# Optimización de rutas en un almacén



**Carlota Royo Ruiz**

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Directoras del trabajo:
Herminia I. Calvete Fernández
Carmen Galé Pola

28 de junio de 2021

# Prólogo

Los almacenes alrededor del mundo presentan diferencias de funcionalidad, dimensión, productos almacenados, etc. A pesar de ello, en todos ellos se realizan los mismos procesos y ligados a ellos surgen varios problemas de optimización cuya resolución contribuyen a la gestión eficiente del almacén. Este trabajo se centra en la preparación de pedidos, en particular, en el cálculo de la ruta óptima que debe realizar el vehículo en el almacén para recoger los productos que componen un pedido. El proceso de preparación de pedidos es, en general, el que consume más recursos, tanto humanos como financieros, de ahí la importancia de proporcionar junto con el listado de productos de un pedido el orden de recogida de los mismos.

Este trabajo está dividido en tres capítulos y un apéndice. A continuación, se da una breve descripción de cada uno de ellos.

- **Capítulo 1. Optimización en la gestión de un almacén**
  En este capítulo se presenta de manera genérica el concepto de almacén y los procesos fundamentales que se realizan en él. Asimismo, se describen los problemas de optimización subyacentes de dichos procesos. En el último apartado de este capítulo se especifica la contribución de este trabajo en el proceso de preparación de pedidos.

- **Capítulo 2. Ruteo en la preparación de un pedido**
  En este capítulo se describe el problema de una forma más detallada y se realiza una breve revisión de las principales aproximaciones que se han seguido en la literatura relacionadas con el ruteo en la preparación de pedidos. La formulación matemática del problema requiere el análisis previo de la estructura de los almacenes, con bloques rectangulares y pasillos paralelos, así como su posible representación como un grafo.

  A continuación, se introducen los modelos de optimización propuestos para determinar las rutas en la preparación del pedido. Una primera aproximación plantea el problema como un caso particular del problema del viajante. La segunda aproximación formula el problema como un problema de ruteo de vehículos. Dentro de este enfoque se plantean tres formulaciones. La primera formulación es la clásica del VRP con dos índices. La segunda formulación incluye variables con tres índices, que dan información directamente sobre la ruta y el vehículo que realiza la ruta. En particular, en los almacenes en los que solo se dispone de un vehículo, las rutas que determina el VRP no se realizan simultáneamente, sino que se corresponden con los viajes que realiza el vehículo. En cada viaje sale y regresa al mismo punto y recoge un subconjunto del total de productos en el pedido. Por ello, la tercera formulación propuesta en el trabajo incluye variables de tiempo. Estas variables permiten introducir restricciones adicionales sobre la recogida de productos que representan situaciones frecuentes en los almacenes, cuando determinados productos han de estar listos en el momento de la llegada al almacén del camión que los recoge.

- **Capítulo 3. Caso de estudio: un almacén de distribución**
  En este último capítulo se resuelve el problema de ruteo en un almacén de distribución real. En primer lugar, se describe el almacén a partir de su representación en bloques rectangulares y

pasillos y, se calculan las distancias entre dos localizaciones cualesquiera con un programa de Python. El peso de los productos se han generado computacionalmente a partir de distribuciones de probabilidad triangulares con varios parámetros. Todas las formulaciones presentadas en el capítulo 2 se han implementado en CPLEX. El interés de calcular las rutas para la preparación de un pedido de forma óptima se ha ilustrado con la generación de varios escenarios, a través de un código de Python. Estos escenarios consideran un total de 60 pedidos que se diferencian en el número de productos, el peso y la localización de los productos en el almacén. Finalmente, se muestra el potencial de la programación matemática al incorporar restricciones de tiempo en la recogida de algunos de los productos del pedido ya que en el almacén real se dispone de un solo vehículo que realiza los viajes uno detrás de otro.

- **Apéndice**
  El trabajo incluye un Apéndice en el que se presentan los códigos de los programas desarrollados para implementar los modelos con Python y CPLEX Studio. También se incluyen los códigos para generar los datos necesarios para su evaluación.

# Summary

Over the years, companies have always been focused on improving their efficiency and, as a result, reducing costs within business operations. Warehouse management is one of the most challenging processes in every company since is the one that consumes most of the human and financial resources of the warehouse. Although warehouses around the world differ in terms of size, type or function the fundamental processes remain. These processes, which are explained in the first chapter, include receiving, storage, picking and despatch. Although there are many problems that could be studied within each of these processes and each must be optimized, this work will focus on picking, in particular, on route optimization for the order picking problem.

Order picking consists in collecting products from a location in the warehouse in a specific quantity given by a customer order and bring them to the depot. Due to the fact that order picking is one of the most crucial process within the warehouse, many studies have been conducted on this subject. The aim of these studies is to find optimal routes that the vehicle should follow within the warehouse to minimize the distance travelled or the time invested. This problem is considered to be NP- hard. However, for the generic case of a rectangular and regular warehouse with a reduced number of aisles and cross-aisles it can be simplified.

The aim of this work is to achieve the following:

- Understanding the processes within the warehouse
- Generating mathematical models for routing with a consideration of various constraints in a warehouse order picking operation.
- Implementing the mathematical model to provide an optimal picking route.
- Performing simulation-based analyses and compare the mathematical models results.

In the second chapter, the most important approaches followed in the literature are presented. They model the warehouse as a directed graph. The layout of the warehouse is one of the most influential aspects in the solution model. When the warehouse consists of two cross-aisles or three cross-aisles, the route problem to be solved is polinomial in the number of aisles and the number of pick locations.

The optimal route may not always be the most frequently used by order pickers in practice. The reason is that the optimal route may not be the easiest to follow and it needs to be solved for each order, which could be time consuming and require and unreasonable computing power. Hence, many heuristics have been proposed to determine order picking routes in a warehouse. Even though these strategies do not necessarily provide the optimal solution, they are easy to learn and intuitive to follow.

The route optimization for the order picking problem can be formulated as a Traveling Salesman Problem (TSP). Nevertheless, this formulation can only be used when the capacity of the vehicle is not considered. In a general case, in which this aspect is relevant, the problem is formulated as a Vehicle Routing Problem (VRP).

First, the two-index vehicle flow formulation of the VRP model is presented. Then, a three-index flow formulation is presented to directly acquire more information about which vehicle collects each product.

In the case of study, there is only one vehicle that execute each route, unlike the classic VRP in which the routes occur simultaneously. Thus, a new formulation for the VRP with three-index is proposed. This formulation allows to add constraints that could be useful in the real world such as demanding that a particular product needs to be collected and brought to the depot before a specific time, in the first tour or in the last tour.

In the third chapter, the case of study is presented. The logistic warehouse that is studied belongs to a company dedicated to store and distribute customer products. This warehouse has a special structure. It has parallel aisles and a rectangular structure, but it has two distinguished blocks of different measures. Hence, the warehouse needs to be analyzed in detail since it has not a standard structure. The first thing that needs to be taken into consideration is how to calculate the distance between two given products. Therefore, a program on Python is developed to compute the distance matrix of the 3105 products stored in the warehouse.

To solve the given problem on the particular warehouse some programs needed to be done. The weight of each product is unknown. Therefore, a program in Python allows to randomly generate them. The products were divided in three categories: light, medium and heavy and, were created considering a triangular distribution for each category where the parameters needed are defined as introduced in this chapter.

Besides, orders are also generated randomly with a code in Python. Simulation has proven that to solve the route problem with CPLEX Studio when the order has more than 12 products took a large amount of computacional time. So, orders generated to simulate the problem consists of 10 or 12 products to pick from the warehouse. The products in the warehouse are stored by layers. Demand of each product is characterized by the weight of the product and the number of layers that pickers needs to pick. Thus, the number of layers demanded is also randomly generated by a program in Python. Orders were generated following two criteria. First, orders were divided by weight, meaning, three groups of orders were created: random weights, light weights and heavy weights products. Second, orders were generated by location in the warehouse, which means that they were also divided in three groups: products from each zone, products from second zone and products from third zone (these zones are defined in this chapter).

In the case of study, the model implemented to solve the problem is based on the VRP formulation, due to the necessity of implementing a model that includes the capacity of the vehicle. To solve the problem, the different formulations of the CVRP were implemented on an optimization software named CPLEX Studio.

Orders classified by weight and location were solved with the two-index and the three-index basic formulations in order to compare computacional time for both formulations. IBoth models achieve the optimal solution, but the three-index formulation takes a considerable amount of time compared with the two-index formulation. However, the three-index formulation provide more information that could be relevant on realistic problems. Lastly, an order is generated to illustrate the proposed formulation that add time constraints in order to show how these constraints affect the optimal solution.

# Índice general

# Capítulo 1

# Optimización en la gestión de un almacén

## 1.1. Introducción

Un aspecto fundamental a considerar en la mayoría de las empresas es la gestión del almacén. Debería implementarse un funcionamiento óptimo para mejorar el rendimiento y la eficiencia de la empresa y conseguir que se cumplan los objetivos propuestos. Con una buena gestión se consigue reducir tiempo y costes que podrían invertirse en otros aspectos de la empresa. La importancia de la gestión de un almacén de las empresas se ve reflejada tanto en la cantidad de estudios que se han realizado en este ámbito como en la cantidad de recursos que las empresas invierten en su mejora.

Un almacén consume muchos recursos de diversos tipos: consume recursos humanos y financieros, tiene inmuebles, estanterías y maquinaria que requiere mantenimiento y, consume tiempo. Todo ello sin que muchas veces se perciba que aporta valor. Sin embargo, en un almacén se recolectan y gestionan materias primas, repuestos, productos semielaborados y productos terminados. Los productos terminados, además de ser almacenados, se preparan para el envío. El almacén es, por tanto, el departamento en el que se realizan muchas de las funciones que hacen posible la actividad económica de una empresa.

Pese a las grandes diferencias que presentan los distintos almacenes alrededor del mundo en cuanto a los productos que se almacenan, sus dimensiones o su funcionalidad, entre otras, todos ellos tienen en común una serie de procesos fundamentales. Ahora bien, es preciso tener en cuenta que la gestión de un almacén no empieza con las actividades que ocurren en él. Previamente, se necesita realizar una serie de actividades para organizar los procesos del almacén.

En primer lugar, el almacén debe planificar las compras a proveedores y tener un acuerdo con estos para evitar problemas y malentendidos. En este acuerdo se deben especificar una serie de condiciones que imponen tanto los proveedores como el almacén:

- Cuando los proveedores envían los productos al almacén, el almacén tiene que asegurarse de que tiene la maquinaria necesaria para la descarga de la mercancía del camión. Los proveedores deberán facilitar esta información. Se debe considerar el tipo de pallets en los que va a venir la mercancía empaquetada y el peso de cada uno de ellos. En algunos casos específicos, por ejemplo, cuando los productos que se reciben son frágiles o peligrosos, se requerirá una maquinaria específica.

- De acuerdo con las limitaciones de capacidad del muelle de descarga y la disponibilidad de los trabajadores, se deben fijar los horarios en los que los camiones tendrán que llegar al almacén.

- El almacén debe comunicar a los proveedores cómo quiere que se organice la mercancía. Así, será más sencillo para los trabajadores localizar y reubicar cada producto.

- Los proveedores tienen que conocer cómo quiere el almacén que se catalogue la mercancía. En las etiquetas deberá aparecer al menos el identificador de cada producto y la cantidad.
- Los proveedores deberán informar de la cantidad de mercancía que necesita ser almacenada.

En conclusión, se deben especificar ciertas condiciones para asegurarse de que el almacén posee los medios mecánicos, humanos e informáticos que se requieren para recibir la mercancía de los proveedores.

Dado un almacén con cualesquiera características específicas, los procesos logísticos básicos que se producen en él se dividen en cuatro grandes grupos: proceso de recepción de mercancía, proceso de ubicación de la mercancía, proceso de preparación de pedidos, y por último, proceso de expedición del pedido. Dentro de cada uno de estos procesos, se realizan diferentes actividades que se describen más detalladamente en los siguientes apartados. Desde el punto de vista de la optimización, dentro de cada uno de los procesos existen una gran cantidad de problemas que se podrían considerar para mejorar la gestión de la empresa. Sin embargo, por las razones que se van a exponer, de entre estos cuatro procesos, este trabajo se centra en la preparación de los pedidos, también conocido como *picking*. En este problema se trata de optimizar los recorridos de pasillos que realizará el vehículo para la recogida de un pedido según las características del almacén.

## 1.2. Proceso de recepción

Este proceso consiste en dar entrada a la mercancía que envían los proveedores. Cuando un camión llega al muelle de descarga, en primer lugar se descarga el pedido. El método más común y tradicional para la descarga de un camión son las carretillas elevadoras. No obstante, se están introduciendo sistemas de descarga automatizados como cintas transportadoras, que agilizan y rentabilizan el tiempo empleado en esta actividad. En cualquier caso, el método de descarga y las máquinas necesarias deben haberse acordado previamente entre proveedor y almacén. En esta fase, se debe comprobar que la mercancía recibida llega al almacén en buenas condiciones y coincide con la información prevista en el pedido, esto es, comprobar que la cantidad y las características se corresponden con lo establecido en el acuerdo del pedido con los proveedores.

En este proceso surgen problemas cuyo estudio se podría abordar con técnicas de optimización. En primer lugar, la organización de la llegada de los camiones es fundamental para el buen funcionamiento de este proceso. A los proveedores se les debe facilitar la información de horarios de llegada del camión al almacén. Por tanto, se podría analizar la organización de los horarios para optimizar tanto los tiempos de llegada de los camiones como la ocupación en el muelle de descarga. Por otro lado, el almacén debe disponer de la maquinaria necesaria . No se debería tener máquinas ociosas ni tampoco tener un número bajo de máquinas que impida realizar la descarga de los camiones eficientemente o impida reponer una máquina por otra en caso de rotura. De igual manera, si se dispone de datos de rotura anteriores de la maquinaria, se puede hacer una predicción sobre las futuras roturas esperables.

## 1.3. Proceso de ubicación

En este proceso el objetivo es ubicar la mercancía en el lugar idóneo del almacén para facilitar la futura recogida. La ubicación de los productos debe ser accesible con las máquinas del almacén y fácilmente localizable. En este proceso también es fundamental que el almacén tenga a su disposición las máquinas necesarias para situar los productos en su localización.

Dado un conjunto de productos a ubicar en el almacén, este proceso no se debería hacer de manera aleatoria. Entre las variables que pueden tenerse en cuenta pueden destacarse la frecuencia con la que cada producto es demandado, la altura en la que se encuentra un producto, etc.

## 1.4. Proceso de preparación o picking

Es este proceso se recoge la mercancía solicitada en el pedido por un cliente o un conjunto de clientes del almacén y, posteriormente, se prepara el pedido. Se necesita la maquinaria adecuada para completar de forma eficiente este proceso, tanto para recoger la mercancía como para transportarla a la zona de expedición. Existen varias posibilidades, como pueden ser vehículos conducidos por un trabajador o, algo más innovador, como son los vehículos de guiado automático también conocidos como AGV (Automated Guided Vehicle) que son vehículos que se mueven de forma automática sin conductor por rutas programadas previamente.

El proceso de *picking* a menudo se considera como el proceso de almacenaje más importante y complicado dentro de las actividades de la empresa. Seleccionar las rutas que deberá realizar el vehículo para recoger los pedidos es fundamental en la gestión del almacén. Conviene señalar que el reciente crecimiento del comercio electrónico da lugar a un gran aumento de las operaciones de preparación de pedidos. Por ello, es la parte a la que se dedican la mayoría de los estudios destinados a mejorar la gestión de los almacenes. De Koster et al.[1] estiman que el *picking* supone un 55 % del gasto operativo total del almacén.

El diseño del almacén juega un papel importante a la hora de minimizar el tiempo de recogida de los productos. Este trabajo, conocido el diseño del almacén, se centra en estudiar las rutas óptimas que deberá realizar el vehículo para recoger los productos de un pedido dado.

## 1.5. Proceso de expedición

Este es el último proceso que se realiza dentro de un almacén y consiste en la salida de la mercancía del almacén para que sea enviada a los clientes. El objetivo de todo almacén es minimizar los recursos necesarios para realizar las actividades de manera eficiente. Por tanto, en el proceso de expedición se debería estudiar la organización interna de los camiones para minimizar el espacio libre y maximizar la carga introducida, y el criterio para dividir la mercancía que se introduce en cada camión. Dentro de este problema habría que considerar, entre otros, aspectos relacionados con el recorrido que va a realizar cada camión para servir a los clientes, así como la urgencia de distribución de cada producto.

## 1.6. Contribución del trabajo

En este capítulo se han introducido brevemente los procesos que se realizan en un almacén general. De entre estos procesos, en este trabajo se ha estudiado el proceso de preparación de pedidos. Actualmente, es el proceso al que se dedican la mayor parte de los estudios cuyo propósito es mejorar la gestión de un almacén. La recogida de los productos de un pedido dispersos en la superficie de un almacén es una de las actividades que más recursos y tiempo consume en su gestión. Por ello, se busca optimizar las rutas que debe realizar el vehículo en el almacén para minimizar la distancia recorrida o el tiempo invertido en recoger los productos. La revisión de la literatura realizada se ha extendido a los modelos de optimización de cálculo de rutas, basados en el problema del viajante, cuando la capacidad del vehículo no es relevante, o el problema de ruteo de vehículos, cuando sí lo es.

Los modelos estudiados en la literatura sobre rutas en la preparación de pedidos se centran en un almacén estándar rectangular. El almacén real analizado en el caso de estudio tiene una estructura parti-

cular que ha habido que tomar en consideración en el cálculo de las distancias entre dos localizaciones cualesquiera del mismo. En el almacén real se gestionan 3105 productos. Los productos están caracterizados por su localización de recogida en el almacén que se determina a partir de su localización física en una estantería. La localización de recogida está situada en el centro de un subpasillo por lo que está asociada a los productos que hay en las estanterías situadas a la izquierda y a la derecha del subpasillo, en las diferentes alturas, ya que el vehículo puede acceder sin problemas a todos ellos. Un algoritmo programado en Python proporciona, dada la localización física de un producto, su localización de recogida, resultando un total de 324 localizaciones en el plano más el punto de recepción, y la matriz de distancias de dimensión $325 \times 325$. Se ha de señalar que la distancia calculada entre dos cualesquiera localizaciones de recogida es la distancia mínima de entre las formas posibles de ir de una a otra.

El almacén real bajo estudio tiene dos importantes restricciones: solo hay un vehículo y tiene una capacidad limitada. La existencia de un solo vehículo supone que, a diferencia del problema de ruteo clásico, las rutas no se realizan simultáneamente, sino que el mismo vehículo realiza varias rutas o viajes hasta completar el pedido, recogiendo en cada viaje un subconjunto de los productos del pedido y volviendo al punto de recepción a descargar cuando ya no dispone de capacidad suficiente. Por otro lado, no ha sido posible obtener toda la información sobre los pesos de los productos. Se ha tenido en cuenta su almacenamiento en mantos para poder transportarlos en pallets. El número de mantos así como el peso de los productos se ha generado de forma realista. Los pesos se han generado a partir de una distribución de probabilidad triangular, cuyos parámetros se han modificado según si el producto era ligero, de peso mediano o pesado. De esta forma, se ha conseguido una aproximación a la variabilidad de los pesos de los productos almacenados razonable.

