[i][0],z3[i][0]), color=”$yellow$”)
ax1.plot((x2[i][0],x6[i][0]),(y2[i][0],y6[i][0]),(z2[i][0],z6[i][0]), color=”$yellow$”)
ax1.plot((x3[i][0],x7[i][0]),(y3[i][0],y7[i][0]),(z3[i][0],z7[i][0]), color=”$yellow$”)
ax1.plot((x1[i][0],x5[i][0]),(y1[i][0],y5[i][0]),(z1[i][0],z5[i][0]), color=”$yellow$”)
elif i>ProductosCliente5 and i<=ProductosCliente6:
ax1.plot((x0[i][0],x1[i][0]),(y0[i][0],y1[i][0]),(z0[i][0],z1[i][0]), color=”green”)
ax1.plot((x1[i][0],x2[i][0]),(y1[i][0],y2[i][0]),(z1[i][0],z2[i][0]), color=”green”)

```
ax1.plot((x2[i][0],x3[i][0]),(y2[i][0],y3[i][0]),(z2[i][0],z3[i][0]), color=”green”)
ax1.plot((x0[i][0],x4[i][0]),(y0[i][0],y4[i][0]),(z0[i][0],z4[i][0]), color=”green”)
ax1.plot((x4[i][0],x5[i][0]),(y4[i][0],y5[i][0]),(z4[i][0],z5[i][0]), color=”green”)
ax1.plot((x5[i][0],x6[i][0]),(y5[i][0],y6[i][0]),(z5[i][0],z6[i][0]), color=”green”)
ax1.plot((x6[i][0],x7[i][0]),(y6[i][0],y7[i][0]),(z6[i][0],z7[i][0]), color=”green”)
ax1.plot((x4[i][0],x7[i][0]),(y4[i][0],y7[i][0]),(z4[i][0],z7[i][0]), color=”green”)
ax1.plot((x0[i][0],x3[i][0]),(y0[i][0],y3[i][0]),(z0[i][0],z3[i][0]), color=”green”)
ax1.plot((x2[i][0],x6[i][0]),(y2[i][0],y6[i][0]),(z2[i][0],z6[i][0]), color=”green”)
ax1.plot((x3[i][0],x7[i][0]),(y3[i][0],y7[i][0]),(z3[i][0],z7[i][0]), color=”green”)
ax1.plot((x1[i][0],x5[i][0]),(y1[i][0],y5[i][0]),(z1[i][0],z5[i][0]), color=”green”)
ax1.plot((0,485),(0,0),(0,0),color=”black”)
ax1.plot((485,485),(0,150),(215,215),color=”black”)
ax1.plot((0,485),(0,0),(215,215),color=”black”)
ax1.plot((485,485),(75,75),(0,215),color=”black”)
ax1.plot((485,485),(0,0),(0,215),color=”black”)
ax1.plot((0,0),(0,0),(0,215),color=”black”)
plt.show()
Solucion_Inicial =np.genfromtxt(’SolucionInicial122.csv’, dtype=int,delimiter=’,’)
Dimensiones=np.genfromtxt(’Dimensiones3_122p.csv’, dtype=int ,delimiter=’,’)
Ceros=np.genfromtxt(’Ceros122.csv’, dtype=int,delimiter=’,’)
Guarda_todo=[]
Guarda_Optimo=[]
tiempo_optimo=[]
tiempo_no_optimo=[]
HA_ik = []
MatrizOptima=[]
MatrizOptima2=[]
MX=[]
MX=[485,150,215]dimensiones camión
Harmonica=[]
Optimo = 0
elapsed_time = 0
```

```
iteracion = 0
r_acept =0.7  Probabilidad en Paper 0.7-0.95
r_pa = 0.2  Probabilidad en Paper 0.1-0.5
Productos =int(input (Çantidad de productos: "))
TiempoDeComputo = int(input ("Tiempo de ejecución: "))
starting_point = time()
a = np.arange(Productos,0,-1)
MA=np.zeros(Productos)
Hik = Ceros
HA_ik = Solucion_Inicial
HA = Solucion_Inicial
def LimitesCamion(HA_ik,i): las cajas no superan los limites del camión
if MX[0]-Dimensiones[i][0]<=HA_ik[i][0]:
HA_ik[i][0]=(MX[0]-Dimensiones[i][0])para que al momento de poner las cajas aleatoriamente no se salga
los limites del camión, todas las lineas de esta función
if MX[1]-Dimensiones[i][1]<=HA_ik[i][1]:
HA_ik[i][1]=(MX[1]-Dimensiones[i][1])
if MX[2]-Dimensiones[i][2]<=HA_ik[i][2]:
HA_ik[i][2]=(MX[2]-Dimensiones[i][2])
if HA_ik[i][0]<0:
HA_ik[i][0]=0
if HA_ik[i][1]<0:
HA_ik[i][1]=0
if HA_ik[i][2]<0:
HA_ik[i][2]=0
return HA_ik,i
def MoverCaja2(HA_ik,k,i,r,t,u):
while True:
A=random.choice([0,1])
