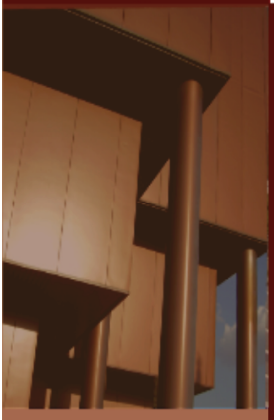
Trabajo de Fin de Máster
Ingeniería Industrial

Modelado y resolución del problema de rutas de preparación de pedidos conjuntos.

Autor: Belén Pascual Montero
Tutor: Pablo Aparicio Ruiz



Dpto. de Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Máster
Ingeniería Industrial

Modelado y resolución del problema de rutas de preparación de pedidos conjuntos.

Autor:
Belén Pascual Montero

Tutor:
Pablo Aparicio Ruiz

Dpto. de Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Resumen

En el presente trabajo se abarca la resolución del problema del agrupamiento de pedidos en lotes y la definición de la ruta para la preparación de los mismos. Este problema es comúnmente conocido como el JOBPRP por sus siglas en ingles (Joint Order Batching and Picker Routing Problem). Este problema supone una alta contribución a los costes logísticos de un almacén por lo que las empresas se preocupan cada vez más por optimizar dicho proceso. Para ello, entre los posibles métodos existentes de resolución de este tipo de problemas se propone la implementación de un modelo matemático para la obtener una solución exacta del mismo. El caso de estudio se centra en almacenes de compra online, haciendo uso de una base de datos disponible en internet que recoge pedidos de usuarios reales, además de un simulador que genera relaciones entre los productos y las posiciones de almacenaje generadas según el layout que se establece como dato de partida.

Para la implementación de este modelo matemático se programa la formulación del mismo en el lenguaje de programación Python, haciendo uso del intérprete Spyder. Se elige este lenguaje de programación debido a la sencillez del mismo y otras características que lo convierten en un lenguaje de fácil aprendizaje. El código del modelo matemático se desarrolla con el fin de ser posteriormente ejecutado con el solver de optimización Gurobi.

Del análisis llevado a cabo para el desarrollo del presente trabajo y la implementación para varios ejemplos, se determina que el problema JOBPRP, resulta difícil de resolver de manera exacta debido a su naturaleza, ya que se trata de una combinación de dos problemas complejos. El tiempo computacional para resolver este problema a través de un método exacto de resolución aumenta exponencialmente con el tamaño del problema por lo que se determina que sería más conveniente abordar este problema mediante métodos de resolución aproximados.

Abstract

The aim of this project is to investigate and solve the Joint Order Batching and Picker Routing Problem (JOBPRP). This problem represents a high contribution to the logistics costs of a warehouse and companies are increasingly concerned with optimizing this process. In order to achieve a solution for it, among the possible existing methods of solving this type of problem, in this project is proposed the implementation of a mathematical model to obtain an exact solution of the problem. The case of study focuses on online supermarket shopping, using a database available on the Internet that collects orders from real users, in addition to a simulator that generates correlations between products and storage positions generated according to the desired layout, given this as an input to the generator.

For the implementation of this mathematical model, the formulation is programmed in the Python programming language, using the Spyder interpreter. This programming language is chosen due to its simplicity and other characteristics that make it an easy to learn language. The code of the mathematical model is developed in order to be later executed with the optimization solver Gurobi.

From the analysis carried out during the development of the present project and the implementation for several examples, it is determined that the JOBPRP problem is difficult to be exactly solved due to its nature, since it is a combination of two complex problems. The computational time to solve this problem through an exact method of resolution increases exponentially with the size of the problem so it is determined that it would be more convenient to address this problem through approximate methods of resolution.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

A medida que avanzamos en la era digital, cada vez es más importante para las empresas de todos los tamaños, desde las pequeñas tiendas familiares hasta las grandes empresas, agilizar sus prácticas de preparación de pedidos en el almacén. Este sentimiento es especialmente cierto para las organizaciones que dependen del personal del almacén o de equipos automatizados para cumplir con los pedidos. Al fin y al cabo, los consumidores de hoy en día se han acostumbrado a recibir sus productos en tiempos de entrega cada vez más reducidos.

Las labores relacionadas con el almacenamiento de la mercancía es un componente clave de la gestión logística. La forma en que las empresas gestionan el picking y el almacenamiento de sus inventarios pueden ser determinantes de la eficiencia de la empresa. El almacenamiento y la logística deben estar alineados dentro de una empresa para tener el efecto más eficiente en la producción general, así como en las actividades de logística de salida como la preparación de pedidos. Estas funciones deben mantener altos niveles de sincronización a fin de obtener la mayor eficiencia y efectividad dentro de la empresa, así como en la cadena de suministro en su conjunto.

La preparación de pedidos en almacén es un concepto simple, pero en la práctica, los procesos de picking pueden ser complejos. En pocas palabras, la preparación de pedidos en el almacén se refiere a la mano de obra y a la maquinaria necesaria para extraer un artículo del inventario y satisfacer el pedido de un cliente. Es un proceso que puede sonar como el aspecto más sencillo de un negocio, pero sin embargo equivale a un alto porcentaje de los costes operativos dentro de cualquier centro de distribución.

1.1 Objetivos

En este proyecto se pretende resolver el problema conjunto de preparación de pedidos por lotes y la definición de la ruta del picker. El término picker hace referencia al operario de almacén encargado de la recogida de mercancía. Se busca un método de resolución exacto para optimizar la distancia recorrida por el picker en el almacén, es decir minimizar las rutas, ahorrando tiempo y por tanto, reduciendo costes.

Esta aplicación además de la minimización de distancias conllevará a su vez la disminución de costes de picking, reduciendo la cantidad de pickers necesarios para la recogida de los pedidos de

cliente. Por tanto, se mejorará la calidad del servicio al cliente reduciendo los tiempos logísticos y se mejorará la competitividad de la empresa en el mercado.

1.2 Estructura del trabajo

En este proyecto se resuelve el problema de preparación de pedidos por lotes y definición de la ruta del picker mediante un método de resolución exacto. Para ello, se plantea un modelo matemático a implementar en diferentes almacenes, variando la demanda de cliente y las limitaciones de capacidad del picker.

En primer lugar, se introducen los conceptos de diseño y gestión de almacenes, donde se hace especial referencia a la operación de picking. A continuación se desarrolla mejor este concepto, explicando las actividades que se llevan a cabo, las distintas técnicas, la contribución de este proceso en la gestión de almacenes, etc. En este punto se explican los dos problemas fundamentales que se pretenden resolver de manera conjunta en este proyecto: la agrupación de pedidos en lotes y la ruta de recogida de pedidos en el almacén.

Posteriormente se define el problema a resolver, exponiendo algunos ejemplos de los casos a los que sería aplicable. Además, se lleva a cabo una revisión de la literatura respecto al problema en sí, variaciones del mismo y estudios previos llevados a cabo dónde se valoran distintos métodos de resolución.

Una vez definido el problema, se lleva a cabo un pequeño estudio sobre los posibles métodos de resolución del mismo, definiendo finalmente el modelo matemático que se va a implementar para la resolución del problema expuesto. Para ello, en primer lugar se traducen las variables que caracterizan el problema a variables matemáticas, es decir, se expone la notación del mismo y seguidamente se formula el modelo matemático en función de estas variables.

Finalmente, se concluirá el proyecto aplicando el modelo desarrollado a distintos ejemplos y analizando los resultados obtenidos en la implementación así como exponiendo las conclusiones alcanzadas a lo largo del desarrollo del proyecto.

2 ANTECEDENTES

Debido a la creciente competencia entre las empresas por un mercado cada vez más exigente, se ha convertido en necesidad la búsqueda de la excelencia. Para ello, las empresas buscan reducir costes y mejorar a su vez la productividad dentro de los almacenes y/o centros de distribución. Con el fin de alcanzar estos objetivos, las operaciones de picking han sido objeto de estudio en numerosas ocasiones. Las operaciones que engloba el término 'picking' se refieren a las relacionadas con la recogida de productos en el almacén para dar respuesta a una solicitud específica del cliente, es decir, cumplir con los pedidos del cliente.

El proceso del picking es uno de los procesos dentro de un almacén que requiere de mayor intensidad de mano de obra cuándo se trabaja con sistemas manuales y supone una gran inversión de capital en los almacenes que disponen de un sistema automatizado. Es por esto, que a juicio profesional se considera la actividad del picking como una de las labores de alta prioridad para mejorar la productividad de una empresa.

En la actualidad, la industria tiende a la búsqueda de lotes de tamaño reducido, así como una reducción en el tiempo de entrega. Desde el punto de vista de la logística de la distribución y con la finalidad de satisfacer al cliente, las empresas pretenden ser capaces de aceptar las distintas órdenes de pedido y servirlas en unos plazos de tiempo muy reducidos. Por este motivo queda altamente reducido el tiempo disponible para el picking y expedición. Por lo general, en almacenes de tamaño considerable, el volumen diario de recogidas es grande y el espacio temporal al que deben ajustarse es corto.

La labor del picking ha ido adquiriendo importancia con el tiempo debido a las nuevas tendencias existentes tanto en la fabricación cómo en la distribución. A continuación se presenta una breve introducción a los almacenes, su diseño y gestión, y a la actividad de picking. Además se exponen algunos de los problemas que surgen en los almacenes a raíz del desarrollo de esta actividad.

2.1 Diseño y gestión de almacenes

El almacén es un lugar destinado a recibir, guardar, manipular, acondicionar y expedir productos, con el objetivo de estabilizar la oferta y la demanda, reducir costes y garantizar el mejor nivel de servicio al cliente.

Dentro de los sistemas de picking, el objetivo es maximizar el nivel de servicio, minimizando las distancias recorridas y cumpliendo las restricciones a las que están sujetos respecto a la limitación de recurso: capital, mano de obra y maquinaria. De aquí surge la necesidad de reducir al mínimo los tiempos dedicados a las operaciones de picking. Dentro del picking se puede invertir tiempo en diferentes actividades, siendo el recorrido a realizar la actividad que mayor tiempo consume. A este parámetro se le conoce como el 'tiempo de viaje'. Este tiempo es el principal candidato para la mejora de la gestión de un almacén (Bartholdi and Hackman, 2014).
Para optimizar este aspecto se puede optar por reducir la distancia media de viaje, reduciendo de este modo la distancia total a recorrer. Las mejoras mencionadas contribuirían de una manera u otra a minimizar el coste total de un almacén desde los gastos operacionales que esto conlleva. A continuación se listan una serie de objetivos adicionales que a menudo se consideran en el diseño y optimización de almacenes (Bartholdi and Hackman, 2014):

- Minimizar el tiempo de operación.
- Minimizar las pérdidas de tiempo en la realización de una orden.
- Maximizar la utilización del espacio.
- Maximizar el aprovechamiento de los equipos.
- Maximizar el uso de la mano de obra.
- Maximizar la accesibilidad de los artículos.

Para alcanzar dichos objetivos, primero se ha de realizar una distribución en plano, lo que se conoce más habitualmente por su término en inglés layout, es decir, el diseño de un almacén plasmado en un plano. La definición del layout del almacén es el primer paso en el trabajo de diseño de la instalación. La organización del espacio es teóricamente una cuestión sencilla, sin embargo resulta complicada de resolver en la práctica.

Entre todas las operaciones que se realizan en un almacén, la más costosa es la dedicada a la preparación de pedidos. Se destacan las siguientes actividades como las más comunes en la preparación de pedidos (Ballou, 2004):

- Recopilación de pedidos

- Elaboración de los documentos de preparación
- Picking
- Traslado a zona de expedición
- Verificación y acondicionamiento de los pedidos

A la hora de abordar el picking desde el punto de vista de la distribución en planta del almacén, es preciso tener en cuenta dos aspectos de gran relevancia: El diseño de las instalaciones para el proceso del picking y el diseño del sistema a llevar a cabo para el picking. El primero de ellos, comúnmente conocido como el diseño de la instalación, consiste en decidir la ubicación física de los distintos departamentos (zona de recepción, carga y descarga, zona de almacenamiento, zona habilitada para el picking, clasificación, zona de expedición). Este problema debe considerar las relaciones de actividades y flujos existentes entre los distintos departamentos para así garantizar la optimización del almacén, minimizando al máximo los costes de manipulación. Estos costes pueden representarse en muchas ocasiones según una función lineal de la distancia recorrida. El segundo problema es comúnmente conocido como el diseño interno, éste abarca la determinación del número y las características de los huecos de almacenamiento y pasillos en las zonas de picking.

Estos dos problemas tienen un objetivo común, mejorar de algún modo el diseño del almacén ajustándose a una serie de requisitos y/o limitaciones. Una vez más, este objetivo se pretende alcanzar mediante la minimización de las distancias a recorrer.

Por lo general, el encargado del diseño de un almacén se encuentra con un espacio determinado en el cuál ciertos factores suponen una cierta limitación de la superficie disponible. Es por esto, que la distribución ha de ser estudiada cuidadosamente con anterioridad. Cuando se trata de decidir la disposición que debe tener un almacén, tanto interna como externa, existen algunos matices que implican la necesidad de abordar el problema de asignación de espacios de manera diferente, estos casos son los siguientes: la instalación de nuevos almacenes, la ampliación de los almacenes ya existentes y la reorganización de los almacenes que actualmente están en servicio.
En el diseño de la distribución deben estar perfectamente definidas las siguientes zonas:

1. Zona de recepción.
2. Zona de almacenaje.
3. Zona de preparación de pedidos.
4. Zona expedición.

En la Figura 1 se presenta un ejemplo de un posible layout de almacén:

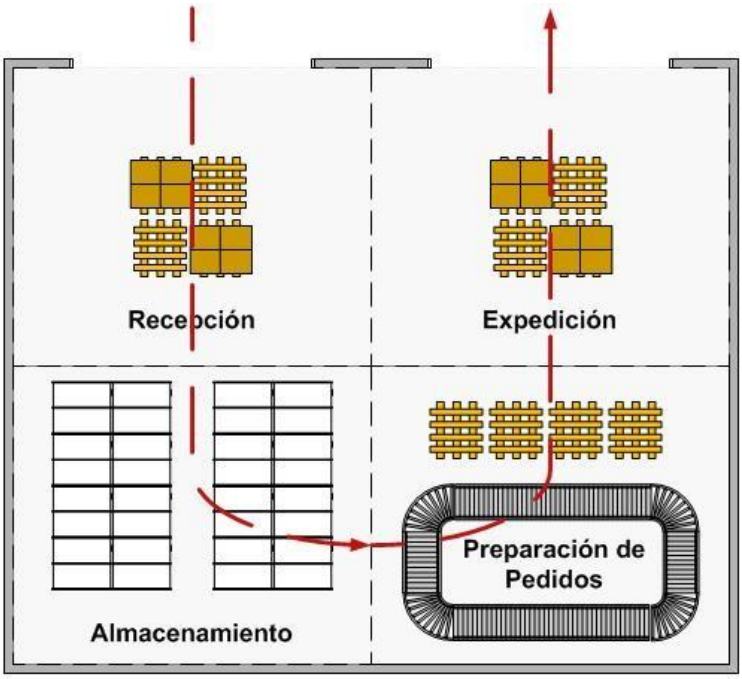


Figura 1: Ejemplo de Layout de un almacén (flujo en U)

Fuente: www.ingenieriaindustrialonline.com

Además de la importancia del diseño, una buena gestión del almacén es fundamental. Esto permite ofrecer el mejor servicio, tener una alta ocupación, disponer del stock necesario, optimizar los tiempos de operación interna, además de otras muchas ventajas. Es por tanto un elemento primordial.

A la hora de diseñar y planificar instalaciones, conocer el funcionamiento de su gestión permite valorar el grado de efectividad y definir qué procesos pueden optimizarse. Por ejemplo, un cambio en la distribución de las estanterías puede simplificar los procesos de picking.

En definitiva, los almacenes han ido adquiriendo una mayor importancia a lo largo del tiempo, llegando a considerarse un elemento estratégico para alcanzar el éxito. Estas instalaciones dedicadas al almacenaje se han convertido a su vez en centros enfocados al servicio y soporte de los negocios, siendo por tanto fundamental un correcto diseño y una correcta gestión del mismo. Esto quiere decir que el almacén debe cumplir su función y poder incluso adaptarse a futuras necesidades. Los factores clave a estudiar a la hora de diseñar un almacén y su layout son los siguientes:

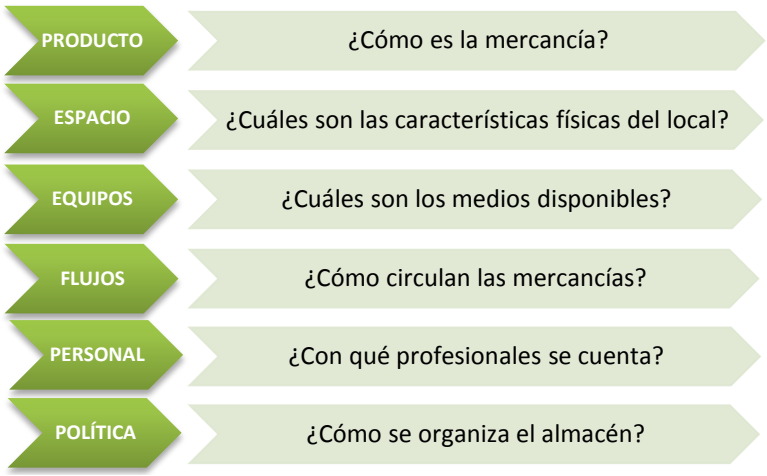


Figura 2: Factores clave para el diseño de un almacén

Fuente: Elaboración propia

2.2 El picking

La gestión de almacenes involucra un alto número de operaciones de manejo de materiales. De estas operaciones, el picking es la más laboriosa y la que generalmente necesita de más maquinaria. Se estima que el coste en el que se incurre en el picking ronda el 60% de los costes totales de un almacén, además estas cifras tienden a aumentar debido al fuerte crecimiento de la popularidad de las compras online en los últimos años. Reducir el impacto del picking a lo mínimo es un objetivo cuyo cumplimiento puede significar la diferencia entre una empresa competitiva y otra que no lo es, es decir, entre permanecer en el mercado o desaparecer.

El picking es la actividad que desarrolla dentro de un almacén un equipo de personal, que consiste en el proceso de retirada de productos de su lugar de almacenamiento con el fin de responder a una demanda de un cliente. Cualquier otra operación de recogida de productos que no obedezca al concepto anterior, tales como: movimientos internos por reorganizaciones de almacén, destrucción de productos obsoletos, traspaso de mercancías de un almacén a otro de la misma empresa (reposiciones), etc., por definición queda fuera de este concepto.

Puede llevarse a cabo en casi cualquier tipo de almacén y se produce siempre que se necesite juntar paquetes, piezas, productos o materiales para, una vez reunidos, proceder a su traslado. Incluye todo el conjunto de operaciones destinadas a extraer y acondicionar los productos

demandados. El objetivo del picking es realizar dichas actividades coordinando estanterías, carretillas, métodos organizativos, informática así como las nuevas tecnologías para mejorar la productividad del almacén. Además, se busca eliminar los errores para alcanzar la calidad requerida por el cliente.

Como se ha mencionado anteriormente, se trata de la actividad más costosa en un almacén. Supone operaciones tales como: desplazamiento de personal para buscar los productos y volver a la zona destinada a la preparación de pedidos, extracción de la mercancía solicitada de las estanterías, acondicionamiento del pedido, control, actualización inmediata del stock, etc. El nivel de automatización del picking suele ser, salvo excepciones, bajo. No obstante esta característica está cambiando progresivamente.

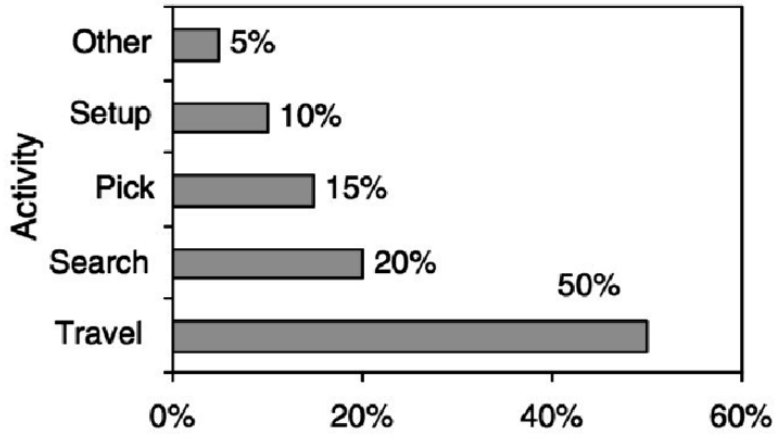


Figura 3: Distribución típica del tiempo en las actividades de picking.

Fuente: Facilities Planning (2010)

En consecuencia, la distancia recorrida es considerada como uno de los principales objetivos de los estudios de optimización de los procesos logísticos. Las políticas de definición de la ruta a seguir tienen como objetivo minimizar esta distancia en diferentes contextos.

Existen diferentes procedimientos de picking, que pueden venir determinadas por la tipología del almacén, de los niveles de servicios prestados por la empresa, etc. Se pueden agrupar de la siguiente forma:

- Según el momento en que se realiza:
  - Picking discrecional.
  - Picking programado.
- Según la dinámica organizativa empleada:

Picking “in situ”.

Estaciones de picking.

- Según los recursos informáticos empleados:

  Picking manual

  Picking asistido por ordenador.

  Sistemas basados en radio control.

Un agente limitante que afecta a la productividad del picking manual es el aumento de pedidos de volumen reducido, lo que supone el reajuste de los niveles de inventario: implica una mayor variedad y exige un trabajo más personalizado para la empresa.

Los recorridos a realizar en la operación de picking son los siguientes:

- Desde la zona de preparación de pedidos hasta el punto de ubicación del producto en cuestión.
- Desde un punto de ubicación hasta el siguiente y así sucesivamente.
- Vuelta al origen desde la última posición visitada.

Las actividades de recogida manual de material se pueden diferenciar en dos tipos de sistemas: 'Parts- to –picker’, que son aquellos sistemas en los que los productos son llevados de manera automatizada hasta el picker cuya localización es fija. Y por otro lado, los sistemas ‘Picker to parts’, en los que los operarios son quienes se desplazan, a pie o mediante vehículos, por el almacén recogiendo los productos demandados. Este segundo tipo es el más común y en el que se desarrollará el caso de estudio de este proyecto, por lo que en los siguientes apartados se explica de forma un poco más detallada. Dentro de este sistema de recogida de pedidos se pueden distinguir tres problemas: la asignación de los productos a las diferentes localizaciones, agrupación de los pedidos de los clientes y por último, definición de la ruta para la recogida de pedidos.

Por otro lado los métodos de recogida de productos mecánicos se pueden también dividir en dos tipos. En primer lugar, el picking automático, que consiste en estanterías inteligentes cuyos slots de almacenamiento son móviles y por tanto la mercancía puede alcanzar la localización en la que se encuentra el picker. El otro método de picking automático es el picking con robots, en el que el trabajo que normalmente realiza el picker humano lo realiza un robot que tiene autonomía para moverse por el almacén y recoger los productos necesarios.

A su vez, dentro del picking-to-parts se puede distinguir entre el picking de alto nivel y el de bajo nivel. La diferencia entre estos radica en la altura máxima a la cual se almacene producto y a las

dimensiones y características de estos productos. El picking de alto nivel es aquel que consiste en preparar pedidos de productos de gran tamaño o en altura suficiente para necesitar una grúa elevadora o carretilla. El picking de bajo nivel sin embargo, no requiere de ninguna máquinaria para alcanzar o recoger los productos, el picker puede acceder a ellos desde el pasillo conforme recorre el almacén.

Por otro lado, existe otra clasificación dentro de la preparación de pedidos picker-to-parts según la tecnología que se use. Un escenario común, que podemos llamar tradicional dentro de la preparación de pedidos, es el uso de listas de pedidos. La lista se imprime en una hoja de papel que contiene mucha información, como la ubicación de los artículos dentro del almacén y las cantidades a preparar. El operador toma la lista de pedidos del centro de información del almacén y comienza la búsqueda y recogida de los artículos listados. Las principales tecnologías que se pueden encontrar en el picker-to-parts son las siguientes. (*Order Picking Technologies | Voice | Pick Light | RF | MWPVL*, no date).

- Picking con lectores de radiofrecuencia: No se necesita papel, ya que las órdenes de trabajo se gestionan mediante el terminal informático que los operarios llevan consigo. Es la tecnología de preparación de pedidos más utilizada. Los dispositivos portátiles de radiofrecuencia se han utilizado en los almacenes desde principios de los años 80, cuando se integraron transmisores de radio y escáners de códigos de barras en unidades portátiles de mano para permitir la recopilación de datos en tiempo real cuando los operarios realizan tareas en el almacén. Hasta la fecha, los dispositivos de radiofrecuencia siguen siendo la tecnología más flexible porque pueden utilizarse en prácticamente todas las operaciones que se llevan a caboen los almacenes (es decir, no se limitan a la preparación de pedidos) y porque son capaces de capturar datos específicos del producto o del cliente (por ejemplo, el número de lote, el número de serie, el peso, etc.). Para aplicaciones de almacenamiento industrial, los principales proveedores de esta tecnlogía son: Motorola, Intermec, Psion Teklogix, LXE y Honeywell.

- Picking por voz: Los operadores usan auriculares con micrófonos y se comunican oralmente con un sistema de software en tiempo real para recibir y confirmar las tareas de picking. El mensaje de voz emitido por un ordenador instruye al operario para que vaya a un lugar donde se encuentra el producto y el operario confirma la ubicación dictando los dígitos que aparecen en una etiqueta colocada en cada posición de almacenamiento. A continuación, el sistema da instrucciones al operario para que tome la cantidad requerida junto con cualquier otra instrucción que sea necesaria para completar la tarea. La ventaja fundamental del picking dirigido por voz es que permite

realizar un picking de "manos libres", lo que significa que el único objetivo del operario es encontrar la ubicación correcta y seleccionar la cantidad correcta. La ventaja de ser manos libres es que el operador tiene ambas manos libres para levantar productos pesados sin la pérdida de tiempo asociada a tener que manipular un dispositivo de radiofrecuencia portátil. La tecnología de voz realmente despegó en la industria de la distribución de comestibles, donde los entornos de almacenamiento en frío requieren que los operadores usen guantes para mantener las manos calientes, lo que dificulta y hace improductivo el uso de dispositivos portátiles de radiofrecuencia.

- Picking por luz: Consiste en pantallas luminosas que se instalan típicamente en cada punto de almacenamiento en las estanterías y racks de almacenamiento. Las tareas de preparación de pedidos se llevan a un subsistema que enciende los dispositivos luminosos uno por uno a medida que los operarios recogen cada línea de pedido. La pantalla luminosa identifica el punto de picking en el que se debe realizar la tarea y además puede indicar la cantidad de picking y la cesta en la que se debe colocar el producto. El operario normalmente pulsa un botón en el dispositivo de luz para confirmar que la tarea de recogida se ha llevado a cabo. Los dispositivos pick-to-light pueden incorporar luces de varios colores, lo que permite preparar varios pedidos a la vez, asignando cada color a cada uno. El pick to light también se puede utilizar para el picking inverso, es decir, se reciben las mercancías y luego se colocan en ubicaciones que representan los pedidos de los clientes. Este último enfoque se utiliza a menudo en los centros de distribución al por menor para líneas de productos como la confección. La ventaja fundamental del pick to light es que es una tecnología visual que soporta altas velocidades de preparación de pedidos y niveles de precisión de clase mundial.

- Picking por visión: es uno de los sistemas más recientes, en el cual el trabajador recibe las indicaciones por medio de gafas especiales. En ellas, se le indica al operario qué productos debe recoger y en qué cantidades. Se ha aplicado tanto a los dispositivos portátiles de radiofrecuencia como a las tecnologías pick to light. En el caso de los dispositivos portátiles de radiofrecuencia, en lugar de mostrar mensajes basados en texto a los operarios para instruirlos, las gafas especiales muestran imágenes gráficas combinadas con instrucciones de texto. Las imágenes se utilizan para mejorar las instrucciones y la precisión (por ejemplo, la imagen del producto, la unidad de medida, la ubicación del contenedor, las instrucciones de embalaje, etc.). En el caso de pick to light, el dispositivo de visualización situado en la cabecera del estante muestra una imagen del SKU para ayudar al operador a seleccionar el artículo y la unidad de medida correctos. Entre los principales proveedores de la aplicación de tecnologías visuales se

encuentran TECSYS (dispositivos portátiles de radiofrecuencia) y ASAP Automation (Pick to Light).

Clasificación según productividad:

El siguiente gráfico muestra cómo cada tecnología de preparación de pedidos se relaciona con las tasas de productividad de la preparación de pedidos. El siguiente gráfico fue construido originalmente por Dematic y MWPVL lo adaptó según su experiencia en el sector. Este gráfico pretende establecer un marco para comparar las tasas de productividad que uno puede esperar obtener de cada tipo de tecnología de preparación de pedidos.