Los datos de entrada de los modelos de optimización describen de forma completa los productos incluidos en el pedido. El conocimiento de la localización física de los productos permite seleccionar la submatriz de distancias de la matriz total de distancias que incluye las filas y las columnas correspondientes a las localizaciones de recogida de los productos en el pedido. En cuanto a la demanda de cada producto, se ha generado al azar el número de mantos solicitado y obtenido así el peso correspondiente a cada producto. En la experiencia computacional se ha considerado una amplia variedad de escenarios para mostrar el potencial de los modelos de programación matemática, en los que se consideran pedidos con productos de diferentes pesos y ubicados en diferentes zonas del almacén. Estos modelos se podrían aplicar del mismo modo si fuera el usuario el que introdujera el listado de productos de un pedido y la demanda en mantos de cada producto. Todos los escenarios se han resuelto con el software de optimización CPLEX Studio y se ha realizado el estudio comparativo entre las formulaciones de dos y tres índices propuestas en términos de los tiempos computacionales.

Finalmente, en este trabajo se ha propuesto un nuevo modelo para resolver variantes del problema no tratadas en la literatura y que pueden ser interesantes en el almacén real bajo estudio, como, por ejemplo, que algún producto llegue al punto de recogida del pedido antes de un tiempo específico o en el primer o último viaje. El proceso de preparación de pedidos se ha de integrar con la planificación de los camiones que llegan al almacén a recoger los pedidos, por lo que las restricciones anteriores son, en ocasiones, necesarias. El nuevo modelo incorpora variables de decisión temporales que permiten añadir dichas restricciones y la experiencia computacional muestra el efecto de las mismas en las soluciones obtenidas.

Este caso de estudio es solo un ejemplo del potencial de la programación matemática en la resolución de los problemas de gestión de los almacenes y ha permitido desarrollar de forma completa un estudio de investigación operativa, desde la recogida de datos hasta la obtención de la solución óptima y el análisis de sensibilidad de la misma.

# Capítulo 2

# Ruteo en la preparación de un pedido

## 2.1. Introducción

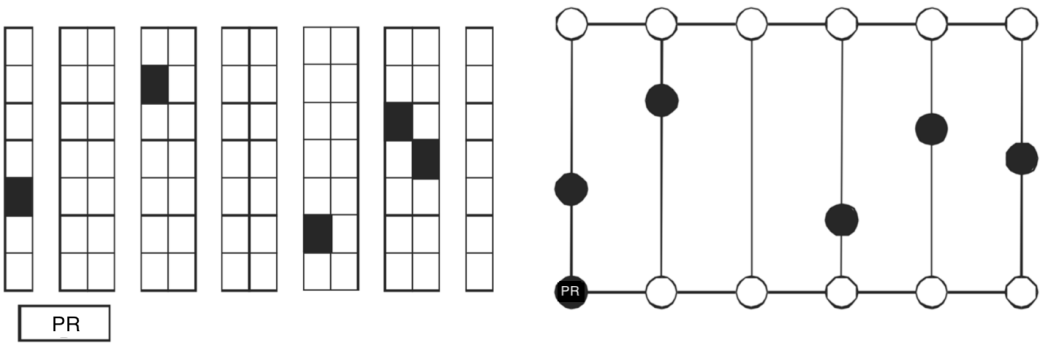
Al problema de ruteo dentro de un almacén se dedican muchos de los estudios destinados a optimizar la gestión de un almacén debido a que es el proceso que consume más recursos en términos de tiempo y mano de obra. En este problema se trata de optimizar las rutas que el vehículo debe seguir en el almacén para la preparación de un pedido. Un pedido consiste en un conjunto de productos, caracterizados por su localización y su peso o volumen, con una determinada demanda que deben recogerse en el almacén. El objetivo es minimizar la distancia total recorrida o el tiempo total invertido por el vehículo.

Un almacén está formado por pasillos y estanterías en las que están localizados los productos. En un almacén estándar, los pasillos y las estanterías están en paralelo, formando así una estructura rectangular. Las dimensiones de los pasillos deben ser adecuadas para el tipo de vehículo y la maquinaria de carga que va a circular por ellos. Los pasillos que se denominan transversales no contienen productos pero permiten que el vehículo pueda circular por el almacén. Los denominados subpasillos están formados por las estanterías en las que están ubicados los productos. Un vehículo al atravesar un subpasillo puede coger productos de cualquiera de las estanterías a las que tiene acceso. Una vez recogido el pedido completo, o una parte de él si el vehículo no puede cargar más productos, este se deposita por la operaria en el punto de recepción (PR). En este punto la operaria recoge también la lista con los productos del pedido o la lista que contiene las rutas que debe realizar en el almacén para recogerlos. Todas las rutas comienzan y finalizan en el PR.

Puesto que, independientemente de cuales sean las rutas que se siguen en el almacén para la recogida de los productos, el tiempo invertido para extraerlos de la estantería en la que están y el tiempo invertido para descargar el pedido del vehículo en el PR no cambia, estos tiempos no son considerados en los modelos de optimización.

## 2.2. Primeros trabajos en la literatura sobre ruteo en almacenes

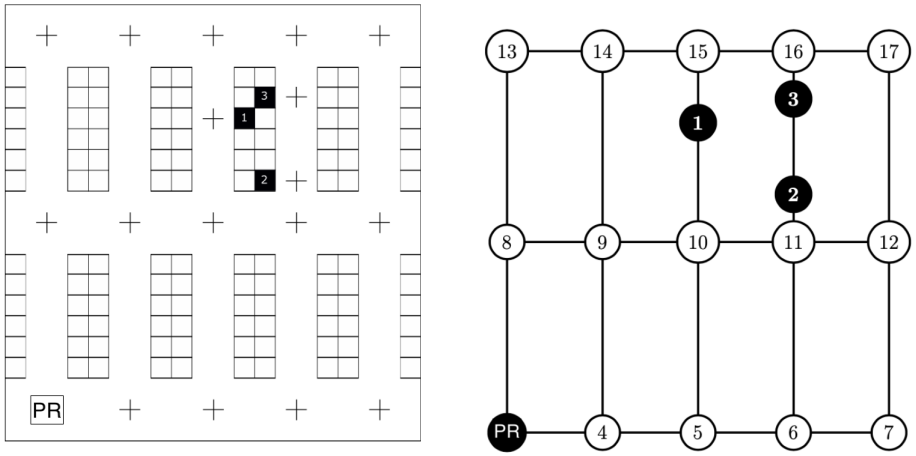
El problema de establecimiento de las rutas en un almacén para la preparación de un pedido ha sido uno de los más estudiados en Investigación Operativa y se introdujo en 1954 por Dantzing, Fulkerson y Johnson [4]. En 1982 se demostró que determinar la solución óptima de este problema es un problema NP-duro dentro de la optimización combinatoria [5]. Por tanto, este problema es computacionalmente complejo. No obstante, los almacenes estándar suelen cumplir ciertas propiedades de diseño: perímetro en forma rectangular, subpasillos paralelos, pasillos transversales, etc. que han permitido el desarrollo de algoritmos polinomiales para la resolución de algunos casos particulares del problema eficientemente.




(a) Almacén con un único bloque (b) Grafo del almacén

Figura 2.1: Almacén con un único bloque y grafo asociado (Figuras editadas extraídas de [2])

El caso de un almacén con un único bloque rectangular fue tratado por Ratliff y Rosenthal [6]. Este tipo de almacén posee dos pasillos transversales localizados en la parte superior e inferior del almacén, únicos lugares donde se puede realizar intercambio de subpasillo. En la Figura 2.1a se representa un ejemplo de este tipo de almacén. Para este problema, Ratliff y Rosenthal en 1983 propusieron por primera vez un modelo de programación dinámica que resuelve el problema basado en la representación del almacén mediante un grafo (véase la Figura 2.1b). Además, tras realizar un estudio sobre las posibles formas de cruzar un pasillo en una solución óptima, en el trabajo se demuestra que cualquier solución óptima contiene cada arco como mucho dos veces (una de ida y otra de vuelta, si es necesario). Este algoritmo se extendió en 2001 al caso en el que el almacén tiene dos bloques por Roodbergen y De Koster [1]. Este diseño de almacén tiene tres pasillos transversales por los que se puede realizar el intercambio entre subpasillos, que están situados en la parte superior, central e inferior del almacén. En la Figura 2.2a se representa un ejemplo de este tipo de almacén y en la Figura 2.2b su grafo asociado. Este algoritmo es polinomial respecto al número de subpasillos y productos del almacén. El algoritmo puede extenderse a cualquier estructura del almacén pero, en ese caso, el algoritmo es exponencial con respecto al número de pasillos transversales.




(a) Almacén con un único bloque (b) Grafo del almacén

Figura 2.2: Almacén con dos bloques y grafo asociado (Figuras editadas extraídas de [5])

## 2.3. Estrategias de picking usuales en las empresas

A lo largo de los años, las empresas han desarrollado procedimientos heurísticos propios para planificar las rutas de recogida de productos de un pedido. Aunque no hay garantías de que estos métodos proporcionen una solución óptima o cercana a la óptima, y en muchos casos la solución que proporcionan está muy alejada de la solución óptima [2], es relativamente frecuente que las empresas recurran a ellos debido a que resultan fáciles de implementar. En la práctica, la ruta óptima no es necesariamente la más utilizada por la operaria que recoge los pedidos. Esto se debe principalmente a que en muchos casos puede no ser una ruta intuitiva de realizar. Además, para cada pedido se deben realizar los cálculos de hallar la ruta óptima, lo que puede consumir mucho tiempo y requerir un alto nivel de computación, del que muchas empresas pueden no disponer. No obstante, gracias al continuo desarrollo de dispositivos electrónicos es posible mandar localizaciones en tiempo real de los vehículos y guías sobre las rutas a realizar que facilitan este trabajo. A continuación se describen las estrategias heurísticas más utilizadas en la práctica para la preparación de pedidos.

### 2.3.1. S-Shape

Esta heurística de ruteo, también llamada estrategia transversal o en forma de S, suele ser considerada como la más básica. Consiste en que todos los subpasillos que contengan algún producto se atraviesan completamente tomando forma de S. Los subpasillos sin producto que recoger se omiten.

La operaria inicia la ruta recorriendo completamente el subpasillo con productos a recoger que esté en el extremo izquierdo o derecho, según cuál sea el más cercano al PR. A partir de ahí, se visitan todos los subpasillos uno a uno pertenecientes al bloque más lejano y se continúa el mismo proceso en cada bloque. En el caso en el que la operaria esté en el último subpasillo a recorrer de cada bloque no hace falta recorrerlo entero; se recoge el último producto y se regresa al pasillo transversal por el que sigue la ruta. Finalmente, la operaria regresa al PR. El proceso puede verse en la Figura 2.3a.

### 2.3.2. Return

Esta estrategia consiste en entrar en cada subpasillo con producto hasta llegar al último producto que hay que recoger para luego volver al pasillo transversal por el que se había entrado. En caso de que el diseño del almacén contenga más de un bloque, la operaria visita los subpasillos de los dos bloques adyacentes al pasillo transversal alternadamente. Finalmente, la operaria regresa al PR. Esta estrategia es más efectiva cuando los productos a recoger están localizados al principio del subpasillo. Si se tiene un almacén en el que las ubicaciones de los productos más demandados están situados al inicio del subpasillo, esta política de ruteo puede esperarse que sea cercana a la óptima. El proceso puede verse en la Figura 2.3b.

### 2.3.3. Largest Gap

Esta heurística consiste en identificar el espacio más largo entre dos productos, llamado *largest gap*, en cada subpasillo y evitar recorrerlo. Los productos que se encuentren por encima del *largest gap* serán recogidos entrando por el pasillo transversal superior y, a continuación, la operaria volverá a este pasillo; los productos restantes se recogerán desde el pasillo transversal inferior. Todos los subpasillos de cada bloque se recorren de esta manera. Finalmente, la operaria regresa al PR. Los únicos subpasillos que se atraviesan completamente son los pertenecientes al primer subpasillo y el último de cada bloque. El proceso puede verse en la Figura 2.3c.

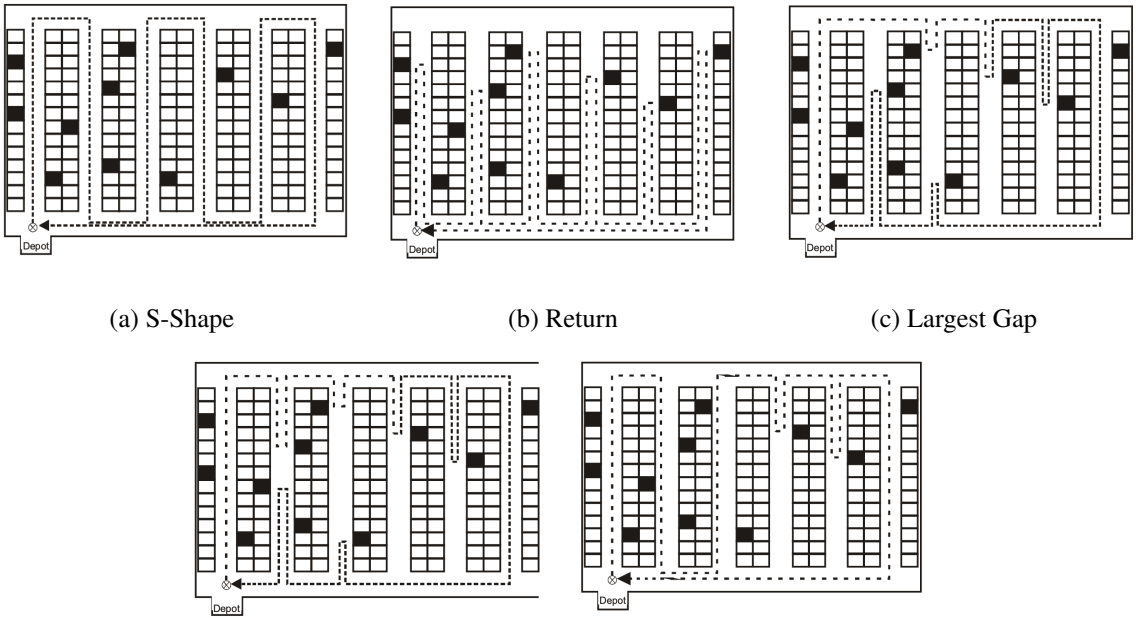
### 2.3.4. Mid-point

En esta política de ruteo, semejante a la *Largest Gap*, cada subpasillo del almacén se divide en dos mitades. La operaria, cuando recorre el pasillo transversal superior, toma únicamente los productos

que se encuentran en la mitad superior del subpasillo, los restantes los recoge cuando recorre el pasillo transversal inferior. En el caso en el que el producto se encuentre justamente en el centro del subpasillo, se recoge desde cualquiera de las dos opciones. Finalmente, la operaria regresa al PR. Si el número de productos a recoger por subpasillo es pequeño, esta política proporciona mejores resultados que la *S-Shape* [2]. El proceso puede verse en la Figura 2.3d.

### 2.3.5. Combined

Esta heurística es una combinación de la *S-Shape* y la *Largest Gap*. Según estas políticas, un subpasillo es atravesado completamente (*S-Shape*) o se entra y se sale por el mismo lado del subpasillo (*Largest Gap*). La decisión se toma en cada momento según la distancia más corta al siguiente producto dentro del mismo bloque. Este proceso se repite hasta que se recorren todos los subpasillos por bloque y finalmente se regresa al PR. El proceso puede verse en la Figura 2.3e.




(a) S-Shape (b) Return (c) Largest Gap

(d) Mid-Point (e) Combined

Figura 2.3: Estrategias heurísticas (Figura extraída de [7], Depot es el PR)

## 2.4. Modelos de optimización

En esta sección se presentan distintos modelos de optimización que permiten resolver el problema en almacenes generales según las características del proceso de ruteo considerado. Los modelos estudiados en la literatura son casos particulares del problema del viajante (Traveling Salesman Problem o TSP) o del problema de ruteo de vehículos (Vehicle Routing Problem o VRP). El TSP consiste en diseñar la ruta que debe seguir un vendedor que comenzando en un origen visita un conjunto de ciudades y vuelve a ese origen, de manera que la distancia total recorrida o el tiempo total invertido es mínimo y cada ciudad sólo se visita una vez. El VRP diseña un conjunto óptimo de rutas para una flota de vehículos con el objetivo de visitar a un conjunto de clientes y satisfacer sus demandas. Esta descripción genérica engloba un conjunto muy variado de problemas que incluyen restricciones de capacidad, de recogida y entrega de productos, etc. Los modelos estudiados en la literatura asocian al almacén un grafo. A continuación se describen las características más importantes.

Sea el grafo dirigido $G = (V_0, A)$ en el que $V_0$ es el conjunto de nodos y $A$ es el conjunto de arcos.

$$A = \{(i, j) : i, j \in V_0, i \neq j\}.$$

Cada arco $(i, j)$ tiene asociada una constante $d_{ij}$ que representa la distancia existente entre un nodo $i$ y un nodo $j$. Suponemos que si $(i, j) \in A$ entonces $(j, i) \in A$ y $d_{ij} = d_{ji}$.

El conjunto de nodos $V_0 = V \cup \{0\}$, siendo 0 el nodo que representa el PR y $V$ el conjunto de los nodos asociados a los productos del almacén.

Notemos que cada producto $i \in V$ está representado por tres componentes $(x_i, y_i, h_i)$. Las componentes $(x_i, y_i)$ proporcionan las coordenadas que localizan el producto en el almacén mientras que la componente $h_i$ proporciona la altura de la estantería. Entre dos productos situados en la misma localización en diferentes alturas de la estantería, su distancia es cero. El resto de las distancias se calculan dependiendo de las características del almacén y de los movimientos permitidos del vehículo.

Por otro lado, si se supone que el vehículo avanza a una velocidad constante $v$ entonces es equivalente considerar la minimización de la distancia o del tiempo ya que si $t_{ij}$ representa el tiempo invertido para desplazarse de $i$ a $j$, entonces $t_{ij} = \dfrac{d_{ij}}{v}$.

Si el peso (o el volumen) de los productos es relevante para el proceso de recogida del pedido por las características del vehículo, cada nodo $i$ tendrá asociada una constante $q_i$ que representa el peso (o volumen) total demandado del producto $i$.

El interés en los modelos de optimización que se proponen es la preparación de pedidos, en los que el número de productos, $n$, es inferior al total de productos almacenados. Por ello, se denota $N \subset V$ el conjunto de nodos que identifican los productos de un pedido particular, $|N| = n$ y sea $N_0 = N \cup \{0\}$, el conjunto de nodos incluido el PR.

### 2.4.1. El problema del viajante

En el caso en el que el vehículo tenga suficiente capacidad para recoger cualquier pedido se puede asumir la no existencia de restricciones de capacidad. Esta hipótesis se traduce en que todos los productos se pueden recoger en una única ruta y el problema se puede modelar como un TSP, en el grafo $G_0 = (N_0, A_0)$ donde $A_0 \subseteq A$ contiene los arcos que conectan los nodos de $N_0$. Se está suponiendo que los nodos que contienen productos del pedido dado sólo se visitan una vez. De entre las formulaciones propuestas para este problema ([8], [9] y [10]) se va a presentar la formulación de Gavish y Graves [8]. En esta formulación, se introduce el concepto de flujo, esto es, la operaria ocupa una unidad del espacio del vehículo en la localización que recoge un producto.