B=random.choice([0,1])
C=random.choice([0,1])
```

```
    if A+B+C==1:
    break
    HA_ik[i][k]=(HA_ik[u][k]+Dimensiones[u][k])*A+HA_ik[i][k]*(1-A)
    HA_ik[i][t]=(HA_ik[u][t]+Dimensiones[u][t])*B+HA_ik[i][t]*(1-B)
    HA_ik[i][r]=(HA_ik[u][r]+Dimensiones[u][r])*C+HA_ik[i][r]*(1-C)
    if MX[k]-Dimensiones[i][k]<HA_ik[i][k]:
    HA_ik[i][k]=MX[k]-Dimensiones[i][k]
    WA=random.choice([0,1])
    HA_ik[i][r]=(min(HA_ik[:Productos,r]))*WA+HA_ik[i][r]*(1-WA)
    HA_ik[i][t]=(min(HA_ik[:Productos,t]))*(1-WA)+HA_ik[i][t]*WA
    if MX[t]-Dimensiones[i][t]<HA_ik[i][t]:
    HA_ik[i][t]=MX[t]-Dimensiones[i][t]
    WA=random.choice([0,1])
    HA_ik[i][r]=(min(HA_ik[:Productos,r]))*WA+HA_ik[i][r]*(1-WA)
    HA_ik[i][k]=(min(HA_ik[:Productos,k]))*(1-WA)+HA_ik[i][k]*WA
    if MX[r]-Dimensiones[i][r]<HA_ik[i][r]:
    HA_ik[i][r]=MX[r]-Dimensiones[i][r]
    WA=random.choice([0,1])
    HA_ik[i][k]=(min(HA_ik[:Productos,k]))*WA+HA_ik[i][k]*(1-WA)
    HA_ik[i][t]=(min(HA_ik[:Productos,t]))*(1-WA)+HA_ik[i][t]*WA
    return HA_ik
    def Overlap(HA_ik,k,i):
    "Se cambia la posicion de las cajas si estan total o parcialmente dentro de otro"
    for t in range(3):
    if k != t:
    for r in range(3):la idea es revisar todas las cajas comparadas a todas las cajas en todas
las posiciones
    if r != t and r != k:
    for u in range(Productos):
    if u!=i:
    while True: if (HA_ik[i][k]<=HA_ik[u][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]
+Dimensiones[i][k] and HA_ik[u][t]<HA_ik[i][t]<HA_ik[i][t]+Dimensiones[i][t]<HA_ik[u][t]
```

```
+Dimensiones[u][t] and HA_ik[u][r]<HA_ik[i][r]<HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]
+Dimensiones[i][r]):
    MoverCaja2(HA_ik,k,i,r,t,u)
    elif (HA_ik[i][k]<=HA_ik[u][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]+Dimensiones[i][k]
    and HA_ik[u][r]<HA_ik[i][r]<HA_ik[i][r]+Dimensiones[i][r]<HA_ik[u][r]+Dimensiones[u][r]
    and HA_ik[i][t]<HA_ik[u][t]<HA_ik[i][t]+Dimensiones[i][t]<HA_ik[u][t]+Dimensiones[u][t]):
    MoverCaja2(HA_ik,k,i,r,t,u)
    elif (HA_ik[i][k]<=HA_ik[u][k] and HA_ik[u][t]<=HA_ik[i][t] and HA_ik[i][r]<HA_ik[u][r]
    and HA_ik[i][t]+Dimensiones[i][t]<HA_ik[u][t]+Dimensiones[u][t]
    and HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]+Dimensiones[i][r]
    and HA_ik[i][k]+Dimensiones[i][k]<HA_ik[u][k]+Dimensiones[u][k]):
    MoverCaja2(HA_ik,k,i,r,t,u)
    elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
    and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]
    and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]):
    MoverCaja2(HA_ik,k,i,r,t,u)
    elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]+Dimensiones[i][k]
    and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
    and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
    MoverCaja2(HA_ik,k,i,r,t,u)
    elif (HA_ik[i][k]<HA_ik[u][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
    and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
    and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
    MoverCaja2(HA_ik,k,i,r,t,u)
    elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
    and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[u][t]+Dimensiones[u][t]<HA_ik[i][t]+Dimensiones[i][t]
    and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
    MoverCaja2(HA_ik,k,i,r,t,u)
    elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
    and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
    and HA_ik[i][r]<HA_ik[u][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
    MoverCaja2(HA_ik,k,i,r,t,u)
```

```
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]+Dimensiones[i][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[i][t]<HA_ik[u][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[u][t]+Dimensiones[u][t]<HA_ik[i][t]+Dimensiones[i][t]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]+Dimensiones[i][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]+Dimensiones[i][k]
and HA_ik[u][t]<HA_ik[i][t]<HA_ik[u][t]+Dimensiones[u][t]<=HA_ik[i][t]+Dimensiones[i][t]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]+Dimensiones[i][k]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]+Dimensiones[i][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[u][t]+Dimensiones[u][t]<HA_ik[i][t]+Dimensiones[i][t]
and HA_ik[i][r]<HA_ik[u][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[i][k]<HA_ik[u][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[u][t]<HA_ik[i][t]<HA_ik[u][t]+Dimensiones[u][t]<=HA_ik[i][t]+Dimensiones[i][t]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[i][k]<=HA_ik[u][k]<HA_ik[i][k]+Dimensiones[i][k]<HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]+Dimensiones[i][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
```

```
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]+Dimensiones[i][k]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[i][r]<HA_ik[u][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif(HA_ik[i][k]<HA_ik[u][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[i][r]<HA_ik[u][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[i][t]<HA_ik[u][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[i][r]<HA_ik[u][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[i][t]<HA_ik[u][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]+Dimensiones[i][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]+Dimensiones[i][k]
and HA_ik[i][t]<HA_ik[u][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[i][k]<HA_ik[u][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[i][t]<HA_ik[u][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]+Dimensiones[i][k]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]+Dimensiones[i][r]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[u][t]+Dimensiones[u][t]<HA_ik[i][t]+Dimensiones[i][t]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[i][k]<HA_ik[u][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]+Dimensiones[i][r]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[u][t]+Dimensiones[u][t]<HA_ik[i][t]+Dimensiones[i][t]):
MoverCaja2(HA_ik,k,i,r,t,u)
```

```