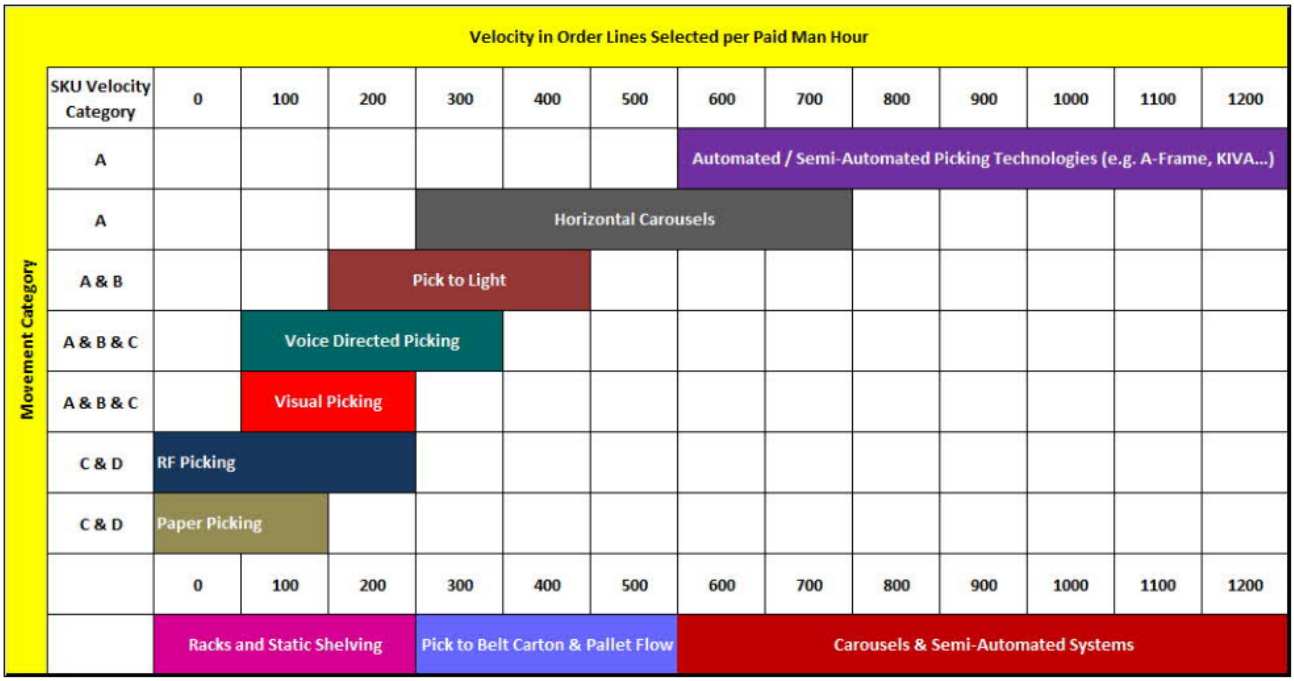


Figura 4: Clasificación de las tecnologías de preparación de pedido según su productividad.
Fuente: MWPVL international.

Estas tecnologías de preparación de pedidos se están transformando y ahora las empresas las utilizan juntas para poder obtener los mejores aspectos de cada tecnología. Por ejemplo, en la siguiente foto, el operador es guiado hasta la posición correcta por unos auriculares con terminal de voz. A continuación, confirma la ubicación leyendo en voz alta el dígito del producto en cuestión o escaneando el código de barras de la localizacion con su escáner de mano. El terminal de voz le indica la cantidad que debe recoger. Cualquier recopilación de datos necesaria se escanea como parte de la transacción de selección y, por último, el operador confirma verbalmente la finalización de la tarea. Este enfoque combina las ventajas del escaneo RF

(recogida de datos) con las ventajas de la tecnología de voz (picking de manos libres).(*Order Picking Technologies | Voice | Pick Light | RF | MWPVL*, no date)

Figura 5: Ejemplo de combinación de varias tecnologías de picking.
Fuente: MWPVL international.

2.2.1 Asignación de localizaciones

Esta operación no forma parte del proceso del picking pero lo condicionará en gran medida. La mayoría de las de las actividades de manejo de material en almacenes suponen unas rutinas repetitivas, de ahí que la localización de los productos defina en gran parte la eficiencia del almacén. La disposición de los productos en los almacenes debe cumplir con ciertas restricciones sobre la ubicación relativa de los artículos, que se conserve su seguridad y la compatibilidad entre los mismos, y además cumplir con los requerimientos para la preparación eficiente y oportuna de los pedidos. El principal objetivo de la asignación de localizaciones, es minimizar la distancia o el tiempo de recorrido dentro del almacén para satisfacer las órdenes de los clientes.

A la hora de decidir la ubicación donde se va a almacenar la mercancía lo más importante es optimizar el uso del área de almacenamiento disponible. Existen dos maneras básicas de ubicar la mercancía en los almacenes, la ubicación aleatoria y la estática. En la primera de estas metodologías, como su propio nombre indica, se asigna al producto el primer slot desocupado que se encuentre en el almacén. De este modo se ahorra tiempo a la hora de asignar una localización pero este ahorro desemboca por otro lado en problemas a la hora de localizarlo, es preciso ser muy organizado con los registros para la gestión de este tipo de ubicación de mercancía. Sin embargo, en el caso de la ubicación estática cada tipo de producto tiene su slot reservado. Esta metodología de almacenamiento, al contrario que la aleaoria permite localizar el producto buscado de manera sencilla, sin embargo, la ubicación estática no optimiza el espacio de almacenamiento ya que en ocasiones las posiciones más accesibles se encuentran vacias y no pueden utilizarse para mercancías distintas.

Cuando se realiza una ubicación estática de productos, una práctica muy común a la hora de asignar las ubicaciones a productos es realizar un análisis ABC de la mercancía. Este análisis hace una clasificación de los productos en función del valor global de los mismos, basándose en el principio de Pareto, que establece que un 20% de los productos del almacén equivalen al 80% del consumo total en valor. Es decir, que no existe una distribución uniforme de la demanda sino que un pequeño grupo de artículos muy demandados superan el conjunto global del resto de artículos. Las reglas que se siguen en el método ABC son las siguientes:

- Tipo A: Son los productos claves. Aproximadamente un 10-20% del total de los productos almacenados que suponen un 70-80% del valor global del almacén en cuanto a demanda del cliente.

- Tipo B: Productos de importancia media, cuyo valor según demanda es intermedio. Aproximadamente un 25-35% del total de productos almacenados que suponen un 15-25% del valor global del almacén en cuanto a demanda del cliente.

- Tipo C: Son los opuestos al tipo A, es decir, aproximadamente un 45-55% del total de los productos almacenados que suponen un 5%-10% del valor global del almacén en cuanto a demanda del cliente.

Además, para el caso de la ubicación de mercancías estática, las distintas ubicaciones en las que se almacenarán los productos deben estar claramente definidas para la posterior localización. Para ello, comúnmente se hace uso de sistemas de coordenadas compuestos por números y letras.

Este sistema permite identificar el lugar específico en el que se encuentra cada producto. La manera más común de realizar dicha identificación a día de hoy es según estos métodos:

- Identificación por estanterías: se enumeran las estanterías, los niveles de profundidad de cada estantería y las distintas alturas de las estanterías. Por ejemplo, Loc. A1-3 representa la estantería A, profundidad 1 y altura 3.

- Identificación por pasillos: del mismo modo que por estantería pero en lugar de enumerar las estanterías se enumeran los pasillos. Por ejemplo, Loc. 1-4-0 representa el pasillo 1, la profundidad 4 y altura 0.

A continuación se muestra un ejemplo de estos dos métodos de identificación de las ubicaciones en los almacenes.

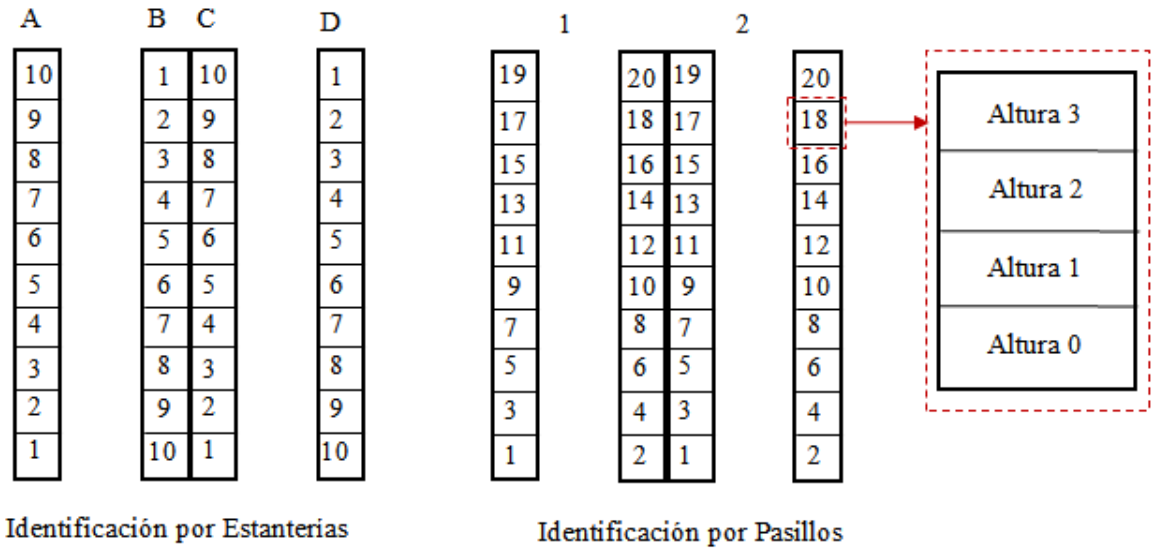


Figura 6: Identificación de ubicaciones.

Fuente: Elaboración propia.

Para la definición del tamaño del almacén, layout del almacén, ubicación de mercancías en almacén y simulación de pedidos de este proyecto, se ha utilizado un simulador online que bebe de una base de datos de pedidos de supermercados reales. Esta base de datos utilizada se explicará con más detalle más adelante. El simulador se puede encontrar en el siguiente link:

https://homepages.dcc.ufmg.br/~arbex/orderpicking.html

2.2.2 Agrupación de pedidos

El proceso de agrupación de pedidos está altamente ligado a optimizar las rutas del picking. Este proceso consiste en preparar varios pedidos en un mismo contenedor. Con este sistema se logra optimizar el tiempo destinado por los operarios en aquellos almacenes en los que se aplique una filosofía de 'Operario a mercancía' y los movimientos de las máquinas en los almacenes automatizados. La manera más común de llevar a cabo la agrupación por lotes es una vez terminada la recolecta de mercancía, se deposita el contenedor en un área destinada a la preparación de pedidos y una vez allí, se manipula la mercancía. Por esto, este sistema es el menos utilizado ya que es necesario separar la carga e ir asignándola a los diferentes pedidos.

También existen otras maneras de realizar el picking según cómo se extrae la mercancía: preparación pedido a pedido, preparación por olas de pedido, etc. En el caso de la preparación pedido a pedido, los operarios sólo pueden preparar los pedidos de uno en uno. Es aconsejable elegir esta opción cuando se trata de pedidos de grandes volúmenes de mercancía o que están conformados por muchas líneas.

2.2.3 Ruta de la recogida de pedidos en almacén

En el picking agrupado, tal y como se ha descrito anteriormente, varios pedidos se agrupan en lotes. A cada trabajador se le asigna un lote y este recoge todos los productos de los pedidos incluidos en este. Como se recogen simultáneamente distintos pedidos, se pueden diseñar rutas eficientes para reducir los tiempos de recogida de pedidos.

La determinación de la ruta establece la secuencia de pasos para recoger todos los productos del mismo lote. Asimismo, determina el tiempo necesario para recoger los pedidos, por tanto, determinar una buena secuencia para que el picker recoja los productos desde sus ubicaciones de almacenamiento colaborará a la mejora de la eficiencia del almacén. De este modo se reducirá la distancia recorrida por el operario y se agilizará todo el proceso de recolecta de los productos de los pedidos pertenecientes a un mismo contenedor.

Está comprobado que existen esquemas de definición de rutas óptimos, pero que en la práctica conducen a rutas complicadas para los trabajadores, pudiendo generar congestiones en los pasillos y errores humanos. Es por eso que suelen ser más prácticos los esquemas heurísticos más sencillos, como S-Shape (en forma de S), Largest Gap o el combinado.

- S-Shape, es la forma más sencilla de todas. El picker recorre por completo todos los pasillos donde se encuentra algún producto que debe ser recogido. Si el número de

pasillos fuese impar, el último pasillo solo se atraviesa hasta el producto más alejado, sin necesidad de recorrer el pasillo entero. En la Figura 7 se muestra un ejemplo de recogida de productos con esta estrategia.

- Largest Gap, es la estrategia en la que se recorren enteros tanto el primer como el último pasillo que contengan productos para recoger. El resto de pasillos se recorren sin atravesar el denominado largest gap. Que se define cómo la máxima distancia entre dos productos consecutivos dentro del mismo pasillo, o la distancia entre el pasillo transversal y el producto más cercano situado en el pasillo paralelo.

- Estratégia combinada, a la hora de recorrer cada pasillo, se realiza una evaluación sobre cuál de las dos metodologías de recorrido, S-Shape o Largest Gap es más corta para recoger todos los productos de ese pasillo en cuestión.

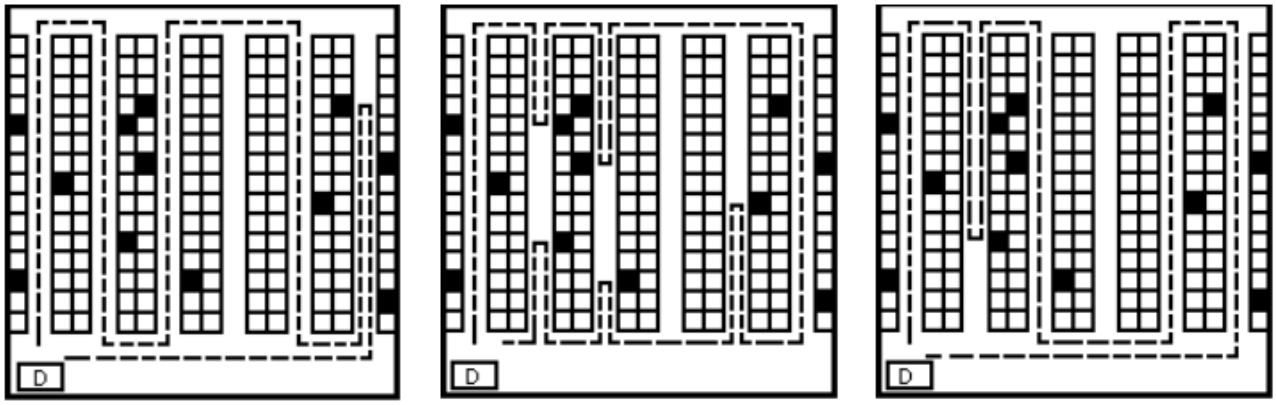


Figura 7: Rutas S-Shape, Larges Gap y Combinado respectivamente.

Fuente: Búsqueda de Vecindad Variable para el problema de la agrupación y recogida de pedidos online en almacenes logísticos (Sergio Gil-Borrás, Abraham Duarte, Antonio Alonso-Ayuso y Eduardo G. Pardo).

En el picking al igual que en todo proceso y, especialmente cuando es manual, la probabilidad de error existe, y su no eliminación desencadena insatisfacción del cliente y genera una fuente de conflictos internos en la organización. El diseño del proceso, reflejado a través del método de picking elegido, el layout, el grado de formalización, el procesamiento de las urgencias y la planificación de los recursos son algunos de los factores que intervienen en la aparición de errores. Conforme las exigencias operativas aumentan, debido al requerimiento de una mayor

velocidad de operación y/o carga de trabajo superior a las definidas como normales, se dan situaciones que suponen una mayor posibilidad de error para el picker. Por tanto, el adecuado conocimiento de las causas, es el primer paso de la mejora necesaria en la preparación de pedidos eficiente.

Para contribuir al éxito de la implantación de mejoras dentro del proceso del picking, es muy importante que el personal involucrado esté bien preparado y tenga motivación. Este proceso está directamente vinculado con la implicación del personal en la planificación y puesta en marcha de las medidas. Según de la metodología del Lean Manufacturing existen diferentes herramientas muy válidas en este aspecto, que permiten la preparación del personal y formación sobre los métodos de resolución de problemas, eliminación de despilfarros, mejora continua y por supuesto, trabajo en equipo. Tener en cuenta estos factores será imprescindible a la hora de implantar mejoras en el proceso del picking.

En general, los elementos que intervienen en el picking y los aspectos a tener en cuenta se ven resumidos en el siguiente cuadro:

Tabla 1: Elementos y aspectos relevantes en el Picking.

<table>
<tr><th>Volumen de Picking</th><th>Almacén</th></tr>
<tr><td rowspan="2">El tipo de producto.
Unidad de carga en producción y ventas.
Número y complejidad de los pedidos.
Longitud de los pasillos y altura de las estanterías.
Niveles de stock.</td><td>Diseño de almacén.
Tipo estanterías.
Tipo carruseles.</td></tr>
<tr><td>Medios materiales</td></tr>
<tr><td>Métodos operativos</td><td>Carretillas.</td></tr>
<tr><td rowspan="2">Mercancía a operario/Operario a mercancía.
Zonificación.
Extracción agrupada.
Extracción en altura.</td><td>Informática</td></tr>
<tr><td>Gestión de ubicaciones.
Papers-less.
Radiofrecuencia, código de barras.</td></tr>
</table>

3 EL PROBLEMA DE AGRUPACIÓN DE PEDIDOS Y RUTA DEL PICKER

3.1 Definición del problema

Para este trabajo en particular, se considera el caso de la compra online en supermercados, dónde los pedidos pueden estar compuestos por docenas de productos. Este caso de estudio es el denominado "Online Order Batching Problem" (OOBP) o el "Joint Order Batching and Picker Routing Problem" (JOBPRP). Este problema aborda la determinación del orden de recogida de los pedidos que se generan en un almacén de supermercado online y la ruta que deberá ejecutar el picker para recoger todos los productos necesarios para satisfacer esta demanda. El "Order Batching Problem" (OBP) se puede defininir de la siguiente manera: Dadas las ubicaciones de almacenamiento de artículos, la estrategia de definición de ruta a utilizar y la capacidad del picker o carro de picking, ¿cómo se puede agrupar el conjunto de pedidos de clientes en lotes de picking de forma que se minimice la suma de las longitudes totales de todos los recorridos necesarios para recoger los artículos solicitados? (Wäscher and Henn, 2010; Henn, Koch and Wäscher, 2011)

Entre las características de este problema cabe destacar: la heterogeneidad entre los productos (pudiendo ser éstos de diferentes tamaños, formas o caducidad) y el hecho de que la recogida de productos se puede llevar a cabo tanto en almacenes cerrados como abiertos al público. Debido a estas características, los operarios encargados del picking suelen hacerlo de manera manual recorriendo ellos mismos los almacenes en lugar de usar sistemas automatizados. Por tanto, para este proyecto se trabajará con un almacén de distribución que tenga un sistema de preparación de pedidos manual de operario a mercancía y de preparación de pedidos a poca altura en el que se deberá recolectar un determinado conjunto de artículos. Como se ha mencionado anteriormente, en un sistema manual de preparación de pedidos, son los operadores humanos quienes realizan todas las tareas necesarias en lugar de autómatas. Por tanto, para el problema estudiado, los pickers recorren el área de picking, visitando los almacenes de los artículos respectivos y retiran la cantidad requerida de unidades del artículo en cuestión. Al tratarse de un sistema de picking de bajo nivel, es decir, a poca altura, los productos deben estar alcanzables, en palets o contenedores colocados en el suelo o en estanterías bajas a las que puede acceder directamente el picker sin necesidad de una carretilla de altura.(Henn and Schmid, 2011)

En este trabajo se pretende formular y resolver de manera conjunta el problema de agrupación de pedidos en lotes y el problema de definición de la ruta en el picking. El objetivo es encontrar los recorridos de coste mínimo, no necesariamente Hamiltonianos, en los que cada operario visita todas las localizaciones requeridas para recoger todos los productos de los pedidos que tengan asignados. Las distintas localizaciones se visitarán en más de una ocasión si fuera necesario.

Como se ha mencionado anteriormente, se considera el caso específico en el que se tienen pedidos compuestos por docenas de productos, dada esta situación se evita mezclar diferentes pedidos en una misma cesta o separar un mismo pedido en diferentes carros, para reducir así los errores en el procesamiento de pedidos.Ya que, aunque permitir la separación de un pedido en varios carros o mezclar varios pedidos en una misma cesta supondría un ahorro en tiempo de recogida y un mejor aprovechamiento del espacio disponible en los carros, estas dos practicas desembocan en una alta probabilidad de error y hace necesario emplear tiempo extraordinario en la verificación de los pedidos antes de su entrega al cliente final.

Se considera un almacén compuesto por pasillos verticales que disponen de espacios de almacenamiento a ambos lados, cada espacio contiene productos de un único tipo; estos espacios de almacenamiento también pueden estar apilados verticalmente en distintos niveles de estanterías. Se asume que el operario encargado de recoger los pedidos, denominado picker, se mueve por el centro de los pasillos pudiendo alcanzar los productos que se encuentran almacenados a ambos lados. Para este caso en cuestión, se supone que las estanterías tienen dos alturas, por tanto, desde cada posición del pasillo se podrán alcanzar hasta 4 tipos de producto diferentes. Además, los almacenes también suelen tener 'pasillos transversales'. De hecho, todos los almacenes están formados por un pasillo transversal al principio y otro al final. A su vez, pueden existir pasillos transversales extra que dividirán los pasillos principales en 'sub-pasillos'. Por medio de estos pasillos transversales, los preparadores de pedidos pueden entrar y salir de un pasillo principial en el que hay posiciones de almacenaje. Los preparadores de pedidos entran desde el área de picking al almacén, para el ejemplo de la foto es el punto denominado ‘origen’ que se encuentra en la esquina inferior izquierda (Figura 8). Este es también el lugar al que los operarios deberán regresar para depositar los artículos recogidos una vez acabada la ruta que se haya definido.

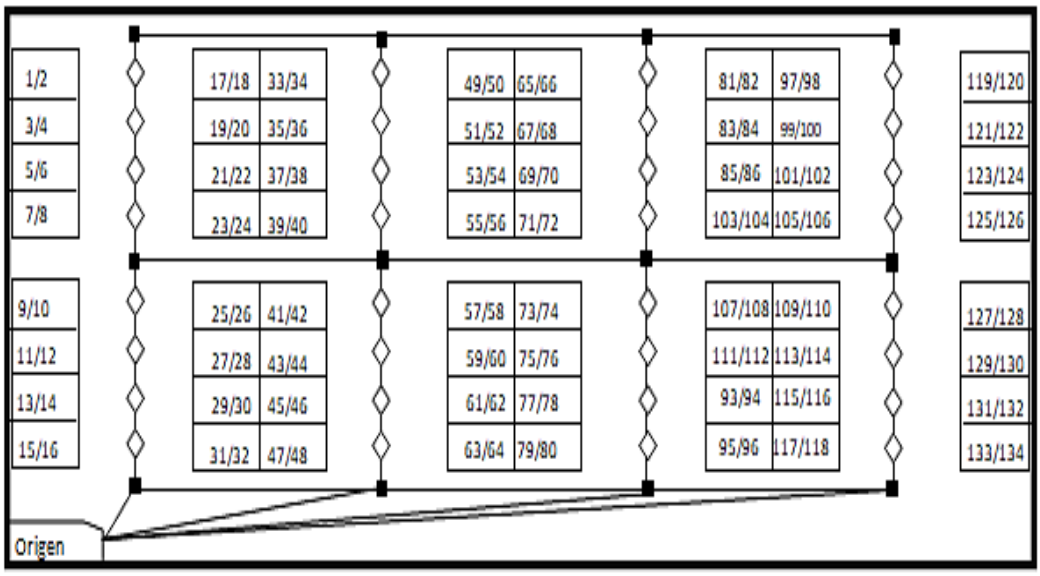


Figura 8: Ejemplo de las ubicaciones de un almacén y su correspondiente grafo.

Fuente: Elaboración propia (adaptación de European Journal of Operational Research 262 (2017) 817-834).

En la Figura 8 se muestra el layout típico de un almacén que consta de 4 pasillos transversales y dos niveles de estanterías, se representa gráficamente para plasmar la máxima similitud con el grafo que se resolverá en este proyecto.

- Las posiciones blancas, representadas mediante un rombo, son las posiciones desde las cuales el operario alcanza los productos, por ejemplo, desde el vértice blanco que se observa en la esquina superior izquierda, el operario alcanza los productos que se encuentran en los espacios 1, 2, 17 y 18.
- Las posiciones negras, representadas mediante un cuadrado, son las denominadas 'posiciones ficticias'. Estas posiciones son las que conectan los pasillos principales y los transversales.
- El origen, indica el punto de salida y retorno de los operarios. Cada ruta que realice el picker para recolectar uno o varios pedidos, debe empezar y terminar en este punto del almacén.
- Las líneas negras representan la conexión entre posiciones, es decir, desde un determinado punto sólo se puede alcanzar los puntos que estén unidos a este por una línea negra. De este modo, si se quiere recoger un producto de una posición determinada y otro de una posición que no es consecutiva a esta última, se deberá necesariamente visitar posiciones en las que no se almacene ningún producto demandado.

Como se ha mencionado, a menudo, no es necesario parar en todas las posiciones del almacén. En la mayoría de las ocasiones, el listado conjunto de todos los pedidos a recoger no engloba productos de todas las posiciones, es decir, pueden demandar varios productos que se alcanzan desde una misma posición pero ninguno de los otros cuatro productos que se alcanzan desde otra posición. Por tanto, cómo se puede ver en la Figura 9 es posible reducir la representación gráfica del almacén eliminando dichas posiciones en las que no se hará una parada cómo tal, sólo se visitarán 'de paso'. Por ejemplo, en esta nueva representación, los espacios de almacenamiento 3, 4, 19 y 20 no contienen ninguno de los productos demandados.

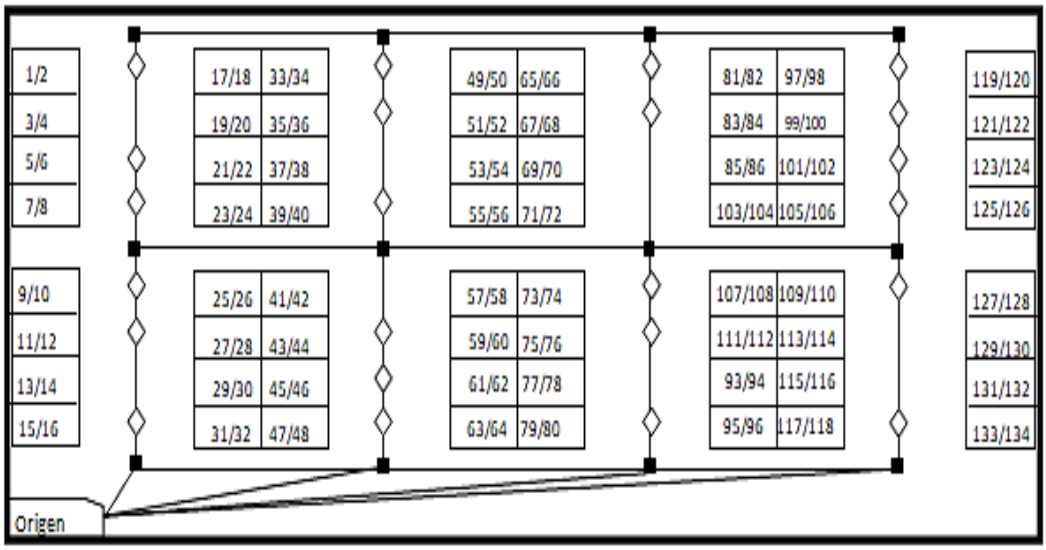


Figura 9: Grafo reducido de un almacén.

Fuente: Elaboración propia (adaptación de European Journal of Operational Research 262 (2017) 817-834).

Para la recogida de pedidos, los pickers usan carros que pueden cargar un número limitado de cestas. Se asume que el número de cestas que se necesitan para cargar cada pedido es un dato conocido. De hecho, en este trabajo se asume que cada cesta contiene un número fijo de productos, independientemente de su forma o tamaño. Además, el número de cestas necesarias para recoger un único pedido no debe exceder la capacidad de un carro. Por tanto, habrá que asumir que la cantidad de producto demandado en una única orden será siempre inferior a la capacidad máxima de un carro.

$$Demanda\ máxima\ en\ un\ pedido\ = \frac{Nº\ de\ Cestas}{Carro}\ x\ \frac{Nº\ de\ Productos}{Cesta}$$

Como se ha mencionado anteriormente y siguiendo el procedimiento general de los supermercados, los pedidos de diferentes clientes no se pueden mezclar en una misma cesta y un único pedido no puede estar separado en diferentes carros. La mezcla y división de pedidos podría reducir el tiempo de preparación de los mismos, sin embargo el tiempo ahorrado luego sería necesario para un post-procesamiento y se aumentaría el riesgo de error.

3.2 Estado del arte

Existe mucha documentación sobre los problemas de agrupación de pedidos y picking. Se presenta una extensa encuesta sobre las diferentes estrategias que se utilizan, en la que se muestra que el picking es la actividad más crítica para el aumento de la productividad en el sector (De-Koster, Le-Duc and Roodbergen, 2007). Un ejemplo es el *Wave picking*, también llamado picking por oleadas o preparación de pedidos por olas en español, es una estrategia que agrupa varios pedidos por un criterio común, por ejemplo por transportista, por cliente, por prioridad de envío, por volumen de mercancía, etc., para ser preparados en un plazo de tiempo determinado. Sin embargo esta estrategia necesita un post-procesamiento de pedidos. En este trabajo se habla de selección por lotes, dónde el operario recolecta los productos para uno o varios pedidos.
Se han propuesto muchos algoritmos para el problema de definir la ruta de un único picker. Se presenta una evaluación de varias heurísticas dónde además se incluye un algoritmo exacto para almacenes que constan de dos pasillos transversales (Petersen, 1997). Se presentó un algoritmo de programación dinámica para almacenes de hasta tres pasillos transversales (Roodbergen and De Koster, 2001b) además de heurísticas para almacenes con múltiples pasillos transversales (Roodbergen and De Koster, 2001a). Este último se compara con un algoritmo de branch-and-bound basado en la formulación clásica TSP (Travelling Salesman Problem) del problema, en la que se asume un circuito hamiltoniano en un gráfico completo en el que todos los vértices deben ser visitados una vez (no se consideran vértices ficticios). Se ha adaptado (Theys *et al.*, 2010) la heurística TSP LKH (Lin and Kernighan, 1973) para fijar la ruta del recorrido de un único operario y se ha llevado a cabo un estudio del caso de una compañía holandesa de venta al por menor cuyo layout consta de múltiples pasillos transversales, dos plantas y diferentes puntos de origen y destino para los pickers (Dekker *et al.*, 2004) .