Para $i, j \in N_0$ se definen las variables:

$$x_{ij} = \begin{cases} 1 & \text{si la ruta utiliza el arco } (i, j) \\ 0 & \text{en caso contrario} \end{cases}$$

$$y_{ij} = \text{cantidad de flujo que pasa por el arco } (i, j)$$

El TSP se formula como el siguiente modelo de programación lineal entera mixta:

$$\min \sum_{i \in N_0} \sum_{j \in N_0} d_{ij} x_{ij} \tag{2.1}$$

$$\text{sujeto a } \sum_{j \in N_0} x_{ij} = 1, \qquad i \in N_0 \tag{2.2}$$

$$\sum_{j \in N_0} x_{ji} = 1, \qquad i \in N_0 \tag{2.3}$$

$$\sum_{j \in N_0} y_{ji} - \sum_{j \in N_0} y_{ij} = 1, \qquad i \in N \tag{2.4}$$

$$y_{ij} \leq n x_{ij}, \qquad i, j \in N_0 \tag{2.5}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j \in N_0 \tag{2.6}$$

$$y_{ij} \geq 0, \qquad i, j \in N_0 \tag{2.7}$$

La función objetivo (2.1) minimiza la distancia total recorrida en la ruta. Las restricciones (2.2) y (2.3) aseguran que cada nodo se visita una vez exactamente. Las restricciones (2.4) aseguran que, excepto por el PR, la operaria deja una unidad de flujo, lo que en términos del problema de preparación de pedidos quiere decir que recoge el producto del nodo $i$, ocupando una unidad del espacio del vehículo, y dejando las que quedan para ser ocupadas en los nodos siguientes de la ruta. Las restricciones (2.5) garantizan que si pasa flujo por el arco $(i, j)$, la ruta utiliza el arco $(i, j)$. Las restricciones (2.6) y (2.7) garantizan que las variables $x_{ij}$ son binarias y las $y_{ij}$ no negativas, respectivamente.

### 2.4.2. Problema de ruteo de vehículos

En el modelo TSP formulado en la sección 2.4.1 se supone que el vehículo no tiene restricción de capacidad y, por tanto, puede recoger todo el pedido en un único viaje. Esta hipótesis, sin embargo, no suele ser muy realista en los problemas de preparación de pedidos ya que el vehículo, generalmente, tiene una capacidad máxima en peso (o en volumen) y por tanto son necesarios más de un viaje o ruta para preparar completamente el pedido. Las características de este sistema pueden modelizarse mediante el VRP. En la versión básica de este modelo (Capacitated VRP o CVRP) se trata de determinar el conjunto de rutas que deben realizarse para visitar un conjunto de clientes desde un almacén central con el objetivo de minimizar la distancia total recorrida. Se supone que los vehículos son idénticos y salen siempre desde el almacén central. Estas restricciones reflejan las características del proceso de preparación de pedidos en el almacén ya que, en este caso, las rutas comienzan y terminan en el PR. Notar, no obstante, que a diferencia del CVRP, en el almacén no se realizan las rutas simultáneamente sino una detrás de otra puesto que sólo hay un vehículo. Este hecho permitirá en la sección 2.4.3 desarrollar una variante del problema que permite modelizar el hecho de que alguno de los productos deban llegar al PR antes de un tiempo especificado. De entre las formulaciones que se han propuesto para el CVRP ([10], [11] y [5]) se va a presentar la formulación con dos índices y la formulación con tres índices [10].

Sea $C$ la capacidad del vehículo y sea $|K|$ una cota superior sobre el número de rutas que puede realizar el vehículo en el almacén para cumplimentar el pedido. Este valor corresponde al número de vehículos disponibles en la formulación general del CVRP. Sea $K = \{1, ..., |K|\}$ el conjunto de los viajes posibles. Dado $S \subseteq N$, se denota por $r(S)$ el mínimo número de viajes necesarios para recoger todos los productos de $S$.

#### 2.4.2.1. Formulación con dos índices

Para $i, j \in N_0$ se define la variable:

$$x_{ij} = \begin{cases} 1 & \text{si la ruta utiliza el arco } (i, j) \\ 0 & \text{en caso contrario} \end{cases}$$

El CVRP se formula como el siguiente modelo de programación entera mixta:

$$\min \sum_{i \in N_0} \sum_{j \in N_0} d_{ij} x_{ij} \tag{2.8}$$

$$\text{sujeto a} \sum_{i \in N_0} x_{ij} = 1, \qquad j \in N \tag{2.9}$$

$$\sum_{j\in N_0} x_{ij} = 1, \qquad i \in N \tag{2.10}$$

$$\sum_{i\in N} x_{i0} = \sum_{j\in N} x_{0j}, \tag{2.11}$$

$$\sum_{j\in N} x_{0j} \leq |K|, \tag{2.12}$$

$$\sum_{i\notin S}\sum_{j\in S} x_{ij} \geq r(S), \qquad S \subseteq N, S \neq \emptyset \tag{2.13}$$

$$x_{ij} \in \{0,1\}, \qquad i,j \in N_0 \tag{2.14}$$

La función objetivo (2.8) minimiza la distancia total recorrida. Las restricciones (2.9) y (2.10) aseguran que cada nodo se visita una sola vez. Las restricciones (2.11) y (2.12) consiguen controlar que el número de veces que sale el vehículo del PR es el mismo que el número de veces que llega y este es menor o igual que el número de viajes posibles del vehículo. Las restricciones (2.13) imponen la eliminación de subtours, es decir, rutas que no incluyen la salida y llegada al PR. Finalmente, con las restricciones (2.14) se imponen que las variables $x_{ij}$ sean variables binarias.

Las restricciones (2.13) que imponen la eliminación de subtours crece exponencialmente al añadir productos a recoger. Lo cual hace que sea casi imposible de resolver directamente el problema de programación entera. En la literatura, se han propuesto otras formulaciones compactas alternativas. De entre ellas, se va a presentar la formulación de Miller-Tucker-Zemlin (MTZ).

Esta formulación precisa definir para cada nodo $i \in N$, la variable $u_i$ que representa la carga que lleva el vehículo al salir del nodo $i$.

La nueva formulación se obtiene sustituyendo las restricciones (2.13) por las restricciones:

$$u_i + q_j - C(1 - x_{ij}) \leq u_j \qquad i,j \in N \tag{2.15}$$

$$q_i \leq u_i, \qquad i \in N \tag{2.16}$$

$$u_i \leq C, \qquad i \in N \tag{2.17}$$

Las restricciones (2.15), (2.16) y (2.17) imponen la restricción de capacidad y los requisitos de conectividad o eliminación de subtours. En efecto, si $x_{ij} = 0$ las restricciones (2.15) no restringen mientras que si $x_{ij} = 1$ imponen que se salga del nodo $j$ con al menos la carga con la que llegó desde el nodo $i$ más la demanda del nodo $j$.

#### 2.4.2.2. Formulación con tres índices

La formulación con dos índices no permite conocer directamente en qué viaje recoge el vehículo cada producto. Esto puede ser de interés si se imponen restricciones de que algún producto se deba recoger en un viaje específico. Por ello, se presenta a continuación la formulación del CVRP con tres índices.

Para $i, j \in N_0$, $k \in K$ se define la variable:

$$x_{ij}^k = \begin{cases} 1 & \text{si la ruta utiliza el arco } (i,j) \text{ en el viaje } k \\ 0 & \text{en caso contrario} \end{cases}$$

El CVRP con tres índices se formula como el siguiente modelo de programación entera mixta:

$$\min \sum_{i\in N_0}\sum_{j\in N_0}\sum_{k\in K} d_{ij} x_{ij}^k \tag{2.18}$$

$$\text{sujeto a } \sum_{i\in N_0}\sum_{k\in K} x_{ij}^k = 1, \qquad j \in N \tag{2.19}$$

$$\sum_{j\in N_0} x_{ij}^k = \sum_{j\in N_0} x_{ji}^k, \qquad i \in N, k \in K \tag{2.20}$$

$$\sum_{i\in N} x_{i0}^k = \sum_{j\in N} x_{0j}^k, \qquad k \in K \tag{2.21}$$

$$\sum_{j\in N} x_{0j}^k \leq \sum_{j\in N} x_{0j}^{k-1}, \qquad k \in K \tag{2.22}$$

$$\sum_{j\in N} x_{0j}^k \leq 1, \qquad k \in K \tag{2.23}$$

$$u_i + q_j - C(1 - x_{ij}^k) \leq u_j, \qquad i, j \in N, k \in K \tag{2.24}$$

$$q_i \leq u_i, \qquad i \in N \tag{2.25}$$

$$u_i \leq C, \qquad i \in N \tag{2.26}$$

$$x_{ij}^k \in \{0,1\}, \qquad i, j \in N_0, k \in K \tag{2.27}$$

La función objetivo (2.18) minimiza la distancia total recorrida. Las restricciones (2.19) y (2.20) aseguran que cada nodo se visita una única vez y que el vehículo con el que se llega a un nodo es el mismo con el que se sale de él. Las restricciones (2.21), controlan que si se inicia un viaje con el vehículo $k$, este termina en el PR. Las restricciones (2.22) y (2.23) controlan que los vehículos se tomen en orden y que cada viaje se haga a lo sumo una única vez. Las restricciones (2.24), (2.25) y (2.26) tienen el significado ya explicado en la sección 2.4.2.1 con la formulación de dos índices. Finalmente, las restricciones (2.27) imponen que las variables $x_{ij}^k$ sean variables binarias.

### 2.4.3. Una variante del VRP con restricciones de tiempo

Como ya se ha indicado, en el ruteo en un almacén las rutas se realizan por el mismo vehículo una detrás de otra, a diferencia del CVRP clásico en el que las rutas son simultáneas. Por ello, se puede estar interesado en conocer cuándo termina cada ruta con objeto de poder garantizar que un producto específico $i_1$ llegue al PR antes de un tiempo preestablecido $T_{i_1}$ desde el momento en el que empieza la preparación del pedido, que un producto llegue en el primer vehículo o que llegue en el último vehículo. Este nuevo modelo que vamos a introducir a continuación precisa la definición de variables adicionales.

Para cada $i \in N$ y $k \in K$, se definen las variables:

$$y_i^k = \begin{cases} 1 & \text{si el producto } i \text{ es recogido en el viaje } k \\ 0 & \text{en caso contrario} \end{cases}$$

$a_i =$ tiempo en el que llega el vehículo que va a recoger el producto del nodo $i$
$a_0^k =$ tiempo en el que llega el vehículo $k$ al PR

Además, se introduce la constante $a_0^0 = 0$.

En este modelo, como se está interesado en restringir el tiempo de llegada a algún producto, se va a trabajar con tiempos $t_{ij}$ en lugar de distancias $d_{ij}$. Como ya se ha indicado anteriormente, como se supone que el vehículo se mueve por el almacén con velocidad constante ambas cantidades son proporcionales.

Con esta notación, se ha construido una nueva formulación con tres índices. Para $M$ suficientemente grande, se tiene:

$$\min \sum_{i\in N_0} \sum_{j\in N_0} \sum_{k\in K} t_{ij} x_{ij}^k \tag{2.28}$$

$$\text{sujeto a } \sum_{k\in K} y_i^k = 1, \qquad i \in N \tag{2.29}$$

$$\sum_{k\in K} y_0^k \leq |K| \tag{2.30}$$

$$\sum_{j\in N_0} x_{ij}^k = \sum_{j\in N_0} x_{ji}^k, \qquad i\in N, k\in K \tag{2.31}$$

$$\sum_{j\in N_0} x_{ij}^k \leq y_i^k, \qquad i\in N, k\in K \tag{2.32}$$

$$\sum_{j\in N} x_{0j}^k \leq \sum_{j\in N} x_{0j}^{k-1}, \qquad k\in K \tag{2.33}$$

$$\sum_{j\in N} x_{0j}^k \leq 1, \qquad k\in K \tag{2.34}$$

$$a_0^{k-1} + t_{0i} - M(1-x_{0i}^k) \leq a_i, \qquad i\in N, k\in K \tag{2.35}$$

$$a_i + t_{ij} - M(1-x_{ij}^k) \leq a_j, \qquad i,j\in N, k\in K \tag{2.36}$$

$$a_j + t_{j0} - M(1-x_{j0}^k) \leq a_0^k, \qquad j\in N, k\in K \tag{2.37}$$

$$a_0^k \leq M\sum_{j\in N} x_{0j}^k, \qquad k\in K \tag{2.38}$$

$$x_{ij}^k \in \{0,1\}, \qquad i,j\in N_0, k\in K \tag{2.39}$$

$$y_i^k \in \{0,1\}, \qquad i,j\in N_0, k\in K \tag{2.40}$$

La función objetivo (2.28) minimiza el tiempo total invertido en las rutas. Las restricciones (2.29) aseguran que cada nodo se visita una única vez y en un único viaje del vehículo. Las restricciones (2.30) controlan que el número de veces que se sale del PR es menor o igual que el número de viajes posibles del vehículo. Las restricciones (2.31) y (2.32) aseguran que el viaje del vehículo con el que se llega a un nodo es el mismo con el que se sale de él y este es el viaje en el que se recoge el producto. Las restricciones (2.33) y (2.34) controlan que los viajes se hagan en orden y que cada viaje se haga a lo sumo una única vez. Las restricciones (2.35), (2.36) y (2.37) garantizan que los tiempos invertidos en realizar cada una de las rutas se van acumulando y por tanto, aseguran los requisitos de conectividad o eliminación de subtours. La restricción (2.38) impone que si el vehículo $k$ no sale del PR, el vehículo llega en el instante cero. Finalmente, las restricciones (2.39) y (2.40) imponen que las variables $x_{ij}^k$ e $y_i^k$ sean variables binarias.

#### 2.4.3.1. Llegada de un producto antes de un tiempo específico

Si se desea garantizar que el producto $i_1$ llegue al PR antes de un tiempo preestablecido $T_{i_1}$ se debe garantizar que el vehículo que lo recoge llega antes de este tiempo $T_{i_1}$. Para $M$ suficientemente grande, esta restricción queda garantizada escribiendo:

$$a_0^k - M(1-y_{i_1}^k) \leq T_{i_1}, \qquad k\in K \tag{2.41}$$

Estas restricciones aseguran que si el producto $i_1$ es recogido por el vehículo $k$, es decir, $y_{i_1}^k = 1$, entonces el vehículo $k$ llega al almacén antes del instante $T_{i_1}$. En otro caso, la restricción no restringe. Estas restricciones se añadirían a la formulación (2.28)-(2.40).

#### 2.4.3.2. Llegada de un producto en el primer vehículo

Si se desea garantizar que el producto $i_1$ llegue al PR en el primer vehículo. Esta restricción queda garantizada escribiendo:

$$\sum_{j\in N_0} x_{i_1 j}^1 = 1 \tag{2.42}$$

Esta restricción garantiza que el vehículo 1 sale desde nodo $i_1$ a algún otro nodo. Esta restricción se añadiría a la formulación (2.28)-(2.40) ó a la formulación de la sección 2.4.2.2 puesto que no es necesario conocer el momento en el que los vehículos llegan al PR.

#### 2.4.3.3. Llegada de un producto en el último vehículo

Sea $r$ el último viaje del vehículo. Entonces,

| viaje r-2 | | viaje r-1 | | viaje r | viaje r+1 | | viaje r+2 |
|---|---|---|---|---|---|---|---|
| $a_0^{r-2}$ | $\leq$ | $a_0^{r-1}$ | $\leq$ | $a_0^r$ | $0 = a_0^{r+1}$ | $=$ | $a_0^{r+2}$ |

Es decir, para el vehículo $r$

$$a_0^{r+1} < a_0^r$$

Por tanto, si $a_0^{r+1} < a_0^r$, ha de garantizarse que el producto $i_1$ se recoge en el viaje $r$, es decir,

$$a_0^{r-1} \leq a_{i_1}$$
$$a_{i_1} \leq a_0^r$$

Esta restricción condicional puede formularse como un conjunto de restricciones lineales introduciendo una variable binaria adicional. Por tanto, en la formulación (2.28)-(2.40) hay que añadir el conjunto de restricciones siguientes:

$$a_0^{k+1} - a_0^k \geq -Mz_k, \qquad k \in K \tag{2.43}$$
$$a_0^{k-1} - a_{i_1} \leq M(1 - z_k), \qquad k \in K \tag{2.44}$$
$$a_{i_1} - a_0^k \leq M(1 - z_k), \qquad k \in K \tag{2.45}$$

Estas restricciones garantizan que la primera vez que $a_0^{k+1} = 0$, como $a_0^{k+1} < a_0^k$, el producto será recogido en el viaje $k$.

# Capítulo 3

# Caso de estudio: un almacén de distribución

## 3.1. Introducción

El almacén logístico que se considera en este estudio está situado en una empresa que se dedica al almacenamiento y distribución de productos. Este almacén no tiene un diseño rectangular estándar. En este capítulo se presentan sus características y se calculan adhoc las distancias entre dos localizaciones de recogida cualesquiera atendiendo a la posibilidad de movimiento del vehículo que se utiliza para recoger los productos. Una vez descrito, se ha diseñado la experiencia computacional con la generación de diferentes escenarios que ilustran el potencial de los modelos de programación matemática introducidos en el capítulo 2.

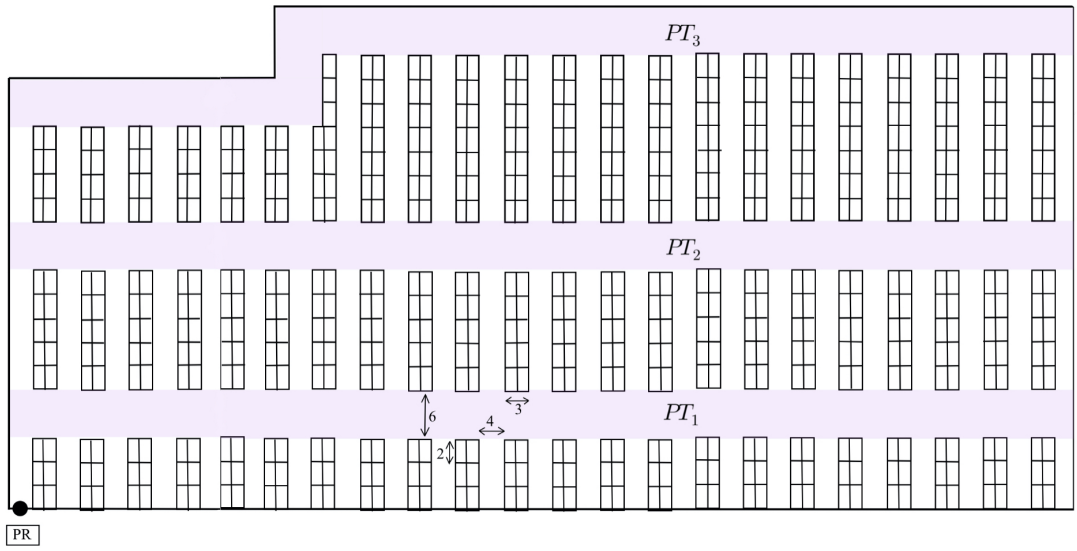


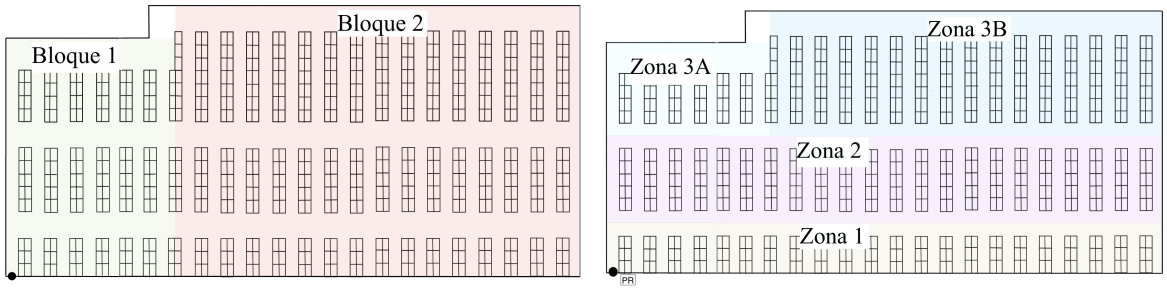

Figura 3.1: Diseño con bloques, pasillos transversales y subpasillos del almacén real bajo estudio

## 3.2. Descripción del almacén

El almacén bajo estudio, cuyo diseño puede verse en la Figura 3.1, está formado por estanterías colocadas en paralelo, con pasillos transversales y subpasillos que facilitan el acceso a las estanterías. El almacén tiene tres pasillos transversales (coloreados en morado en la Figura 3.1), $PT_1, PT_2$ y $PT_3$ y, veintitrés subpasillos. Los pasillos transversales tienen una anchura de 6 metros y los subpasillos de

4 metros. Las estanterías tienen 1,5 metros de fondo y 5 alturas. En total hay 621 estanterías y 3105 productos diferentes. Desde un pasillo transversal no se puede acceder a las estanterías para colocar o recoger productos. El almacén tiene una dimensión de $7300m^2$ y dispone de un único vehículo con capacidad máxima 700 kg.