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]+Dimensiones[i][k]
and HA_ik[i][r]<HA_ik[u][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[u][t]+Dimensiones[u][t]<HA_ik[i][t]+Dimensiones[i][t]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]+Dimensiones[i][k]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]+Dimensiones[i][r]
and HA_ik[i][t]<HA_ik[u][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[i][k]<HA_ik[u][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[i][r]<HA_ik[u][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]
and HA_ik[u][t]<=HA_ik[i][t]<HA_ik[u][t]+Dimensiones[u][t]<HA_ik[i][t]+Dimensiones[i][t]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[i][k]<HA_ik[u][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[u][r]<=HA_ik[i][r]<HA_ik[u][r]+Dimensiones[u][r]<HA_ik[i][r]+Dimensiones[i][r]
and HA_ik[i][t]<HA_ik[u][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[u][k]<=HA_ik[i][k]<HA_ik[u][k]+Dimensiones[u][k]<HA_ik[i][k]+Dimensiones[i][k]
and HA_ik[i][r]<HA_ik[u][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]
and HA_ik[i][t]<HA_ik[u][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[i][k]<HA_ik[u][k]<HA_ik[i][k]+Dimensiones[i][k]<=HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[i][r]<HA_ik[u][r]<HA_ik[i][r]+Dimensiones[i][r]<=HA_ik[u][r]+Dimensiones[u][r]
and HA_ik[i][t]<HA_ik[u][t]<HA_ik[i][t]+Dimensiones[i][t]<=HA_ik[u][t]+Dimensiones[u][t]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[i][k]+Dimensiones[i][k]==HA_ik[u][k]+Dimensiones[u][k]
and HA_ik[i][t]+Dimensiones[i][t]==HA_ik[u][t]+Dimensiones[u][t]
and HA_ik[i][r]+Dimensiones[i][r]==HA_ik[u][r]+Dimensiones[u][r]):
MoverCaja2(HA_ik,k,i,r,t,u)
elif (HA_ik[i][k]==HA_ik[u][k]
and HA_ik[i][r]==HA_ik[u][r]
and HA_ik[i][t]==HA_ik[u][t]):
MoverCaja2(HA_ik,k,i,r,t,u)
```

```
else:
break return HA_ik
def Gravedad2(HA_ik):
MA = np.zeros(Productos)
productos en cota cero
for i in range(Productos):
if HA_ik[i][2] == 0:
MA[i] = 1
productos inmediatamente sobre otros productos que no vuelan
for i in range(Productos):
if MA[i] == 0:
compatibles = []
for u in range(Productos):
if i != u and MA[u] == 1:
if HA_ik[u][2]+Dimensiones[u][2] == HA_ik[i][2]:cotas compatibles
if( HA_ik[u][1] <= HA_ik[i][1]
and HA_ik[u][1]+Dimensiones[u][1] >= HA_ik[i][1]+Dimensiones[i][1]
and HA_ik[u][0] <= HA_ik[i][0]
and HA_ik[u][0]+Dimensiones[u][0] >= HA_ik[i][0]+Dimensiones[i][0]):
MA[i] = 1la caja i está justo sobre caja u
break
else:agrega a lista de compatibles
compatibles.append(u)
if MA[i] == 0 and len(compatibles) != 0:
for u in compatibles:
if ((HA_ik[u][0]+Dimensiones[u][0] > HA_ik[i][0]
and HA_ik[u][0] < HA_ik[i][0]+Dimensiones[i][0]
and
(HA_ik[u][1]+Dimensiones[u][1] > HA_ik[i][1]
and HA_ik[u][1] < HA_ik[i][1]+Dimensiones[i][1]))):
MA[i] = 1
break
```

```
else:
compatibles.remove(u)
productos que vuelan
vuelan = diccionario con producto y altura
for i in range(Productos):
if MA[i] == 0:
vuelan.setdefault(i,HA_ik[i][2])
for w in sorted(vuelan.items(), key=lambda x:x[1]):
i = w[0]
compatibles = []