Se han realizado estudios de un caso en particular que resulta muy interesante en el que se asume un único picker y se conocen las localizaciones requeridas, este caso del problema viajante, conocido como STSP por sus siglas en inglés, Steiner Travelling Salesman Problem. La formulación que se presenta en este trabajo está inspirada en dos formulaciones compactas del STSP (Letchford, Nasiri and Theis, 2012). Además de tener múltiples carros, el problema que se

propone en este trabajo difiere del STSP en que la asignación de los vértices requeridos para cada ruta no está definida a priori.

También se han propuesto varias heurísticas para el agrupamiento en lotes y la ruta de recogida para múltiples pickers, en muchas ocasiones las distancias de ruta se estiman utilizando heurísticas para un único picker para resolver el problema de agrupación de pedidos. Se ha realizado también un estudio sobre métodos de agrupamiento en lotes (Henn, Koch and Wäscher, 2011).

Para almacenes con dos pasillos transversales, se propone una heurística VNS -Variable Neighborhood Search, Búsqueda de Entorno Variable en español- (Albareda-Sambola *et al.*, 2009) y más tarde, (Wäscher and Henn, 2010) presentaron la Búsqueda Tabú y el algoritmo de Escala. Estos últimos métodos se adaptaron más tarde para un problema que también considerase la secuenciación de pedidos (Henn and Schmid, 2011). En otro articulo, se plantean heurísticas que resuelven en primer lugar la secuenciación y agrupación de pedidos y en un segundo plano la definición de la ruta a seguir (Azadnia *et al.*, 2013). Para almacenes con tres pasillos transversales, existen otros estudios en los que se proponen un algoritmo de recocido simulado en el que se incluyen las relaciones de precedencia (por ejemplo, que las cajas más pesadas tengan que situarse en el fondo del contenedor), en este método, la definición de la ruta de recogida de los lotes candidatos se resuelve con el algoritmo A* (Hart, Nilsson and Raphael, 1972).

En (Gibson and Sharp, 1992) introdujeron heurísticas basadas en curvas que rellenan el espacio para dos y cuatro dimensiones. (Hsu, Chen and Chen, 2005) propusieron un algoritmo genético que también considera de forma conjunta la definición de la ruta y la agrupación de pedidos. (Hsu, Chen and Chen, 2005) presentan un enfoque de agrupamiento basado en la asociación para la división en lotes teniendo en cuenta las demandas de los clientes.

4 METODOLOGÍA DE RESOLUCIÓN

El problema que se aborda en este proyecto se puede asimilar al clásico problema de ruteo de vehículos (VRP) ya que consiste en encontrar la mejor ruta/combinación de vehículos para visitar una serie de nodos necesarios para cubrir la demanda partiendo desde un nodo de origen y volviendo al mismo. El nodo 'origen' es lo que en el VRP clásico sería el depósito central y las posiciones a visitar se asemejarían a los clientes, siendo los productos a recoger de cada uno de ellos lo que limitaría la capacidad del vehículo, es decir, similar a la demanda del cliente. Existe un análisis comparativo entre el problema que se pretende resolver (el problema de conformación de lotes con ruteo en la preparación de pedidos) respecto al HVRP - Heterogeneous Vehicle Routing Problem (Gómez *et al.*, 2016).

El problema de la agupación de pedidos por lotes es un problema considerado NP-hard, por lo que es extremadamente difícil obtener soluciones óptimas para problemas a gran tamaño en un periodo de tiempo de cálculo razonable (Hsu, Chen and Chen, 2005; Valle, Beasley and da Cunha, 2016). Obtener una solución exacta para los problemas de agrupación de pedidos por lotes es muy difícil y lleva mucho tiempo (Chen et al., 2005; De Koster et al., 2007). Por otro lado, para el problema de definición de la ruta del picker, el tiempo de ejecución aumenta exponencialmente con el número de posiciones a visitar, siendo este tipo de problemas muy difíciles de tratar en escalas reales (Hsu, Chen and Chen, 2005). Por este motivo se ha determinado que los problemas de tamaño real para la ruta del picker en almacén no pueden resolverse en un tiempo de cálculo razonable (Albareda-Sambola *et al.*, 2009). Debido a que ambos problemas por separado lo son, separación de pedido en lotes y la definición de la ruta del picker, el problema conjunto de ambos, que es el que se pretende resolver en este trabajo, también será considerado NP-hard. Se han planteado diferentes métodos para la resolución de este tipo de problemas. Estos métodos, pueden ser clasificados en dos grupos, exactos y aproximados. Estos últimos se dividen a su vez en heurísticas de resolución y metaheuristicas.

Métodos exactos de resolución

Mientras los métodos aproximados, heurísticas y metaheurísticas, son capaces de obtener soluciones factibles a un problema determinado en un tiempo computacionalmente aceptable, tal y como se verá en las siguientes secciones, los métodos exactos son capaces de proporcionar la solución óptima global del problema. Aunque las soluciones que se obtienen con estos métodos

son óptimas, para llegar a ellas hace falta un mayor tiempo computacional. La mayoría de los métodos exactos tienen como base un modelo matemático, del que se centra en las variables reales, se busca la mejor solución para estas varibles y después se busca la solución optima entera partiendo de esta solución inicial. La inversión de tiempo que necesitaría un método exacto para resolver de manera óptima un problema NP-hard, suponiendo que existiera un método exacto capaz de resolver tal problema, es de mucho mayor orden de magnitud que el tiempo computacional invertido en heurísticas y metaheuristicas. Algunos de los métodos exactos más comunes para la resolución de problemas son los siguientes:

- Branch and Bound: Un algoritmo de Branch and Bound (o Ramificación y Acotación) consiste en una enumeración sistemática de soluciones candidatas por medio de la búsqueda en el espacio aprovechando el hecho de que un número muy grande de soluciones no válidas pueden ser descartadas mediante la estimación de cotas inferiores y superiores del valor que desea encontrarse. El conjunto de soluciones candidatas se considera como la formación de un árbol ramificado siendo el conjunto completo la raíz del árbol. El algoritmo explora las ramas de este árbol, que representan subconjuntos del conjunto de soluciones.

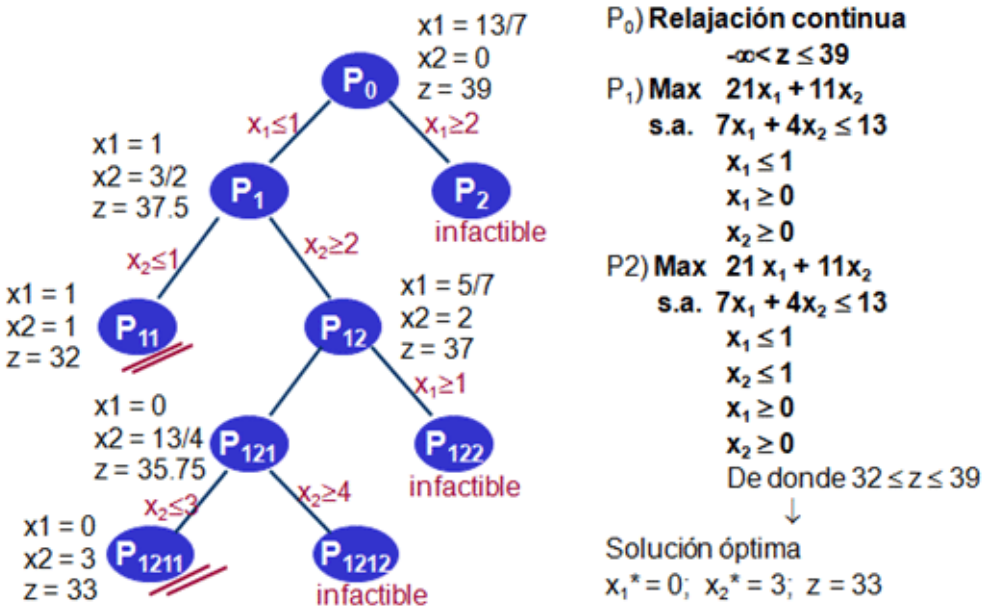


Figura 10: Branch and Bound example.

Fuente: Programación Entera y Ejemplos. (Investigación de Operaciones, 2015).

Este algoritmo se diseñó para buscar soluciones a modelos matemáticos de programación entera. El procedimiento parte de una linealización del modelo. Una vez resuelto el modelo como si se tratara de un modelo matemático de programación lineal,

lo que también se conoce como resolver la relajación del modelo, se deben definir las cotas inferiores y superiores siempre y cuando exista al menos una variable de decisión entera cuyo valor no sea entero. El algoritmo consiste en generar de forma iterativa cotas (o restricciones adicionales) que conducirán a los valores enteros para las variables de decisión.

- Branch and Cut: Este método es un algoritmo exacto que consiste en una combinación de un método de plano de corte y un algoritmo de branch and bound. En definitiva, consiste en la aplicación del método de plano de corte a la relajación del problema (problema linealizado) y más tarde resolver la relajación mediante un algoritmo clásico de Branch and Bound. El Branch and Cut surge de que al ejecutar el algoritmo de planos de corte sobre los la relajación lineal de los nodos, se pueden obtener cotas mas restrictivas, que permita podar más nodos que antes no habían sido descartado.(Severin, 2012). Este método disminuye mucho el tiempo de resolución empleado. El procedimiento que se sigue es el siguiente:

  - ✓ Elección de un nodo para evaluar (inicialmente el nodo raíz es el problema original relajado) y resolución del mismo.
  - ✓ Decidir si generar o no planos de corte. Si se obtienen desigualdades válidas incumplidas, se añaden al problema y se resuelve éste.
  - ✓ Podar y ramificar según los criterios del método de ramificación y acotamiento.

- Branch and Price: En este método de resolución que tiene un enfoque de generación de columnas, en lugar de fijar precios de las variables no básicas mediante enumeración, el precio reducido (más negativo o positivo) se encuentra resolviendo un problema de optimización.

  En este método algunas clases de desigualdades válidas, ordenadas en columnas, se dejan fuera de la relajación del problema lineal porque hay demasiadas restricciones para manejar de manera eficiente los problemas y la mayoría de ellas no serán vinculantes en una solución óptima del mismo. Entonces, si no es posible hallar una solución óptima para la relajación de un problema lineal, se resuelve un subproblema llamado problema de separación para tratar de identificar las desigualdades incumplidas en una clase. Si se identifican desigualdades incumplidas, se agregan a la formulación del problema y se re-optimiza. La ramificación se da cuando no se encuentran desigualdades incumplidas que corten una solución inviable.

Heurísticas de resolución

En la literatura se recogen muchas definiciones diferentes de algoritmo heurístico, entre las que destaca la siguiente (De-Armas, 2013):

"Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución."

En la actualidad, existe una fuerte tendencia al uso de heurísticas como método de resolución de problemas de optimización. La principal característica que diferencia los métodos heurísticos de los exactos es que las técnicas heurísticas proporcionan una solución razonable, considerada bastante buena pero no necesariamente la óptima. Sin embargo, las técnicas de optimización exactas conducen siempre a la solución óptima del problema en cuestión.

La principal causa de esta fuerte tendencia al desarrollo de técnicas de resolución de problemas basadas en heurísticas pueden ser algunas de sus características tales como: flexibilidad, bajo coste computacional, determinación de soluciones factibles que pueden ser utilizadas como solución inicial a un problema para, a partir de ésta, aplicar otro procedimiento.

- Algoritmos de semilla: (Albareda-Sambola *et al.*, 2009) hace referencia a este tipo de algoritmos para la resolución del problema de segregación de pedidos en lotes, que es uno de los que se abordan en este proyecto. Consiste en construir los lotes uno a uno, siguiendo dos pasos. Primero, se selecciona una orden de semillas del conjunto de órdenes aún no seleccionadas, que servirá como orden de referencia para el nuevo lote. El segundo paso consistiría en la adición de pedidos, el lote se amplía añadiendo secuencialmente nuevos pedidos, siempre que no se exceda la capacidad máxima de picking. Los nuevos pedidos se seleccionan utilizando alguna medida de proximidad al pedido de partida (pedido de semillas). Se pueden definir diferentes métodos de siembra eligiendo diferentes reglas para cada uno de los dos pasos, la selección de la semilla y el criterio de proximidad.

  Además, todas las reglas de selección de semillas pueden aplicarse tanto en modo simple como en modo acumulativo. En el modo simple, la semilla se selecciona una sola vez, mientras que en el modo acumulativo el orden de las semillas se actualiza cada vez que se añade una orden al lote.

- Algoritmos de Ahorro: Este tipo de algoritmos es comúnmente aplicado para la resolución de problemas como el que se pretende resolver en el presente documento. Problemas de definición de rutas en función de pedidos que se deben recoger tras organizarse en lotes. Es muy útil tanto para la optimización de la ruta cómo para la formación de los lotes. (Albareda-Sambola *et al.*, 2009) muestra cómo funciona este algoritmo para la segregación de pedidos en lotes. Se parte de tantos lotes como ordenes se tenga, y de manera secuencial, se trata de reducir el número total de lotes combinando dos lotes en uno solo. El criterio utilizado para seleccionar los lotes que deben fusionarse cada vez es en función del ahorro de costes que supone cada fusión.

  Este método de resolución se basa en el algoritmo de ahorro de Clarke y Wright que se utiliza con frecuencia para la resolución del VRP. Este método se aplica generalmente a problemas en los que el número de vehículos se modela como una variable de decisión, y funciona igualmente bien para problemas dirigidos y no dirigidos. Siendo T_B el recurso empleado en recolectar un lote B, se puede definir una matriz de ahorro, donde el elemento (p, q) es el ahorro obtenido por la fusión de los lotes Bp y Bq en un solo y se calcula de la siguiente manera:

$$Ahorro\ (Bp, Bq)\ =\ T_{Bp}\ +\ T_{Bq}\ -\ T_{Bp \cup Bq}$$

  Si este valor es positivo y el valor de no se excede la capacidad del recogedor, entonces los lotes Bp y Bq se fusionan en un solo lote.

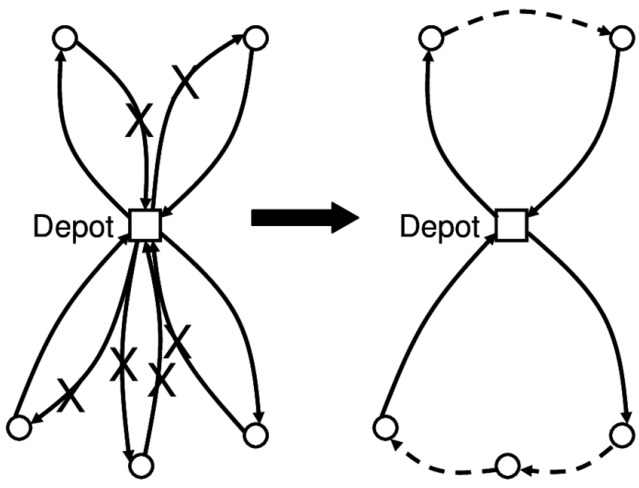


Figura 11: Saving Algorithms.

Fuente: The pickup and delivery problem with split loads (Nowak, 2005).

- Algoritmos basados en la regla de prioridad: Para el problema en cuestión de este proyecto (Gibson and Sharp, 1992), establece que en los algoritmos basados en reglas de

prioridad, los pedidos de los clientes se clasifican primero según un valor de prioridad y luego se asignan a los lotes siguiendo esta clasificación. La forma probablemente más conocida y más directa de asignación, consiste en la aplicación de la regla del orden de llegada (FCFS, por sus siglas en inglés, First Come First Served). La asignación de pedidos de cliente a lotes puede hacerse secuencialmente, lo que se denomina 'regla de ajuste siguiente' o simultáneamente, lo que se denomina 'regla de ajuste primero o regla de ajuste mejor'.

Metaheurísticas de resolución

El término metaheurística se refiere a algoritmos heurísticos de alto nivel que por lo general, integran varios procesos heurísticos de bajo nivel para lograr sus objetivos de optimización estratégica de alto nivel (Yang, 2009). Los algoritmos de optimización metaheurística de alto nivel buscan de manera eficiente un espacio de solución factible que es demasiado grande para ser muestreado en su totalidad. La "intensificación" y la "diversificación" son dos atributos clave de la metaheurística moderna (Yang, 2009):

"Para que un algoritmo sea eficiente y efectivo, debe ser capaz de generar una amplia gama de soluciones, incluyendo las soluciones potencialmente óptimas para explorar todo el espacio de búsqueda de manera efectiva, mientras que intensifica su búsqueda alrededor de la vecindad de una solución óptima o casi óptima".

En defintiva, se trata de una estratégia de resolución que combinan diferentes métodos de resolución heurísticos de manera que el proceso total permite salir de los óptimos locales a través de diversas estratégias y exploran espacios de búsqueda de gran tamaño. Se centran y caracterizan por su capacidad de descartar regiones del espacio de búsqueda que no son susceptibles de formar parte de la solución global, evitando explorar regiones repetidas veces. A continuación se muestran algunas de las propiedades que caracterizan a las metaheurísticas:

- Búsqueda eficiente que encuentra soluciones razonablemente cercanas al óptimo.
- Cuentan con mecanismos que optimiza la exploración del espacio de búsqueda, descartando regiones según distintos criterios.
- Poseen un procedimiento genérico, adaptable a la resolución de cualquier problema.
- Se basan en otras heurísticas más específicas que se procesan bajo el control de una estratégia de búsqueda de más alto nivel.

- Hacen uso de funciones de bondad para evaluar la calidad de las soluciones obtenidas.

A continuación se hace un pequeño estudio sobre algunos de los métodos metaheurísticos más comunes para la resolución de este tipo de problemas:

- Algoritmo Genético: El algoritmo genético es un método para resolver problemas de optimización tanto restringidos como no restringidos que se basa en la selección natural, el proceso que impulsa la evolución biológica. Este algoritmo hace repetidas modificaciones en una población de soluciones individuales. En cada paso, el algoritmo genético selecciona al azar a los individuos de la población actual para que sean padres y los utiliza para producir los hijos, la siguiente generación. A lo largo de generaciones sucesivas, la población "evoluciona" hacia una solución óptima.

  Se puede aplicar el algoritmo genético para resolver una variedad de problemas, incluidos los problemas en los que la función objetivo es discontinua, no diferenciable, estocástica o no lineal. El algoritmo genético puede abordar problemas de programación de números enteros mixtos, donde algunos componentes están restringidos para ser valorados como números enteros. Este método utiliza tres tipos de reglas en cada paso para crear la siguiente generación a partir de la población actual:

  I. Las reglas de selección: seleccionan a los individuos elegidos padres.
  II. Las reglas de cruce: combinan a dos padres para formar hijos.
  III. Las reglas de mutación: aplican cambios aleatorios a los padres.

  A continuación se muestra un ejemplo de la estructura del algoritmo genético aplicado al problema de segregación de pedidos en lotes.

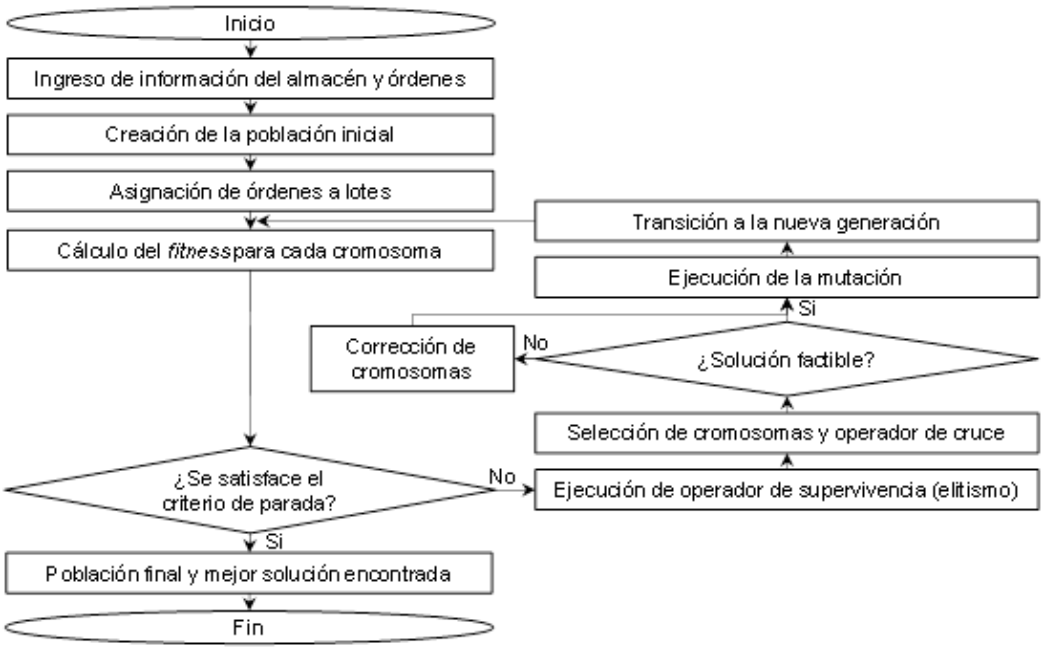


Figura 12: Genetic Algorithm for the OBP.

Fuente: (Cano, Correa-Espinal and Gómez-Montoya, 2018)

- Algoritmos de Búsqueda de la Vecindad Variable: Este método de resolución de problemas es una metaheurística basada en modificaciones sistemáticas de la estructura de vecindad. De esta manera, no se focaliza el método en estudiar los optimos locales sino que se hace un estudio de todo el espacio. El algoritmo de búsqueda de la vecindad variable, es un método que se basa en la búsqueda local pero que incorpora una fase de re-optimización en las fases intermedias, de este modo, se aumenta la probabilidad de explorar todas las regiones del espacio de soluciones factibles.

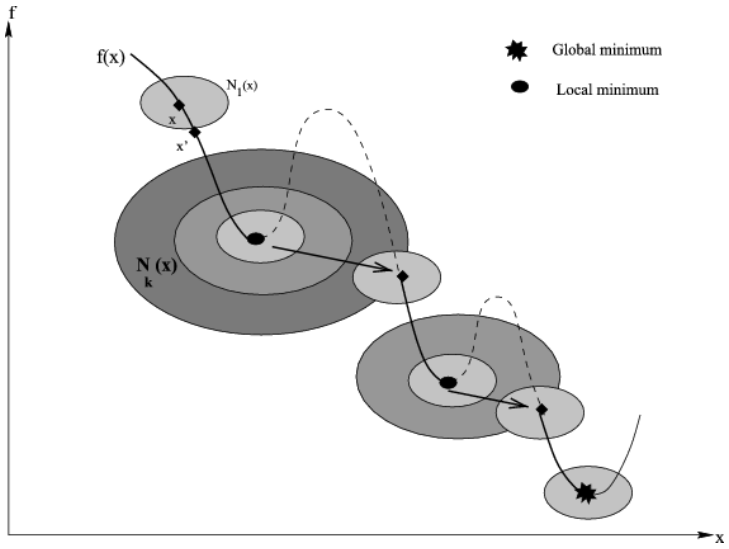


Figura 13: Variable Neighborhood Search, VNS.

Fuente: Metaheurísticas para Búsqueda y Optimización (Sancho Caparrini, 2018).

Se trata de un algoritmo muy genérico, por lo que se caracteriza por su flexibilidad para los cambios, debido a los grados de libertad del método es posible adaptarlo a la resolución de diferentes problemas a través de variaciones en el mismo.

- Algoritmo de la Búsqueda Tabú: Se trata de una metodología iteratíva de mejora, de manera que partiendo de una solución inicial factible, se explora una estructura de vecindario. Además, del mismo modo que los anteriores, confome avanzan las iteraciones y se va explorando el vecindario, se consigue mejorar el valor de la función objetivo evitando quedar atrapado en un óptimo local. Este algoritmo es considerado uno de los más populares en la optimización. Se trata de un método iterativo en el que la mejor solución obtenida de cada iteración se registra en una 'lista tabú`y se considera un movimiento. La aplicación de estos movimientos está prohibida ("tabu") para un número determinado de iteraciones con el fin de evitar el ciclismo y diversificar la búsqueda. El algoritmo se detiene cuando se cumple una condición de terminación. Esta lista es lo que se considera la memoria o el historial de este algoritmo, el cuál permite excluir estas soluciones ya exploradas en las iteraciones futuras. Es en este aspecto en el que se ve que esta estrategia de resolución de problemas de optimización combinatoria se basa en los principios fundamentales de la Inteligencia Artificial (AI).

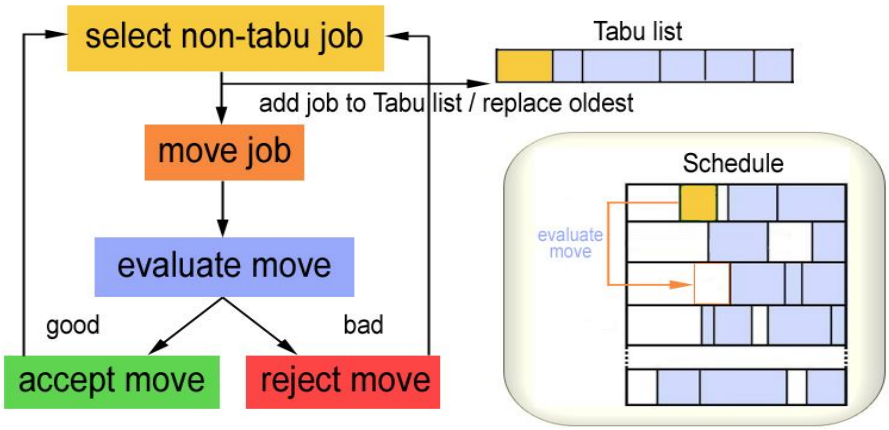


Figura 14: Tabu Search (TS) Algorithm.

Fuente: Metaheurísticas para Búsqueda y Optimización (Sancho Caparrini, 2018).

- Iterated Local Search: Es un método iterativo basado en pequeñas modificaciones que se le aplican a la solución obtenida en cada iteración procesada. Este cambio da lugar a una nueva solución, denominada 'solución intemedia' que es tratada posteriormente con un

método heurístico de mejora local. El output de este método de mejora es un óptimo local, que en función de una serie de criterios de aceptación, pasará o no a ser la nueva solución actual.

- Algoritmo de las Colonias de Hormigas: La optimización de las colonias de hormigas (ACO según sus siglas en inglés), se basa en el comportamiento natural de las hormigas cuando buscan comida, que son capaces de encontrar el camino más corto desde el hormiguero hasta una fuente específica e identificada de alimento. Esta técnica es explotada para encontrar el camino más corto entre un determinado número de rutas alternativas distribuidas en un espacio de solución factible. El concepto del algoritmo de las colonias de Hormigas es el siguiente (Shekhawat, Poddar and Boswal, 2009):

    - Las hormigas se desplazan desde el hormiguero a la fuente de alimento.
    - Las hormigas son ciegas.
    - El camino más corto se descubre a través del rastro de feromonas.
    - Cada hormiga se mueve de manera aleatoria.
    - En el camino se va dejando un rastro de feromona.
    - Más feromonas en el camino aumenta la probabilidad de que se siga el mismo.

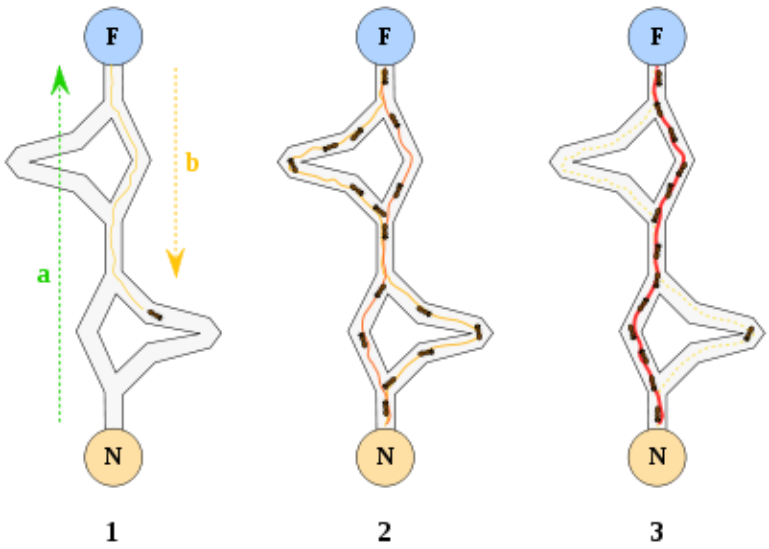


Figura 15: Ant Colony Optimization (ACO).
Fuente: Ant colony Optimization Algorithms : Introduction and Beyond (Shekhawat, Poddar and Boswal, 2009).

El problema de la preparación de pedidos por lotes y ruta del picker en el almacén (JOBPRP) se clasifica en la revisión bibliográfica llevada a cabo como NP-Hard, debido a las relaciones establecidas entre los productos y posiciones de almacenamiento, y la complejidad de resolver problemas grandes debido a las limitaciones de capacidad computacional. Por este motivo, en la literatura para problemas de preparación de pedidos por lotes con rutas de gran tamaño, se han utilizado metaheurísticos, los cuales han generado buenas soluciones.(Gómez *et al.*, 2016)

4.1 Aplicación del modelo matemático

En este trabajo se trata de encontrar una herramienta capaz de encontrar soluciones buenas para el problema conjutnto de agrupación de pedidos en lotes y ruta del picker en almacén, de manera eficiente y eficaz. Se pretende llevar a cabo una técnica flexible, con la posibilidad de adaptase a cualquier tipo de almacenes, capaz de buscar un solución optima al problema en un tiempo computacional aceptable.