Las características del almacén permiten su zonificación de varias maneras. En esta memoria, cuando convenga, podrá utilizarse la zonificación en dos bloques que puede verse en la Figura 3.2a o la zonificación en cuatro zonas mostrada en la Figura 3.2b.




(a) Almacén por bloques (b) Almacen por zonas

Figura 3.2: Zonificaciones del almacén

## 3.3. Cálculo de distancias

Uno de los aspectos relevantes en el proceso de modelización del almacén es el cálculo de las distancias entre el PR y/o los productos. En el trabajo cuando se dice distancia entre dos productos se refiere a la distancia entre las localizaciones de recogida de los productos. Este cálculo debe tener en cuenta que el vehículo solo puede realizar movimientos horizontales y verticales (supuesto el almacén situado sobre el plano). A continuación se presentan dos procedimientos equivalentes para el cálculo de las distancias, una compacta en la que se utiliza la zonificación por bloques y otra más detallada en la que se utiliza la zonificación por zonas.

La localización en el plano $(x_i, y_i)$ en el centro de un subpasillo, está asociada a los productos que hay en las estanterías situadas a la izquierda y a la derecha del subpasillo, en las cinco alturas, ya que el vehículo situado en esa posición puede acceder sin problemas a todos ellos. Cada producto tiene asociada su localización, su altura en la estantería y un identificador que indica en qué lado del pasillo se encuentra. Obviamente, a efectos del movimiento del vehículo, la distancia entre dos productos con la misma localización de recogida es cero. Por lo que, los 3105 productos requieren únicamente considerar 324 localizaciones en el plano más el PR.

### 3.3.1. Forma compacta: Zonificación por bloques

En este procedimiento, dados dos productos $i$ y $j$ se identifican formas posibles de acceder desde $i$ hasta $j$ siguiendo los pasillos existentes y se calcula el mínimo de las tres rutas que intuitivamente son las posibles rutas de distancia mínima. Este procedimiento es el que se ha programado en Python para el almacén de estudio (véase apéndice B.1). Estas rutas se muestran en la Figura 3.3. Se distinguen los siguientes casos:

**Caso 1:** Las dos localizaciones se encuentran en el mismo subpasillo, esto es, $x_i = x_j$:

$$d_{ij} = |y_i - y_j|$$

**Caso 2:** Las dos localizaciones no se encuentran en el mismo subpasillo. Se supone, sin pérdida de generalidad, que $x_i < x_j$. Sean en el almacén de estudio, $B = 9$, $M = 25$, $T_1 = 39$ y $T_2 = 45$.

**Caso 2.1:** Las dos localizaciones están en el mismo bloque $B_k$, $k = 1,2$:

$$\begin{aligned} d_{ij} = \min\{&|y_i - B| + x_j - x_i + |y_j - B|, \\ &|y_i - M| + x_j - x_i + |y_j - M|, \\ &|y_i - T_k| + x_j - x_i + |y_j - T_k|\} \end{aligned}$$

**Caso 2.2:** Las dos localizaciones están en diferente bloque, esto es, $(x_i, y_i) \in B_1$ y $(x_j, y_j) \in B_2$:

$$\begin{aligned} d_{ij} = \min\{&|y_i - B| + |x_j - x_i| + |y_j - B|, \\ &|y_i - M| + |x_j - x_i| + |y_j - M|, \\ &|y_i - T_1| + |x_i - P| + |T_2 - T_1| + |x_j - P| + |y_j - T_2|\} \end{aligned}$$

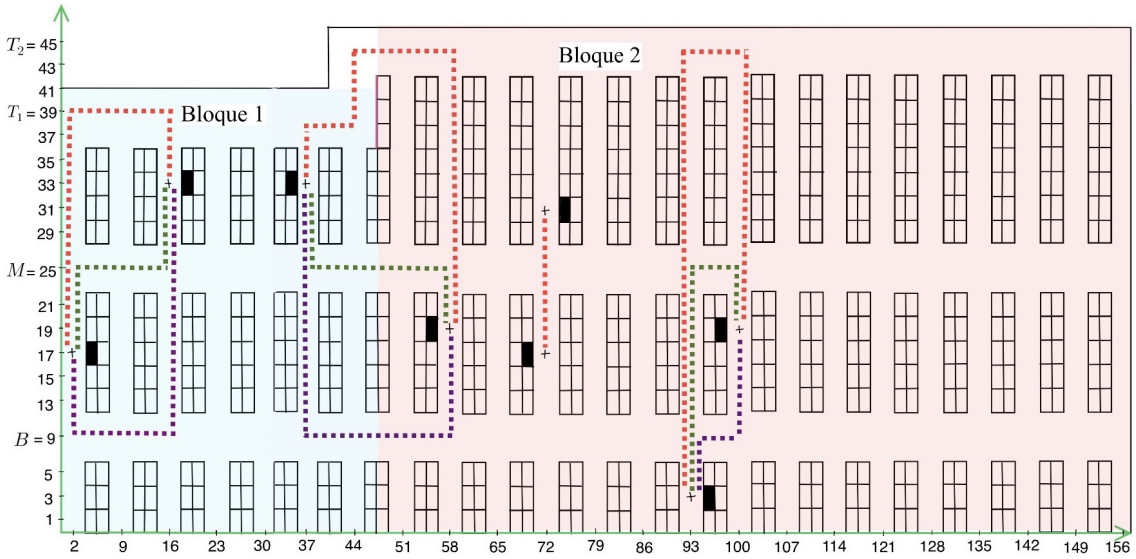



Figura 3.3: Rutas factibles entre dos localizaciones de recogida según el bloque

### 3.3.2. Forma detallada: Zonificación por zonas

En este procedimiento, dependiendo de la zona en la que está la localización de recogida del producto, se detalla la ruta que debe seguir el vehículo para recorrer la distancia mínima. Estas rutas se muestran en la Figura 3.4. Se distinguen los siguientes casos:

**Caso 1:** Las dos localizaciones se encuentran en el mismo subpasillo, esto es, $x_i = x_j$ entonces la ruta es directa de $i$ a $j$ o de $j$ a $i$ según el sentido del vehículo.

**Caso 2:** Las dos localizaciones no están en el mismo subpasillo, esto es, $x_i \neq x_j$. Se supone ahora, sin pérdida de generalidad, que $y_i \leq y_j$:

**Caso 2.1:** Las dos localizaciones están en la misma zona $Z_k$, $k = 1,2,3A,3B$:

**Caso 2.1.1:** Si están en la primera zona, es decir, en $Z_1$, se toma la ruta que atraviesa el primer pasillo transversal $PT_1$.

**Caso 2.1.2:** Si están en $Z_k$, $k = 2,3A,3B$:

**Caso 2.1.2.1:** Si están por encima del punto medio, esto es $y_i, y_j > M_k$, se toma el pasillo transversal que delimita la zona $Z_k$ superiormente.

**Caso 2.1.2.2:** Si están por debajo del punto medio, esto es $y_i, y_j < M_k$, se toma el pasillo transversal que delimita la zona $Z_k$ inferiormente.

**Caso 2.1.2.3:** Si están en el punto medio, esto es $y_i = y_j = M_k$, se toma indistintamente el pasillo transversal que delimita la zona $Z_k$ superiormente o inferiormente.

**Caso 2.1.2.4:** Si la localización de $j$ está en el punto medio, esto es $y_i < y_j = M_k$, se toma el pasillo transversal que delimita la zona $Z_k$ inferiormente.

**Caso 2.1.2.5:** Si la localización de $i$ está en el punto medio, esto es $y_i = M_k < y_j$, se toma el pasillo transversal que delimita la zona $Z_k$ superiormente.

**Caso 2.1.2.6:** Si la localización de $i$ está por debajo del punto medio y la de $j$ por encima, esto es, $y_i < M_k < y_j$:

**Caso 2.1.2.6.1:** Si la distancia de $i$ al punto medio es menor que la de $j$ al punto medio, se toma el pasillo transversal que delimita la zona $Z_k$ superiormente.

**Caso 2.1.2.6.2:** Si la distancia de $i$ al punto medio es mayor que la de $j$ al punto medio, se toma el pasillo transversal que delimita la zona $Z_k$ inferiormente.

**Caso 2.2:** Las dos localizaciones están en diferente zona:

**Caso 2.2.1:** Si la localización de $i$ está en la zona $Z_1$ y la de $j$ en $Z_2$, se toma la ruta que atraviesa el pasillo transversal $P_1$.

**Caso 2.2.2:** Si la localización de $i$ está en la zona $Z_1$ y la de $j$ en $Z_3$, se toma indistintamente la ruta que atraviesa el pasillo transversal $P_1$ ó $P_2$.

**Caso 2.2.3:** Si la localización de $i$ está en la zona $Z_2$ y la de $j$ en $Z_3$, se toma la ruta que atraviesa el pasillo transversal $P_2$.

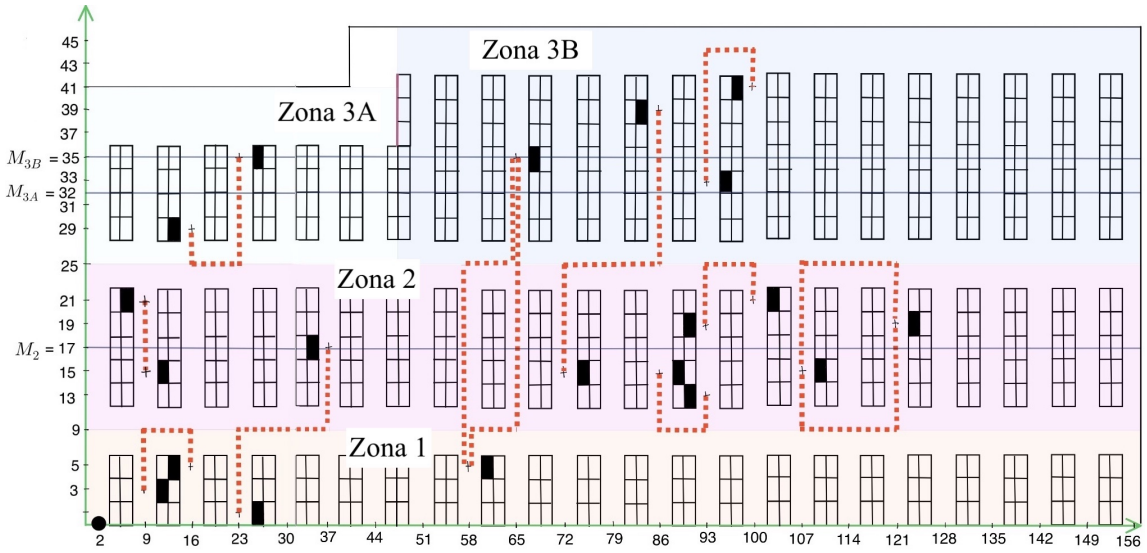



Figura 3.4: Rutas de distancia mínima entre dos localizaciones de recogida según la zona

## 3.4. Experimento computacional

En esta experiencia computacional se va a mostrar el potencial de los modelos CVRP propuestos en el capítulo 2. Las rutas para la preparación de los pedidos dependen de la localización de recogida de los productos, pero también del peso de los mismos. La restricción de capacidad del vehículo, implica cuando el peso de los productos de un pedido supera dicha capacidad, la necesidad de realizar más de un viaje de recogida de productos. Por ello, se van a generar una variedad de escenarios en los que se va modificando el peso de los productos en el pedido y la localización de los mismos. Dado su carácter

ilustrativo, se ha decidido generar los pesos de los productos al azar atendiendo a sus características realistas.

En el almacén bajo estudio, la demanda del producto se realiza por mantos, ya que el vehículo transporta pallets. El peso de un manto de cada uno de los 3105 productos en el almacén se ha generado a partir de una distribución de probabilidad triangular. Los parámetros de la distribución dependen del tipo de producto, ya que se han considerado productos ligeros, de peso mediano y pesados. En particular, de los 3105 productos diferentes, se ha establecido que el 20 % son productos ligeros, cuyo peso en kilogramos sigue una distribución triangular $(10, 25, 40)$; el 70 % son medianos, cuyo peso sigue una distribución triangular $(30, 90, 130)$; y el 10 % son pesados, cuyo peso sigue una distribución triangular $(130, 140, 180)$. En la Figura 3.5 se puede ver un histograma de la distribución de los pesos generados de los mantos de cada producto. Los pesos se han transformado en números enteros por truncamiento. Además, en adelante, cuando se refiere al peso de un producto, se estará refiriendo al peso de un manto del producto.

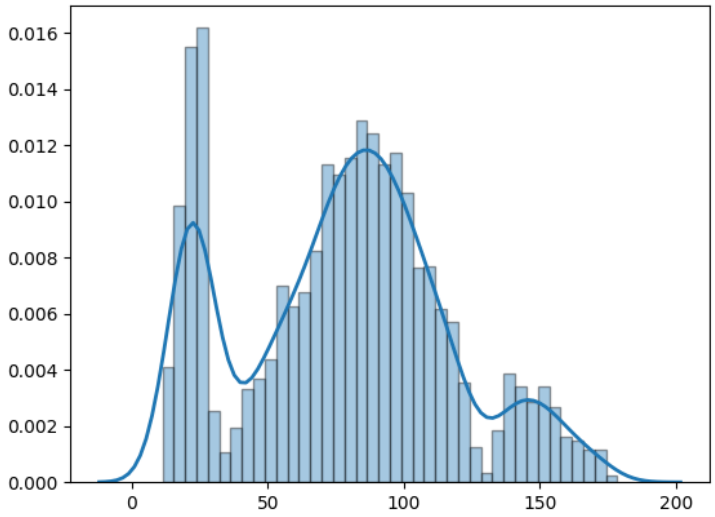



Figura 3.5: Histograma del peso de los 3105 productos

Para el cálculo de la matriz de distancias y la generación de los pesos de los productos se han escrito varios programas en PyCharm (versión 2021.1.2), un IDE que utiliza Python como lenguaje de programación, que se incluyen en los apéndices B.1 y B.4 . Además, en los apéndices B.2 y B.5 se incluyen una parte de la matriz de distancias y de la tabla de los pesos generados. Además, se desarrolló otro programa en Python que dada la identificación del producto obtiene las coordenadas en el plano del producto (véase apéndice B.3).

El listado de productos junto con su demanda es el dato de entrada a los modelos de programación matemática propuestos. En el almacén bajo estudio, la demanda del producto se realiza por mantos, por ello en el trabajo se ha considerado que en cada pedido, el número de mantos que se pueden solicitar de un producto tiene un comportamiento uniforme discreto con rango $\{1, 2, 3\}$. Si el pedido incluye H mantos del producto $i$, el peso que debe recoger el vehículo es $q_i = H\gamma_i$, siendo $\gamma_i$ el peso del producto $i$.

Los pedidos se generan seleccionando, aleatoriamente, el número de productos requeridos de los 3105 productos. El programa de Python que genera los pedidos se incluye en el apéndice B.6. Además, en el apéndice B.8 se incluye el programa en Python para la generación de pedidos que permite al usuario seleccionar los productos que forman el pedido y la demanda de estos. En el apéndice B.7 se incluye un ejemplo del documento que se genera a partir de la información del listado de productos del

pedido y su demanda. Este documento incluye todos los datos que necesita el modelo de optimización implementado en CPLEX para calcular las rutas. En la Figura 3.6 se muestra la tabla con los datos de un pedido de 12 productos.

| idProd | tipo | peso | NumDemanda | PesoDemanda |
|---|---|---|---|---|
| **331** | Pesado | 170 | 3 | 510 |
| **1571** | Pesado | 147 | 2 | 294 |
| **2007** | Pesado | 152 | 2 | 304 |
| **2617** | Pesado | 138 | 1 | 138 |
| **1947** | Mediano | 106 | 2 | 212 |
| **515** | Mediano | 95 | 2 | 190 |
| **1732** | Mediano | 75 | 3 | 225 |
| **595** | Mediano | 87 | 1 | 87 |
| **26** | Mediano | 110 | 1 | 110 |
| **603** | Mediano | 56 | 1 | 56 |
| **1815** | Mediano | 53 | 2 | 106 |
| **1735** | Ligero | 25 | 3 | 75 |

Figura 3.6: Listado de los 12 productos de un pedido

Los modelos de optimización, cuyos códigos se incluyen en los apéndices A.1, A.2 y A.3 se han resuelto usando el programa IBM ILOG CPLEX Optimization Studio (versión 12.8.0.0) con licencia académica en un MacBook Pro 2016 con Intel core i5 CPU 2 GHz y 8 GB RAM. La información sobre los resultados de la resolución se ha recogido en un fichero de texto para facilitar la comparación entre escenarios. Un ejemplo de fichero de resultados se incluye en el apéndice A.4.

En un análisis preliminar se ejecutaron varios escenarios y se observó que pedidos con más de 12 productos exigían un tiempo de computación para su resolución demasiado grande, por encima de 10 horas, y se consideró elevado para poder realizar una experiencia computacional completa. Por tanto, en el estudio computacional se han resuelto problemas con 10 y 12 productos. Además, se ha establecido como criterio de parada 15 minutos. Al terminar, CPLEX genera un documento que incluye el valor de la función objetivo, que es óptima si la resolución ha finalizado antes de los 15 minutos o es la mejor solución encontrada antes de su interrupción a los 15 minutos. También incluye el tiempo de computación del programa, el número de viajes que realiza el vehículo, las rutas óptimas de recogida del pedido y, para cada viaje del vehículo, el número de productos que recoge, el tiempo que tarda en realizarlo y el peso que carga (Véase un ejemplo de este documento en el apéndice A.4).

En problemas de naturaleza combinatoria, como el tratado en este trabajo, es habitual el desarrollo de algoritmos heurísticos cuando el tamaño del problema requiere gran esfuerzo computacional para su resolución exacta que garantizaría la optimalidad de la solución. En el apéndice A.5 se ha incluido a modo ilustrativo, la resolución con el procedimiento heurístico disponible en `https://sites.google.com/site/vrphlibrary/` de un pedido de 40 productos.

### 3.4.1. Evaluación de los modelos de dos y tres índices en términos del tiempo de computación.

El objetivo de esta sección es comparar los resultados de las formulaciones del CVRP presentados en 2.4.2.1 y 2.4.2.2. Para ello, se han generado varios escenarios en los que los pedidos se diferencian

en el peso y la localización de recogida de los productos que lo componen para evaluar su influencia en el tiempo computacional y las soluciones óptimas.

#### 3.4.1.1. Escenarios con pedidos según características de peso

En este apartado se han generado pedidos con 10 y con 12 productos. Para poder valorar como influye el peso de los productos, se han generado en cada caso 5 pedidos que incluyen solo productos elegidos entre los ligeros, 5 pedidos que incluyen solo productos elegidos entre los pesados y 5 pedidos generales con productos elegidos aleatoriamente de entre los 3105 existentes.