for u in range(Productos):
if i != u and MA[u] == 1:
if (HA_ik[u][2]+Dimensiones[u][2] < HA_ik[i][2]):cotas compatibles i está sobre u
compatibles.append(u)
for u in compatibles:
borrar = []
if not (HA_ik[u][0]+Dimensiones[u][0] > HA_ik[i][0]
and HA_ik[u][0] < HA_ik[i][0]+Dimensiones[i][0]
and HA_ik[u][1]+Dimensiones[u][1] > HA_ik[i][1]
and HA_ik[u][1] < HA_ik[i][1]+Dimensiones[i][1]):
borrar.append(u)
compatibles = list(set(compatibles)-set(borrar))
asignar cota al más alto de la proyecciónXY
if len(compatibles) != 0:
cotaMax = -1
cajaMax = 0
for u in compatibles:
if HA_ik[u][2]+Dimensiones[u][2] >= cotaMax:
cotaMax = HA_ik[u][2]+Dimensiones[u][2]
cajaMax = u
if cotaMax != -1:
HA_ik[i][2] = HA_ik[cajaMax][2]+Dimensiones[cajaMax][2]
```

MA[i] = 1
else:
HA_ik[i][2] = 0
MA[i] = 1
return HA_ik
while (True):
.Aquí empieza La metaheurística ”Harmony Search”, con sus 3 casos (encontrar una melodía ya utilizada , usar una melodía ya utilizada variando ciertas notas y usar una melodia con notas totalmente aleatorias.) ”
Harmonica.append(HA_ik)
rand=random.random()
if rand > r_acept:
p=random.randrange(len(Harmonica))
HA_ik=Harmonica[p]
Gravedad2(HA_ik)
for i in range(Productos):
for k in range(3):
Overlap(HA_ik,k,i)
elif rand > r_pa:
f=random.randrange(Productos)cantidad de productos que se mueven escogido aleatoriamente
for q in range(f): moviendo los productos
x=random.randrange(Productos) se mueve un producto aleatorio
z=random.randrange(2) en una dimension aleatoria en el plano xy
HA_ik[x][z]=HA_ik[x][z]-np.random.randint(0,50) se mueve hasta un maximo de 50 unidades
LimitesCamion(HA_ik,x)
Gravedad2(HA_ik)
for i in range(Productos):
for k in range(3):
Overlap(HA_ik,k,i) else:
for i in range(Productos):

HA_ik[i][0]=random.randrange(MX[0]-Dimensiones[i][0])
HA_ik[i][1]=random.randrange(MX[1]-Dimensiones[i][1])
HA_ik[i][2]=random.randrange(MX[2]-Dimensiones[i][2])
Gravedad2(HA_ik)
for i in range(Productos):
for k in range(3):
Overlap(HA_ik,k,i)
for i in range(Productos): Volumen=(HA_ik[i][0])*a[i]+(HA_ik[i][1])*a[i]+(HA_ik[i][2])*a[i]
"Función objetivo, con penalidad para las primeras cajas que se suben al camión son mas tienen una mayor ponderación "
Optimo=Optimo+Volumen
if iteracion==0:
MatrizOptima2.append(100000000000000000)
Guarda_todo.append(Optimo)
tiempo_no_optimo.append(int(elapsed_time))
F=0
F=int(min(MatrizOptima2))
elapsed_time = time() - starting_point
if bool(Optimo<F):
"Se obtiene el minimo de las pruebas"
HA=HA_ik.copy()
Guarda_Optimo.append(Optimo)
tiempo_optimo.append(int(elapsed_time))
print ("HA_ik ", iteracion, ."es":", HA_ik[0:Productos], ,."El optimo es:", Optimo," En" , elapsed_time, "segundos")
else:
HA_ik=HA.copy()
if (iteracion==0):
MatrizOptima2=[]
MatrizOptima2.append(Optimo)
Optimo=0
iteracion=iteracion+1

```
elapsed_time = time() - starting_point
if elapsed_time>=TiempoDeComputo:
break
print (”Numero de iteraciones”, iteracion)
print (”Optimo es:”, min(MatrizOptima2))
print (”tiempo”, elapsed_time)
print (”la mejor iteración es: ”, HA[0:Productos])
dibujar(HA,Productos, Dimensiones)
```