En primer lugar, como se mostrará en apartados posteriores, se ha desarrollado un modelo matemático para definir el problema, se ha determinado una función objetivo de minimizar distancias y se han descrito las restricciones que lo componen. Debido a la naturaleza del prrblema, que trata de recogida de pedidos en almacenes, se ha planteado un modelo matemático basado en grafos; en este sistema de grafos las posiciones de salida, de almacenamiento y de cruce se traducen como los nodos del grafo; los posibles desplazamientos que se pueden realizar dentro del almacén se representan como los arcos de unión entre los nodo.

Tras el planteamiento del modelo matemático, se aplica a ejemplos de pedidos de clientes a un supermercado online, programando el modelo con el lenguaje de programación Python y resolviendo el mismo con el solver Gurobi Optimizer para cada uno de los casos. Estos pedidos de clientes se sacan de una base de datos real que esta disponible en la red y que recopila pedidos de clientes reales a lo largo del tiempo, combinándolos para crear instancias mayores y más variadas. Tras la pequeña revisión literia anteriormente expuesta, se puede deducir que el modelo matemático exacto es una metodología de resolución perfecta a la hora de resolver problemas de dimensiones pequeñas y complejidad media. Sin embargo, cuando se pretenden tratar almacenes que superen un cierto nivel de complejidad o tamaño, el modelo planteado como método exacto de resolución no es capaz de encontrar soluciones admisibles en un tiempo computacional aceptable y sin errores de memoria si no se dispone de medios de computación especiales. Esto se debe a que el número de variables aumenta de forma exponencial a medida que aumenta el número de posiciones de almacenamiento que se permiten visitar en el problema a resolver.

Como se ha mencionado anterioremente, para la resolución del problema en este proyecto, se ha utilizado el software de optimización Gurobi, un solver comercial que se utiliza frecuentemente en programación lineal entera mixta (MILP), además de muchos otros modelos de optimización matemática que se pueden tratar a través de Gurobi. Los modelos han sido implementados en lenguaje de programación Python y para su correcta codificación se han consultado distintos tutoriales de Python.

La versión de Python con la que se ha trabajado para el desarrollo de este proyecto es la 2.7.15 y la versión de Gurobi con la que se ha trabajado es la 8.1.0 habiendo adquirido para ello una licencia académica.

4.2 Introducción al Python

4.2.1 Historia del desarrollo del lenguaje

Debido al rápido avance tecnológico que se está experimentando, el software libre está cobrando cada día más importancia, pudiendo considerarse uno de los movimientos tecnológicos de mayor auge en la actualidad. La clave de su éxito radica en las herramientas y bases utilizadas en su desarrollo, que son sencillas para la comprensión del usuario y permiten que su utilización sea óptima. Una de estas herramientas sencillas y óptimas es el Python, que se encuentra en la lista de recomendaciones para el desarrollo de softwares libres. A continuación se descibirá este lenguaje en mayor profundida, analizando las características que presenta y cuáles son los usos principales del mismo.

El denominado 'software libre' es un movimiento tecnológico en el que cobra importancia la manipulación del software para satisfacer las necesidades del propio usuario. Es decir, que el software desarrollado según este nuevo concepto, admite su uso, ejecución, distribución y modificación de manera libre.

Como parte de este gran avance tecnológico, Linus Torvalds lanza en el año 1991 la primera versión del núcleo Linux, y es también este año cuando la primera versión de este lenguaje de programación, Python es liberada. El lenguaje de programación se puede considerar una de las bases fundamentales de la programación en general, siendo este la herramienta principal de la construcción de programas. El motivo por el que Python ha ido cobrando importancia con el paso de los años en el ámbito matemático, científico y educacional, es debido a la sencillez que presenta que permite que la dificultad de los problemas a resolver radique en el problema en sí y no en el lenguaje de programación.

Python es un lenguaje de programación que se encuentra dentro del marco de lenguajes de alto nivel. Esto es debido a que dispone de una serie importante de estructuras de datos tales como diccionarios, listas, conjuntos y tuplas, que mediante una correcta utilización de las mismas, permite realizar tareas de alto nivel y resoluciones complejas sin necesidad de generar un código de gran tamaño y permitiendo de este modo su fácil comprensión por parte de los usuarios.

El creador de este lenguaje de programación fue el programador holandés Guido van Rossum, quien a finales de los años 80 y principio de los 90 trabajaba en el desarrollo del sistema operativo Amoeba. Es entonces, cuando se libera el lenguaje Python, con el fin de tener interfaces con Amoeba y siendo así el sucesor del lenguaje ABC.

A comienzos del siglo XXI se lanza Python 2.0, una versión del anterior que presentaba un amplio soporte a Unicode. Sin embargo, la versión 3.0 de este lenguaje de programación tardó cerca de una década en terminar su periodo de pruebas, lanzándose en 2008 y siendo incompatible con las dos versiones anteriores en bastantes aspectos. Durante los últimos años, Guido van Rossum ha trabajado en Google y en Dropbox, sin embargo, durante este periodo ha seguido liderando el desarrollo del Python (Challenger Pérez, Díaz Ricardo and Becerra García, 2014).

Se trata de un lenguaje interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos (González Duque, 2008).

- Se denomina lenguaje interpretado o de script a todo aquel para el cual es necesario utilizar un programa intermedio llamado intérprete a la hora de ejecutarlo, en lugar de compilar el código a lenguaje máquina que sea comprensible por una computadora y esta lo pueda ejecutar directamente, este último caso es lo que se denomina lenguaje compilado. Sin embargo, Python presenta muchas de las características típicas de los lenguajes compilados, por lo que se podría decir que es semi interpretado.

- Se dice de un lenguaje de programación que usa un tipado dinámico cuando la comprobación de tipificación se realiza durante su ejecución en vez de durante la compilación. Es decir, no existe la necesidad de declarar con antelación el tipo de dato que será cada una de las variables, sino que este puede variar en todo momento según el valor asignado a la misma.

- El hecho de que se considere un lenguaje fuertemente tipado se traduce en que los cambios de tipos de valor requieren una conversión explícita, es decir, aunque se puedan cambiar, este cambio no se realiza de manera repentina. Sólo después de haber realizado el cambio de tipo de variable se podrá trabajar sobre el nuevo tipo.

- Que el lenguaje Python sea multiplataforma quiere decir que está disponible y es sostenible en diferentes plataformas tales como: Linux, Windows, MacOS, Solaris, etc. Por este motivo el programa generado puede ejecutarse en cualquiera de estas plataformas siempre y cuando no se haga uso de ninguna librería específica de alguna de ellas.

- Por otro lado, que un lenguaje sea orientado a objetos quiere decir que las variables que intervienen en nuestro problema real, se traducen en clases y objetos dentro de nuestro programa. Cuando se ejecuta el programa, se dan diferentes interacciones entre los objetos que componen nuestro problema. Otros tipos de programaciones también están permitidas en Python, como son la orientada a aspectos, la funcional y la imperativa.

4.2.2 La filosofía Python

En la página oficial de Python hacen referencia a una lista de principios que constituyen una especie de filosofía de trabajo denominada 'El Zen del Python' cuyo autor es Tim Peters. Es una colección de 20 principios de software que influyen en el diseño del Lenguaje de Programación Python, de los cuales 19 fueron escritos por Tim Peters en junio de 1999. El texto es distribuido como dominio público.

Esta filosofía se basa en los siguientes principios (Peters, 2004):

- Bonito es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Liso es mejor que enredado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son lo suficientemente especiales para romper las reglas.
- Lo práctico suele vencer a lo puro.

- Los errores nunca deben pasar desapercibidos.
- Excepto si se ha especificado este comportamiento.
- Ante lo ambiguo, descarte la tentación de adivinar.
- Debe haber una - y preferiblemente solo una- manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que seas holandés.
- Ahora es mejor que nunca.
- Aunque nunca a menudo es mejor que ahora mismo.
- Si su implementación es difícil de explicar, es una mala idea
- Si su implementación es fácil de explicar, puede ser una buena idea.
- Los espacios de nombre son una Buena idea – ¡Hagamos mas de esos!

Esta lista son las bases de una especie de subcultura, que implica una metodología específica a la hora de escribir un código y es una forma atractiva de trabajar para los desarrolladores. En grandes rasgos, lo que esta lista representa es que cuánto más sencillas y claras estén las cosas, mejor será el resultado obtenido.

4.2.3 Sintaxis en Python

A la hora de comunicarnos, la manera en la que se organizan las palabras y el uso que se le da a las mismas es denominada la sintaxis. En Python, es necesario cumplir una serie de requisitos a la hora de conformar el código, de este modo, evitamos errores de comprensión por parte del intérprete. En primer lugar, es necesario conocer una serie de palabras en inglés, su significado, y el aprender a utilizarlas adecuadamente dentro del lenguaje de programación. En la sintaxis de Python, cada expresión que se formula tiene una manera específica y correcta de escribirla y varias de estas expresiones juntas conformarían lo que se denomina un bloque de código.

En el lenguaje de programación Python, la sintaxis es muy sencilla, llegando incluso a parecer más bien un pseudocódigo. Para ejemplificar esto, a continuación se muestran las diferencias entre el clásico programa de 'Hola Mundo' escrito en Python y escrito en C++, que es otro de los lenguajes de programación de alto nivel más comunes a día de hoy (Challenger Pérez, Díaz Ricardo and Becerra García, 2014).

- Programa 'Hola Mundo' en C++:

```
#include <iostream>
using namespace std;
```

```
int main()
{
cout  << ”Hola Mundo” << endl;
return 0;
}
```

- Programa ‘Hola Mundo’ en Pyhton:

```
print “Hola Mundo”
```

Como puede observarse a simple vista, la diferencia en la complejidad de la sintaxis de ambos lenguajes para lograr una misma tarea es notable, Python se caracteriza por su sencillez respecto a otros lenguajes como puede ser C++. Es gracias a esta sencillez a la hora de escribir, por lo que este lenguaje de programación encabeza la lista en el ámbito educacional, ya que hace fácil su comprensión y el proceso de aprendizaje se simplifica en comparación con otros lenguajes.

Cuando la disposición de las palabras es incorrecta o incoherente dentro del bloque de código, Python avisará de que se ha cometido un error mostrando el siguiente mensaje: Sintax Error. Esto quiere decir que el fin de la sintaxis, que es indicar una determinada instrucción al intérprete, no se ha podido acometer.

A continuación se explica en mayor detalle qué es una instrucción en Python.

4.2.4 Instrucciones en Python

Una instrucción es un conjunto de caracteres que permiten al desarrollador definir una acción que debe gestionar su algoritmo. Esta acción puede ser la asignación de un valor a una variable, la ejecución de una función, la declaración de una clase, la escritura de una condición, la entrada en una iteración o cualquier otra actividad (Chazallet, 2015).

Para declarar una variable en Python o declarar una función, existen unas instrucciones expecíficas y sólo puede llevarse a cabo la acción a través de éstas.
Tipos de instrucciones:

- Instrucciones simples: Estas instrucciones son todas aquellas que están compuestas por una sola línea.

```
a = 'Hola'
print (a)
```

- Instrucciones compuestas: En este otro caso, su sintaxis comieza con una clausula de sentencia compuesta que finaliza con <<:>> y que es continuada con una indención más abajo donde se comprende el resto del bloque de código.

```
if  a == 12:
    print (a)
```

4.2.5 La biblioteca estándar

Uno de los puntos fuertes de Python frente a otros lenguajes de programación es que cuenta con una librería estándar muy amplia. Esta librería esta compuesta por decenas de módulos que permiten agilizar en gran medida el trabajo de un programador, satisfaciendo las necesidades básicas de cualquier programa. Algunos de los campos que engloba esta biblioteca son los siguientes:

- Estructuras de datos.
- Interfaz al sistema operativo.
- Formatos de archivo.
- Funciones numéricas.
- Funciones matemáticas.
- Manejo de errores y excepciones.
- Medición del rendimiento.

La biblioteca estándar de Python es muy amplia y ofrece una amplia gama de servicios, tal y como los que se indican en la lista anterior y muchos otros. La biblioteca contiene módulos incorporados, escritos en C, que proporcionan acceso a la funcionalidad del sistema, tales como E/S de archivos que de otro modo serían inaccesibles para los programadores de Python, así como módulos escritos en Python que proporcionan soluciones estandarizadas para muchos problemas que ocurren en la programación diaria.

4.2.6 Entornos de Desarrollo Integrado

A la hora de diseñar un software, los desarrolladores hacen uso de lo que se denomina un entorno de desarrollo integrado, que es a su vez una aplicación de software. Esta aplicación es una compilación de todas la herramientas necesarias que se almancenan en un mismo entorno compuesto por las distintas herramientas automáticas de construcción, un editor de código fuente y un depurador de bugs.

A continuación se listan algunos de estos desarrolladores que han sido elaborados para el uso de Python:

- PyDev para Eclipse: El Eclipse es una plataforma de desarrollo llamada RCP, en la cual con el fin de ampliar su funcionalidad, se montan una serie de plugins. Por otro lado, el PyDev es idóneo para organizar las distintas aplicaciones. Este último posee buenas características tales como: completado inteligente del código, resaltado de código, depurardor, gestión de la documentación, etc. Esta serie de características hacen que este entorno sea ideal para proyectos de tamaño considerable.

- IDLE: Su nombre corresponde a las iniciales de Integrated DeveLopment Enviroment. Se trata de un entorno relativamente simple que es desarrollado en el propio Python, por lo tanto tiene incorporado un intérprete al vuelo que es una consola que permite realizar operaciones y pruebas sin necesidad de crear un módulo para ello. Este entorno de desarrollo es ideal para aplicaciones de tamaño reducido ya que es muy simple y además es distribuido junto con el resto de paquetes de Python para los sistemas operativos más usados: Windows, Linux y MacOS.

- PyCharm: Es un IDE desarrollado por JetBrains con muy buena acogida en el mundo de los desarrolladores que trabajan con Python. Se trata de un entorno desarrollo integrado con distintas funciones, esto hace que el entorno sea algo pesado pero aumenta enormemente su potencial a la hora de programar.

- Spyder: Spyder es otro interesante entorno científico escrito en Python, para Python, y diseñado por y para científicos, ingenieros y analistas de datos. Ofrece una combinación única de la funcionalidad avanzada de edición, análisis, depuración y perfilado de una herramienta de desarrollo integral con el análisis de datos, ejecución interactiva, inspección en profundidad y capacidades de visualización de un paquete científico.

Más allá de las numerosas funciones que este entorno trae incorporadas, sus capacidades pueden ampliarse aún más a través de su sistema de plugins y API. Debido a su alta capacidad de análisis de datos, este entorno se está convirtiendo en uno de los más populares a dia de hoy.

Este proyecto se ha desarrollado en spyder debido a la previa utilización de este entorno y por tanto, al fácil acceso a la licencia de estudiantes. Para ello, se ha gestionado a través de Anaconda Distributions que es una distribución de código abierto que incluye un conjunto de aplicaciones, librerías y conceptos diseñados para el desarrollo específicamente con el fin de satisfacer la Ciencia de datos con Python. En líneas generales Anaconda Distribution es una distribución de Python cuyas funciones son gestionar el entorno, gestionar los paquetes y que además posee una colección de más de 720 paquetes de código abierto (Toro, 2017).

4.2.7 Ventajas y desventajas

A continuación, se muestra un breve resumen sobre las ventajas e inconvenientes que presenta Python como lenguaje de programación.

- Ventajas que presenta el lenguaje:

Simplificado y rápido

Como se ha comentado en apartados anteriores, se trata de un lenguaje muy sencillo, que permite resumir lo que en otros lenguajes necesita más de 3 líneas, en una sóla. Es un lenguaje que simplifica en gran medida la programación y por tanto reduce los tiempos empleados.

Ordenado y limpio

Debido a la sencillez en su estructura y a la filosofía de Python de la que se ha hablado anteriormente y la cuál es popular entre los desarrolladores, es considerado un código de fácil lectura. Además, los módulos se encuentran bien organizados y con sus respectivas indentaciones para que la estructura sea clara a simple vista. Esta ventaja es una de las que lo hace destacar frente a otros códigos del mismo sector.

Flexible

Debido a la amplia colección de herramientas que proporciona el lenguaje se considera un lenguaje flexible. Un ejemplo de esta flexibilidad sería la facilidad para cambiar el tipo de las variables, sin necesidad de declarar cada tipo de dato.

Amplia cartera de usuarios

El hecho de que este lenguaje sea cada vez más común en el mundo de la programación, promueve a su vez su uso debido a la cantidad de código disponible en internet y los numerosos foros y tutoriales. Este aspecto facilita la obtención de ayuda de manera sencilla y rápida en caso de necesitarlo.

Portabilidad

La mayoría de los programas en Python se pueden ejecutar indistintamente en la gran parte de las plataformas disponibles. Por otro lado, Python ofrece múltiples opciones para desarrollar interfaces gráficas de usuario portables, entre ella: TKinter, wxPython, PyGTK y PyQT.

- Inconvenientes que presenta el lenguaje:

Tiempo de ejecución

Los programas interpretados son más lentos que los compilados. Sin embargo los programas interpretados suelen ser más cortos, compensando esto el tiempo de ejecución. Por tanto, según el caso, esta lentitud puede ser inapreciable frente a otros lenguajes.

Detección de errores

Como se trata de un lenguaje interpretado, al cometer un error en la escritura, no se indicará nada hasta que no se ejecute el código y se trate de ejecutar la línea en cuestión. Mientras el código esté sintácticamente bien, las operaciones ilógicas no son detectadas sin llevar a cabo la ejecución.

4.2.8 Casos de éxito

Para hacer una buena estimación de como de bueno es un lenguaje, es conveniente analizar cuál es su cartera de usuarios y qué tipo de usuarios son, es decir, cuál es el fin con el que se utiliza dicho lenguaje. A continuación se analizan brevemente cuáles son algunos de los usos que se le da al lenguaje de programación Pyhton hoy en día.

Uno de los usos actuales de Python que indica lo potente que es, es Google. Este gigante de internet y de los sistemas informáticos, trabaja con una enorme cantidad de algoritmos de búsqueda y para muchos de estos algoritmos se ha empleado Python. Cómo se comenta en anteriores apartados de este proyecto, el creador de Python acabó trabajando para esta gran empresa. Sin embargo Google no es la única empresa que hace un uso comercial de Python, lo

acompañan otras grandes empresas como: Industrial Light and Magic (ILM), Facebook, Spotify, Dropbox, Yahoon, etc. El uso de Python está muy extendido en el análisis datos y la extracción de información útil para empresas.

Otro ejemplo del éxito de Python en la actualidad es el uso del mismo para el desarrollo de productos de software libre. De hecho, este lenguaje surgió a raíz del fenómeno de 'software libre' que ha tenido cada vez más influencia en la última década. Un buen ejemplo de esto es la distibución de Linux Ubuntu, que usa Python para escribir la mayoría de sus aplicaciones. Otros ejemplos del uso de Python en software libre son: Gimp y Blender.

Además del ámbito comercial y el software libre, Python es muy popular en el ámbito educacional. Como se ha descrito anteriormente en este proyecto, este lenguaje se ha concebido basándose en la sencillez, la limpieza y principalmente, la facilidad de su aprendizaje. Muchas universidades lo utilizan hoy en día para impartir asignaturas relacionadas con el mundo de la programación. Además, se desarrollan muchos proyectos docentes utilizando este lenguaje debido a su popularidad.

Por último, cabe destacar la importancia del uso de este lenguaje de programación en la ciencia. A la hora de resolver un problema de cualquier ámbito de la ciencia, se evita que la manera de resolverlo suponga un aditivo de dificultad, es por esto que Python es el lenguaje por excelencia elegido para este fin ya que se trata de una herramienta simple y eficiente. Las herramientas que este enguaje proporciona, como son las estructuras de datos, las librerías gráficas o el análisis de datos, hacen que la tendencia del uso de Python en los centros de investigación esté en pleno crecimiento, así como en otras ramas de la ciencia tales como: Neurofisiología, Informática, Física, Matemáticas, etc. Hay en particular dos paquetes de Python que han cobrado gran importancia en el mundo de la ciencia, estos son el NumPy y el SciPy que se refieren al Python numérico y científico respectivamente.

La tendencia del mercado laboral muestra que la demanda de profesionales de Python crece exponencialmente cada año. Esta tendencia se ve reflejada en un famoso artículo de la web Stackoverflow, que es una fuente de información electrónica muy popular entre los desarrolladores de todo el mundo, que ha presentado la evolución del volumen de búsquedas por tipo de lenguaje. En estas gráficas se puede distinguir Python como el lenguaje que encabeza las listas a día de hoy (Robinson, 2017).

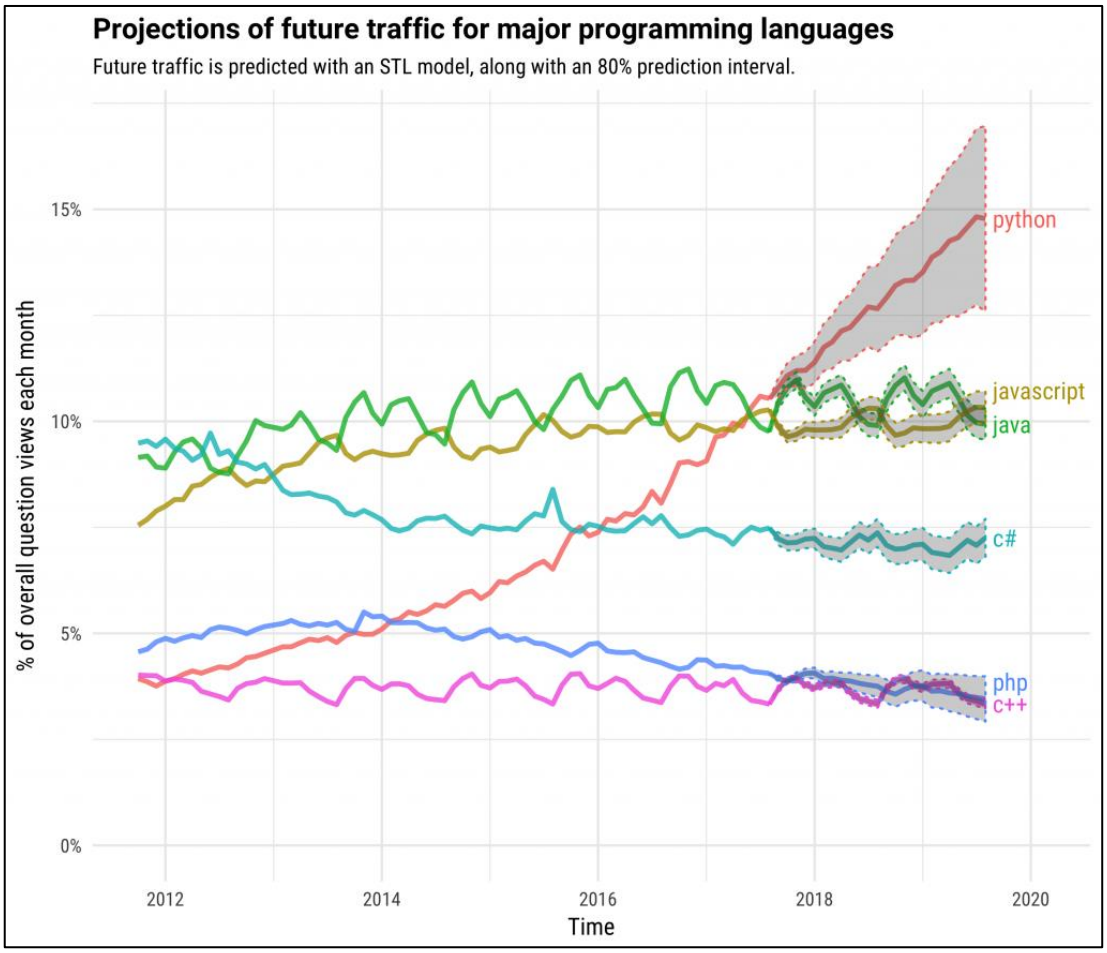


Figura 16: Tráfico de los grandes lenguajes de programación.

Fuente: The Incredible Growth of Python (2017).

4.3 Introducción a los solver de optimización

4.3.1 Optimización y programación Matemática

La optimización matemática consiste en la agrupación de distintas técnicas de modelado matemático que hacen posible hallar soluciones para problemas. Los ejemplos más comunes de estos problemas para los que se usa optimización son aquellos en los que se pretende hallar una asignación o planificación óptima de recuersos limitados. La optimización matemática, también conocida como la investigación operativa, trata en general de dar apoyo, de una manera eficiente y lo más rápida posible, al proceso de toma de decisiones.

Esta 'investigación operátiva' se utiliza para resolver problemas complejos a través de modelos matemáticos. Los fines de la resolución de estos modelos suelen ser: reducir costes, minimizar tiempo o distancias recorridas, maximizar beneficios, etc. Las soluciones optimas a estos

problemas se enuentran en el punto máximo de equilibrio en el que se cumplen una serie de requisitos predeterminados.

Estás técnicas de optimización se encuentran en plena expansión, aumentando su popularidad cada día debido a los grandes beneficios que puede aportar su implementación. Las situaciones que pueden afrontarse con la programación matemática se suelen presentar en ingeniería, empresas comerciales y en ciencias sociales y físicas. Uno de estos beneficios, entre otros, es el gran valor de negocio que puede aportarle a una empresa, viéndose esto relfejado directamente en el resultado financiero de la misma. Esto es debido a que con la optimización matemática se pueden optimizar costes, incrementar ventas o de una manera u otra, maximizar márgenes. La creciente complejidad de los problemas que se resulven mediante esta disciplina y la eficencia en los resultados obtenidos, han contribuido de manera enormemente a su incremento de popularidad.

Los primeros modelos en los que se trabajó y por lo tanto los que más desarrollados han sido a lo largo del tiempo en el ámbito de la programación matemática es el modelo lineal. Este tipo de programación, la lineal, estudia la manera de optimizr una función lineal de manera que la solución satisfaga una serie de restricciones lineales que pueden ser de igualdades o desigualdades. Fue un consejero de los controladores de la Fuerza Aérea de los Estados Unidos, George B. Dantzig quien en el año 1947 desarrolló el método simplex para resolver este tipo de problemas. (Universitat Politecnica de Valencia, 2012) El procedimiento de este método de resolución de problemas lineales, permite mejorar las respuestas paso a paso, con el fin de alcanzar la solución óptima de un problema.

Por otro lado, la programación entera consiste en problemas de optimización en los que algunas o todas sus variables de decisión se restringen a un conjunto de valores discretos. Este tipo de programación permite incorportar a los modelos algunos aspectos que van más allá de lo permitido en la programación lineal. En este sentido los algoritmos de resolución de los modelos de Programación Entera son más completos que modelos de Programación Lineal. Entre la lista de este tipo de programación cabe destacar: el Algoritmo de Ramificación y Acotamiento (o Branch & Bound), Branch & Cut, Planos Cortantes, Relajación Lagrangeana, entre otros. Dentro de la programación entera se puede establecer una división entre dos ramas diferentes: la programación entera mixta (PEM) y la programación entera pura (PEP).

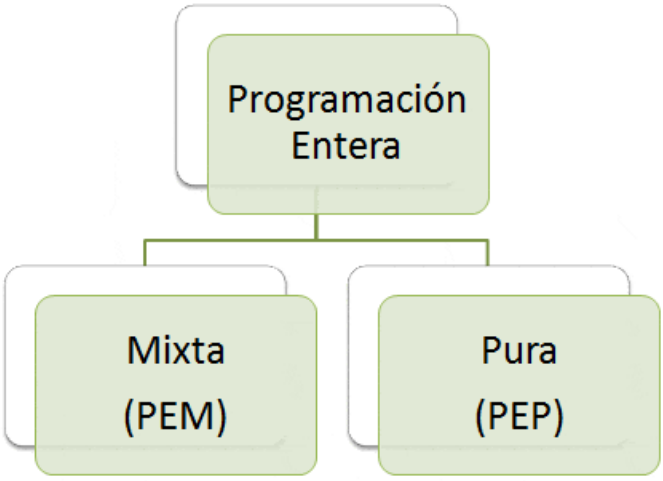


Figura 17: Ramas de la Programación Entera.
Fuente: Web sobre la Gestión e Investigación de Operaciones (2016).

4.3.2 Modelos de optimización en Python

Hoy en día, muchos en la industria necesitan realizar planificaciones y tomar decisiones óptimas de manera regular como parte de sus operaciones semanales, diarias o incluso horarias. Los recientes avances computacionales nos han proporcionado la infraestructura para incorporar modelos de optimización para soluciones de software analítico. Esto significa que los profesionales dedicados a la investigación operativa hoy en día necesitan diseñar, modelar e implementar motores de software robustos basados en modelos LP/MILP. Necesitan utilizar un lenguaje de programación como C++, Java, C#, Python, etc. para ello.

Uno de los mejores lenguajes de programación, más popular y por tanto el recomendado por muchos de los expertos en investigación operativa y en las comunidades de la ciencia de datos es Python. Es fácil de aprender, flexible y potente, y cómo se ha mencionado con anterioridad, dispone de una amplia variedad de bibliotecas para el Machine Learning, la optimización y el modelado estadístico.

A continuación se muestra una breve calsificación de las librerías de Python que resultan de gran utilidad a la hora de enfrentarse a un problema de optimización:

Tabla 2: Aplicación de librerías de Python.

Librería de Python	Aplicación
PuLP	Lenguaje para optimización programación lineal. Requiere solver adicional.
Scipy.optimize	Rutinas numéricas para optimización problemas no lineales.
Pyomo	Lenguaje para optimización. Requiere solver adicional.
CVXopt	Optimización convexa.

Muchos solvers de optimización, tanto comerciales cómo de código abierto, usan interfaces Python para modelar LPs, MILPs y QPs. Algunos de los solvers comerciales más populares son CPLEX y Gurobi, y un ejemplo de paquete de modelado de código abierto en Python que es muy potente es PuLP. Con el debido estudio de las estructuras de estos solvers, es relativamente fácil traducir cualquier problema de un modelo a otro (Moarefdoost, 2018b).