Los datos de entrada de cada pedido se han recogido en ficheros que se denotan por $P_{n,A}$, donde $n$ indica el número de productos y $A \in \{G, L, P\}$ para indicar si los productos en el pedido son ligeros $(L)$, pesados $(P)$ o cualesquiera $(G)$. En total se han resuelto 30 problemas. Los pedidos diferenciados por peso, se han generado con el programa de Python que se inlcuye en el apéndice B.9. Además, la cota superior del número de viajes del vehículo $|K|$, se estableció en 8 para los pedidos de 10 productos y en 9 para los de 12 productos.

| | | | Dos índices | | Tres índices | |
|---|---|---|---|---|---|---|
| Pedidos | Viajes | Objetivo | Tiempo (CPU) | Gap | Tiempo (CPU) | Gap |
| $P_{10,G}$ | 2 - 3 | 694.0 | 10.72 | | 139.52 | |
| $P_{10,L}$ | 1 | 467.6 | 1.65 | | 54.86 | |
| $P_{10,P}$ | 5 - 7 | 1307.2 | 0.70 | | 8.67 | |
| $P_{12,G}$ | 3 - 4 | 869.6 | 294.98 | 0.03 | 900.00 | 0.43 |
| $P_{12,L}$ | 1 - 3 | 510.4 | 195.94 | 0.03 | 395.10 | 0.13 |
| $P_{12,P}$ | 6 - 8 | 1567.6 | 1.17 | | 132.56 | |

Tabla 3.1: Resultados obtenidos de los modelos con dos y tres índices según peso (Tiempo de CPU en segundos)

La tabla 3.1 resume los resultados obtenidos. Las columnas tiempo (CPU) indican el tiempo en segundos invertido por CPLEX en la resolución. Para todos los pedidos con 10 productos se ha alcanzado la solución óptima. En esta tabla puede verse que el número de viajes empleado en la solución varía entre 2 y 3 para los pedidos generales, solo necesita un viaje en los cinco pedidos de los productos ligeros, y necesita entre 5 y 7 viajes en los pedidos de productos pesados. En la columna objetivo se indica el valor, en media, de las distancias totales recorridas en los cinco pedidos de cada categoría.

En los pedidos con 12 productos, CPLEX alcanza la solución óptima con la formulación de dos índices en todos los pedidos de productos pesados y en cuatro de los cinco pedidos en los que los productos son ligeros o generales. En la formulación con tres índices, CPLEX alcanza la solución óptima en todos los pedidos con productos pesados, en cuatro de los cinco de productos ligeros y se detiene al aplicar el criterio de parada en todos los pedidos de productos generales. En los problemas en los que se ha interrumpido la ejecución al alcanzar los 15 minutos, el valor de la función objetivo es el mejor valor obtenido hasta ese momento. Una idea sobre la proximidad de dicha solución al óptimo lo proporciona el valor del gap relativo definido como:

$$\text{gap relativo} = \frac{|\text{mejor solución relajada} - \text{mejor solución entera}|}{10^9 + |\text{mejor solución entera}|}$$

Con respecto a los tiempos de cálculo, como sería esperable, es mayor en la formulación con tres índices. Notar que, por ejemplo, en un pedido con 12 productos el modelo con dos índices tiene 181

variables mientras que el modelo con tres índices tiene 1533 variables. Por otro lado, incrementar el número de productos también eleva mucho en ambas formulaciones el tiempo de cálculo. Igualmente, merece la pena destacar que cuando el pedido consta solo de productos pesados, el tiempo de CPU es mucho más pequeño. Una posible explicación es que el vehículo es capaz de cargar menos productos en el mismo viaje y, por tanto, aunque hace más viajes estos son "menos complicados"de obtener porque recogen menos productos.

#### 3.4.1.2. Escenarios con pedidos según la localización de los productos

En este apartado, se han generado también pedidos con 10 y con 12 productos. Para poder analizar cómo influye la localización de recogida de los productos, se han generado diferentes pedidos según su zona. En los pedidos de 10 productos se han generado 5 pedidos que incluyen tres productos de la zona 1, tres productos de la zona 2, dos productos de la zona 3A y dos de la zona 3B; 5 pedidos que incluyen todos los productos de la zona 2; y 5 pedidos que incluyen cinco productos de la zona 3A y cinco de la zona 3B. En los pedidos de 12 productos, se han generado 5 pedidos que incluyen cuatro productos de la zona 1, cuatro productos de la zona 2, dos productos de la zona 3A y dos de la zona 3B; 5 pedidos que incluyen todos los productos de la zona 2; y 5 pedidos que incluyen seis productos de la zona 3A y seis de la zona 3B. Todos los productos se han elegido de forma aleatoria con las características impuestas.

Los datos de entrada de cada pedido se han recogido en ficheros que se denotan por $\mathrm{P}_{n,A}$, donde $n$ indica el número de productos y $A \in \{T, Z_2, Z_3\}$ para indicar si los productos en el pedido son de todas las zonas $(T)$, de la zona 2 $(Z_2)$ o de la zona 3 $(Z_3)$. En total se han resuelto 30 problemas. Conviene notar que, igual que en el caso anterior, se eligen aleatoriamente los productos y el número de mantos de demanda lo que permite calcular el peso total que el vehículo debe recoger. Los pedidos diferenciados por zona, se han generado con el programa de Python incluido en el apéndice B.11.

| | | | Dos índices | | Tres índices | |
|---|---|---|---|---|---|---|
| Problema | Viajes | Objetivo | Tiempo (CPU) | Gap | Tiempo (CPU) | Gap |
| $P_{10,T}$ | 2 - 4 | 668.4 | 6.48 | | 456.50 | 0.05 |
| $P_{10,Z_2}$ | 3 | 724.0 | 6.97 | | 596.73 | 0.05 |
| $P_{10,Z_3}$ | 2 - 4 | 675.6 | 7.04 | | 479.26 | 0.03 |
| $P_{12,T}$ | 3 - 4 | 795.2 | 128.12 | | 900.00 | 0.38 |
| $P_{12,Z_2}$ | 2 - 3 | 737.2 | 645.03 | 0.09 | 900.00 | 0.45 |
| $P_{12,Z_3}$ | 3 - 4 | 820.0 | 197.73 | | 900.00 | 0.34 |

Tabla 3.2: Resultados obtenidos de los modelos con dos y tres índices según zona (Tiempo de CPU en segundos)

La tabla 3.2 muestra los resultados obtenidos. Las columnas Tiempo (CPU) indican el tiempo en segundos invertido por CPLEX en la resolución. Para todos los pedidos con 10 productos se ha alcanzado la solución óptima con la formulación de dos índices. En la formulación con tres índices, alcanza la solución en tres de los cinco pedidos con productos de todas las zonas, en cuatro de cinco de los pedidos con productos de la zona 2 y en cuatro de los cinco pedidos con productos de la zona 3. En ella puede verse que el número de viajes necesarios para recoger todos los productos varía entre 2 y 4 viajes para los pedidos con productos en todas las zonas, necesita 3 viajes en los cinco pedidos de los productos de la zona 2, y necesita entre 2 y 4 viajes en los pedidos de productos de la zona 3. En la columna objetivo se indica el valor, en media, de las distancias totales recorridas en los cinco pedidos de cada categoría.

En los pedidos con 12 productos CPLEX alcanza la solución óptima con la formulación de dos ín-

dices en todos los pedidos de productos de todas las zonas y de la zona 3 y, en dos de los cinco cuando los productos son de la zona 2. En la formulación con tres índices, no alcanza la solución óptima en ninguno de los pedidos. Sin embargo, en todas se obtiene la misma solución que con la formulación de dos índices.

Con respecto a los tiempos de cálculo, como en el apartado 3.4.1.1, es mayor en la formulación con tres índices. Para los pedidos con 10 productos la zona no parece ser relevante en el tiempo de CPU. Sin embargo, en los problemas con 12 productos, en la formulación con dos índices, sí que hay una diferencia notable en los tiempos para alcanzar la solución óptima según la categoría del pedido, precisando bastante más los pedidos de la categoría $Z_2$. Por otro lado, incrementar el número de productos también eleva mucho en ambas formulaciones el tiempo de cálculo.

### 3.4.2. Formulación con tres índices y restricciones adicionales de tiempo

Uno de los aspectos novedosos de este trabajo es la inclusión de una variante del CVRP que tiene en cuenta la existencia de restricciones sobre el tiempo de llegada al PR de algunos de los productos (sección 2.4.3). Esta variante se ilustra en el caso de estudio con un pedido generado de 10 productos. Además se supone que el vehículo recorre el almacén a velocidad constante de $1m/s$, por lo que la distancia recorrida en metros es igual al tiempo invertido en segundos, es decir, $t_{ij} = d_{ij}$. Al resolver el modelo (2.28)-(2.40) se obtienen las siguientes rutas.

| Viaje | Recorrido | Peso total | Llegada al PR |
|---|---|---|---|
| $R_1$ | 0 - 7 - 1 - 0 | 634 | 280 |
| $R_2$ | 0 - 2 - 4 - 0 | 565 | 532 |
| $R_3$ | 0 - 3 - 6 - 8 - 0 | 628 | 818 |
| $R_4$ | 0 - 5 - 9 - 10 - 0 | 604 | 1202 |

Notar que el tiempo total invertido en los viajes es de 1202 segundos y el tiempo de CPU invertido en la resolución es de 252.17 segundos.

A continuación, se presentan varios escenarios imponiendo las restricciones temporales de capítulo 2. El código para implementar el modelo con estas restricciones se incluye en el apéndice A.3. En primer lugar, se impone que los productos $i_1 = 5, i_2 = 8$ lleguen al PR antes de los 290 segundos (restricción (2.41)). La solución obtenida incluye las siguientes rutas:

| Viaje | Recorrido | Peso total | Tiempo de llegada al PR |
|---|---|---|---|
| $R_1$ | 0 - 8 - 5 - 4 - 0 | 612 | 290 |
| $R_2$ | 0 - 7 - 1 - 0 | 634 | 570 |
| $R_3$ | 0 - 6 - 3 - 0 | 572 | 856 |
| $R_4$ | 0 - 2 - 9 - 10 - 0 | 613 | 1240 |

Los productos 5 y 8 se han incluido ahora en el primer viaje, que también recoge el producto 4. Este primer viaje llega al PR en 290 segundos. La solución del modelo original sin restricciones añadidas no verifica las restricciones de tiempo de recogida de los productos 5 y 8, que llegan alPR al cabo de 1202 y 818 segundos respectivamente. Por ello, el tiempo total invertido en los viajes de la nueva solución es mayor, 1240 segundos. El tiempo de CPU es de 450.96 segundos.

Si se impone que, por ejemplo, el producto $i_1 = 4$ llegue al PR en el primer vehículo (restricción (2.42)) las rutas obtenidas son:

| Viaje | Recorrido | Peso total | Tiempo de llegada al PR |
|---|---|---|---|
| $R_1$ | 0 - 2 - 4 - 0 | 565 | 252 |
| $R_2$ | 0 - 3 - 6 - 8 - 0 | 628 | 538 |
| $R_3$ | 0 - 5 - 9 - 10 - 0 | 604 | 922 |
| $R_4$ | 0 - 7 - 1 - 0 | 634 | 1202 |

En este caso, se garantiza que el producto 4 llega en el primer vehículo sin incrementar el tiempo total de los viajes, que sigue siendo 1202 segundos. Es decir, el problema tiene, para un valor de la función objetivo de 1202, múltiples soluciones óptimas que reordenan los viajes y al resolver el modelo con la restricción añadida ha colocado el viaje que recoge el producto 4 como primer viaje. Notar que el tiempo de CPU invertido en resolver el modelo (2.18)-(2.27) con la restricción adicional (2.42) es de 351.84 segundos.

Si se impone que, por ejemplo, los productos $i_1 = 5, i_2 = 8$ lleguen al PR en el primer vehículo (restricción (2.42)) y los productos $i_3 = 4, i_4 = 6$ lleguen al PR en el último vehículo (restricciones (2.43)-(2.45)) se obtienen las rutas:

| Ruta | Recorrido | Peso total | Tiempo de llegada al PR |
|---|---|---|---|
| $R_1$ | 0 - 8 - 5 - 9 - 10 - 0 | 660 | 384 |
| $R_2$ | 0 - 1 - 7 - 0 | 634 | 664 |
| $R_3$ | 0 - 2 - 0 | 432 | 916 |
| $R_4$ | 0 - 3 - 0 | 282 | 1194 |
| $R_5$ | 0 - 4 - 6 - 0 | 423 | 1438 |

En este caso, los productos 5 y 8 se han incluido en el primer viaje, que también recoge los productos 9 y 10. En el último viaje se han incluido únicamente los productos 4 y 6. El tiempo total invertido en los cinco viajes es 1438 segundos, como es de esperar, es mayor puesto que el modelo tiene ahora más restricciones. El tiempo de CPU es de 639.41 segundos.

# Bibliografía


[1] K.J. ROODBERGEN AND R. DE KOSTER, Routing order pickers in a warehouse with a middle aisle, *European Journal of Operational Research* **133** (2001), 32–43.

[2] N. SHETTY AND B. SAH AND S.H. CHUNG, Route optimization for warehouse order picking operations via vehicle routing and simulation, *Springer Nature Applied Sciences* **2** (2020), 1–18.

[3] T. PETERS, Warehouse processes: receiving and put-away, *Amer. Math. Monthly* **96** (1) (1989), 41–42.

[4] T. DANTZIG AND R.FULKERSON AND S. JOHNSON, Solution of a large-scale traveling- salesman problem, *Journal of the operations research society of America* **2** (4) (1954), 393–410.

[5] L. PANSART, Exact algorithms for picking problems, Université Grenoble Alpes ,2016.

[6] H. RATLIFF AND A. ROSENTHAL, Order-Picking in a rectangular warehouse: A solvable case of the Traveling Salesman Problem, *Operations Research* **31** (1983), 507–521.

[7] G. DUKIC, Order-picking routing policies : Simple heuristics, advanced heuristics or optimal algorithm, *Journal of Mechanical Engineering 50(2004)* **50** (2004), 530–531.

[8] B. GAVISH AND S. GRAVES, The travelling salesman problem and related problems, *Operations research center* (1978)

[9] G. CORNUÉJOLS AND J. FONLUPT, AND D. NADDEF, The traveling salesman problem on a graph and some related integer polyhedra, *Mathematical programming* **33** (1) (1985), 1–27.

[10] P. TOTH AND D. VIGO, The Vehicle Routing Problem, Universitá degli Studi di Bologna, Bologna, Italy, 2002.

[11] L. PANSART AND N. CATUSSE AND H. CAMBAZARD, Exact algorithms for the order picking problem, *Computers and Operations Research* **100** (2018), 117–127.

[12] T. CHABOT AND R. LAHYANI AND L. COELHO AND J. RENAUD, Order Picking Problems under Weight , Fragility and Category Constraints, *Cirrelt* **49** (2010), 1000–1005.

[13] A. LETCHFORD AND D. NASIRI AND D. THEIS, Compact formulations of the Steiner Traveling Salesman Problem and related problems, *European Journal of Operational Research* **228** (2013), 83–92.

[14] E. ZUNIC AND A. BESIREVIC AND S. DELALIC AND K. HODZIC AND H. HASIC, *A generic approach for order picking optimization process in different warehouse layouts*, University of Sarajevo, Faculty of Science, Bosnia and Herzegovina, 2018.

# Apéndice A

# Modelos de optimización en CPLEX

A continuación, se van a presentar los códigos utilizados en CPLEX Studio para implementar las tres formulaciones presentadas en el Capítulo 2 del problema de optimización.

## A.1. Formulación con dos índices

[ ]:

```
//Criterio de parada
 execute PARAMS {
     cplex.tilim = 900;
     //cplex.epagap = 20;
     //cplex.epgap =0.01;
    }

 // Defino las variables fijas

 int NProd = ...; // número de productos
 range r=0..NProd;
 range r1=1..NProd;
 int C = ... ; //capacidad del vehículo
 int NVehic = ... ; //número de vehículos

 int dem[1..NProd] = ... ; //demanda de cada producto
 int d[0..NProd][0..NProd] = ... ; //distancia entre dos productos


 // Defino variables de decisión

 dvar int u[1..NProd]; //carga al salir del nodo i
 dvar boolean x[0..NProd][0..NProd]; //arco en ruta óptima

 minimize sum (i in 0..NProd) sum (j in 0..NProd) d[i][j]*x[i][j];

 subject to {

   //cada nodo se visita una sola vez
   forall (j in 1..NProd)
     sum(i in 0..NProd) x[i][j] == 1;
```

```
forall (i in 1..NProd)
  sum(j in 0..NProd) x[i][j] == 1;


//vehículos que entran = vehículos que salen
sum(i in 1..NProd) x[i][0] == sum(j in 1..NProd) x[0][j] ;


//total de viajes de vehículos menor o igual que los disponibles
sum(j in 1..NProd) x[0][j] <= NVehic;


//no se supera la carga máxima del vehículo
forall (i in 1..NProd)
  forall(j in 1..NProd)
    u[i]+dem[j]-C*(1-x[i][j]) <= u[j];


 //en cada nodo se coge toda la demanda que no supera la capacidad del
 vehículo
 forall(i in 1..NProd){
   dem[i] <= u[i];
   u[i] <= C;
 }
}


   main{
     //creo el documento de texto
     var f = new IloOplOutputFile("Ruta12_4L_2.txt");
     thisOplModel.generate();

     var time0 = new Date();
     var OK = cplex.solve();
     var time1 = new Date();
     var numViaje = 0;
     var pTotal = 0;

     var tTotal = (time1.getTime()-time0.getTime())/1000;

     if (OK) {
         f.writeln("Ruta:");
         for(var i in thisOplModel.r){
             for (var j in thisOplModel.r){
                 if(thisOplModel.x[i][j] != 0){
                     f.writeln("De "+i+" a "+j);
                 }
             }
         }
         for (var l in thisOplModel.r1){
```

```
                    numViaje =  numViaje + thisOplModel.x[0][l];
            }

            for (var j in thisOplModel.r1){
                pTotal = pTotal + thisOplModel.dem[j];
            }
        f.writeln(" ");
        f.writeln("El peso total que debe llevar la carretilla es "+pTotal);
        f.writeln(" ");
        f.writeln("El número de viajes de la carretilla es "+numViaje);
        f.writeln(" ");
        f.writeln("Tiempo: ",tTotal," segundos.Solución con objetivo: "+cplex.
        getObjValue( ));
        f.writeln(" ");
        f.writeln("Gap relativo:", cplex.getMIPRelativeGap());
        f.writeln(" ");
        f.writeln("Gap absoluto:", cplex.getMIPRelativeGap()*cplex.getObjValue(
         ));

        f.close()
        }
    }
```