4.3.3 Solver de optimización: Gurobi

Gurobi fue diseñado desde sus inicios para ser el solver más rápido y potente disponible para problemas de LP, QP, QCP, y MIP (MILP, MIQP, y MIQCP). Su código fue construido para explotar al máximo el paralelismo. No se trata de un código secuencial que fue paralelizado, sino un código fundamentalmente paralelo que presenta la opción de ser ejecutado secuencialmente. Los desarrolladores de este solver han ido más allá de las versiones de vanguardia de todos los planos de corte estándar y han desarrollado nuevas clases de cortes que sólo se encuentran disponibles en Gurobi. Este solver trabaja con una potente heurística MIP avanzada que es capaz de encontrar en un breve periodo de tiempo soluciones factibles, es decir, proporciona soluciones de buena calidad para problemas que otros solvers son incapaces de resolver. Sus algoritmos de barrera aprovechan al máximo las características que presentan las últimas arquitecturas informáticas. Y sus APIs están diseñadas para ser ligeras, modernas e intuitivas, con el fin de minimizar su curva de aprendizaje y maximizar su productividad.

Se trata de un software de optimización escrito en C y que es accesible desde diferentes lenguajes de programación. Además de otras características que lo hacen destacar y de las que se hablará más adelante, posee una interfaz potente e interactiva de Python, una manera simple de ejecución en línea de comandos. Este solver aprovecha al máximo la evolución de las matemáticas y las mejoras en las metodologías de optimización, así como los avances en el hardware moderno de computación de escritorio y entornos de programación.

Trabajar con solvers como Gurobi es muy útil cuando se quiere resolver problemas del mundo real en el que un problema a gran escala debe dividirse en problemas más pequeños y manejables que, a su vez, se resuelven de forma secuencial. La solución de un problema se convierte en la entrada a otro problema, y así sucesivamente. Con esta metodología de trabajo, se consigue resolver el problema grande de una manera mucho más eficiente. Este enfoque algorítmico no sólo requiere experiencia en modelado avanzado, sino también un solver robusto sobre el cual construir el modelo. Para resolver la mayoría de los problemas de la vida real con modelos en Gurobi, se requieren muchas entradas, requisitos y restricciones interconectadas entre sí. Además, es aconsejable construir los modelos de manera que presenten la máxima felxibilidad para adaptarlos a posibles cambios a lo largo del tiempo. Gurobi es un solver comercial con la potencia y estabilidad necesarias para manejar problemas de gran tamaño, alcance y complejidad. Las clases, funciones y estructuras de datos de Gurobi como column, tuplelist y quicksum, no sólo ayudan en la implementación de los modelos, sino que también mejoran el rendimiento general del mismo (Moarefdoost, 2018a).

A continuación se listan algunas de las características principales de este software de optimización:

- Una API de Python que proporciona las ventajas de un lenguaje de modelado y un lenguaje de programación completo.
- Soporte para modelos en los que compiten múltiples objetivos y dos opciones para resolverlos.
- La capacidad de resolver de manera directa modelos no-lineales subdivididos en modelos más pequeños con funciones objetivos lineales, para capturar mejores soluciones para dicho problema no lineal.
- La capacidad de expresar construcciones de modelado comunes como min/max o si/entonces a un nivel superior, haciendo que tales modelos sean más fáciles de construir y mantener.
- Potentes y flexibles capacidades cliente-servidor y de nube para ayudar a desarrollar los trabajos dónde y cómo se desee, para cualquier número de usuarios en una amplia gama de plataformas.

Este proyecto ha sido desarrollado haciendo uso del solver comercial Gurobi, y en particular, se ha trabajado con la versión 8.1.0 del mismo. Para el uso de este software de optimización se ha adquerido una licencia académica que permite hacer uso del mismo sin limitaciones de relevancia para este problema en cuestión.

En definitiva, se ha llevado a cabo la resolución del modelo de agrupación de pedidos y rutas de recogida a través del entorno de desarrollo Spyder, codificando con el lenguaje Python y llamando al solver Gurobi para la optimización del modelo. Para ello, en primer lugar, es necesario vincular Gurobi con Anaconda, que es la distribución de la que proviene Spyder. Para llevar a cabo esta conexión en un ordenador con Windows que es donde se desarrolla este proyecto, desde una terminal de Anaconda se ejecutan los comandos necesarios para vincular el paquete gurobi:

1. Añade el canal de Gurobi a la lista de canales predeterminados:

```
conda config --add channels http://conda.anaconda.org/gurobi
```

2. Instalar el paquete de Gurobi:

```
conda install gurobi
```

5 MODELADO DEL PROBLEMA

Cómo se ha mencionado anteriormente, el problema que se desea estudiar es el de agrupación de pedidos y definición de la ruta del recorrido del picker en un almacén. Este problema se presenta como un problema complejo, ya que su tamaño aumenta exponencialmente según el tamaño del almacén en cuestión. En nuestro caso, se trabaja con datos reales provenientes de una base de datos que recoge las compras electrónicas de un establecimiento, por tanto se manejan unos datos de tamaño considerable. Además, la mayoría de las variables que intervienen en el problema en cuestión son enteras, lo que aumenta a su vez su complejidad en cierto modo.

El modelo en el que se basa este proyecto es un modelo de optimización mediante grafo. El modelado del problema y la notación seguida han sido extraidos de (Valle, Beasley and da Cunha, 2016) al cuál se le han intriducido pequeñas modificaciones. Este modelo matemático, que se explicará en profundidad a continuación, pretende agrupar los diferentes pedidos en lotes (carros) para calcular a su vez la ruta óptima que debe recorrer cada carro asegurando que todos los pedidos son recogidos y minimizando además la distancia recorrida.

El modelo formulado posteriormente será resuelto por el solver Gurobi, que usa Python 2.7 como lenguaje de programación.

5.1 Notación del modelo matemático

Como se ha mencionado en apartados anteriores, el problema que se estudia consiste en buscar las rutas cerradas que son necesarias para visitar todas las localizaciones en las que se almacenan productos demandados. Para el problema en cuestión, se supone que la recogida de pedidos se lleva a cabo mediante una metodología "picker-to-parts" en la que es el operario el que se mueve por el almacén para alcanzar los productos. Por esto, en este caso se supone que el mayor "coste" del problema de la definición de la ruta lo determina la distancia recorrida, por lo que se busca minimizar dicho parámetro.

Para la correcta formulación del problema se deben tener una serie de consideraciones:

i. Se dispone de un número suficiente de carros para recoger todos los pedidos, y cada carro a su vez se compone de un número determinado de cestas.
ii. Mezclar y dividir pedidos ahorra tiempo a la hora de recoger pedidos pero aumenta de manera exponencial la cantidad de errores acarreados.

iii. Los pedidos pueden llegar a alcanzar grandes cantidades de productos sin llegar en ningún caso a superar la capacidad de un único carro. Por tanto, debido a la cantidad de errores que genera en la recogida de pedidos, no se permitirá dividir un pedido en varios carros.

iv. Por el mismo motivo mencionado en el punto anterior, se evitará también mezclar diferentes pedidos en una misma cesta. Esto último no limita el uso de un mismo carro para distintos pedidos siempre y cuando su capacidad lo permita y los productos correspondientes a cada pedido se almacenen en sus correspondientes cestas.

v. Se consideran almacenes convencionales, compuestos de pasillos rectos con huecos de almacenamiento (slots) a ambos lados del pasillo.

vi. Dichos huecos de almacenamiento se apilan a su vez verticalmente en distintas alturas, es decir, en estanterías de almacenaje.

vii. El número de cestas por cada carro es limitado y conocido, así como el número de cestas necesarias para recoger cada pedido.

viii. Cada picker puede llevar a la vez un único carro, por lo que podemos asumir que picker ≡ carro.

ix. No se tendrán en cuenta limitaciones de espacio o peso, por lo que todas las cestas tenfran una capacidad fija, independientemente de la forma o tamaño de los productos.

x. El picker puede visitar las posiciones más de una vez, pero deberá llegar a dicha posición por distintos caminos cada vez que se visite.

Antes de definir la función objetivo y las retricciones que forman nuestro problema, es necesario posentar la notación que se usa en el entorno del modelo. Esta notación expondrá, tanto los conjuntos de datos que determinan el problema, como las variables que se emplean para resolverlo.

5.1.1 Conjunto de datos del problema

La cantidad de datos que maneja un problema esta fuertemente relacionado con las dimensiones del mismo, en el caso de un almacén de productos alimentarios como el que se presenta, las cantidades pueden ser muy variables y por tanto se podrá trabajar con distintos volúmenes de datos. Se podrán utilizar las instancias de menor tamaño para simplificar las simulaciones en el solver y después aplicarlo a instancias de mayores dimensiones.

Se considera dato todo aquello que define el problema que se pretende abordar. Para el problema trabajado se definen los siguientes datos y conjuntos de datos:

- **Conjunto P:** hace referencia al conjunto de productos existentes en el almacén. Todos ellos tienen un espacio de almacenamiento asignado y conocido.

- **Conjunto L:** hace referencia al conjunto de localizaciones (posiciones en el centro del pasillo) desde las cuales el picker puede alcanzar una serie determinada de productos ubicados a ambos lados.

  En el grafo que define el problema, los nodos serán un conjunto compuesto de dicho conjunto de localizaciones L además de:

  - El nodo origen, representado como {s}, que será punto de partida y de retorno para todos los carros utilizados.
  - Una serie de localizaciones artificiales L_A, que conectan los nodos del conjunto L para formar el layout del almacén pero desde los cuales no se alcanza ningún producto. Representa los cruces de pasillos y/o subpasillos.

  Los nodos, también denominados vértices, que componen el grafo son los siguientes
  $V = \{L, L_A, s\}$

- **Conjunto Pos:** hace referencia al conjunto de posiciones de almacenamiento disponibles (slots), dónde se almacenan, o no, productos. Desde cada localización L se podrán alcanzar un subconjunto de posiciones. Cómo se ha mencionado en la definición del conjunto P, cada producto tiene asignado su posición de almacenaje y son previamente conocidas.

- **Conjunto O:** hace referencia al conjunto de órdenes que se reciben por parte de los clientes y de las cuales deben recogerse todos los productos que las componen. Se representará mediante el índice 'o', es decir, P_O será el subconjunto de productos P que componen el pedido o, y a su vez L_O representará el subconjunto de localizaciones L desde las que se alcanzan todos los P_O.

- **Dij:** hace referencia a la distancia que hay entre el nodo i y el nodo j del grafo, es decir, distancias entre posiciones consecutivas del almacén. Esta matriz de datos es la que define el Layout del almacén. Se trata de una matriz simétrica, ya que para cada arco i-j existe otro arco j-i tal que dij ≡ dji. Todas las distancias han de ser positivas o cero en caso de que no exista conexión entre los nodos implicados.

- **Parámetro B:** Capacidad de los carros, es decir, el número máximo de cestas que pueden almacenar productos para un mismo recorrido, carro o picker. Es conocido y fijado al inicio del problema.

- **Capacidad cestas:** Número máximo de productos por cestas. Es conocido y fijado al inicio del problema. Junto con el parámetro B definen la capacidad total de productos que tienen los carros:

$$Capacidad\ Total\ carros = B * Capacidad\ cestas$$

- **Conjunto T:** hace referencia al conjunto de carros o pickers disponibles. Se representa mediante el índice 't'. En nuestro caso se define de la siguiente manera:

$$T = \frac{\sum_{o \in O} b_o}{B} + 0{,}2$$

Además, de los conjuntos previamente definidos se obtiene una serie de datos adicionales como son los siguientes:

- Número de pedidos a recoger.
- El código de identificación de cada producto demandado dentro de cada pedido.
- Las cantidades que se demanda de cada producto dentro de cada pedido.

5.1.2 Variables que intervienen en el problema

Como en todo problema matemático, las variables del modelo serán aquellas cuyo valor podrá variar dando lugar a una solución u otra según los criterios establecidos. Para el problema de agrupación de pedidos y definición de la ruta del picker en el almacén, el número de variables involucradas, al igual que los datos, aumentará según aumente el tamaño del almacen y/o pedidos a satisfacer. Una vez conocido el valor de todas las variables se puede conocer la solución exacta del problema estudiado. Hallar la solución óptima a tal problema no es computacionalmente fácil. Conforme aumenta el tamaño del mismo, crece considerablemente el número de posibles soluciones y, por tanto, el tiempo requerido para resolverlo.

Las variables involucradas en el problema que se estudia son las que se definen a continuación:

- **Zot:** se trata de una variable binaria. Su valor en la solución determina el carro que recoge el pedido. Si esta variable Zot vale 1, es decir, se activa, indicará que el pedido 'o' es asignado al carro 't'. Si por el contrario el valor que toma la varible es 0, es decir, no se activa, el pedido 'o' no será asignado al carro 't'.

- **Xtij:** se trata de una variable binaria. Su valor en la solución determina la ruta que realiza cada carro. Si esta variable Xtij vale 1, es decir, se activa, indicará que el carro 't' recorre el arco descrito por los nodos consecutivos i-j, donde 'i' representa el nodo de inicio y 'j' el nodo de salida. Si por el contrario el valor que toma la variable es 0, es decir, no se activa, el arco i-j no formará parte del recorrido del carro 't'.

- **Yti:** se trata de una variable binaria. Su valor en la solución determina los nodos visitados por cada carro. Si esta variable Yti vale 1, es decir, se activa, indicará que el carro 't' visita el nodo 'i'. Si por el contrario el valor que toma la variable es 0, es decir, no se activa, el nodo 'i' no será ninguno de los nodos visitados por el carro 't'.

- **Etij:** Se trata de una variable binaria. Se ha definido de manera auxiliar para indicar que un carro recorre el camino entre dos nodos consecutivos i-j, sin tener en cuenta en qué dirección se realice el movimiento. Si esta variable Etij vale 1, es decir, se activa, indicará que el carro 't' recorre el arco i-j y/o el arco j-i. El orden de los subincides de la variable no indican en este caso el orden de visita.

- **Gti:** se trata de una variable entera. Su valor en la solución determina el grado de cada nodo para el recorrido de cada carro. El grado de un nodo o vértice, tal y como se muestra en la siguiente imagen, indica el número de arcos parten de él. El valor entero que tome la variable Gti indicará el número de arcos que recorre el carro 't' con nodo de inicio 'i'. Por ejemplo, si G_{12} toma el valor 3, el **carro 1** recorrerá tres arcos que parten del **nodo 2** hacia otros nodos.

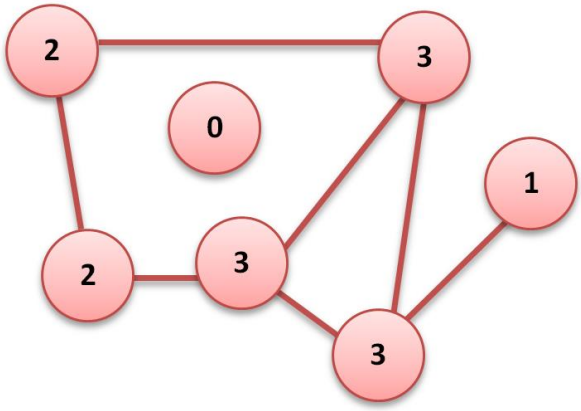


Figura 18: Un grafo con vértices etiquetados según su grado.

Fuente: Elaboración propia.

5.2 Formulación del modelo matemático

En el modelo matemático se utilizan fórmulas matemáticas para representar la relación entre distintas variables, parámetros y datos definidos anteriormente. De manera general, un modelo matemático se compone de una función objetivo y una serie de restricciones a cumplir. La función objetivo determina la finalidad del problema y las restricciones definen las limitaciones a tener en cuenta o los requisitos que se quiere que cumplan las soluciones obtenidas en cada caso.
A continuación se describen estas relaciones matemáticas con el fin de describir el modelo que se estudia en mayor detalle.

El código en Python 2.7 que se ha generado basado en estas relaciones y que se ha utilizado para resolver el problema mediante el solver Gurobi, se encuentra en el anexo.

5.2.1 Restricciones

1. Como se ha definido en apartados anteriores, uno de los datos del problema es la limitación de capacidad que tienen los carros/pickers que se identifica cómo el parámetro B. Además, una de las variables que formará parte de la solución final del problema es $Z_{o\,t}$, que en caso de activarse, indicará que el pedido 'o' es recogido por el carro 't'. La restricción que se muestra a continuación indica que para cada carro/picker, la suma de todas las cestas necesarias para recoger todos los pedidos que este carro recoja, no debe superar el número máximo de cestas permitidas para un único carro. Esta relación se expresa matemáticamente de la siguiente manera:

$$\sum_{o \in O} b_o * Zot \leq B \qquad \forall t \in T$$

2. Una de las consideraciones mencionadas a la hora de definir el problema establece que, para evitar errores, no se debe dividir un pedido en varios carros. Es decir, todas las cestas pertenecientes a un mismo pedido, no deben necesitar más de un carro para ser recogidas. Además, con esta restricción se fuerza la parte más importante de la solución del problema, que indica que todos los pedidos deben ser recogidos por algún carro. Esta relación se expresa matemáticamente de la siguiente manera:

$$\sum_{t \in T} \boldsymbol{Z_{ot} = 1} \qquad \forall o \in O$$

3. Como se ha explicado en apartados anteriores, el problema que se estudia se representa como un grafo D = {V, A}. En éste, los nodos representan las localizaciones del almacén desde las cuales se pueden alcanzar una serie de productos almacenados o los cruces entre pasillos y los arcos la unión entre estas localizaciones, que estarán definidos como 0 o positivos según el layout del almacén. Lo que se quiere asegurar con esta restricción es que si un pedido está asignado a un carro ($Z_{o\,t}$ activa) entonces cada nodo que contiene productos de ese pedido deberá ser visitado al menos una vez por el carro 't'. Esta relación se expresa matemáticamente de la siguiente manera:

$$\sum_{(i,j) \in i^+} \boldsymbol{X_{tij} \geq Z_{ot}} \qquad \forall o \in O, \quad t \in T, \quad i: l(i) \in L_o$$

4. Se ha definido anteriormente que para cada arco ij existe un arco ji, por tanto, aunque se alcance un nodo 'i' desde el nodo 'i-1' y se pretenda recorrer el camino inverso, es decir, regresar al nodo 'i-1', se recorrerán arcos diferentes. De este modo, un carro no recorrerá un mismo arco más de una vez. Para regularizar la continuidad de flujo en el grafo, se establece la siguiente restricción que asegura que cada carro que alcance un nodo i por alguno de sus arcos contenidos en i-, debe abandonar el mismo por alguno de los arcos pertenecientes a i+. Esta relación se expresa matemáticamente de la siguiente manera:

$$\sum_{(i,j) \in i^+} \boldsymbol{X_{tij}} = \sum_{(j,i) \in i^-} \boldsymbol{X_{tji}} \qquad \forall i \in V, \quad t \in T$$

5. Uno de los requisitos que deben cumplirse a la hora de la recogida de pedidos, es que los pickers deben comenzar su ruta desde la posición origen, definida como {s} y

deberán terminar su ruta en la misma posición. Por tanto, esta restricción indica que si un carro 't' recoge un determinado pedido 'o', es decir $Z_{o\,t}$ se activa, se tiene que activar un arco de salida del origen s para este carro. Esta relación se expresa matemáticamente de la siguiente manera:

$$\sum_{(s,j)\in s^{+}} X_{tsj} \geq Z_{ot} \qquad \forall t \in T, \qquad o \in O$$

6. Se ha definido el grado de un nodo como el número de arcos parten de él. Esta es una de las variables enteras del problema que se estudia por lo que será necesario una restricción que mediante su relación con otras varibles permita definir su significado. Por esto, se establece la siguiente igualdad que indica que la suma de los arcos de salida de un nodo 'i' que son recorridos por el carro 't', equivaldrá al grado de dicho nodo para ese carro en cuestión. Esta relación se expresa matemáticamente de la siguiente manera:

$$\sum_{(i,j)\in i^{+}} X_{tij} = G_{ti} \qquad \forall i \in V, \qquad t \in T$$

7. Otra de las relaciones entre variables del problema que debe establecerse es que si un carro recorre un arco, por fuerza ese carro ha debido visitar el nodo del que parte. Es decir, si la variable $X_{t\,i\,j}$ se activa, deberá activarse también $Y_{t\,i}$ ya que implica que el carro 't' visita el nodo 'i' y en caso contrario sería imposible que $X_{t\,i\,j}$ se activase. Esta relación se expresa matemáticamente de la siguiente manera:

$$Y_{ti} \geq X_{tij} \qquad \forall (i,j) \in A, \qquad t \in T$$

8. La restricción 7 hace necesario que si un arco se recorre se tenga que visitar el nodo de partida del arco, sin embargo, esto no es suficiente para definir la variable $Y_{t\,i}$, ya que con sólo esa restricción se permitiría que $Y_{t\,i}$ se activase aunque no se recorriera ningún arco que saliese de ella. Por eso se ha de establecer una relación más, que además de afinar el valor que tome la variable en cuestión, permitirá que se recorran sub-rutas, más conocidas como 'ciclos' en la teoría de grafos. Para que se dé este caso, al menos uno de los nodos visitados por el caro 't' deberá tener un grado superior a 1. Esta relación se expresa matemáticamente de la siguiente manera:

$$\sum_{j \in W} G_{tj} \geq Y_{ti} + \sum_{(j,k) \in A(W)} X_{tjk} \qquad \forall i \in W, W \subseteq V \backslash \{s\}, |W| > 1, \qquad t \in T$$

5.2.2 Función Objetivo

La función objetivo, es la función lineal que se va a maximizar o minimizar en el proceso de optimización de un modelo matemático.

En el caso del problema de agrupación de pedidos por lotes y definición de la ruta del picker en el almacén, se trata de encontrar la colección de carros/pickers óptima T* perteneciente al conjunto de carros disponibles T, que mediante recorridos cerrados sea capaz de recoger todos los pedidos cumpliendo con las restricciones previamente definidas y con el último fin de minimizar la distancia total recorrida. Esta relación se expresa matemáticamente de la siguiente manera:

$$MIN \left\{ \sum_{t \in T} \sum_{(i,j) \in A} Dij * Xtij \right\}$$

5.3 El modelo completo

F.O

$$MIN\left\{\sum_{t\in T}\sum_{(i,j)\in A} Dij * Xtij\right\}$$

s.a:

$$\sum_{o\in O} b_o * Zot \leq B \qquad \forall t \in T$$

$$\sum_{t\in T} Z_{ot} = 1 \qquad \forall o \in O$$

$$\sum_{(i,j)\in i^+} X_{tij} \geq Z_{ot} \qquad \forall o \in O, \quad t \in T, \quad i: l(i) \in L_o$$

$$\sum_{(i,j)\in i^+} X_{tij} = \sum_{(j,i)\in i^-} X_{tji} \qquad \forall i \in V, \quad t \in T$$

$$\sum_{(s,j)\in s^+} X_{tsj} \geq Z_{ot} \qquad \forall t \in T, \quad o \in O$$

$$\sum_{(i,j)\in i^+} X_{tij} = G_{ti} \qquad \forall i \in V, \quad t \in T$$

$$Y_{ti} \geq X_{tij} \qquad \forall (i,j) \in A, \quad t \in T$$

$$\sum_{j\in W} G_{tj} \geq Y_{ti} + \sum_{(j,k)\in A(W)} X_{tjk} \qquad \forall i \in W, W \subseteq V\backslash\{s\}, |W| > 1, t \in T$$

6 IMPLEMENTACIÓN Y RESULTADOS OBTENIDOS

6.1 Detalles de la Implementación

El alcance inicial de este proyecto era la implementación de este modelo en un gran almacén de productos de un supermercado on-line de tamaño real. A continuación se describe el escenario en cuestión.

Los datos que hacen referencia a los productos y pedidos, como se ha mencionado en apartados anteriores, proceden de una base de datos reales disponible en la red. La base de datos a componen pedidos de comprasonline anónimas de un periodoaproximado de más de dos años. Los clientes de los cuales se han recogido estos datos son clientes de una cadena de supermercados. Cuenta con un total de 1560 productos distintos, separados en clases de productos que contienen 4 diferentes niveles de categorías. También contiene aproximadamente 2.700.000 pedidos para el período 1997-1998, cada uno de ellos muestra el detalle de la lista de productos distintos solicitados, la cantidad pedida para cada producto, así como otros detalles como la fecha de realización del pedido y el identificador del cliente en cuestión. Todos estos datos provienen originalmente de la base de datos: MySQL Foodmart Database. La lista de productos con la que se trabaja en este proyecto se encuentra en el Anexo.

Los datos se encuentran compilados en diferentes archivos que muestran más o menos detalle según el uso que se le pretenda dar a la información. De manera general, los datos vienen recopilados en los siguientes archivos:

1) ProductsDB_1560_list: Un archivo que contiene la lista de 1560 productos, incluyendo el nombre y la categoría a la que corresponden cada uno de ellos.
2) ProductsDB_1560_locations: Un archivo que contiene un mapa de productos a las ubicaciones del almacén.
3) Una recopilación de archivos que combinan diferentes ordenes para crear una amplia variedad de instancias con las que poder trabajar. Los archivos se llaman: instances_dXX_ordYY, donde XX es el número de días que se fusionan para producir una mayor y YY son el número de órdenes en el archivo. De este modo, una instancia llamada: instances_d5_ord5, será una recopilación de datos de 5 días de compras online y los muestra como 5 pedidos diferentes.

Para los datos relativos al layout del almacén, distribución de productos en el mismo así como cantidad de posiciones de almacenaje, se ha hecho uso de un generador desarrollado por (Valle, Beasley and da Cunha, 2016). Este generador crea almacenes a gusto del usuario, que deben ser capaces de contener un número mínimo predeterminado de productos distintos. Para la generación de estos almacenes, en primer lugar se debe asignar un número fijo de pasillos, pasillos transversales y alturas de estanterias. También se dan longitudes arbitrarias, en metros, para la anchura de pasillos y pasillos transversales, así como para la profundidad de la estantería y la anchura de las ranuras y la distancia desde el origen hasta su vértice artificial más cercan.En definitiva, se dan los datos suficientes para crear una matriz de distancias de un almacén completo.

El almacén más complejo que este generador es capaz de generar es el que está dimensionado para almacenar los 1560 productos distintos que la base de datos original contempla. El archivo generado para este almacén, puede darse según las siguientes características: warehouse_X_Y_Z_XXXX, donde:

- X indica el número de pasillos.
- Y indica el número de pasillos transversales.
- Z indica el número de alturas que presentan las estanterías.
- XXXX indica el número de productos almacenados.

Tras extraer los datos para el almacén más complejo y crear la matriz de distancias y posiciones alcanzables, se obtuvo una representación gráfica del almacén cómo se muestra a continuación.

Warehouse_8_1_3_1560: Almacén compuesto por 8 pasillos, 1 pasillo transversal adicional (es decir, 3 pasillos transversales en total), 3 alturas de estantería y con capacidad para almacenar al menos 1560 productos. Esto implica, un total de 289 vértices de los cuales 264 son posiciones a visitar desde las que se alcanza productos, 1 es el origen del picker y 24 representan los cruces (8 pasillos x 3 pasillos transversales). Según esto, este almacén tiene capacidad para almacenar 1584 productos (264 posiciones x 2 estanterias, una a cada lado de la posición x 3 alturas). Además, el generador establece las siguientes distancias:

- Distancia entre dos posiciones de almacenamiento consecutivas: 1 metro.
- Distancia entre posición de almacenamiento y posición artificial: 2 metros.
- Distancia entre dos posiciones artificiales: 5 metros.

- Distancia entre el origen y las posiciones artificiales alcanzables desde este: Proporcionales.

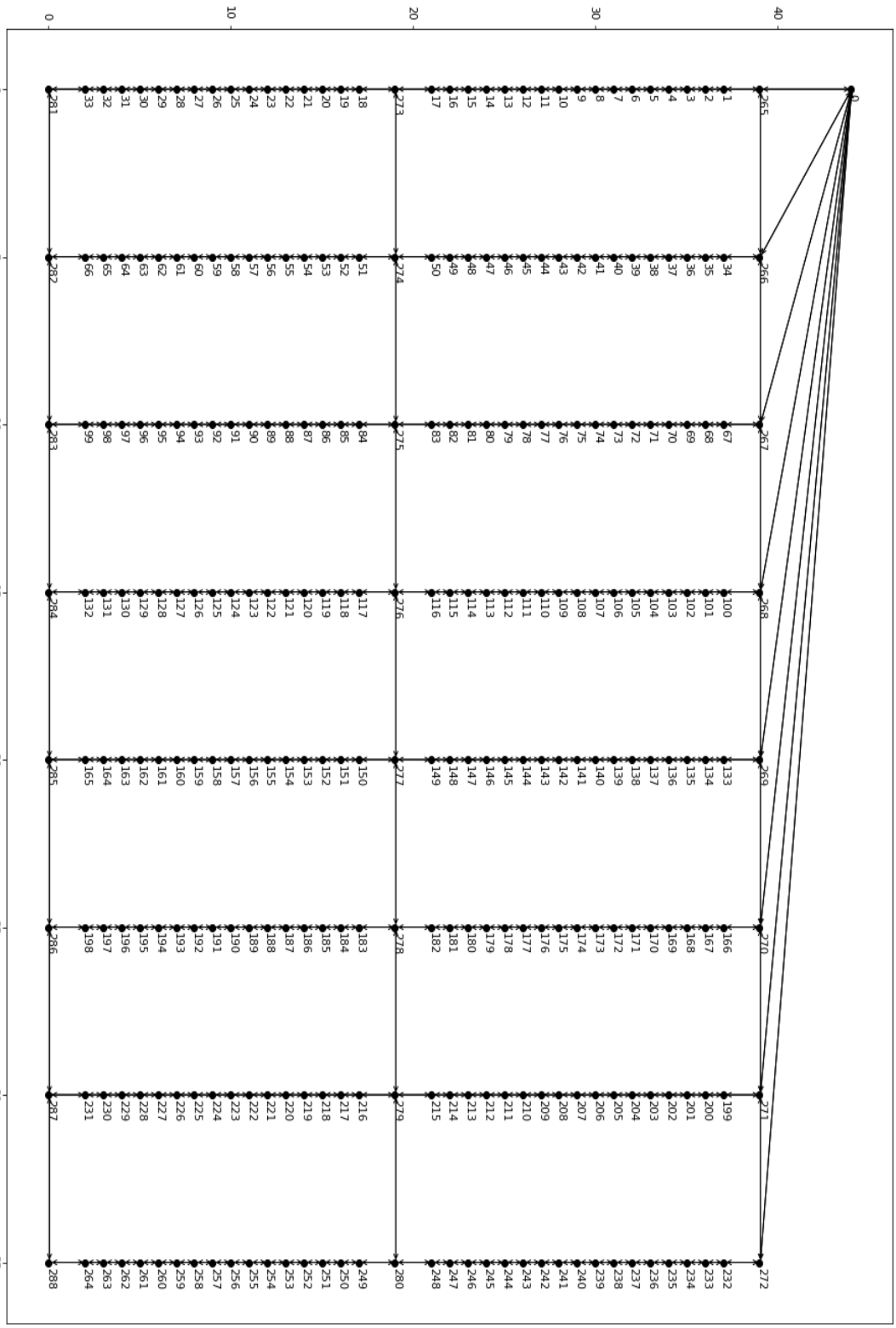


Figura 19: Representación gráfica en Python de la mayor matriz de distancias disponible.