## A.2. Formulación con tres índices

[ ]:
```
    //Criterio de parada
 execute PARAMS {
      cplex.tilim = 900;
    //cplex.epagap = 20;
     //cplex.epgap =0.01;
}

 // Defino las variables fijas

 int NProd = ...; // número de productos
 range r=0..NProd;
 range r1=1..NProd;
 int C = ... ; //capacidad del vehículo
 int NVehic = ... ; //número de vehículos
 range r2=1..NVehic;

 int dem[1..NProd] = ... ; //demanda de cada producto

 int d[0..NProd][0..NProd] = ... ; //distancia entre dos productos


 // Defino variables de decisión

 dvar int u[1..NProd]; //carga al salir del nodo i
 dvar boolean x[0..NProd][0..NProd][1..NVehic]; //arco en ruta óptima


 minimize sum (i in 0..NProd) sum (j in 0..NProd) sum (k in 1..NVehic)␣
 ↪d[i][j]*x[i][j][k];

 subject to {

   //llego a todos los productos
   forall (j in 1..NProd)
     sum(i in 0..NProd) sum (k in 1..NVehic) x[i][j][k] == 1;


   //entras y sales con el mismo vehículo
   forall (i in 1..NProd, k in 1..NVehic)
     sum(j in 0..NProd) x[i][j][k] == sum(j in 0..NProd) x[j][i][k];


   //vehiculos que entran = vehículos que salen
   forall (k in 1..NVehic)
     sum(i in 1..NProd) x[i][0][k] == sum(j in 1..NProd) x[0][j][k];

   //Impongo que coja vehículos por orden
      forall(k in 2..NVehic)
        sum(j in 1..NProd) x[0][j][k] <= sum(j in 1..NProd) x[0][j][k-1];
```

```
  //Con cada vehículo un único viaje
  forall (k in 1..NVehic)
    sum(j in 1..NProd) x[0][j][k] <= 1;

  //no se supera la carga máxima del vehículo
  forall (k in 1..NVehic, i in 1..NProd, j in 1..NProd)
    u[i]+dem[j]-C*(1-x[i][j][k]) <= u[j];


  //en cada nodo se coge toda la demanda que no supera la capacidad del␣
↪vehículo
  forall(i in 1..NProd){
     dem[i] <= u[i];
     u[i] <= C;

     }

  }

   main{
       //creo el docuento de texto
       var f = new IloOplOutputFile("Ruta12_Z_3.txt");
       thisOplModel.generate();

       var time0 = new Date();
       var OK = cplex.solve();
       var time1 = new Date();
       var pTotal = 0;
       var numViaje = 0;
       var suma = 0;
       var peso = 0;
       var tiempo = 0;

       var tTotal = (time1.getTime()-time0.getTime())/1000;

       if (OK) {
           f.writeln("Ruta:");
           for(var k in thisOplModel.r2){
               for (var j in thisOplModel.r){
                   for(var i in thisOplModel.r){
                       if(thisOplModel.x[i][j][k] != 0){
                           f.writeln("De "+i+" a "+j+" en el vehículo " + k);
                       }
                   }
               }
           }
           for (var l in thisOplModel.r1){
               for (var k in thisOplModel.r2){
                   numViaje =  numViaje + thisOplModel.x[0][l][k];
               }
```

```
            }
            f.writeln(" ");
            f.writeln("El número de viajes de la carretilla es "+numViaje);
            f.writeln(" ");
            for(var k in thisOplModel.r2){
                for (var j in thisOplModel.r){
                    for(var i in thisOplModel.r){
                        suma = suma +thisOplModel.x[i][j][k];
                        tiempo = tiempo +thisOplModel.x[i][j][k]*thisOplModel.
↪d[i][j];
                    }
                }
                for (var j in thisOplModel.r1){
                    for(var i in thisOplModel.r){
                        peso = peso +thisOplModel.x[i][j][k]*thisOplModel.
↪dem[j];
                    }
                }
                if (suma!=0){
                    f.writeln("El número de productos en la carretilla "+k+"␣
↪es "+(suma-1));
                    f.writeln("El peso cargado por la carretilla "+k+" es␣
↪"+peso);
                    f.writeln("El tiempo recorrido por la carretilla "+k+" es␣
↪"+tiempo);
                    f.writeln(" ");
                    suma = 0;
                    peso =0;
                    tiempo = 0;
                }
            }

            for (var j in thisOplModel.r1){
                pTotal = pTotal + thisOplModel.dem[j];
            }
        }

    f.writeln("El peso total que debe llevar la carretilla es "+pTotal);
    f.writeln(" ");
    f.writeln("Tiempo: ",tTotal," segundos. Solución con objetivo: "+cplex.
↪getObjValue( ));
    f.writeln(" ");
    f.writeln("Gap relativo:", cplex.getMIPRelativeGap());
    f.writeln(" ");
    f.writeln("Gap absoluto:", cplex.getMIPRelativeGap()*cplex.getObjValue(␣
↪));


    f.close()
  }
```

## A.3. Variante de la formulación con tres índices

A continuación se presenta el programa introducido en CPLEX Studio con la variante de la formulación de tres índices del VRP junto con los códigos que se deberían añadir al programa para imponer las restricciones adicionales.

[ ]:
```
execute PARAMS {
          cplex.tilim = 900;
        //cplex.epagap = 20;
         //cplex.epgap =0.01;
}

// Defino las variables fijas

int NProd = ...;
range r=0..NProd;
range r1=1..NProd;
int C = ... ; //capacidad vehículo
int NVehic = ... ; //número vehículos
range r2 = 1..NVehic;

int M = 10000 ; // distancia máximo de vehículo

int dem[1..NProd] = ... ; //demanda de cada producto

int d[0..NProd][0..NProd] = ... ; //distancia entre dos productos


// Defino variables de decisión

dvar int u[1..NProd]; //carga al salir del nodo i
dvar int a[1..NProd]; //instante de llegada al nodo i
dvar int a0[1..NVehic]; //instante de llegada al nodo 0 del vehículo k
dvar boolean x[0..NProd][0..NProd][1..NVehic]; //arco en ruta óptima
dvar boolean y[0..NProd][1..NVehic]; //producto i en viaje k

minimize sum (i in 0..NProd) sum (j in 0..NProd) sum (k in 1..NVehic)␣
↪d[i][j]*x[i][j][k];

subject to {

  //Llego a todos los productos
  forall(i in 1..NProd)
          sum(k in 1..NVehic) y[i][k] == 1;


       //No supero el numero de vehículos disponibles
  sum(k in 1..NVehic) y[0][k] <= NVehic;

  //Vehic que salen = vehic que entran
  forall (i in 1..NProd, k in 1..NVehic)
    sum(j in 0..NProd) x[i][j][k] == sum(j in 0..NProd) x[j][i][k];
```

```
  //Con cada vehículo un único viaje
  forall (k in 1..NVehic)
    sum(j in 1..NProd) x[0][j][k] <= 1;

  //Impongo que coja vehículos por orden
         forall(k in 2..NVehic)
           sum(j in 1..NProd) x[0][j][k] <= sum(j in 1..NProd) x[0][j][k-1];


        //Si recorro arco con un vehículo, este es con el que recojo el␣
↪producto
  forall (i in 1..NProd, k in 1..NVehic)
    sum(j in 0..NProd) x[i][j][k] == y[i][k];


  //No se supera la carga máxima del vehículo
  forall (k in 1..NVehic, i in 1..NProd, j in 1..NProd)
    u[i]+dem[j]-C*(1-x[i][j][k]) <= u[j];


   //En cada nodo se coge toda la demanda que no supera la capacidad del␣
↪vehículo
   forall(i in 1..NProd){
     dem[i] <= u[i];
     u[i] <= C;
     d[0][i] <= a[i];
     }

   //Restricciones adicionales: tiempos
       forall (i in 1..NProd, j in 1..NProd){
         d[0][i]-M*(1-x[0][i][1]) <= a[i];
         a[i]+d[i][j]-M*(1-x[i][j][1]) <= a[j];
        a[j]+d[j][0]-M*(1-x[j][0][1]) <= a0[1];
         }

       forall (k in 2..NVehic, i in 1..NProd, j in 1..NProd){
         a0[k-1]+d[0][i]-M*(1-x[0][i][k]) <= a[i];
         a[i]+d[i][j]-M*(1-x[i][j][k]) <= a[j];
         a[j]+d[j][0]-M*(1-x[j][0][k]) <= a0[k];
         }

       forall (k in 1..NVehic)
          a0[k] <= M* sum( j in 1..NProd) x[0][j][k];


  }



   main{
```

```
        //creo el docuento de texto
        var f = new IloOplOutputFile("Ruta10_RA1.txt");
        thisOplModel.generate();

        var time0 = new Date();
        var OK = cplex.solve();
        var time1 = new Date();
        var pTotal = 0;
        var numViaje = 0;
        var suma = 0;
        var peso = 0;
        var tiempo = 0;


        var tTotal = (time1.getTime()-time0.getTime())/1000;

        if (OK) {
            f.writeln("Ruta:");
            for(var k in thisOplModel.r2){
                for (var j in thisOplModel.r){
                    for(var i in thisOplModel.r){
                        if(thisOplModel.x[i][j][k] != 0){
                            f.writeln("De "+i+" a "+j+" en el vehículo " + k);
                        }
                    }
                }
            }
            for (var l in thisOplModel.r1){
                for (var k in thisOplModel.r2){
                    numViaje =  numViaje + thisOplModel.x[0][l][k];
                }
            }
            f.writeln(" ");
            f.writeln("El número de viajes de la carretilla es "+numViaje);
            f.writeln(" ");
            for(var k in thisOplModel.r2){
                for (var j in thisOplModel.r){
                    for(var i in thisOplModel.r){
                        suma = suma +thisOplModel.x[i][j][k];
                        tiempo = tiempo +thisOplModel.x[i][j][k]*thisOplModel.
↪d[i][j];
                    }
                }
                for (var j in thisOplModel.r1){
                    for(var i in thisOplModel.r){
                        peso = peso +thisOplModel.x[i][j][k]*thisOplModel.
↪dem[j];
                    }
                }
                if (suma!=0){
```

```
                    f.writeln("El número de productos en la carretilla "+k+"␣
↪es "+(suma-1));
                    f.writeln("El peso cargado por la carretilla "+k+" es␣
↪"+peso);
                    f.writeln("El tiempo recorrido por la carretilla "+k+" es␣
↪"+tiempo);
                    f.writeln("El tiempo total hasta iniciar con la␣
↪carretilla "+k+" es "+thisOplModel.a0[k]);
                    f.writeln(" ");
                    suma = 0;
                    peso =0;
                    tiempo = 0;
                }
            }

            for (var j in thisOplModel.r1){
                pTotal = pTotal + thisOplModel.dem[j];
            }
        }

    f.writeln("El peso total que debe llevar la carretilla es "+pTotal);
    f.writeln(" ");
    f.writeln("Tiempo: ",tTotal," segundos. Solución con objetivo: "+cplex.
↪getObjValue( ));
    f.writeln(" ");
    f.writeln("Gap relativo:", cplex.getMIPRelativeGap());
    f.writeln(" ");
    f.writeln("Gap absoluto:", cplex.getMIPRelativeGap()*cplex.getObjValue(␣
↪));

    f.close()
  }
```

[ ]:
```
//Cojo los productos 5 y 8 antes de 290 segundos
          forall(k in 1..NVehic){
                  a0[k]-M*(1-y[5][k]) <= 290;
                  a0[k]-M*(1-y[8][k]) <= 290;
                  }
```

[ ]:
```
//Impongo que el producto 4 se recoja en el primer viaje
          sum(j in 0..NProd) x[4][j][1] == 1;
```

[ ]:
```
//Impongo que el vehículo recoja el producto
          a0[2] - a0[1] >= -M*z[1];
          0 - a[4] <= M*(1-z[1]);
          a[4] - a0[2] <= M*(1-z[1]);

          forall (k in 2..(NVehic-1)){
                  a0[k+1] - a0[k] >= -M*z[k];
                  a0[k-1] - a[4] <= M*(1-z[k]);
                  a[4] - a0[k] <= M*(1-z[k]);
```

## A.4. Solución CPLEX

Al implementar el modelo de tres índices en CPLEX Studio, se genera un fichero de texto con los resultado obtenidos que se presenta a continuación.

[ ]:
```
Ruta:
De 1 a 0 en el vehículo 1
De 0 a 1 en el vehículo 1
Ruta1:0-1-0

De 10 a 0 en el vehículo 2
De 9 a 6 en el vehículo 2
De 6 a 8 en el vehículo 2
De 0 a 9 en el vehículo 2
De 8 a 10 en el vehículo 2
Ruta2: 0-9-6-8-10-0

De 5 a 0 en el vehículo 3
De 0 a 3 en el vehículo 3
De 3 a 4 en el vehículo 3
De 4 a 5 en el vehículo 3
Ruta3: 0-3-4-5-0

De 11 a 0 en el vehículo 4
De 0 a 2 en el vehículo 4
De 12 a 7 en el vehículo 4
De 7 a 11 en el vehículo 4
De 2 a 12 en el vehículo 4
Ruta4: 0-2-12-7-11-0

El número de viajes de la carretilla es 4
El peso total que debe llevar la carretilla es 2307


El número de productos en la carretilla 1 es 1
El peso cargado por la carretilla 1 es 510
El tiempo recorrido por la carretilla 1 es 80

El número de productos en la carretilla 2 es 4
El peso cargado por la carretilla 2 es 443
El tiempo recorrido por la carretilla 2 es 164

El número de productos en la carretilla 3 es 3
El peso cargado por la carretilla 3 es 654
El tiempo recorrido por la carretilla 3 es 332

El número de productos en la carretilla 4 es 4
El peso cargado por la carretilla 4 es 700
El tiempo recorrido por la carretilla 4 es 260

Tiempo: 900.039666992 segundos. Solución con objetivo: 836
```

```
Gap relativo: 0.468288103
Gap absoluto: 391.488853789
```

## A.5. Estrategia heurística para un pedido de 40 productos

A continuación, se presenta un ejemplo resolución con el procedimiento heurístico disponible en `https://sites.google.com/site/vrphlibrary/` para un pedido de 40 productos.

| Ruta | Recorrido |
|---|---|
| $R_1$ | 0 - 4 - 2 - 35 - 0 |
| $R_2$ | 0 - 6 - 10 - 27 - 34 - 26 - 23 - 0 |
| $R_3$ | 0 - 7 - 24 - 19 - 18 - 0 |
| $R_4$ | 0 - 12 - 9 - 1 - 40 - 0 |
| $R_5$ | 0 - 14 - 20 - 0 |
| $R_6$ | 0 - 16 - 37 - 8 - 29 - 31- 0 |
| $R_7$ | 0 - 25 - 11 - 5 - 22 - 28 - 0 |
| $R_8$ | 0 - 30 - 3 - 15 - 33 - 38 - 32 - 0 |
| $R_9$ | 0 - 36 - 21 - 13 - 17 - 39 - 0 |

Tabla A.1: Solución con objetivo: 2238 y CPU: 1.26

# Apéndice B

# Programas en Python

## B.1. Cálculo de las distancias

A continuación se presentan dos códigos en Python. En el primero se ha programado una función que calcula la distancia entre cualesquiera dos localizaciones del almacén y el segundo genera la matriz de distancias para todos los productos del almacén de estudio.

[ ]:
```
# coding=utf-8
#Programa para funcion de distancias en almacen
import math

#ordenadas de los pasillos transversales
B = 9
M = 25
T1 = 39
T2 = 45

#abcisas de los cambios de bloque
P1 = 44

d = []
dB = 0
dM = 0
dT = 0

pB1 = []
pB2 = []
def distancia(p1,p2):
    dB = abs(p1[1] - B) + abs(p1[0] - p2[0]) + abs(p2[1] - B)
    dM = abs(p1[1] - M) + abs(p1[0] - p2[0]) + abs(p2[1] - M)

    if p1[0] == p2[0]:
         d = abs(p1[1]-p2[1])
            return d
    elif p1[0] <= P1 and p2[0] <= P1: #ambos pertenecen al Bloque1
        dT = abs(p1[1] - T1) + abs(p1[0] - p2[0]) + abs(p2[1] - T1)

    elif p1[0] > P1 and p2[0] > P1: #ambos pertenecen al Bloque2
```

```
        dT = abs(p1[1] - T2) + abs(p1[0] - p2[0]) + abs(p2[1] - T2)
    else:
        if p1[0] <= P1:  #el p1 es del Bloque1
            pB1 = p1
            pB2 = p2
        else:          # el p2 es del Bloque1
            pB1 = p2
            pB2 = p1
        dT = abs(pB1[1] - T1) + abs(pB1[0] - 44) + abs(T1 - T2) + abs(44 -
        pB2[0]) + abs(pB2[1] - T2)
        # realmente podría escribirse como dT = abs(p1[1] - T2) + abs(p1[0]
        - p2[0]) + abs(p2[1] - T2)
    d = min(dB, dM, dT)
    return d
```

[ ]:
```
#Programa para matriz de distancias en almacen
from funcionDistancia import distancia
import pandas as pd
import numpy as np


#Genero las localizaciones de los productos con sus coordenadas:
#producto=(idLocalizacion, coordenadas)


coordY1 =[1,3,5,13,15,17,19,21,29,31,33,35]
coordY2 =[1,3,5,13,15,17,19,21,29,31,33,35,37,39,41]

coordX1=[]
for i in range(7):
    coordX1.append(2+7*i)

print(coordX1)

coordX2=[]
for i in range(16):
    coordX2.append(51+7*i)

print(coordX2)

p1 = np.arange(84)
prod1 = p1 + 1

p2 = np.arange(240)
prod2 = p2 + 85


producto = [[0,2,0]]#deposito
for k in range(len(coordX1)):
    for i in range(len(coordY1)):
            producto.append([prod1[i+12*(k)], coordX1[k], coordY1[i]])
```

```
for k in range(len(coordX2)):
    for i in range(len(coordY2)):
            producto.append([prod2[i+15*(k)], coordX2[k], coordY2[i]])
print(producto)
print(producto[0][-2:],producto[1][-2:])

#Calculo las distancias según las coordenadas con la función distancia

d = distancia(producto[0][-2:],producto[1][-2:])

print(d)

matriz = np.zeros((len(producto), len(producto)),int)

for i in range(len(producto)):
    for j in range(len(producto)):
        matriz[i][j] = distancia(producto[i][-2:],producto[j][-2:])

print(matriz)


#Compruebo que es simetrica
tras=np.transpose(matriz)
print(tras)


print(np.array_equal(matriz, tras))

columnas = [i for i in range(325)]

#Exporto la matriz distancia a una excel

df = pd.DataFrame(matriz, columns=columnas)
df2 = pd.DataFrame(matriz)

print(df2)

df2.to_csv ('MatrizDistancias.csv', index = False, header=True)
```

## B.2. Submatriz de distancias

A continuación, se muestra la matriz de distancias que el programa anterior exporta a una Excel. La matriz de distancias del almacén de estudio es una matriz 325$x$325 por lo que se muestra una submatriz.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | 1 | 3 | 5 | 13 | 15 | 17 | 19 | 21 | 29 | 31 | 33 | 35 | 24 | 22 | 20 | 20 | 22 | 24 | 26 | 28 | 36 | 38 | 40 | 42 | 31 | 29 | 27 | 27 |
| **1** | **1** | 0 | 2 | 4 | 12 | 14 | 16 | 18 | 20 | 28 | 30 | 32 | 34 | 23 | 21 | 19 | 19 | 21 | 23 | 25 | 27 | 35 | 37 | 39 | 41 | 30 | 28 | 26 | 26 |
| **2** | **3** | 2 | 0 | 2 | 10 | 12 | 14 | 16 | 18 | 26 | 28 | 30 | 32 | 21 | 19 | 17 | 17 | 19 | 21 | 23 | 25 | 33 | 35 | 37 | 39 | 28 | 26 | 24 | 24 |
| **3** | **5** | 4 | 2 | 0 | 8 | 10 | 12 | 14 | 16 | 24 | 26 | 28 | 30 | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 23 | 31 | 33 | 35 | 37 | 26 | 24 | 22 | 22 |
| **4** | **13** | 12 | 10 | 8 | 0 | 2 | 4 | 6 | 8 | 16 | 18 | 20 | 22 | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 23 | 23 | 25 | 27 | 29 | 26 | 24 | 22 | 22 |
| **5** | **15** | 14 | 12 | 10 | 2 | 0 | 2 | 4 | 6 | 14 | 16 | 18 | 20 | 21 | 19 | 17 | 17 | 19 | 21 | 23 | 21 | 21 | 23 | 25 | 27 | 28 | 26 | 24 | 24 |
| **6** | **17** | 16 | 14 | 12 | 4 | 2 | 0 | 2 | 4 | 12 | 14 | 16 | 18 | 23 | 21 | 19 | 19 | 21 | 23 | 21 | 19 | 19 | 21 | 23 | 25 | 30 | 28 | 26 | 26 |
| **7** | **19** | 18 | 16 | 14 | 6 | 4 | 2 | 0 | 2 | 10 | 12 | 14 | 16 | 25 | 23 | 21 | 21 | 23 | 21 | 19 | 17 | 17 | 19 | 21 | 23 | 32 | 30 | 28 | 28 |
| **8** | **21** | 20 | 18 | 16 | 8 | 6 | 4 | 2 | 0 | 8 | 10 | 12 | 14 | 27 | 25 | 23 | 23 | 21 | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 34 | 32 | 30 | 30 |
| **9** | **29** | 28 | 26 | 24 | 16 | 14 | 12 | 10 | 8 | 0 | 2 | 4 | 6 | 35 | 33 | 31 | 23 | 21 | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 42 | 40 | 38 | 30 |
| **10** | **31** | 30 | 28 | 26 | 18 | 16 | 14 | 12 | 10 | 2 | 0 | 2 | 4 | 37 | 35 | 33 | 25 | 23 | 21 | 19 | 17 | 17 | 19 | 21 | 19 | 44 | 42 | 40 | 32 |
| **11** | **33** | 32 | 30 | 28 | 20 | 18 | 16 | 14 | 12 | 4 | 2 | 0 | 2 | 39 | 37 | 35 | 27 | 25 | 23 | 21 | 19 | 19 | 21 | 19 | 17 | 46 | 44 | 42 | 34 |
| **12** | **35** | 34 | 32 | 30 | 22 | 20 | 18 | 16 | 14 | 6 | 4 | 2 | 0 | 41 | 39 | 37 | 29 | 27 | 25 | 23 | 21 | 21 | 19 | 17 | 15 | 48 | 46 | 44 | 36 |
| **13** | **24** | 23 | 21 | 19 | 19 | 21 | 23 | 25 | 27 | 35 | 37 | 39 | 41 | 0 | 2 | 4 | 12 | 14 | 16 | 18 | 20 | 28 | 30 | 32 | 34 | 23 | 21 | 19 | 19 |
| **14** | **22** | 21 | 19 | 17 | 17 | 19 | 21 | 23 | 25 | 33 | 35 | 37 | 39 | 2 | 0 | 2 | 10 | 12 | 14 | 16 | 18 | 26 | 28 | 30 | 32 | 21 | 19 | 17 | 17 |
| **15** | **20** | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 23 | 31 | 33 | 35 | 37 | 4 | 2 | 0 | 8 | 10 | 12 | 14 | 16 | 24 | 26 | 28 | 30 | 19 | 17 | 15 | 15 |
| **16** | **20** | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 23 | 23 | 25 | 27 | 29 | 12 | 10 | 8 | 0 | 2 | 4 | 6 | 8 | 16 | 18 | 20 | 22 | 19 | 17 | 15 | 15 |
| **17** | **22** | 21 | 19 | 17 | 17 | 19 | 21 | 23 | 21 | 21 | 23 | 25 | 27 | 14 | 12 | 10 | 2 | 0 | 2 | 4 | 6 | 14 | 16 | 18 | 20 | 21 | 19 | 17 | 17 |
| **18** | **24** | 23 | 21 | 19 | 19 | 21 | 23 | 21 | 19 | 19 | 21 | 23 | 25 | 16 | 14 | 12 | 4 | 2 | 0 | 2 | 4 | 12 | 14 | 16 | 18 | 23 | 21 | 19 | 19 |
| **19** | **26** | 25 | 23 | 21 | 21 | 23 | 21 | 19 | 17 | 17 | 19 | 21 | 23 | 18 | 16 | 14 | 6 | 4 | 2 | 0 | 2 | 10 | 12 | 14 | 16 | 25 | 23 | 21 | 21 |
| **20** | **28** | 27 | 25 | 23 | 23 | 21 | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 20 | 18 | 16 | 8 | 6 | 4 | 2 | 0 | 8 | 10 | 12 | 14 | 27 | 25 | 23 | 23 |
| **21** | **36** | 35 | 33 | 31 | 23 | 21 | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 28 | 26 | 24 | 16 | 14 | 12 | 10 | 8 | 0 | 2 | 4 | 6 | 35 | 33 | 31 | 23 |
| **22** | **38** | 37 | 35 | 33 | 25 | 23 | 21 | 19 | 17 | 17 | 19 | 21 | 19 | 30 | 28 | 26 | 18 | 16 | 14 | 12 | 10 | 2 | 0 | 2 | 4 | 37 | 35 | 33 | 25 |
| **23** | **40** | 39 | 37 | 35 | 27 | 25 | 23 | 21 | 19 | 19 | 21 | 19 | 17 | 32 | 30 | 28 | 20 | 18 | 16 | 14 | 12 | 4 | 2 | 0 | 2 | 39 | 37 | 35 | 27 |
| **24** | **42** | 41 | 39 | 37 | 29 | 27 | 25 | 23 | 21 | 21 | 19 | 17 | 15 | 34 | 32 | 30 | 22 | 20 | 18 | 16 | 14 | 6 | 4 | 2 | 0 | 41 | 39 | 37 | 29 |
| **25** | **31** | 30 | 28 | 26 | 26 | 28 | 30 | 32 | 34 | 42 | 44 | 46 | 48 | 23 | 21 | 19 | 19 | 21 | 23 | 25 | 27 | 35 | 37 | 39 | 41 | 0 | 2 | 4 | 12 |
| **26** | **29** | 28 | 26 | 24 | 24 | 26 | 28 | 30 | 32 | 40 | 42 | 44 | 46 | 21 | 19 | 17 | 17 | 19 | 21 | 23 | 25 | 33 | 35 | 37 | 39 | 2 | 0 | 2 | 10 |
| **27** | **27** | 26 | 24 | 22 | 22 | 24 | 26 | 28 | 30 | 38 | 40 | 42 | 44 | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 23 | 31 | 33 | 35 | 37 | 4 | 2 | 0 | 8 |
| **28** | **27** | 26 | 24 | 22 | 22 | 24 | 26 | 28 | 30 | 30 | 32 | 34 | 36 | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 23 | 23 | 25 | 27 | 29 | 12 | 10 | 8 | 0 |
| **29** | **29** | 28 | 26 | 24 | 24 | 26 | 28 | 30 | 28 | 28 | 30 | 32 | 34 | 21 | 19 | 17 | 17 | 19 | 21 | 23 | 21 | 21 | 23 | 25 | 27 | 14 | 12 | 10 | 2 |
| **30** | **31** | 30 | 28 | 26 | 26 | 28 | 30 | 28 | 26 | 26 | 28 | 30 | 32 | 23 | 21 | 19 | 19 | 21 | 23 | 21 | 19 | 19 | 21 | 23 | 25 | 16 | 14 | 12 | 4 |
| **31** | **33** | 32 | 30 | 28 | 28 | 30 | 28 | 26 | 24 | 24 | 26 | 28 | 30 | 25 | 23 | 21 | 21 | 23 | 21 | 19 | 17 | 17 | 19 | 21 | 23 | 18 | 16 | 14 | 6 |
| **32** | **35** | 34 | 32 | 30 | 30 | 28 | 26 | 24 | 22 | 22 | 24 | 26 | 28 | 27 | 25 | 23 | 23 | 21 | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 20 | 18 | 16 | 8 |
| **33** | **43** | 42 | 40 | 38 | 30 | 28 | 26 | 24 | 22 | 22 | 24 | 26 | 28 | 35 | 33 | 31 | 23 | 21 | 19 | 17 | 15 | 15 | 17 | 19 | 21 | 28 | 26 | 24 | 16 |
| **34** | **45** | 44 | 42 | 40 | 32 | 30 | 28 | 26 | 24 | 24 | 26 | 28 | 26 | 37 | 35 | 33 | 25 | 23 | 21 | 19 | 17 | 17 | 19 | 21 | 19 | 30 | 28 | 26 | 18 |
| **35** | **47** | 46 | 44 | 42 | 34 | 32 | 30 | 28 | 26 | 26 | 28 | 26 | 24 | 39 | 37 | 35 | 27 | 25 | 23 | 21 | 19 | 19 | 21 | 19 | 17 | 32 | 30 | 28 | 20 |

## B.3. Producto a localización

A continuación se muestra el programa que contiene la función que dado un producto cualquiera, devuelve la localización en el almacén de estudio.

[ ]:
```
# coding=utf-8
#funcion que dado el id del producto en el almacén devuelve el id de la␣
 ↪localizacion en la que está

import numpy as np

#Generamos las localizaciones de los productos
a = np.arange(12)+1
b = np.repeat(a,5)

for i in range(6):
    a = np.arange(12)+(12*(i+1)+1)
    a = np.repeat(a, 5)
    a = np.tile(a, 2)
    b = np.concatenate((b, a))

for i in range(15):
    a = np.arange(15)+(15*i+85)
    a = np.repeat(a, 5)
    a = np.tile(a, 2)
    b = np.concatenate((b, a))


a=np.arange(15)+310
a = np.repeat(a, 5)
b = np.concatenate((b, a))


#Generamos los id de los productos
numProd=3105
prod=[]

for i in range(numProd):
    prod.append(i+1)


#Asignamos a cada producto su localización
def ProdALoc(p):
    if p == 0:
        return 0
    for i in range(numProd):
        if p == prod[i]:
            return b[i]
```

## B.4. Generación de pesos

A continuación se incluye el programa que genera los pesos de los productos de manera aleatoria y exporta esta información a una Excel.

[ ]:
```
# coding=utf-8

import random
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

np.random.seed(0)

numProd = 3105
prod = []

#Mezclamos los id de los productos para que los pesos no tengan un orden␣
 ↪específico
for i in range(numProd):
    prod.append(i + 1)

random.shuffle(prod)

#Dividimos los productos en tres grupos según los porcentajes de cada peso

ref_P = np.split(prod, [int(.1 * len(prod)), int(.8 * len(prod))])[0]
ref_M = np.split(prod, [int(.1 * len(prod)), int(.8 * len(prod))])[1]
ref_L = np.split(prod, [int(.1 * len(prod)), int(.8 * len(prod))])[2]


#Generamos los pesos con distribuciones triangulares

pesosP = np.random.triangular(130, 140, 180, len(ref_P))
pesosM = np.random.triangular(30, 90, 130, len(ref_M))
pesosL = np.random.triangular(10, 25, 30, len(ref_L))

#En un principio dejamos los pesos con dos decimales
pesos = list(pesosP)
for i in range(len(pesosP)):
    pesos[i] =round(pesos[i],2)

for i in range(len(pesosM)):
    pesos.append(round(list(pesosM)[i],2))

for i in range(len(pesosL)):
    pesos.append(round(list(pesosL)[i],2))
```

```
#Creamos una columna con la característica del peso de cada prodcuto

tipo = []
for i in range(len(ref_P)):
    tipo.append("Pesado")
for i in range(len(ref_M)):
    tipo.append("Mediano")
for i in range(len(ref_L)):
    tipo.append("Ligero")

#Realizamos un histograma de los pesos
h = plt.hist(pesos, bins=200,density=True)
plt.show()


#Exportamos los pesos a una Excel
datos = {'idProd': prod,
        'tipo': tipo,
        'peso': pesos }

df = pd.DataFrame(datos, columns = ['idProd', 'tipo', 'peso'])

df.to_csv ('TablaPesos.csv', index = False, header=True)
```

## B.5. Tabla de pesos

A continuación se presenta la tabla de pesos generada por el programa anterior. Puesto que hay 3105 productos, se muestra una parte de esta tabla.

| idProd | tipo | peso |
|---|---|---|
| 1 | Mediano | 54.28 |
| 2 | Mediano | 109.0 |
| 3 | Ligero | 25.11 |
| 4 | Mediano | 55.56 |
| 5 | Mediano | 71.54 |
| 6 | Mediano | 92.63 |
| 7 | Mediano | 98.98 |
| 8 | Ligero | 20.71 |
| 9 | Mediano | 88.2 |
| 10 | Mediano | 90.41 |
| 11 | Ligero | 23.54 |
| 12 | Ligero | 18.84 |
| 13 | Ligero | 14.95 |
| 14 | Mediano | 112.91 |
| 15 | Pesado | 146.95 |
| 16 | Mediano | 86.72 |
| 17 | Ligero | 14.04 |
| 18 | Pesado | 138.13 |
| 19 | Mediano | 95.55 |
| 20 | Mediano | 69.06 |
| 21 | Mediano | 65.46 |
| 22 | Mediano | 99.61 |
| 23 | Mediano | 50.33 |
| 24 | Mediano | 64.06 |
| 25 | Ligero | 24.39 |
| 26 | Mediano | 110.04 |
| 27 | Mediano | 116.87 |
| 28 | Mediano | 85.86 |
| 29 | Mediano | 75.95 |
| 30 | Mediano | 87.13 |

## B.6. Generación de pedidos

A continuación se presenta el código creado para generar un pedido aleatorio de 12 productos.

[ ]:
```
# coding=utf-8
import os
import random
import pandas as pd
import numpy as np
from funcionProdALoc import ProdALoc
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import seaborn as sns

np.random.seed(0)

#Importamos la tabla con los pesos de cada producto ya fijados

tablaPes= pd.read_csv("TablaPesosFija.csv", header = 0, sep = ',' )

#print(tablaPes)

matriz = tablaPes.to_numpy()

#Importamos la matriz de distancia entre cada producto
MatDis= pd.read_csv("MatrizDistancias.csv", header = 0, sep = ',')

matrizD =MatDis.to_numpy()


#Indicamos la longitud del pedido que vamos a generar

longPedido = 12

#Seleccionamos al azar los productos del pedido
idProdPedi=[]
for i in range(longPedido):
    idProdPedi.append(random.randint(1,3105))

print(idProdPedi)

#Completamos la informacion del pedido con los pesos de los productos
pedido=[]

for i in range(3105):
    for j in range(longPedido):
        if idProdPedi[j] == matriz[i][0]:
            pedido.append(matriz[i])

print(pedido)
```

```
#La demanda se genera aleatoriamente entre 1,2 ó 3

demanda=[]
for i in range(len(pedido)):
    demanda.append(random.randint(1,3))

for i in range(len(pedido)):
    pedido[i][2] = int(pedido[i][2]) #pesos a enteros


print(pedido)

#Calculamos la demanda (en peso) total según la demanda del producto
demandaP=[]
for i in range(len(pedido)):
    demandaP.append(demanda[i]*pedido[i][2])

#Exportamos a una excel la información de cada pedido

df2 = pd.DataFrame(pedido, columns = ['idProd', 'tipo', 'peso'])


df2['NumDemanda'] = demanda
df2['PesoDemanda'] = demandaP

print(df2)

df2.to_csv ('Pedido10_3.csv', index = False, header=True)


#distancias entre productos especificos

prod=[0]
for i in range(len(pedido)):
    prod.append(pedido[i][0])
print(prod) #productos

l = np.zeros((len(prod)),int)

for i in range(len(prod)):
    l[i] = ProdALoc(prod[i])

print(l) #localizaciones

#Hallamos la submatriz de distancias para los productos específicos
subm = np.zeros((len(pedido)+1, len(pedido)+1),int)

for i in range(len(pedido)+1):
    for j in range(len(pedido) + 1):
        subm[i][j] = matrizD[l[i]][l[j]]
```

```
print(subm)


#Exportamos a un documento de texto los datos necesarios para CPLEX

file = open("Pedido12_3.txt", "w")

# Numero de productos
file.write("//Los productos del pedido son " + str(prod[1:]) + os.linesep)

file.write("//Numero de productos en el pedido" + os.linesep)
file.write("NProd = " + str(len(pedido)))
file.write(";" + os.linesep)
file.write( os.linesep)

#Capacidad Vehículo
file.write("//Capacidad en peso del vehículo" + os.linesep)
file.write("C = 700" )
file.write(";" + os.linesep)
file.write( os.linesep)

#Numero de Vehículos
file.write("//Numero de vehiculos" + os.linesep)
file.write("NVehic = 5" )
file.write(";" + os.linesep)
file.write( os.linesep)


#Demanda por producto
file.write("//Demanda por producto" + os.linesep)
file.write("dem =" + str(demandaP) )
file.write(";" + os.linesep)
file.write( os.linesep)

#Distancias entre productos
file.write("//Distancia entre productos" + os.linesep)
file.write("d = " + str(subm) )
file.write(";" + os.linesep)



file.close()
```

## B.7. Documento pedido para CPLEX

A continuación, se presenta el documento de texto de un pedido de 12 productos generado con el programa anterior que se introduce a CPLEX Studio.

[ ]:
```
//Los productos del pedido son [331, 1571, 2007, 2617, 1947, 515, 1732, 595,␣
 ↪26, 603, 1815, 1735]
//Numero de productos en el pedido
NProd = 12;

//Capacidad en peso del vehículo
C = 700;

//Numero de vehiculos
NVehic = 5;


//Demanda por producto
dem =[510, 294, 304, 138, 212, 190, 225, 87, 110, 56, 106, 75];

//Distancia entre productos
d = [[  0  40 113 122 154 127  47 124  68  17  52 126 124]
 [ 40   0  73  98 122  87  19  84  28  35  32  86  84]
 [113  73   0  33  57  22  66  19  61  96  77  21  19]
 [122  98  33   0  40  19  91  30  86 121  86  32  30]
 [154 122  57  40   0  43 115  54 110 145 118  56  54]
 [127  87  22  19  43   0  80  19  75 110  91  21  19]
 [ 47  19  66  91 115  80   0  77  21  42  25  79  77]
 [124  84  19  30  54  19  77   0  72 107  88   2   0]
 [ 68  28  61  86 110  75  21  72   0  51  32  74  72]
 [ 17  35  96 121 145 110  42 107  51   0  51 109 107]
 [ 52  32  77  86 118  91  25  88  32  51   0  90  88]
 [126  86  21  32  56  21  79   2  74 109  90   0   2]
 [124  84  19  30  54  19  77   0  72 107  88   2   0]];
```

## B.8. Generación de pedidos por usuario

A continuación se presenta el código del programa en el que el usuario introduce el programa con el pedido deseado y se exporta un documento con los datos necesarios para resolver el modelo en CPLEX.