Fuente: Elaboración propia.

Sin embargo, debido a las limitaciones de capacidades de los equipos con los que se ha trabajado y a las grandes dimensiones de este almacén, ha resultado imposible resolver dicho problema sin dar lugar a errores de memoria. El principal motivo de esta limitación radica en la restricción [8] del modelo matemático que se pretende implantar para resolver este problema. Esta restricción es la siguiente:

$$\sum_{j \in W} G_{tj} \geq Y_{ti} + \sum_{(j,k) \in A(W)} X_{tjk} \qquad \forall i \in W, W \subseteq V \backslash \{s\}, |W| > 1, \qquad t \in T$$

Donde, como su expresión indica, W hace referencia a todos los subconjuntos posibles del conjunto total de vértices V. Es decir, W es el conjunto potencia de V. Por definición, el conjunto potencia de un conjunto de n elementos tendrá 2^n elementos, por tanto, para un almacén de 289 vértices como el que se pretende resolver, esta restricción estará elevada a 2^{289}= 9,9465x10^{86}. Esta cantidad de datos es intratable sin trabajar con equipos especiales ni en un tiempo computacional aceptable. Por tanto, se decidió trabajar con un almacén más pequeño, cómo el que se pone de ejemplo en apartados anteriores.

Debido a las limitaciones de los sistemas utilizados para la resolución de este problema, se aplica el modelo matemático en cuestión a un almacén de tamaño reducido, con un total de 16 posiciones. Este almacén consta de una capacidad de almacenamiento de 36 productos diferentes. Se trata de un almacén de 3 pasillos, con 3 sub-pasillos. Aunque en este caso cada pasillo realmente es el acceso a una única posición de almacenamiento, desde cada una de las cuales se pueden alcanzar 6 productos diferentes ya que constan de estantería a ambos lados y cada una de estas estanterías está compuesta por 3 alturas diferentes de slots. Al ser 3 pasillos y 3 sub-pasillos, existen 9 posiciones artificiales, que definen la interesección entre cada uno de estos pasillos. Se trabaja con un almacén tan reducido debido a las limitaciones de memoria del problema, siendo para este caso la restricción [8] aplicada a 15 posiciones, es decir, repetida para 2^{15} casos, lo que equivale a 32.768 restricciones.

Se genera por tanto una matriz de distancias entre los 16 nodos del problema, que será el input en la función del código (anexo) que permitirá representar gráficamente el almacén. Además, esta matriz de distancias es la que establece las relaciones de conectividad entre los nodos. Es decir, si dos nodos i, j se encuentran conectados entre sí, existirá un termino en la matriz de distancias Dist[i][j] que tomará un valor distinto de 0 y por tanto, habrá tantos valores distintos de 0 como arcos tenga el problema gráfico. Como se ha comentado en apartados anteriores, los arcos podrán ser recorridos en ambas direcciones, por lo que la matriz de distancias será una matriz simétrica donde Dist[i][j]=Dist[j][i].

A continuación se muestra la matriz de distancias para este almacén de 16 posiciones totales, de las cuales 1 es la posición de origen desde la que parten y a la que vuelven los operarios, 6 posiciones en las que se encuentran productos almacenados a ambos lados, y 9 posiciones artificiales, necesarias para poder realizar movimientos de una posicioón de almacenamiento a otra. Las distancias que se han tenido en cuenta son las siguientes:

- Distancias entre una posición artifical y una de almacenamiento: 2 metros.
- Distancia entre dos posiciones artificiales: 5 metros.
- Distancia entre el origen y las posiciones artificiales alcanzables desde el mismo: proporcionales.

De este modo, la matriz de distancias queda como sigue:

Tabla 3: Matriz de distancias para un almacén de 16 posiciones.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	**3,5**	0	0	0	0	**4,5**	0	0	0	0	**6,5**	0	0	0	0
1	**3,5**	0	**2**	0	0	0	**5**	0	0	0	0	0	0	0	0	0
2	0	**2**	0	**2**	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	**2**	0	**2**	0	0	0	**5**	0	0	0	0	0	0	0
4	0	0	0	**2**	0	**2**	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	**2**	0	0	0	0	0	**5**	0	0	0	0	0
6	**4,5**	**5**	0	0	0	0	0	**2**	0	0	0	**5**	0	0	0	0
7	0	0	0	0	0	0	**2**	0	**2**	0	0	0	0	0	0	0
8	0	0	0	**5**	0	0	0	**2**	0	**2**	0	0	0	**5**	0	0
9	0	0	0	0	0	0	0	0	**2**	0	**2**	0	0	0	0	0
10	0	0	0	0	0	**5**	0	0	0	**2**	0	0	0	0	0	**5**
11	**6,5**	0	0	0	0	0	**5**	0	0	0	0	0	**2**	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	**2**	0	**2**	0	0
13	0	0	0	0	0	0	0	0	**5**	0	0	0	**2**	0	**2**	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**	0	**2**
15	0	0	0	0	0	0	0	0	0	0	**5**	0	0	0	**2**	0

Esta matriz de distancias se puede representar gráficamente según el código programado en Python en este trabajo que se encuentra en el anexo. Para realizar la representación gráfica tanto de los almacenes en versión problema de grafos, como de la solución obtenida en cada ejecución, se ha utilizado la librería de Python **matplotlib.pyplot**.

A continuación se muestra una imagen de lo que sería este espacio de almacenamiento, con sus posiciones de almacenaje, y de la representación gráfica del mismo a través de Python.

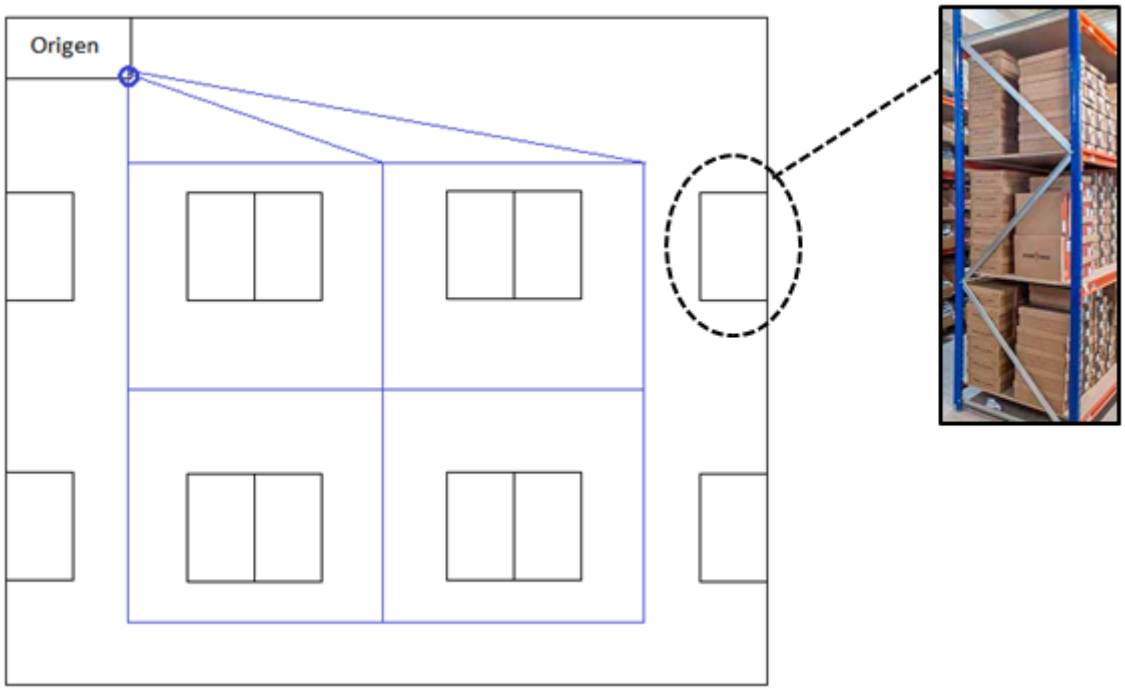


Figura 20: Representación del almacén con tres alturas de almacenamiento.

Fuente: Elaboración propia.

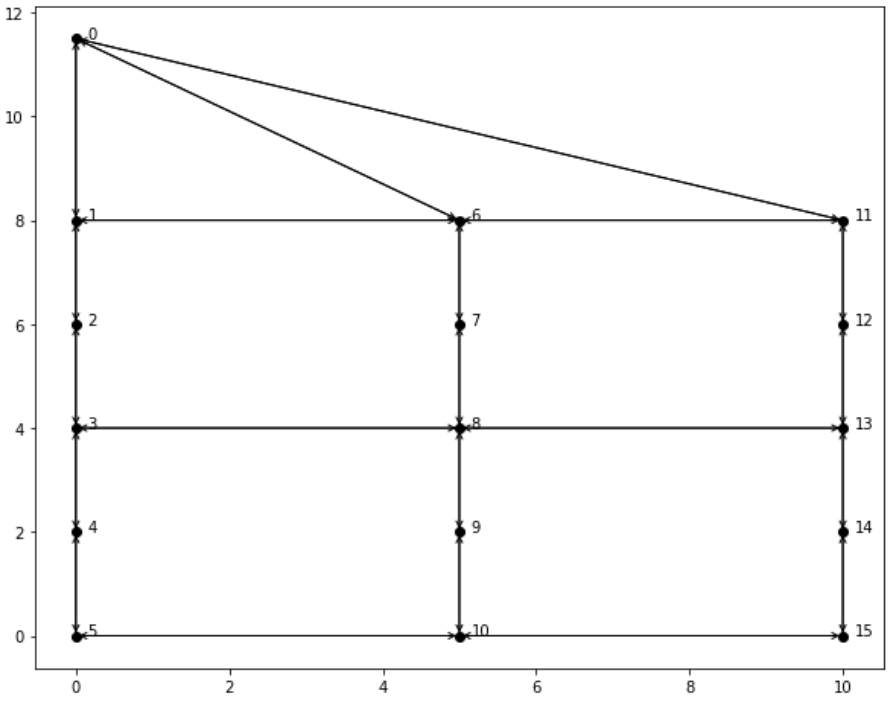


Figura 21: Representación del almacén de 16 posiciones como problema de grafos.

Fuente: Elaboración propia.

Dentro de este almacén se podrán almacenar 36 productos, 6 productos alcanzables desde cada una de las posiciones de almacenamiento. Para esta simulación, se han seleccionado 36 productos de la base de datos con la que se trabaja, y se han distribuido aleatoriamente en las posiciones de almacenamiento. Una lista más extensa de estos productos se puede encontrar en el anexo y podrá utilizarse para futuras aplicaciones del problema.

Tabla 4: Lista de productos almacenados en las 36 posiciones del almacén.

ID	family	name	Localization	Position
85	Drink	Good Imported Beer	1	1
31	Drink	Pearl Chablis Wine	2	1
45	Drink	Excellent Cola	3	1
108	Drink	Excellent Mango Drink	4	1
44	Drink	Excellent Strawberry Drink	5	1
46	Drink	BBB Best Hot Chocolate	6	1
54	Drink	CDR Hot Chocolate	7	2
34	Drink	Super Hot Chocolate	8	2
57	Drink	BBB Best Columbian Coffee	9	2
71	Drink	Excellent Orange Juice	10	2
98	Drink	Fabulous Apple Juice	11	2
60	Drink	Booker 1% Milk	12	2
97	Food	Modell Bagels	13	3
66	Food	Colony English Muffins	14	3
76	Food	Colony White Bread	15	3
58	Food	BBB Best Corn Oil	16	3
24	Food	Plato Vegetable Oil	17	3
47	Food	CDR Tomato Sauce	18	3
107	Food	Super Salt	19	4
15	Food	CDR Brown Sugar	20	4
101	Food	CDR White Sugar	21	4
10	Food	BBB Best Strawberry Jam	22	4
51	Food	CDR Apple Jam	23	4
105	Food	CDR Grape Jam	24	4
90	Food	CDR Strawberry Jam	25	5
96	Food	CDR Apple Jelly	26	5
8	Food	CDR Grape Jelly	27	5
30	Food	CDR Strawberry Jelly	28	5
106	Food	CDR Chunky Peanut Butter	29	5
42	Food	CDR Creamy Peanut Butter	30	5

50	Food	BBB Best Low Fat Apple Butter	31	6
70	Food	Just Right Fancy Canned Oysters	32	6
43	Food	Better Fancy Canned Sardines	33	6
86	Food	Blue Label Large Canned Shrimp	34	6
26	Food	Better Chicken Noodle Soup	35	6
67	Food	Bravo Turkey Noodle Soup	36	6

Como se ha comentado en anteriores apartados, la implementación de este modelo matemático ha sido codificada en el intérprete de programación Spyder, para ello se ha utilizado el lenguaje de programación Python debido a sus características explicadas en el apartado 4 y finalmente se lleva a cabo la resolución del mismo mediante el solver comercial Gurobi.

6.2 Resultados obtenidos

Habiendo definido el almacén a resolver y los datos relativos a los productos almacenados que serán parte del input necesario para el modelo matemático, se procede a resolver este almacén para varios ejemplos distintos.

Ejemplo 1.

En este ejemplo se pretende resolver el problema para las siguientes características:

- 2 pedidos a recoger.
- Sin limitación de capacidad del picker, es decir, el nº de cestas * nº de ítems por cesta se asumirá siempre superior a la cantidad total de productos a recoger.

Tabla 5: Pedidos Ejemplo 1.

Pedido o=0		Pedido o=1	
Producto	**Qty**	**Producto**	**Qty**
Pearl Chablis Wine	1	Excellent Mango Drink	1
Good Imported Beer	3	Modell Bagels	3
Excellent Cola	3	BBB Best Strawberry Jam	3
Excellent Mango Drink	5	CDR Tomato Sauce	5
Super Hot Chocolate	4	BBB Best Columbian Coffee	4
Excellent Cola	5	Colony White Bread	5
Excellent Strawberry Drink	3	Just Right Fancy Canned Oysters	3
CDR Hot Chocolate	8	CDR White Sugar	4
		Super Hot Chocolate	2
		Modell Bagels	3
TOTAL	**32**	**TOTAL**	**33**

Como los pedidos de este ejemplo piden recoger 32 y 33 uds respectivamente, la capacidad del picker debe ser superior a 65, por lo que se establece que la capacidad máxima de cestas por carro es 4, y el máximo número de ítems por cesta será 20. De este modo, se asegura que un único picker será suficiente para recoger todo lo que se pide. Sin embargo, como input del problema, habrá que definir un número de pickers/carros disponibles según el cálculo definido en el apartado 4:

$$T = \frac{\sum_{o \in O} b_o}{B} + 0{,}2$$

Lo que establece que para este ejemplo en cuestión, se dispondrá de 2 carros, aunque en el resultado óptimo no todos los carros disponibles deben ser utilizados.

Por otro lado, aunque esta tarea la realiza el propio código en función de los datos de entrada al problema, por dar mayor visibilidad del lector, a continuación se muestra la relación de los productos demandados con las posiciones de almacenaje en cuestión:

Tabla 6: Relación ID producto- Localización en almacen para Ejemplo 1.

Pedido o=0			Pedido o=1		
Pos	ID	Producto	Pos	ID	Producto
2	31	Pearl Chablis Wine	4	108	Excellent Mango Drink
1	85	Good Imported Beer	13	97	Modell Bagels
3	45	Excellent Cola	22	10	BBB Best Strawberry Jam
4	108	Excellent Mango Drink	18	47	CDR Tomato Sauce
8	34	Super Hot Chocolate	9	57	BBB Best Columbian Coffee
3	45	Excellent Cola	15	76	Colony White Bread
5	44	Excellent Strawberry Drink	32	70	Just Right Fancy Canned Oysters
7	54	CDR Hot Chocolate	21	101	CDR White Sugar
			8	34	Super Hot Chocolate
			13	97	Modell Bagels

El código programado en Spyder, mediante el lenguaje de programación Python y que se resuelve a través del solver de optimización Gurobi, se puede encontrar en el anexo de este documento. A continuación se muestran los resultados de esta ejecución:

En primer lugar se obtienen los valores de $\mathbf{Z_{ot}}$ que definen qué carro 't' recoge cada pedido 'o'.

Z[0,1]1
Z[1,1]1

Según se puede ver, tanto el pedido o=0 como el pedido o=1 son recogidos por el carro t=1. Por tanto, lo siguiente será determinar las siguientes variables:

- **Xtij:** determina la ruta que realiza cada carro. Si esta variable Xtij vale 1, es decir, se activa, indicará que el carro 't' recorre el arco descrito por los nodos consecutivos i-j, donde 'i' representa el nodo de inicio y 'j' el nodo de salida.
- **Yti:** determina los nodos visitados por cada carro. Si esta variable Yti vale 1, es decir, se activa, indicará que el carro 't' visita el nodo 'i'.
- **Gti:** determina el grado de cada nodo para el recorrido de cada carro. El grado de un nodo o vértice, indica el número de arcos parten de él. El valor entero que tome la variable Gti indicará el número de arcos que recorre el carro 't' con nodo de inicio 'i'.

A continuación se muestra el resultado de estas variables para la implementación del modelo matemático en este ejemplo 1. Los valores que se muestran en la tabla son solo aquellas variables binarias con valor verdadero (=1) en la solución y el valor de las variables enteras, por simplificar, se omiten los valores binarios falsos (=0). En la primera columna de la tabla se muestran los arcos recorridos por el carro t=1, siendo un total de 12 arcos recorridos. En la segunda columna de la tabla, se muestran los nodos visitados en la solución del problema, siendo un total 13 de nodos visitados (12 en la tabla + origen). Y por último, en la tercera columna de la tabla, se muestran los valores de los grados de cada uno de estos nodos visitados, donde cabe destacar que el nodo 11 tiene grado 2, lo que significa que de él parten dos arcos en direcciones diferentes, es decir, en la solución hay un "ciclo".

Tabla 7: Valor en la solución de las variables del modelo para el Ejemplo 1.

X[t, i, j]	Y[t, i]	G[t, i]
X[1,0,11]1	Y[1,1]1	G[1,0]1
X[1,1,0]1	Y[1,2]1	G[1,1]1
X[1,2,1]1	Y[1,3]1	G[1,2]1
X[1,3,2]1	Y[1,4]1	G[1,3]1
X[1,4,3]1	Y[1,5]1	G[1,4]1
X[1,5,4]1	Y[1,6]1	G[1,5]1
X[1,6,7]1	Y[1,7]1	G[1,6]1
X[1,7,8]1	Y[1,8]1	G[1,7]1
X[1,8,9]1	Y[1,9]1	G[1,8]1
X[1,9,10]1	Y[1,10]1	G[1,9]1
X[1,10,5]1	Y[1,11]1	G[1,10]1
X[1,11,6]1	Y[1,12]1	G[1,11]2

X[1,11,12]1	G[1,12]1
X[1,12,11]1	

Por último, y lo más importante, el valor que toma la función objetivo para la solución óptima de este ejemplo es → Obj: 40. Es decir, la distancia total recorrida por el picker será de 40 metros. Esto tiene fácil comprobación ya que conocemos la matriz de distancias entre todos los nodos y ahora también conocemos los arcos recorridos en la solución.

$$\boldsymbol{MIN}\left\{\sum_{t\in T}\sum_{(i,j)\in A}\boldsymbol{Dij}*\boldsymbol{Xtij}\right\} = \mathbf{40\ m}$$

Por ultimo, como se ha comentado anteriormente, se ejecuta un bloque de código que permite obtener una representación gráfica del recorrido que ejecuta el picker según la solución obtenida. Según esto último, el recorrido será el siguiente:

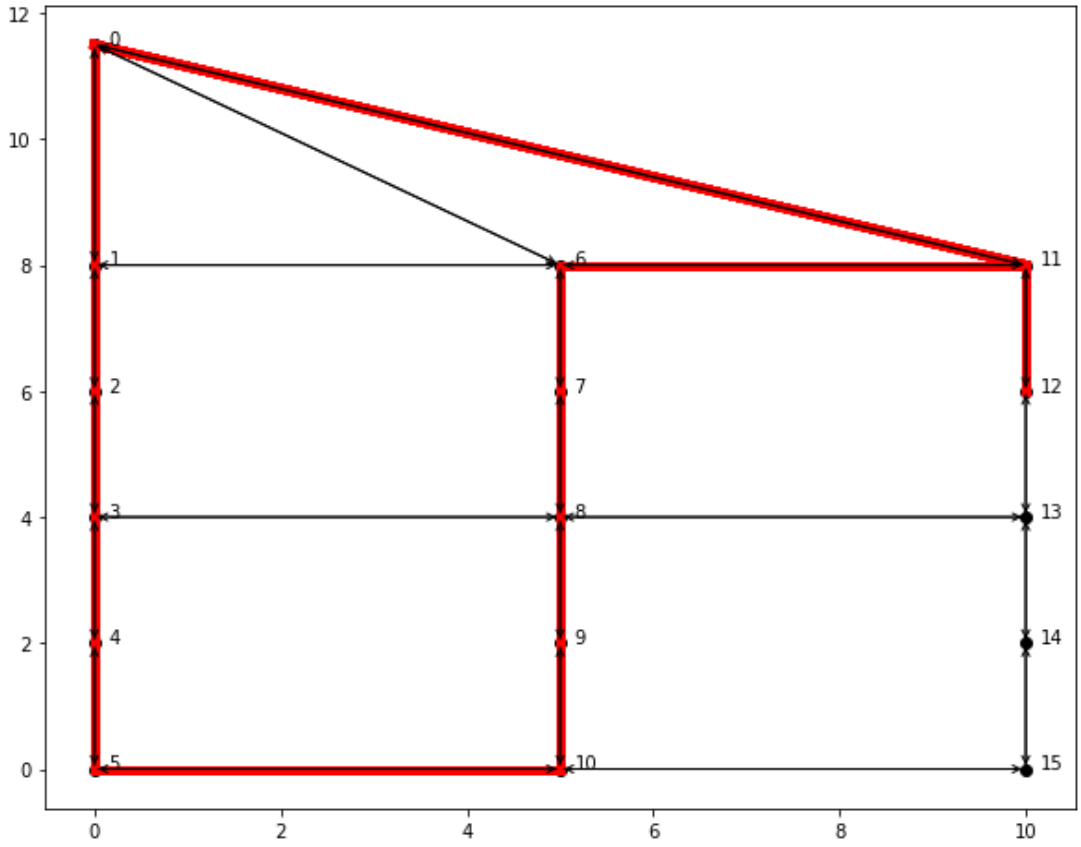


Figura 22: Recorrido del carro t=1 para el ejemplo 1.

Fuente: Elaboración propia.

El porcentaje de utilización del picker para el Ejemplo 1 será del 65/80 = 81.25%.

Ejemplo 2.

En este ejemplo se pretende resolver el problema para las siguientes características:

- 3 pedidos a recoger.
- Con limitación de capacidad del picker, es decir, el nº de cestas * nº de ítems por cesta será tal, que necesite más de un picker para satisfacer toda la demanda de los clientes.

Tabla 8: Pedidos Ejemplo 2.

Pedido o=0		Pedido o=1		Pedido o=2	
Producto	Qty	Producto	Qty	Producto	Qty
BBB Best Strawberry Jam	5	Better Chicken Noodle Soup	1	CDR Strawberry Jelly	2
Super Hot Chocolate	7	Better Fancy Canned Sardines	3	Super Hot Chocolate	3
Excellent Mango Drink	8	Bravo Turkey Noodle Soup	5	CDR Creamy Peanut Butter	4
Excellent Strawberry Drink	8	CDR Brown Sugar	7	Excellent Cola	2
CDR Apple Jam	4	CDR Chunky Peanut Butter	2	BBB Best Low Fat Apple Butter	1
CDR Tomato Sauce	10	CDR Grape Jelly	4	Bravo Turkey Noodle Soup	1
BBB Best Columbian Coffee	4	CDR White Sugar	3	BBB Best Columbian Coffee	3
BBB Best Corn Oil	6	Colony English Muffins	6	Colony English Muffins	9
Booker 1% Milk	11	Colony White Bread	3	Blue Label Large Canned Shrimp	1
Fabulous Apple Juice	7	Good Imported Beer	7	CDR Strawberry Jam	2
CDR Grape Jam	5	Just Right Fancy Canned Oysters	1	CDR Apple Jelly	1
Plato Vegetable Oil	3	Pearl Chablis Wine	1	Modell Bagels	2
BBB Best Hot Chocolate	5	Super Salt	2	Good Imported Beer	4
TOTAL	**83**	**TOTAL**	**45**	**TOTAL**	**35**

Como los pedidos de este ejemplo piden recoger 83, 45 y 35 uds respectivamente, la capacidad del picker se establecerá de 100 uds, por lo que se supondrá que la capacidad máxima de cestas por carro es 5, y el máximo número de ítems por cesta será 20. De este modo, un picker tendrá capacidad de recoger el pedido o=0 completo y un segundo picker tendrá capacidad suficiente para recoger los productos correspondientes a los pedidos o=1 y o=2. Siendo las capacidades correspondientes a cada carro las siguientes:

Tabla 9: Reparto carros para Ejemplo 2.

Nº ordenes	Capacidad carro			Utilizacion	% Utilización
o=0	5 cestas	20 uds/cesta	100	83	**83%**
o=1 o=2	5 cestas	20 uds/cesta	100	45 35	**80%**

Del mismo modo que en el ejemplo anterior, aunque esta tarea la realiza el propio código en función de los datos de entrada al problema, por dar mayor visibilidad del lector, a continuación se muestra la relación de los productos demandados con las posiciones de almacenaje en cuestión:

Tabla 10: Relación ID producto- Localización en almacen para Ejemplo 2.

Pedido o=0		
pos	ID	Producto
22	10	BBB Best Strawberry Jam
8	34	Super Hot Chocolate
4	108	Excellent Mango Drink
5	44	Excellent Strawberry Drink
23	51	CDR Apple Jam
18	47	CDR Tomato Sauce
9	57	BBB Best Columbian Coffee
16	58	BBB Best Corn Oil
12	60	Booker 1% Milk
11	98	Fabulous Apple Juice
24	105	CDR Grape Jam
17	24	Plato Vegetable Oil
6	46	BBB Best Hot Chocolate

Pedido o=1			Pedido o=2		
pos	ID	Producto	pos	ID	Producto
35	26	Better Chicken Noodle Soup	28	30	CDR Strawberry Jelly
33	43	Better Fancy Canned Sardines	8	34	Super Hot Chocolate
36	67	Bravo Turkey Noodle Soup	30	42	CDR Creamy Peanut Butter
20	15	CDR Brown Sugar	3	45	Excellent Cola
29	106	CDR Chunky Peanut Butter	31	50	BBB Best Low Fat Apple Butter
27	8	CDR Grape Jelly	36	67	Bravo Turkey Noodle Soup
21	101	CDR White Sugar	9	57	BBB Best Columbian Coffee
14	66	Colony English Muffins	14	66	Colony English Muffins
15	76	Colony White Bread	34	86	Blue Label Large Canned Shrimp
1	85	Good Imported Beer	25	90	CDR Strawberry Jam
32	70	Just Right Fancy Canned Oysters	26	96	CDR Apple Jelly
2	31	Pearl Chablis Wine	13	97	Modell Bagels
19	107	Super Salt	1	85	Good Imported Beer

A continuación se muestran los resultados de esta ejecución:

En primer lugar se obtienen los valores de $\mathbf{Z_{ot}}$ que definen qué carro 't' recoge cada pedido 'o':

Z[0,0]1
Z[1,1]1
Z[2,1]1

Donde, tal y como se había previsto, se refleja que el carro/picker t=0 recogerá los productos del pedido o=0 y el carro/picker t=1 recogerá los productos correspondientes a los pedidos o=1 y o=2.

Del mismo modo que en el ejemplo anterior, se recogen los valores que el resto de variables del problema toman en la solución óptima del ejemplo 2. Como en esta ocasión es necesario el uso de dos carros para dar soporte a toda la demanda del cliente, se analizarán las soluciones para cada uno de estos carros por separado. Finalmente, el valor de la función objetivo, englobará ambos recorridos y es por esto que la solución conjunta es la óptima.

Resultados obtenidos para el carro t=0:

De los resultados obtenidos para este carro, se observa que se recorren un total de 11 arcos, visitando tan solo 9 nodos (los mostrados en la tabla además del nodo origen). Para que esto se pueda dar, será necesario que se formen "ciclos" en el recorrido, lo cual se demuestra en los valores del grado de los nodos, donde tanto e nodo 3 como el nodo 8 tienen grado superior a 1, es decir, de ellos parten más de un arco en distintas direcciones como se observa en la figura 23.