[ ]:
```
# coding=utf-8
#Programa para calculas los datos necesarios en CPLEX preguntadno al usuario
import os
import pandas as pd
import numpy as np
from funcionProdALoc import ProdALoc


#Tomamos la matriz de distancias

MatDis= pd.read_csv("MatrizDistancias.csv", header = 0, sep = ',')

matriz =MatDis.to_numpy()


#Pedimos al usuario el número de productos, los productos y la demanda
NProd = int(input("Dígame el número de productos : "))
print("El numero de productos es:", NProd )

p=[0]
d=[]
for i in range(NProd):
    a = int(raw_input("Dígame el producto " +str(i+1)+" : "))
    b = int(raw_input("Dígame la demanda del producto " + str(i + 1) + " : "))
    p.append(a)
    d.append(b)

print("Los productos a recoger son:", p[1:])
print("con demanda:", d)

#Calculamos la localización de cada producto
l = np.zeros((len(p)),int)
for i in range(len(p)):
    l[i] = ProdALoc(p[i])


#Hallamos las distancias entre los productos seleccionados
subm = np.zeros((NProd+1, NProd+1),int)

for i in range(NProd+1):
    for j in range(NProd+1):
        subm[i][j] = matriz[l[i]][l[j]]

print("La matriz de distancias entre estos productos es: ")
print(subm)
```

```
#Creamos un documneto de texto e introducimos los datos necesarios para el␣
 ↪programa de CPLEX

file = open("DatosPedido.txt", "w")

# Numero de productos
file.write("//Los productos del pedido son " + str(p[1:]) + os.linesep)

file.write("//Numero de productos en el pedido" + os.linesep)
file.write("NProd = " + str(NProd))
file.write(";" + os.linesep)
file.write( os.linesep)

#Capacidad Vehículo
file.write("//Capacidad en peso del vehículo" + os.linesep)
file.write("C = 3" )
file.write(";" + os.linesep)
file.write( os.linesep)

#Numero de Vehículos
file.write("//Numero de vehiculos" + os.linesep)
file.write("NVehic = 3" )
file.write(";" + os.linesep)
file.write( os.linesep)

#Demanda por producto
file.write("//Demanda por producto" + os.linesep)
file.write("D =" + str(d) )
file.write(";" + os.linesep)
file.write( os.linesep)

#Distancias entre productos
file.write("//Distancia entre productos" + os.linesep)
file.write("d = " + str(subm) )
file.write(";" + os.linesep)



file.close()
```

## B.9. Pedidos según peso

En este trabajo se han dividido los pedidos según diferentes variables, la primera de ellas es el peso. El siguiente código genera pedidos aleatorios según el peso. En esté código se encuentran comentados los fragmentos que efectivamente generan pedidos de productos ligeros, pedidos de productos pesados y pedidos de 6 pesados y 6 al azar.

[ ]:
```
# coding=utf-8
import os
import random
from funcionDistancia import distancia
import pandas as pd
import numpy as np
from funcionProdALoc import ProdALoc
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import seaborn as sns

np.random.seed(0)

#Importamos la tabla con los pesos de cada producto ya fijados y la matriz de␣
 ↪distancias entre productos

tablaPes= pd.read_csv("TablaPesosFija.csv", header = 0, sep = ',' )

matriz = tablaPes.to_numpy()


MatDis= pd.read_csv("MatrizDistancias.csv", header = 0, sep = ',')

matrizD =MatDis.to_numpy()


#Seleccionamos la longitud del pedido
longPedido = 12


#Seleccionamos los ligeros
ligeros=[]
for i in range(3015):
    if (matriz[i][1]=="Ligero"):
        ligeros.append(matriz[i])


#Seleccionamos los pesados
pesados=[]
for i in range(3015):
    if (matriz[i][1]=="Pesado"):
        pesados.append(matriz[i])
```

```
'''
#Generamos un pedido aleatorio de productos ligeros
idProdPedi=[]

for i in range(12):
    idProdPedi.append(random.choice(ligeros)[0])


print(idProdPedi)

'''

'''
#Generamos un pedido aleatorio de productos pesados
idProdPedi=[]

for i in range(12):
    idProdPedi.append(random.choice(pesados)[0])

print(idProdPedi)
'''

'''
#Generamos un pedido con seis productos al azar y seis pesados
#Seleccionamos al azar seis productos del pedido
idProdPedi=[]
for i in range(6):
    idProdPedi.append(random.randint(1,3105))

print(idProdPedi)

#Generamos seis aleatorio de entre los productos pesados

for i in range(6):
    idProdPedi.append(random.choice(pesados)[0])

print(idProdPedi)

'''

#Completamos la informacion del pedido con los pesos de los productos
pedido=[]

for i in range(3105):
    for j in range(longPedido):
        if idProdPedi[j] == matriz[i][0]:
            pedido.append(matriz[i])

print(pedido)

#La demanda se genera aleatoriamente entre 1,2 ó 3
```

```
demanda=[]
for i in range(len(pedido)):
    demanda.append(random.randint(1,3))

for i in range(len(pedido)):
    pedido[i][2] = int(pedido[i][2]) #pesos a enteros


print(pedido)

#Calculamos la demanda (en peso) total según la demanda del producto
demandaP=[]
for i in range(len(pedido)):
    demandaP.append(demanda[i]*pedido[i][2])

#Exportamos a una excel la información de cada pedido

df2 = pd.DataFrame(pedido, columns = ['idProd', 'tipo', 'peso'])

df2['NumDemanda'] = demanda
df2['PesoDemanda'] = demandaP

print(df2)

df2.to_csv ('Pedido12_4P.csv', index = False, header=True)



#distancias entre productos especificos

prod=[0]
for i in range(len(pedido)):
    prod.append(pedido[i][0])
print(prod) #productos

l = np.zeros((len(prod)),int)

for i in range(len(prod)):
    l[i] = ProdALoc(prod[i])

print(l) #localizaciones


#Hallamos la submatriz de distancias para los productos específicos
subm = np.zeros((len(pedido)+1, len(pedido)+1),int)

for i in range(len(pedido)+1):
    for j in range(len(pedido) + 1):
        subm[i][j] = matrizD[l[i]][l[j]]

print(subm)
```

```
#Exportamos a un documento de texto los datos necesarios para CPLEX
file = open("Pedido12_4P.txt", "w")

# Numero de productos
file.write("//Los productos del pedido son " + str(prod[1:]) + os.linesep)

file.write("//Numero de productos en el pedido" + os.linesep)
file.write("NProd = " + str(len(pedido)))
file.write(";" + os.linesep)
file.write( os.linesep)

#Capacidad Vehículo
file.write("//Capacidad en peso del vehículo" + os.linesep)
file.write("C = 700" )
file.write(";" + os.linesep)
file.write( os.linesep)

#Numero de Vehículos
file.write("//Numero de vehiculos" + os.linesep)
file.write("NVehic = 5" )
file.write(";" + os.linesep)
file.write( os.linesep)

#Distancia maxima recorrida por vehículo
file.write("//Distancia maxima recorrida por vehículo" + os.linesep)
file.write("//distMax = 300" )
file.write(";" + os.linesep)
file.write( os.linesep)

#Demanda por producto
file.write("//Demanda por producto" + os.linesep)
file.write("dem =" + str(demandaP) )
file.write(";" + os.linesep)
file.write( os.linesep)

#Distancias entre productos
file.write("//Distancia entre productos" + os.linesep)
file.write("d = " + str(subm) )
file.write(";" + os.linesep)



file.close()
```

## B.10. Tabla resultado

La siguiente tabla resume los resultados obtenidos al resolver el modelo con todos los pedidos de 10 y 12 productos divididos según el peso. No está completa puesto que hay pedidos que requieren hasta 8 vehículos.

| | Productos | Nº Productos | Tº ejecución para 2 índices | Gap relativo | Gap absoluto | Tº ejecución para 3 índices | Gap relativo | Gap absoluto | Función Objetivo | Nº viajes | Peso Total | Productos en viaje 1 | Tiempo en viaje 1 | Peso en viaje 1 | Productos en viaje 2 | Tiempo en viaje 2 | Peso en viaje 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Pedido10_0** | Aleatorios | 10 | 19,03418190908 | 0 | 0 | 315,4467788809 | 0 | 0 | 740 | 3 | 1486 | 1 | 114 | 278 | 2 | 208 | 546 |
| **Pedido10_1** | Aleatorios | 10 | 8,359153076 | 0 | 0 | 51,273666016 | 0 | 0 | 782 | 3 | 1753 | 3 | 138 | 571 | 5 | 378 | 699 |
| **Pedido10_2** | Aleatorios | 10 | 0,634026855 | 0 | 0 | 7,178242188 | 0 | 0 | 592 | 2 | 964 | 5 | 166 | 281 | 5 | 426 | 683 |
| **Pedido10_3** | Aleatorios | 10 | 21,979487061 | 0 | 0 | 311,548746826 | 0 | 0 | 856 | 3 | 1752 | 3 | 282 | 666 | 2 | 168 | 466 |
| **Pedido10_4** | Aleatorios | 10 | 3,610608154 | 0 | 0 | 12,195903809 | 0 | 0 | 500 | 2 | 1062 | 7 | 346 | 673 | 3 | 154 | 346 |
| **Pedido10_0L** | Ligeros | 10 | 2,267717041 | 0 | 0 | 167,079370117 | 0 | 0 | 458 | 1 | 382 | 10 | 458 | 382 | 0 | 0 | 0 |
| **Pedido10_1L** | Ligeros | 10 | 2,602473877 | 0 | 0 | 53,635411865 | 0 | 0 | 486 | 1 | 501 | 10 | 486 | 501 | 0 | 0 | 0 |
| **Pedido10_2L** | Ligeros | 10 | 0,921254883 | 0 | 0 | 33,129213867 | 0 | 0 | 470 | 1 | 443 | 10 | 470 | 443 | 0 | 0 | 0 |
| **Pedido10_3L** | Ligeros | 10 | 0,722686035 | 0 | 0 | 10,147452148 | 0 | 0 | 444 | 1 | 417 | 10 | 444 | 417 | 0 | 0 | 0 |
| **Pedido10_4L** | Ligeros | 10 | 1,741640869 | 0 | 0 | 10,340840088 | 0 | 0 | 480 | 1 | 465 | 10 | 480 | 465 | 0 | 0 | 0 |
| **Pedido10_0P** | Pesados | 10 | 0,219504883 | 0 | 0 | 0,829861084 | 0 | 0 | 1484 | 6 | 3283 | 2 | 160 | 607 | 2 | 314 | 606 |
| **Pedido10_1P** | Pesados | 10 | 0,299396973 | 0 | 0 | 7,745958008 | 0 | 0 | 1096 | 5 | 2560 | 3 | 398 | 589 | 2 | 170 | 592 |
| **Pedido10_2P** | Pesados | 10 | 0,225682129 | 0 | 0 | 0,353278076 | 0 | 0 | 1478 | 7 | 3537 | 1 | 248 | 411 | 3 | 346 | 690 |
| **Pedido10_3P** | Pesados | 10 | 1,874333008 | 0 | 0 | 26,239666016 | 0 | 0 | 1262 | 5 | 2599 | 1 | 90 | 286 | 2 | 382 | 640 |
| **Pedido10_4P** | Pesados | 10 | 0,895474854 | 0 | 0 | 8,2223215918 | 0 | 0 | 1216 | 5 | 2824 | 2 | 224 | 597 | 2 | 314 | 578 |
| **Pedido12_0** | Aleatorios | 12 | 92,95875 | 0 | 0 | 900 | 0.491592545 | 416.870478013 | 848 | 3 | 1709 | 4 | 334 | 670 | 5 | 402 | 675 |
| **Pedido12_1** | Aleatorios | 12 | 900 | 0.15502859 | 123.722816617 | 900 | 0.45334534 | 361.769581056 | 798 | 3 | 1537 | 4 | 316 | 650 | 6 | 384 | 645 |
| **Pedido12_2** | Aleatorios | 12 | 42,691632813 | 0 | 0 | 900 | 0.257789573 | 214.996504045 | 834 | 3 | 1579 | 2 | 78 | 187 | 5 | 350 | 693 |
| **Pedido12_3** | Aleatorios | 12 | 77,177787109 | 0 | 0 | 900 | 0.468288103 | 391.488853789 | 836 | 4 | 2307 | 1 | 80 | 510 | 4 | 164 | 443 |
| **Pedido12_4** | Aleatorios | 12 | 362,06762793 | 0 | 0 | 900 | 0.502901354 | 520 | 1032 | 3 | 1977 | 4 | 392 | 687 | 5 | 288 | 620 |
| **Pedido12_0L** | Ligeros | 12 | 44,555035156 | 0 | 0 | 900 | 0.200807783 | 79.51988218 | 396 | 1 | 470 | 12 | 396 | 470 | 0 | 0 | 0 |
| **Pedido12_1L** | Ligeros | 12 | 900 | 0.164204526 | 131.035211705 | 900 | 0.452819358 | 361.349847477 | 798 | 3 | 1537 | 4 | 316 | 650 | 6 | 384 | 645 |
| **Pedido12_2L** | Ligeros | 12 | 3,803305176 | 0 | 0 | 55,609447998 | 0 | 0 | 462 | 1 | 543 | 12 | 462 | 543 | 0 | 0 | 0 |
| **Pedido12_3L** | Ligeros | 12 | 0,768290039 | 0 | 0 | 2,178088135 | 0 | 0 | 480 | 1 | 342 | 12 | 480 | 342 | 0 | 0 | 0 |
| **Pedido12_4L** | Ligeros | 12 | 28,600333984 | 0 | 0 | 117,691451904 | 0 | 0 | 416 | 1 | 415 | 1 | 416 | 415 | 0 | 0 | 0 |
| **Pedido12_0P** | Pesados | 12 | 1,072714111 | 0 | 0 | 32,697588867 | 0 | 0 | 1392 | 6 | 3670 | 2 | 92 | 654 | 2 | 204 | 609 |
| **Pedido12_1P** | Pesados | 12 | 0,159433838 | 0 | 0 | 0,910096924 | 0 | 0 | 1538 | 8 | 4191 | 2 | 256 | 608 | 1 | 168 | 450 |
| **Pedido12_2P** | Pesados | 12 | 2,057993164 | 0 | 0 | 31,85124707 | 0 | 0 | 1632 | 6 | 3633 | 3 | 352 | 691 | 1 | 58 | 447 |
| **Pedido12_3P** | Pesados | 12 | 2,254072021 | 0 | 0 | 590,60975708 | 0 | 0 | 1318 | 6 | 3521 | 2 | 206 | 672 | 3 | 414 | 696 |
| **Pedido12_4P** | Pesados | 12 | 0,312919922 | 0 | 0 | 6,766578125 | 0 | 0 | 1958 | 8 | 4003 | 1 | 216 | 423 | 2 | 244 | 455 |

## B.11. Pedidos según localización

El siguiente código genera pedidos aleatorios según la localización. En este código se encuentran comentados los fragmentos que efectivamente generan pedidos de cada zona, de la zona 2 y de las zonas 3 .

[ ]:
```
# coding=utf-8
import os
import random
import pandas as pd
import numpy as np
from funcionProdALoc import ProdALoc
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import seaborn as sns

np.random.seed(0)

#Importamos la tabla con los pesos de cada producto ya fijados y la matriz de␣
 ↪distancias entre productos
tablaPes= pd.read_csv("TablaPesosFija.csv", header = 0, sep = ',' )

#print(tablaPes)

matriz = tablaPes.to_numpy()

MatDis= pd.read_csv("MatrizDistancias.csv", header = 0, sep = ',')

matrizD =MatDis.to_numpy()

#print(matriz)

longPedido = 12

#Hago cuatro vectores con los productos de cada zona
a = np.arange(15)+1
for i in range(12):
    b= np.arange(15)+(60*(i+1)+1)
    a = np.concatenate((a, b))


for i in range(31):
    b= np.arange(15)+(75*i+781)
    a = np.concatenate((a, b))

Zona1=a


a = np.arange(25)+16
for i in range(12):
```

```
    b= np.arange(25)+(60*(i+1)+16)
    a = np.concatenate((a, b))


for i in range(31):
    b= np.arange(25)+(75*i+796)
    a = np.concatenate((a, b))

Zona2=a


a = np.arange(20)+41
for i in range(12):
    b= np.arange(20)+(60*(i+1)+41)
    a = np.concatenate((a, b))

Zona3A= a


a = np.arange(35)+821
for i in range(30):
    b= np.arange(35)+(75*(i+1)+821)
    a = np.concatenate((a, b))

Zona3B = a


idProdPedi=[]
zona=[]
'''
#elijo tres de la zona 1, tres de la zona 2, dos de la zona 3A y dos de la␣
 ↪zona 3B
for i in range(4):
    idProdPedi.append(random.choice(Zona1))
    zona.append("Zona1")
    idProdPedi.append(random.choice(Zona2))
    zona.append("Zona2")

for i in range(2):
    idProdPedi.append(random.choice(Zona3A))
    zona.append("Zona3A")
    idProdPedi.append(random.choice(Zona3B))
    zona.append("Zona3B")

print(idProdPedi)
'''

'''
#Elijo los diez productos de la zona 2
for i in range(longPedido):
    idProdPedi.append(random.choice(Zona2))
```

```
    zona.append("Zona2")

'''
'''
#Elijo los doce productos de la zona 3; la mitad de la Zona3A y la otra mitad␣
 ↪de Zona3B
for i in range(6):
    idProdPedi.append(random.choice(Zona3A))
    zona.append("Zona3A")
    idProdPedi.append(random.choice(Zona3B))
    zona.append("Zona3B")
print(idProdPedi)
'''

#Completamos la informacion del pedido con los pesos de los productos
pedido=[]

for j in range(longPedido):
    for i in range(3105):
        if idProdPedi[j] == matriz[i][0]:
            pedido.append(matriz[i])

print(pedido)
#La demanda se genera aleatoriamente entre 1,2 ó 3
demanda=[]
for i in range(len(pedido)):
    demanda.append(random.randint(1,3))

for i in range(len(pedido)):
    pedido[i][2] = int(pedido[i][2])#pesos a enteros


print(pedido)

#Calculamos la demanda (en peso) total según la demanda del producto
demandaP=[]
for i in range(len(pedido)):
    demandaP.append(demanda[i]*pedido[i][2])


#Exportamos a una excel la información de cada pedido
df2 = pd.DataFrame(pedido, columns = ['idProd', 'tipo', 'peso'])

df2['NumDemanda'] = demanda
df2['PesoDemanda'] = demandaP
df2['Zona'] = zona

print(df2)

df2.to_csv ('Pedido12_4_Z3.csv', index = False, header=True)
```

```
#distancias entre productos específicos

prod=[0]
for i in range(len(pedido)):
    prod.append(pedido[i][0])
print(prod) #productos

l = np.zeros((len(prod)),int)

for i in range(len(prod)):
    l[i] = ProdALoc(prod[i])

print(l) #localizaciones

#Hallamos la submatriz de distancias para los productos específicos
subm = np.zeros((len(pedido)+1, len(pedido)+1),int)

for i in range(len(pedido)+1):
    for j in range(len(pedido) + 1):
        subm[i][j] = matrizD[l[i]][l[j]]

print(subm)

#Exportamos a un documento de texto los datos necesarios para CPLEX
file = open("Pedido12_4_Z3.txt", "w")

# Numero de productos
file.write("//Los productos del pedido son " + str(prod[1:]) + os.linesep)

file.write("//Numero de productos en el pedido" + os.linesep)
file.write("NProd = " + str(len(pedido)))
file.write(";" + os.linesep)
file.write( os.linesep)

#Capacidad Vehículo
file.write("//Capacidad en peso del vehículo" + os.linesep)
file.write("C = 700" )
file.write(";" + os.linesep)
file.write( os.linesep)

#Numero de Vehículos
file.write("//Numero de vehiculos" + os.linesep)
file.write("NVehic = 5" )
file.write(";" + os.linesep)
file.write( os.linesep)

#Distancia maxima recorrida por vehículo
file.write("//Distancia maxima recorrida por vehículo" + os.linesep)
file.write("//distMax = 300" )
file.write(";" + os.linesep)
```

```
file.write( os.linesep)

#Demanda por producto
file.write("//Demanda por producto" + os.linesep)
file.write("dem =" + str(demandaP) )
file.write(";" + os.linesep)
file.write( os.linesep)

#Distancias entre productos
file.write("//Distancia entre productos" + os.linesep)
file.write("d = " + str(subm) )
file.write(";" + os.linesep)



file.close()
```