Tabla 11: Valor en la solución de las variables del modelo relativas a t=0 para el Ejemplo 2.

X[t, i, j]	Y[t, i]	G[t, i]
X[0,0,1]1	Y[0,1]1	G[0,0]1
X[0,1,2]1	Y[0,2]1	G[0,1]1
X[0,2,3]1	Y[0,3]1	G[0,2]1
X[0,3,4]1	Y[0,4]1	G[0,3]2
X[0,3,8]1	Y[0,6]1	G[0,4]1
X[0,4,3]1	Y[0,7]1	G[0,6]1
X[0,6,0]1	Y[0,8]1	G[0,7]1
X[0,7,6]1	Y[0,9]1	G[0,8]2
X[0,8,7]1		G[0,9]1
X[0,8,9]1		
X[0,9,8]1		

El término de la función objetivo correspondiente a la suma de las distancias recorridas por el picker t=0, alcanza por tanto un valor de 29 metros.

Teniendo todo esto en cuenta, el código ejecutado nos muestra la representación gráfica del recorrido que ejecuta el picker t=0 según la solución obtenida.

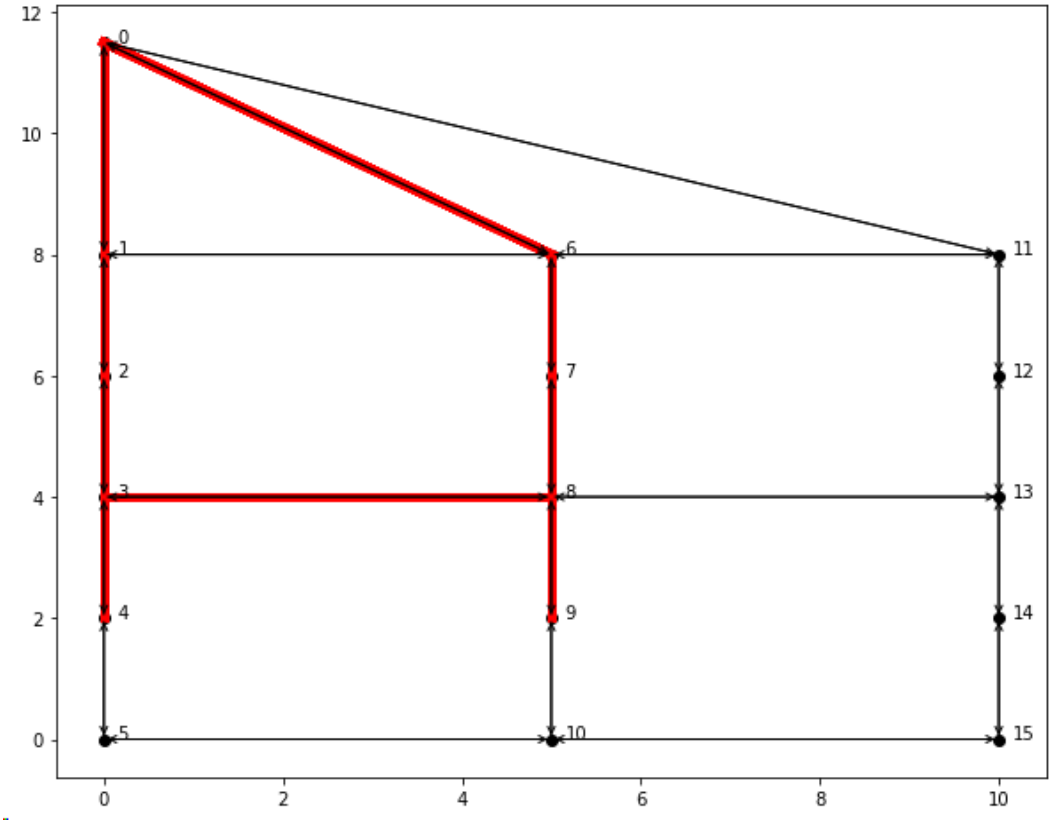


Figura 23: Recorrido del carro t=0 para el ejemplo 2.

Fuente: Elaboración propia.

Resultados obtenidos para el carro t=1:

De los resultados obtenidos para este carro, se observa que se recorren un total de 12 arcos, visitando tan solo 10 nodos (los mostrados en la tabla además del nodo origen). Una vez más, para que esto se pueda dar, será necesario que se formen "ciclos" en el recorrido, lo cual se demuestra en los valores del grado de los nodos, donde tanto e nodo 8 como el nodo 13 tienen grado superior a 1, es decir, de ellos parten más de un arco en distintas direcciones como se observa en la figura 24.

Tabla 12: Valor en la solución de las variables del modelo relativas a t=1 para el Ejemplo 2.

X[t, i, j]	Y[t, i]	G[t, i]
X[1,0,11]1	Y[1,1]1	G[1,0]1
X[1,1,0]1	Y[1,2]1	G[1,1]1
X[1,2,1]1	Y[1,3]1	G[1,2]1
X[1,3,2]1	Y[1,7]1	G[1,3]1
X[1,7,8]1	Y[1,8]1	G[1,7]1
X[1,8,3]1	Y[1,11]1	G[1,8]2
X[1,8,7]1	Y[1,12]1	G[1,11]1
X[1,11,12]1	Y[1,13]1	G[1,12]1
X[1,12,13]1	Y[1,14]1	G[1,13]2
X[1,13,8]1		G[1,14]1
X[1,13,14]1		
X[1,14,13]1		

El término de la función objetivo correspondiente a la suma de las distancias recorridas por el picker t=1, alcanza por tanto un valor de 36 metros.

Teniendo todo esto en cuenta, el código ejecutado nos muestra la representación gráfica del recorrido que ejecuta el picker t=1 según la solución obtenida.

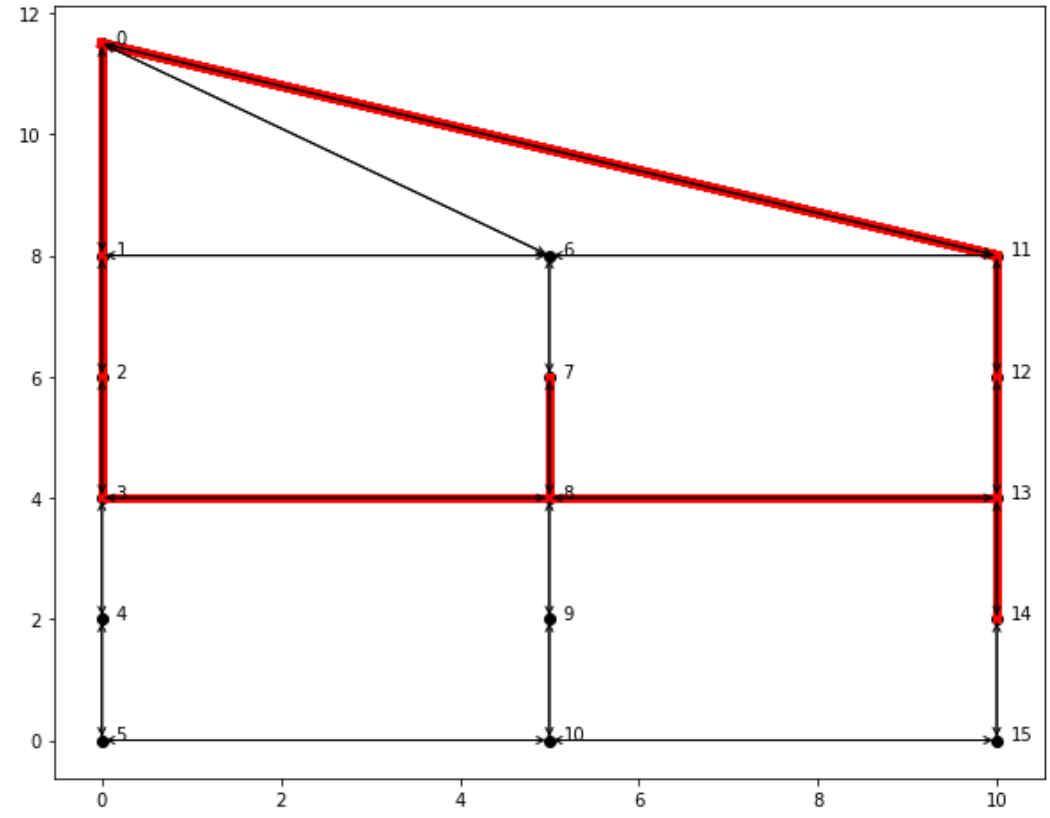


Figura 24: Recorrido del carro t=1 para el ejemplo 2.

Fuente: Elaboración propia

Por último, el valor que toma la función objetivo completa para la solución óptima de este ejemplo es → Obj: 65. Es decir, la distancia total recorrida por el picker será de 65 metros. Esto tiene fácil comprobación ya que conocemos la matriz de distancias entre todos los nodos y ahora también conocemos los arcos recorridos en la solución para cada uno de los carros. Teniendo todo esto en cuenta se obtiene lo siguiente:

$$\boldsymbol{MIN}\left\{\sum_{t\in T}\sum_{(i,j)\in A}\boldsymbol{Dij}*\boldsymbol{Xtij}\right\}\textbf{ = 65 m}$$

A continuación se muestra un pequeño resumen de los resultados obtenidos en estos pequeños ejemplos de aplicación del modelo.

Tabla 13: Resultados obtenidos.

Ejemplo	NºCarros	NºPedidos	NºProductos	NºArcos recorridos	NºNodos visitados	F.O.
1	1	1	65	12	13	**40**
2	2	2	83	11	9	**60**
		1	80	12	10	

6.2.1 Mejoras en el código implementado

Durante el desarrollo de este proyecto se encontraron ciertas dificultades debido a la naturaleza del problema así como a las limitaciones de los equipos de trabajo. Debido a estas dificultades es por lo que se trabaja con unos casos prácticos muy reducidos, que pueden encontrarse en la realidad de pequeños establecimientos pero no sería realista para el caso de grandes almacenes. Por este motivo, se trabajó en la búsqueda de alternativas para la resolución de la restricción limitante [8] así como posibles mejoras en el proceso de aplicación de manera que el código utilizado sea extrapolable a almacenes de mayor tamaño.

Para el caso de la restricción [8] que se ha definido como sigue:

Permite que se recorran sub-rutas, más conocidas como 'ciclos' en la teoría de grafos. Para que se dé este caso, al menos uno de los nodos visitados por el caro 't' deberá tener un grado superior a 1. Esta relación se expresa matemáticamente de la siguiente manera:

$$\sum_{j\in W}\boldsymbol{G_{tj}}\geq\boldsymbol{Y_{ti}}+\sum_{(j,k)\in A(W)}\boldsymbol{X_{tjk}}\qquad\forall\boldsymbol{i}\in\boldsymbol{W},\boldsymbol{W}\subseteq\boldsymbol{V}\backslash\{\boldsymbol{s}\},|\boldsymbol{W}|>\boldsymbol{1},\qquad\boldsymbol{t}\in\boldsymbol{T}$$

Se propuso una alternativa que supone eliminar dicha restricción (lo que suponía perder la continuidad del recorrido) y modificar el grafo del problema a un grafo en el que todos los nodos se encuentren conectados entre sí, siendo la distancia entre ellos la suma mínima de distancias entre los nodos que los unen. De este modo, se obtiene una matriz completa de distancias entre nodos y se aplicaría la siguiente restricción:

$$T_{jt} \geq T_{it} + Dist_{ij} - M[1 - X_{tij}]$$

Donde Tjt es el instante en el que el carro t alcanza el nodo j y M un coeficiente suficientemente grande. Con esta restricción propuesta, se aseguraría la continuidad del recorrido. Sin embargo, para llevar a cabo esta mejor será necesario crear la matriz de distancias completa para todos los nodos, lo cual es un trabajo tedioso y que igualmente crece exponencialmente con el tamaño del almacén a resolver.

En la búsqueda de mejoras para el proceso de aplicación del código se desarrolló una función que reduce el almacén de la forma que se explicaba en la introducción del presente trabajo, es decir, reduciendo los nodos del problema sólo a aquellos que deben ser visitados y recalculando la matriz de distancias para dichos nodos. Se ha desarrollado una función en Python para dicha reducción. Esto permitirá resolver almacenes de mayor tamaño ya que no todas sus posiciones deben ser visitadas, por lo que no limitaría el tamaño del almacén en sí, sino la variabilidad en los pedidos a recoger que hará que aumenten las posiciones a visitar, y por tanto, el tamaño del problema. Este caso se va a explicar a continuación.

Se trata de resolver de manera exacta un almacén de 37 posiciones. Para ello, se genera un almacén de 3 pasillos y 1 sub-pasillo. Este alamacén por tanto constará de un nodo origen, 9 nodos de intersección y 27 nodos desde los cuales se pueden alcanzar hasta 4 productos almacenados, es decir, esta vez se trabajará con dos alturas por simplifciar. Debido a estas características, el almacén que se pretende resolver tendrá capacidad de almacenar hasta 108 productos diferentes.

A continuación se muestran cual es esta matriz de dimensiones, para la cual se han tomado valores como los que se tomaban en el problema original. Una distancia de un metro entre nodos con productos almacenados que se encuentran disposicionados de manera consecutiva, una distancia de 2 metros entre un nodo artificial y sus nodos adyacentes que contengan productos, una distancia de 5 metros entre nodos artificiales y unas distancias entre el origen y los nodos

artificiales alcanzables desde este que se establecen de manera proporcional. Dicho esto, la matriz de distancias para el almacén a resolver queda de la siguiente manera:

Tabla 14: Matriz de distancias para un almacén de 37 posiciones.

	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	29	30	31	32	33	34	35	36
0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	6	8	-	-	-	-	-	-
1	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-
2	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-
6	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-
7	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-
10	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-
24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	2	-	-	-
25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-
26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-
27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	2
28	4	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-
29	6	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	5	-	-	-	-	-	-
30	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-
31	-	-	-	-	-	2	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-
32	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	5	-	-	-
33	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2	-	-	-	-	-	-	-	5	-	-	-	-
34	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-
35	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	5
36	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	5	-

Esta matriz de distancias se puede representar gráficamente según el código programado en Python en este trabajo que se encuentra en el Anexo I. Para realizar la representación gráfica tanto de los almacenes en versión problema de grafos, como de la solución obtenida en cada ejecución, se ha utilizado la librería de Python **matplotlib.pyplot**.

A continuación se muestra la representación del almacén de 37 posiciones, en el cual se almacenan hasta 108 variedades de productos con el que se va a trabajar en este proyecto:

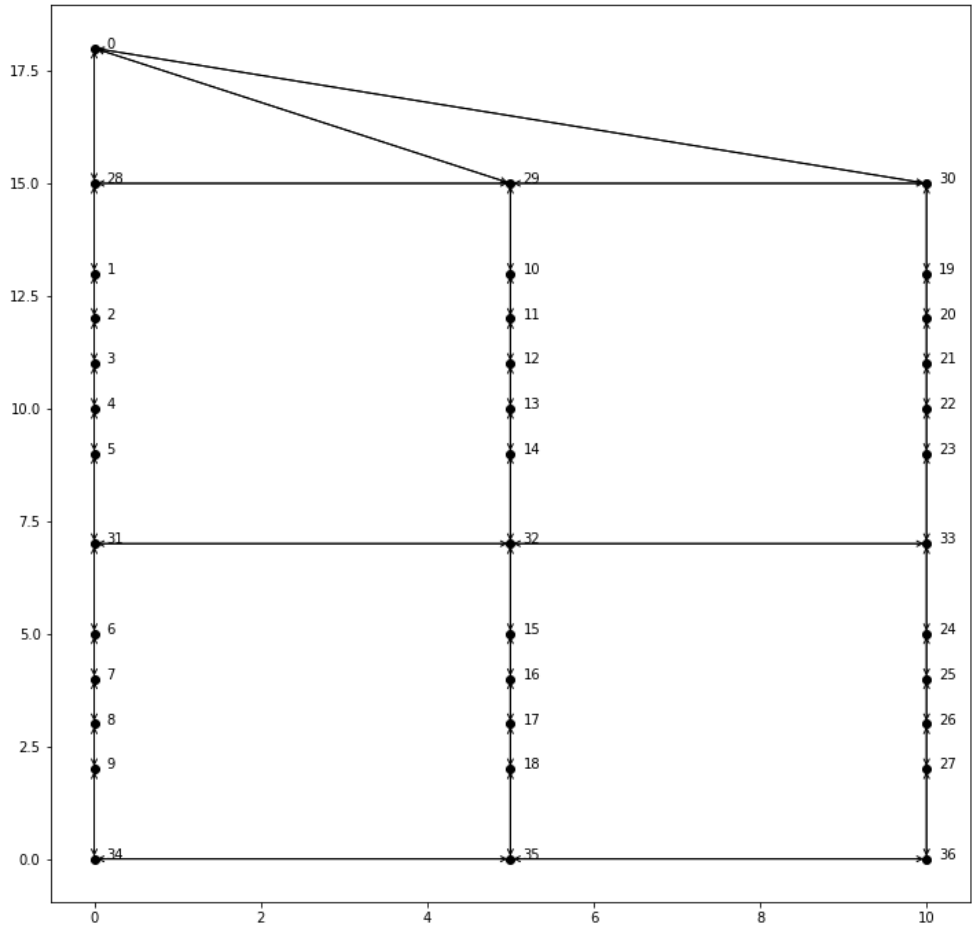


Figura 25: Representación gráfica en Python de la matriz de distancias para un almacén de 37 posiciones.

Fuente: Elaboración propia

Aunque el almacén a estudiar inicalmente se sigue viendo reducido considerablemente, como se ha mencionado anteriormente, la restricción que establece que está permitida la realización de ciclos en la solución de la ruta de este almacén, restricción [8], crece exponencialmente con las posiciones. Para este nuevo grafo a resolver, esta restricción se repetiría al menos 2^{36}=6.872x10^{10} veces. Esto agiliza la resolución del problema, sin embargo, con el fin de optimizar el código, se desarrolla una función en Pyhton que permitirá reducir los nodos del grafo, de manera que de los nodos en los que está permitido almacenar productos, sólo se trabaje con aquellos de los cuales haya que recolectar alguno de los 4 productos alcanzables desde el mismo. De esta manera, se evitará analizar todas las restricciones para nodos que sólo van a ser recorridos ‘de paso’, es decir, porque se encuentran entre dos nodos que deben ser visitados para cumplir con los requerimientos del cliente.

6.2.1.1 Función reducción de almacén

A continuación se muestra en qué consiste esta función.

Si se parte de un almacén de 108 localizaciones de almacenamiento, es decir, 37 posiciones totales como el que se muestra en la figura 25, y se tienen que recoger los productos de los siguientes pedidos:

Tabla 15: Ejemplo de una orden de recogida compuesta por dos pedidos.

Pedido 1		Pedido 2	
Producto	**Cantidad**	**Producto**	**Cantidad**
Super Salt	2	Footnote Seasoned Hamburger	5
Hermanos Summer Squash	5	CDR White Sugar	2
Carrington Waffles	8	Hermanos Elephant Garlic	4
Hermanos Lettuce	2	Carlson Sour Cream	2
Colony White Bread	3	Hermanos Lettuce	2
Booker Mild Cheddar Cheese	2	Carrington Apple Cinnamon Waffles	5
Big Time Frozen Mushroom Pizza	3	Blue Medal Small Brown Eggs	6
Carrington Ice Cream	1	Carlson Strawberry Yogurt	4
Carlson Sour Cream	1	Hermanos Shitake Mushrooms	2

Para cada uno de los productos, existe un código ID para identificarlos más fácilmente y una relación de este ID con su posición en el almacén. Estas relaciones se pueden encontrar en el Anexo.

Cómo para este proyecto no se contemplan restricciones de tamaño, forma o compatibilidad de productos, sino que la única restricción de la recogida será la capacidad del picker en cuanto a número de productos, esta información ID/Posición será con la que se ejecuta el modelo matemático a la hora de resolver el problema.

Para el ejemplo mostrado en la Tabla 15, se muestra a continuación esta relación de ID de producto - Posición de almacenamiento:

Tabla 16: Relación ID producto- Localización en almacen.

Pedido 1		Pedido 2	
ID producto	Localización	ID producto	Localización
107	19	7	65
94	100	101	21
32	56	104	87
55	91	25	43
76	15	55	91
88	40	9	53
102	61	20	51
100	57	33	45
25	43	74	98

Por tanto, una vez listados los productos a recoger y sus localizaciones, deben analizarse desde qué posiciones del almacén se alcanza cada una de estas posiciones de almacenamiento.

Tabla 17: Posiciones a visitar en el almacén.

Localización en almacén	19	100	56	91	15	40	61	57	43	65	21	87	53	51	45	98
Posiciones a visitar	1	23	10	19	8	11	13	11	13	15	2	26	18	17	14	22

Se puede prescindir de las posiciones 3, 4, 5, 6, 7, 9, 12, 16, 20, 21, 24, 25 y 27 ya que no se requiere ningún producto de los que se alcanzan desde ellas. Esto reducirá por tanto el número de restricciones tipo [8] que como se ha visto anteriormente, aumenta exponencialmente con el número de posiciones del almacén.

Siendo estas las 14 posiciones a visitar, se reducirá el almacén de 36 posiciones a un almacén de 24 posiciones. Una vez definidas las nuevas posiciones desde las que se debe recoger material, teniendo en cuenta que se deben añadir además los nodos artificiales (de intersección entre pasillos y subpasillos) así como le nodo origen ‘s’, la representación gráfica del almacén reducido para este ejemplo en cuestión será la siguiente:

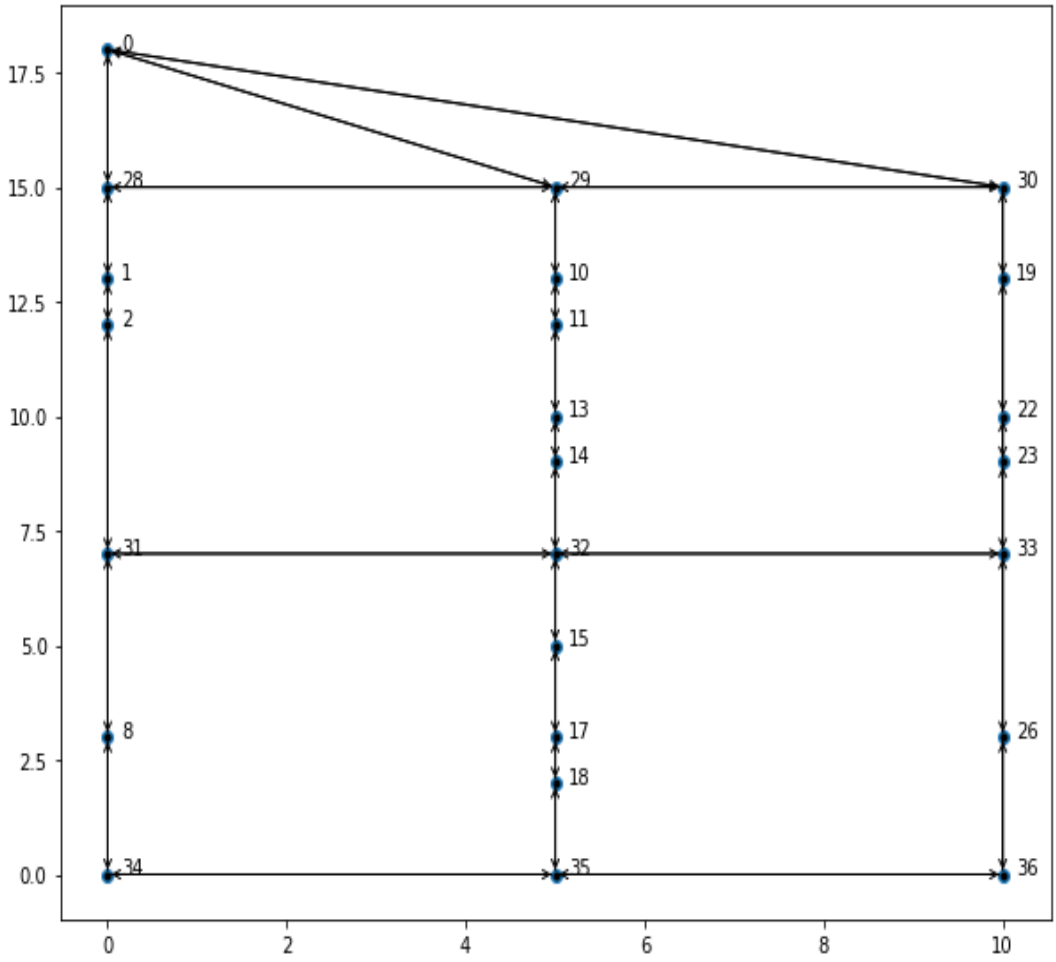


Figura 26: Representación gráfica en Python del almacén reducido.

Fuente: Elaboración propia.

Para llevar a cabo la reducción de este alamacén se ejecuta el siguiente bloque de código en Pyhton, correspondiente a una función llamada "red" de elaboración propia:

```
def red (Dist,Loc_total_red):
    Red=[]
    i=0
    j=0
    while i<len(Loc_total_red):
        Red.append([])
        while j<len(Loc_total_red):
            Red[i].append(0)
            j=j+1
        j=0
        i=i+1
    i=0
    for i in range (0, len(Dist)):
        if i in Loc_total_red and i<28:
```

```
    for j in range (0, len(Dist)):
        d=0
        if Dist[i][j] != 0:
            if j in Loc_total_red:
                Red[Loc_total_red.index(i)][Loc_total_red.index(j)] =Dist[i][j]
            elif j not in Loc_total_red:
                k=j
                d=0
                while k not in Loc_total_red and Dist[k][k-1]!=0:
                    d=d+Dist[k][k-1]
                    k=k+1
                if k!=i and Dist[k][k-1]!=0:
                    Red[Loc_total_red.index(i)][Loc_total_red.index(k)] =d+ Dist[k][k-1]
                    Red[Loc_total_red.index(k)][Loc_total_red.index(i)] =d+ Dist[k][k-1]
                else:
                    pass
            else:
                pass
        else:
            pass
elif i in Loc_total_red and i>=28:
    for j in range (0, len(Dist)):
        d=0
        if Dist[i][j] != 0:
            if j in Loc_total_red:
                Red[Loc_total_red.index(i)][Loc_total_red.index(j)] =Dist[i][j]
            elif j not in Loc_total_red:
                d=Dist[i][j]
                k=j
                while k not in Loc_total_red and Dist[k][k-1]!=0:
                    d=d+Dist[k][k-1]
                    k=k-1
                if k!=i and Dist[k][k-1]!=0:
                    Red[Loc_total_red.index(i)][Loc_total_red.index(k)] =d
                    Red[Loc_total_red.index(k)][Loc_total_red.index(i)] =d
                else:
                    pass
                d=Dist[i][j]
                k=j
                while k not in Loc_total_red and Dist[k][k+1]!=0:
                    d=d+Dist[k][k+1]
                    k=k+1
                if k!=i and Dist[k][k+1]!=0:
                    Red[Loc_total_red.index(i)][Loc_total_red.index(k)] =d
                    Red[Loc_total_red.index(k)][Loc_total_red.index(i)] =d
                else:
                    pass
        else:
```

```
            pass
        else:
            pass
    return Red
```

A continuación se muestra una comprobación empírica de la aplicación del modelo matemático con dicha mejora.

Ejemplo 3.

En este ejemplo se pretende resolver el problema para un almacén original de 37 posiciones como el mostrado anteriormente y las siguientes características:

- 3 pedidos a recoger.
- Con limitación de capacidad del picker.

A continuación se muestran los pedidos que se pretenden recoger en este ejemplo.

Tabla 18: Pedidos Ejemplo 3.

Pedido o=0		Pedido o=1		Pedido o=2	
Producto	**Qty**	**Producto**	**Qty**	**Producto**	**Qty**
Excellent Strawberry Drink	4	Club Low Fat Cottage Cheese	2	Colony White Bread	7
CDR Apple Jam	3	Best Choice Fudge Brownies	3	BBB Best Hot Chocolate	6
Carrington Popsicles	3	CDR Grape Jam	3	Carrington Blueberry Waffles	4
Blue Medal Large Eggs	5	Blue Medal Large Brown Eggs	4	Blue Label Large Canned Shrimp	4
Hermanos Green Pepper	4	Ebony Limes	1	Ebony Plums	5
Best Choice Frosted Cookies	5	Hermanos Green Pepper	2	CDR Grape Jam	10
Gorilla Head Cheese	3	BBB Best Hot Chocolate	1	CDR Apple Jam	3
CDR Grape Jam	4	Big Time Frozen Cheese Pizza	4	Excellent Strawberry Drink	3
Blue Medal Large Brown Eggs	3			BBB Best Corn Oil	4
Ebony Macintosh Apples	3			Carrington Apple Cinnamon Waffles	3
				Better Fancy Canned Sardines	4
				Ebony Peaches	2
TOTAL	**37**	**TOTAL**	**20**	**TOTAL**	**55**

Para forzar a que sea necesario el uso de más de un picker para satisfacer dicha demanda, se definirá la capacidad máxima del picker en 75, de tal manera que un único picker como máximo solo podrá recoger 2 pedidos. Para ello se define un nº máximo de cestas por carro de 3 y un nº máximo de ítems por cesta de 25 productos.

Para conocer los nodos a visitar, y por tanto, a los que se podrá reducir el grafo del problema, es necesario conocer las posición de almacenamiento de cada uno de los productos demandados.

Tabla 19: Relación ID producto- Localización en almacen para el Ejemplo 3.

Pedido o=0		
Loc	ID	Producto
5	44	Excellent Strawberry Drink
23	51	CDR Apple Jam
59	28	Carrington Popsicles
50	62	Blue Medal Large Eggs
90	19	Hermanos Green Pepper
107	56	Best Choice Frosted Cookies
41	95	Gorilla Head Cheese
24	105	CDR Grape Jam
49	22	Blue Medal Large Brown Eggs
68	69	Ebony Macintosh Apples

Pedido o=1			Pedido o=2		
Loc	ID	Producto	Loc	ID	Producto
42	83	Club Low Fat Cottage Cheese	15	76	Colony White Bread
108	81	Best Choice Fudge Brownies	6	46	BBB Best Hot Chocolate
24	105	CDR Grape Jam	54	79	Carrington Blueberry Waffles
49	22	Blue Medal Large Brown Eggs	34	86	Blue Label Large Canned Shrimp
67	21	Ebony Limes	72	5	Ebony Plums
90	19	Hermanos Green Pepper	24	105	CDR Grape Jam
6	46	BBB Best Hot Chocolate	23	51	CDR Apple Jam
60	17	Big Time Frozen Cheese Pizza	5	44	Excellent Strawberry Drink
			16	58	BBB Best Corn Oil
			53	9	Carrington Apple Cinnamon Waffles
			33	43	Better Fancy Canned Sardines
			71	53	Ebony Peaches

En primer lugar, se obtiene la agrupación de pedidos en lotes, asignándolos a distintos carros de recogida:

Z[0,0]1
Z[1,0]1
Z[2,1]1

De donde se concluye que el carro t=0 recoge los pedidos o=0 y o=1 (carga total de 57 uds) y el carro t=1 recogerá el pedido o=2 (carga de 55 uds). El valor que toman las variables del problema en la solución final para cada uno de los carros se muestran a continuación.

Resultados obtenidos para el carro t=0:

Tabla 20: Valor en la solución de las variables del modelo relativas a t=0 para el Ejemplo 3.

X[t, i, j]	Y[t, i]	G[t, i]
X[0,0,8]1	Y[0,1]1	G[0,0]1
X[0,1,7]1	Y[0,3]1	G[0,1]1
X[0,3,11]1	Y[0,4]1	G[0,3]1
X[0,4,5]1	Y[0,5]1	G[0,4]2
X[0,4,11]1	Y[0,6]1	G[0,5]2
X[0,5,4]1	Y[0,7]1	G[0,6]1
X[0,5,14]1	Y[0,8]1	G[0,7]1
X[0,6,15]1	Y[0,10]1	G[0,8]1
X[0,7,0]1	Y[0,11]1	G[0,10]1
X[0,8,3]1	Y[0,14]1	G[0,11]2
X[0,10,1]1	Y[0,15]1	G[0,14]2
X[0,11,4]1		G[0,15]2
X[0,11,10]1		
X[0,14,5]1		
X[0,14,15]1		
X[0,15,6]1		
X[0,15,14]1		

Lo que tras ejecutar el bloque de código pertinente, se traduce gráficamente al siguiente recorrido:

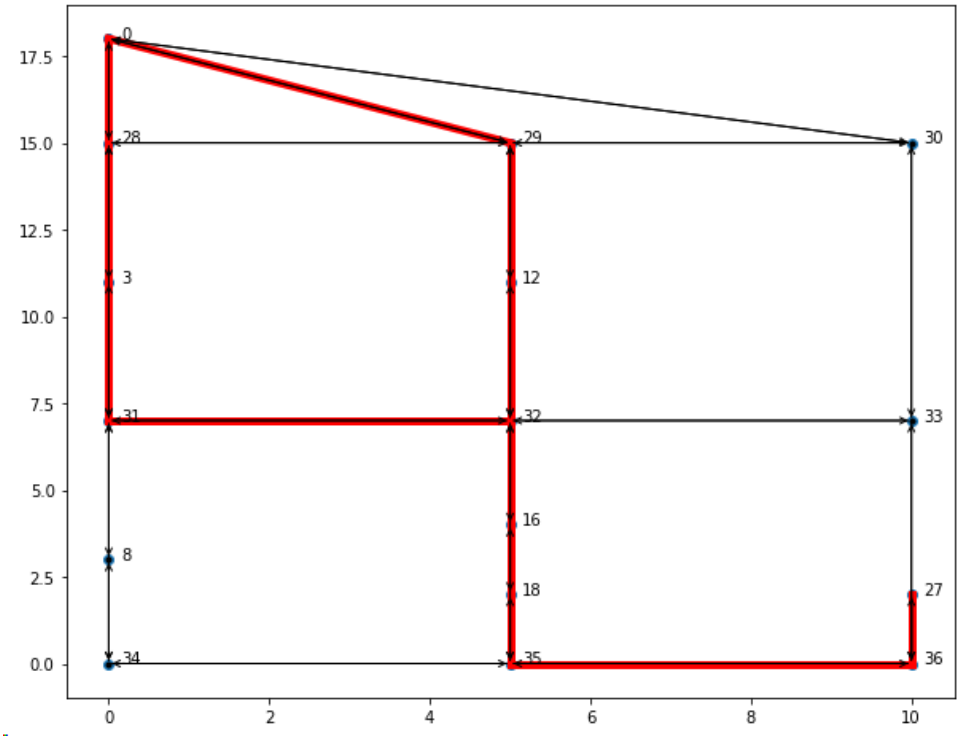


Figura 27: Recorrido del carro t=0 para el ejemplo 3.

Fuente: Elaboración propia.

Resultados obtenidos para el carro t=1:

Tabla 21: Valor en la solución de las variables del modelo relativas a t=1 para el Ejemplo 3.

X[t, i, j]	Y[t, i]	G[t, i]
X[1,0,8]1	Y[1,1]1	G[1,0]1
X[1,1,7]1	Y[1,2]1	G[1,1]1
X[1,2,10]1	Y[1,3]1	G[1,2]1
X[1,3,11]1	Y[1,4]1	G[1,3]1
X[1,4,5]1	Y[1,5]1	G[1,4]1
X[1,5,14]1	Y[1,7]1	G[1,5]1
X[1,7,0]1	Y[1,8]1	G[1,7]1
X[1,8,3]1	Y[1,10]1	G[1,8]1
X[1,10,1]1	Y[1,11]1	G[1,10]1
X[1,11,4]1	Y[1,13]1	G[1,11]1
X[1,13,2]1	Y[1,14]1	G[1,13]1
X[1,14,13]1		G[1,14]1

Lo que tras ejecutar el bloque de código pertinente, se traduce gráficamente al siguiente recorrido:

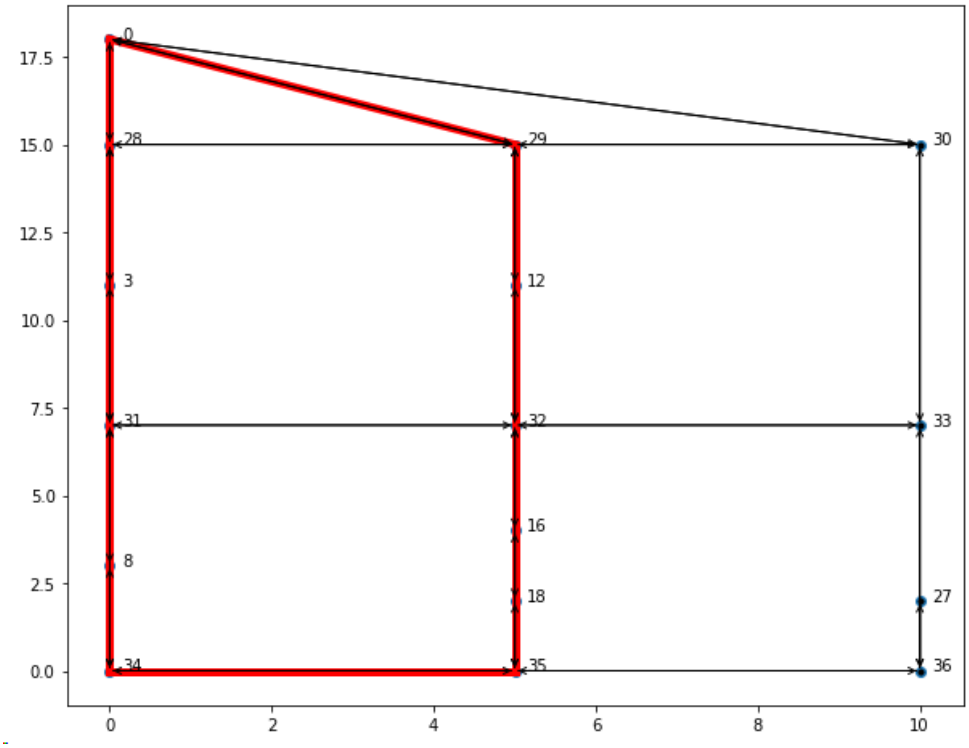


Figura 28: Recorrido del carro t=1 para el ejemplo 3.

Fuente: Elaboración propia.

Como puede observarse en ambas figuras, el almacén se ha reducido al ejecutarse el código, de manera que sólo se analizan los nodos que deben ser visitados para satisfacer los pedidos de clientes, esto relaja la ejecución del código y la representación gráfica pero no afecta al valor de la solución objetivo, ya que en esta solo intervienen las distancias entre nodos que son recalculadas en la función 'red'.

Por último, el valor que toma la función objetivo completa para la solución óptima de este ejemplo es → Obj: 140. Es decir, la distancia total recorrida por el picker será de 140 metros, de los cuales 59 metros los recorre el t=0 y 45 metros los recorre el t=1. Esto tiene fácil comprobación ya que conocemos la matriz de distancias entre todos los nodos y ahora también conocemos los arcos recorridos en la solución para cada uno de los carros. Teniendo todo esto en cuenta se obtiene lo siguiente:

$$\boldsymbol{MIN\left\{\sum_{t\in T}\sum_{(i,j)\in A} Dij * Xtij\right\}} = \mathbf{140\ m}$$

Por tanto, como se comprueba con este ejemplo, esta pequeña mejora permite extrapolar el modelo a almacenes de un tamaño algo superior.

7 CONCLUSIÓN

El problema que se ha abordado en este proyecto trata de optimizar las operaciones del picking en el almacén. Para ser exactos, la agrupación de pedidos de cliente en lotes y la definición de la ruta del picker. Como se menciona en diferentes ocasiones durante el proyecto, este es un problema que engloba operaciones del ámbito de la gestión de almacenes que a priori pueden parecer algo triviales, sin embargo, hoy en día se buscan estratégias óptimas para llevar a cabo estas tareas ya que se ha estudiado que pueden llegar a alcanzar hasta un 60% de los costes de un almacén.

Para tratar de resolver este problema se decidió aplicar un modelo matemático ya que mediante la aplicación del mismo se pueden obtener resultados exactos. No obstante, una vez realizada la revisión bibliográfica del problema en cuestión, se define el problema del JOBPRP (Joint Order Batching and Picker Routing Problem) como un problema complejo, NP-hard. Pues tanto el problema del agrupamiento de pedidos por lotes, como el problema de definición de la ruta del picker en el almacén lo son. Este tipo de problemas se caracterizan por que su complejidad crece exponencialmente con su tamaño.

Para llevar a cabo este estudio se decidió aplicarlo a un almacén de supermercados de compra online. En un primer alcance del proyecto, se abordó un problema de tamaño real, aplicando el modelo a un almacén con un lay-out realista de una gran cadena de supermercados online, con capacidad de almacenar un total de 1584 productos. Sin embargo, la búsqueda de solución para un problema de tal tamaño no consiguió llegar a completarse en ninguno de los intentos, a pesar de modificar levemente el modelo con el fin de buscar solución. No obstante, tras varias reducciones del almacén se llegó a la conclusión de que con la formulación del modelo matemático planteado y las limitaciones de los equipos utilizados, este problema sólo ha sido posible resolverlo para un almacén de tamaño muy reducido, muy lejano al alcance inicial del proyecto. Si bien, para problemas de este tamaño, el modelo es capaz de obtener soluciones óptimas en buenos tiempos computacionales, además se desarrolla la posibilidad de mostrar gráficamente el recorrido, lo que facilitaría en cierta medida la tarea del picker.

Además, como bien se ha llevado a cabo en este proyecto, se observa la posibilidad de reducir el tamaño de cualquier problema real, partiendo del grafo inicial del almacén, a un grafo reducido

que permite no estudiar posiciones que no son necesarias visitar. Esta conclusión mejora considerablemente el modelo matemático en tiempo dedicado a la exploración de nodos, y a su vez, permite extrapolar el modelo a almacenes de mayor tamaño, siendo el limitante la variabilidad de los productos demandados, que se asume que en pocas ocasiones cubrirá el 100% de las posiciones de un almacén.

Con todo esto, se concluye que este problema, sería conveniente resolverlo mediante un método de resolución aproximado, por ejemplo la Búsqueda Tabú, ya que estos aunque no aseguran alcanzar una solución exacta, son capaces de obtener soluciones válidas bastante buenas.

7.2 Lineas de futuro

En relación a estudios futuros relativos a este problema, sería interesante llevar a cabo la resolución del mismo por otros de los métodos de resolución expuestos o bien mediante un modelo matemático diferente. De este modo se podrían comparar resultados y sacar conclusiones sobre la eficiencia de los mismos.

Por otro lado, un aspecto interesante para líneas de futuro sería la variación del problema inicial, por ejemplo: limitar la compatibilidad de productos de distintas familias en un mismo carro, considerar restricciones de peso y/o volumen del carro, asumir heterogeneidad en la flota de carros disponibles, etc. Esto daría una visión interesante y más realista del problema de la recogida de pedidos en el almacén.

8 REFERENCIAS

Albareda-Sambola, M. *et al.* (2009) *Variable Neighborhood Search for Order Batching in a Warehouse 1*. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.545.4598&rep=rep1&type=pdf (Accessed: 8 May 2019).

Azadnia, A. H. *et al.* (2013) 'Order batching in warehouses by minimizing total tardiness: a hybrid approach of weighted association rule mining and genetic algorithms.', *TheScientificWorldJournal*. Hindawi, 2013, p. 246578. doi: 10.1155/2013/246578.

Ballou, R. H. (2004) *Administración de la cadena de suministro*. Available at: https://es.slideshare.net/RafaelSanabria1/ballou-logstica-administracin-de-la-cadena-de-suministro-40033088 (Accessed: 11 April 2018).

Bartholdi, J. J. and Hackman, S. T. (2014) 'Warehouse & Distribution Science'. Available at: https://www2.isye.gatech.edu/~jjb/wh/book/editions/wh-sci-0.96.pdf (Accessed: 11 April 2018).

Cano, J. A., Correa-Espinal, A. A. and Gómez-Montoya, R. A. (2018) 'Solving the Order Batching Problem in Warehouses using Genetic Algorithms', *Informacion Tecnologica*. Centro de Informacion Tecnologica, 29(6), pp. 235–244. doi: 10.4067/S0718-07642018000600235.

Challenger Pérez, I., Díaz Ricardo, Y. and Becerra García, R. A. (2014) 'El lenguaje de programación Python', *Ciencias Holguín*, XX(2), pp. 1–13. Available at: https://www.redalyc.org/pdf/1815/181531232001.pdf (Accessed: 5 October 2019).

Chazallet, S. (2015) *Python 3 : los fundamentos del lenguaje*. ENI. Available at: https://www.ediciones-eni.com/open/mediabook.aspx?idR=9103588b3f4c184b8f2497bcf75605a9 (Accessed: 6 October 2019).

De Armas, J. (2013) *Problemas de corte: métodos exactos y aproximados para formulaciones mono y multi-objetivo.*

De Koster, R., Le-Duc, T. and Roodbergen, K. J. (2007) 'Design and control of warehouse order picking: a literature review', *European Journal of Operational Research*, 182(2), pp. 481–501. Available at: http://roodbergen.com/publications/EJOR2007.pdf (Accessed: 8 May 2018).

Dekker, R. *et al.* (2004) 'Improving Order-Picking Response Time at Ankor's Warehouse', *Interfaces*, 34(4). doi: 10.1287/inte.1040.0083.

Gibson, D. R. and Sharp, G. P. (1992) 'Order batching procedures', *European Journal of Operational Research*. North-Holland, 58(1), pp. 57–67. doi: 10.1016/0377-2217(92)90235-2.

Gómez, R. *et al.* (2016) *Comparative analysis of order batching and routing problem in the picking regarding classical HVRP (hetero-geneous vehicle routing problem), Dirección y Organización*.

González Duque, R. (2008) 'Python para todos'. Available at: https://launchpadlibrarian.net/18980633/Python para todos.pdf (Accessed: 7 October 2019).

Hart, P. E., Nilsson, N. J. and Raphael, B. (1972) 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths', *ACM SIGART Bulletin*, (37), pp. 28–29. doi: 10.1145/1056777.1056779.

Henn, S., Koch, S. and Wäscher, G. (2011) *Order Batching in Order Picking Warehouses: A Survey of Solution Approaches*. Available at: http://www.fww.ovgu.de/femm (Accessed: 8 May 2019).

Henn, S. and Schmid, V. (2011) *Metaheuristics for Order Batching and Sequencing in Manual Order Picking Systems*. Available at: http://www.fww.ovgu.de/femm (Accessed: 8 May 2019).

Hsu, C.-M., Chen, K.-Y. and Chen, M.-C. (2005) 'Batching orders in warehouses by minimizing travel distance with genetic algorithms', *Computers in Industry*, 56(2), pp. 169–178. doi: 10.1016/j.compind.2004.06.001.

Letchford, A. N., Nasiri, S. D. and Theis, D. O. (2012) *Compact Formulations of the Steiner Traveling Salesman Problem and Related Problems*. Available at: https://arxiv.org/pdf/1203.3854.pdf (Accessed: 8 May 2019).

Lin, S. and Kernighan, B. W. (1973) 'An Effective Heuristic Algorithm for the Travelling-Salesman Problem'. Available at: http://160592857366.free.fr/joe/ebooks/ShareData/An Effective Heuristic Algorithm for the Traveling-Salesman Problem.pdf (Accessed: 9 May 2018).

Moarefdoost, M. (2018a) *Applying Gurobi in the Real World - The Opex Analytics Blog - Medium*. Available at: https://medium.com/opex-analytics/applying-gurobi-in-the-real-world-c9c2fd5e7310 (Accessed: 12 October 2019).

Moarefdoost, M. (2018b) *Optimization modeling in Python: Pulp, Gurobi and CPLEX, Medium*. Available at: https://medium.com/opex-analytics/optimization-modeling-in-python-pulp-gurobi-and-cplex-83a62129807a (Accessed: 11 October 2019).

Nowak, M. A. (2005) *The Pickup and Delivery Problem with Split Loads The Pickup and Delivery Problem with Split Loads ACKNOWLEDGEMENTS*.

Order Picking Technologies | Voice | Pick Light | RF | MWPVL (no date). Available at: http://www.mwpvl.com/html/order_pick_technologies.html (Accessed: 19 October 2019).

Peters, T. (2004) *PEP 20 -- The Zen of Python | Python.org*. Available at: https://www.python.org/dev/peps/pep-0020/ (Accessed: 5 October 2019).

Petersen, C. G. (1997) 'An evaluation of order picking routeing policies', *International Journal of Operations & Production Management*. MCB UP Ltd, 17(11), pp. 1098–1111. doi: 10.1108/01443579710177860.

Robinson, D. (2017) *The Incredible Growth of Python | Stack Overflow, Stackoverflow Blog*. Available at: https://stackoverflow.blog/2017/09/06/incredible-growth-python/ (Accessed: 8 October 2019).

Roodbergen, K. J. and De Koster, R. (2001a) 'Routing methods for warehouses with multiple cross aisles', *International Journal of Production Research*, 39(9), pp. 1865–1883. Available at: http://www.roodbergen.com/publications/IJPR2001.pdf (Accessed: 9 May 2018).

Roodbergen, K. J. and De Koster, R. (2001b) 'Routing order pickers in a warehouse with a middle aisle', *European Journal of Operational Research*, 133(1), pp. 32–43. Available at: https://pdfs.semanticscholar.org/da2e/c3690487232da2daf29ed77cd9f85d85d7dc.pdf (Accessed: 22 March 2018).

Sancho Caparrini, F. (2018) *Metaheurísticas para Búsqueda y Optimización (Parte 2) -*. Available at: http://www.cs.us.es/~fsancho/?e=208 (Accessed: 31 October 2019).

Severin, D. E. (2012) *Estudio poliedral y algoritmo branch-and-cut para el problema de coloreo equitativo en grafos*.

Shekhawat, A., Poddar, P. and Boswal, D. (2009) *Ant colony Optimization Algorithms : Introduction and Beyond*.

Theys, C. *et al.* (2010) 'Using a TSP heuristic for routing order pickers in warehouses', *European Journal of Operational Research*. North-Holland, 200(3), pp. 755–763. doi: 10.1016/J.EJOR.2009.01.036.

Toro, L. (2017) *Anaconda Distribution: La Suite más completa para la Ciencia de datos con Python*. Available at: https://blog.desdelinux.net/ciencia-de-datos-con-python/ (Accessed: 7 October 2019).

Universitat Politecnica de Valencia (2012) *Optimización y programación matemática – Proyecto de investigación HORSOST*. Available at: https://horsost.blogs.upv.es/2012/01/07/optimizacion-y-programacion-matematica/ (Accessed: 10 October 2019).

Valle, C. A., Beasley, J. E. and da Cunha, A. S. (2016) 'Modelling and Solving the Joint Order Batching and Picker Routing Problem in Inventories', in. Springer, Cham, pp. 81–97. doi: 10.1007/978-3-319-45587-7_8.

Wäscher, G. and Henn, S. (2010) *Tabu Search Heuristics for the Order Batching Problem in Manual Order Picking Systems OTTO-VON-GUERICKE-UNIVERSITY MAGDEBURG*

FACULTY OF ECONOMICS AND MANAGEMENT Working Paper Series. Available at: http://www.ww.uni-magdeburg.de/ (Accessed: 8 May 2019).

Yang, X.-S. (2009) *Harmony Search as a Metaheuristic Algorithm*. Springer.

9 ANEXOS

Lista de relación productos-posiciones de almacenaje:

id	family	name	Posición
1	Food	Hermanos Mushrooms	92
2	Food	Hermanos Prepared Salad	96
3	Food	Ebony Mandarin Oranges	69
4	Food	Ebony Lemons	66
5	Food	Ebony Plums	72
6	Food	Ebony Red Delcious Apples	73
7	Food	Footnote Seasoned Hamburger	65
8	Food	CDR Grape Jelly	27
9	Food	Carrington Apple Cinnamon Waffles	53
10	Food	BBB Best Strawberry Jam	22
11	Food	Ebony Mixed Nuts	77
12	Food	Applause Canned Mixed Fruit	39
13	Food	Carrington Low Fat Waffles	55
14	Food	Hermanos Asparagus	80
15	Food	CDR Brown Sugar	20
16	Food	Hermanos Cauliflower	84
17	Food	Big Time Frozen Cheese Pizza	60
18	Food	Better Canned Tuna in Oil	37
19	Food	Hermanos Green Pepper	90
20	Food	Blue Medal Small Brown Eggs	51
21	Food	Ebony Limes	67
22	Food	Blue Medal Large Brown Eggs	49
23	Food	Hermanos Red Pepper	97
24	Food	Plato Vegetable Oil	17
25	Food	Carlson Sour Cream	43
26	Food	Better Chicken Noodle Soup	35
27	Food	Fast BBQ Potato Chips	104
28	Food	Carrington Popsicles	59
29	Food	Hermanos Broccoli	83
30	Food	CDR Strawberry Jelly	28
31	Drink	Pearl Chablis Wine	2

32	Food	Carrington Waffles	56
33	Food	Carlson Strawberry Yogurt	45
34	Drink	Super Hot Chocolate	8
35	Food	American Sliced Turkey	47
36	Food	Hermanos Fresh Lima Beans	88
37	Food	Ebony Tangerines	74
38	Food	Hermanos Tomatos	103
39	Food	Hermanos Sweet Peas	102
40	Food	Carlson Blueberry Yogurt	44
41	Food	Best Choice Chocolate Chip Cookies	106
42	Food	CDR Creamy Peanut Butter	30
43	Food	Better Fancy Canned Sardines	33
44	Drink	Excellent Strawberry Drink	5
45	Drink	Excellent Cola	3
46	Drink	BBB Best Hot Chocolate	6
47	Food	CDR Tomato Sauce	18
48	Food	Hermanos Onions	94
49	Food	Blue Medal Small Eggs	52
50	Food	BBB Best Low Fat Apple Butter	31
51	Food	CDR Apple Jam	23
52	Food	Big Time Frozen Sausage Pizza	63
53	Food	Ebony Peaches	71
54	Drink	CDR Hot Chocolate	7
55	Food	Hermanos Lettuce	91
56	Food	Best Choice Frosted Cookies	107
57	Drink	BBB Best Columbian Coffee	9
58	Food	BBB Best Corn Oil	16
59	Food	Ebony Oranges	70
60	Drink	Booker 1% Milk	12
61	Food	Hermanos Beets	82
62	Food	Blue Medal Large Eggs	50
63	Food	Blue Medal Egg Substitute	48
64	Food	American Sliced Ham	46
65	Food	Hermanos Potatos	95
66	Food	Colony English Muffins	14
67	Food	Bravo Turkey Noodle Soup	36
68	Food	Big Time Frozen Pepperoni Pizza	62
69	Food	Ebony Macintosh Apples	68
70	Food	Just Right Fancy Canned Oysters	32

71	Drink	Excellent Orange Juice	10
72	Food	Fast Corn Chips	105
73	Food	Carrington Ice Cream Sandwich	58
74	Food	Hermanos Shitake Mushrooms	98
75	Food	Hermanos Baby Onion	81
76	Food	Colony White Bread	15
77	Food	Ebony Walnuts	79
78	Food	Hermanos Squash	99
79	Food	Carrington Blueberry Waffles	54
80	Food	Hermanos Garlic	89
81	Food	Best Choice Fudge Brownies	108
82	Food	Footnote Extra Lean Hamburger	64
83	Food	Club Low Fat Cottage Cheese	42
84	Food	Ebony Party Nuts	78
85	Drink	Good Imported Beer	1
86	Food	Blue Label Large Canned Shrimp	34
87	Food	Ebony Almonds	75
88	Food	Booker Mild Cheddar Cheese	40
89	Food	Hermanos Dried Mushrooms	86
90	Food	CDR Strawberry Jam	25
91	Food	Hermanos Corn on the Cob	85
92	Food	Hermanos New Potatos	93
93	Food	Better Canned Peas	38
94	Food	Hermanos Summer Squash	100
95	Food	Gorilla Head Cheese	41
96	Food	CDR Apple Jelly	26
97	Food	Modell Bagels	13
98	Drink	Fabulous Apple Juice	11
99	Food	Ebony Canned Peanuts	76
100	Food	Carrington Ice Cream	57
101	Food	CDR White Sugar	21
102	Food	Big Time Frozen Mushroom Pizza	61
103	Food	Hermanos Sweet Onion	101
104	Food	Hermanos Elephant Garlic	87
105	Food	CDR Grape Jam	24
106	Food	CDR Chunky Peanut Butter	29
107	Food	Super Salt	19
108	Drink	Excellent Mango Drink	4

Código en Python, programado para su resolución con el solver de optimización Gurobi:

```
# IMPORTAMOS FUNCIONES

from gurobipy import *
from gurobipy import Model
from gurobipy import GRB
import xlrd
from distancias1 import Dist1
from Loc_prod import Loc_prod
from reducir import red
import matplotlib.pyplot as plt
import numpy as np
from itertools import chain
from itertools import combinations
from powerset import powerset

#DEFINICION DE DATOS#

Max_CestasPorCarro = 3
Max_ItemsPorCesta = 25

# creamos la matriz distancias (DATOS)
book = xlrd.open_workbook("Distancias1.xlsx")
sheet = book.sheet_by_index(0)
Dist = Dist1(book, sheet)

# DATOS ORDENES DESDE EXCEL (num_ordenes,num_items orden, num_productos orden,
num cestas ordenes)

book = xlrd.open_workbook("ORDENPR.xlsx")
sheet1 = book.sheet_by_index(0)
num_ordenes = int(sheet1.cell(2, 0).value)
num_items = []
num_cestas = []
for i in range(4, num_ordenes+4, 1):
    N_prod_ord = int(sheet1.cell(i, 0).value)
    num_items.append([])
    num_cestas.append([])
    C = 0
    for j in range(2, N_prod_ord*2+1, 2):
        C = C+int(sheet1.cell(i, j).value)
        num_items[i-4] = C
        num_cestas[i-4] = C/Max_ItemsPorCesta+1

# Numero de cestas totales: T
C_total = float(0)
for j in range(0, num_ordenes):
```

```
    C_total = C_total+num_cestas[j]

T = float(C_total/Max_CestasPorCarro)+0.2
T_inf = int(C_total/Max_CestasPorCarro+0.2)

if T-T_inf != 0:
    T = int((C_total/Max_CestasPorCarro)+0.2)+1
else:
    T = int((C_total/Max_CestasPorCarro)+0.2

try:
 #creamos el modelo
    m = Model()

# AUXILIARES #

    # nos devuelve una matriz productos_orden[i][j] donde i es la orden y cada j un prod

    book = xlrd.open_workbook("ORDENPR.xlsx")
    sheet1 = book.sheet_by_index(0)
    productos_orden = []
    for i in range(4, num_ordenes+4):
        productos_orden.append([])
        N_prod_ord = int(sheet1.cell(i, 0).value)
        for j in range(1, N_prod_ord*2+1, 2):
            productos_orden[i-4].append(int(sheet1.cell(i, j).value))

    # nos devuelve una lista de vectores [ID_prod][Loc/vertice]

    book = xlrd.open_workbook("Loc1.xlsx")
    sheet1 = book.sheet_by_index(0)
    Loc = []
    k = 0
    col = 0
##ojo##
    while k < 109:
        Loc.append([])
        while col < 2:
            Loc[k].append(int(sheet1.cell(k, col).value))
            col = col+1
        col = 0
        k = k+1
    k = 0

    # Devuelve un vector de vectores de la loc de cada producto para cada orden

Loc_prod = Loc_prod(num_ordenes, productos_orden, Loc)
```

```
    # Devuelve esas localizaciones en un vector total.

    Loc_total = []
    for o in range(0, num_ordenes):
        for i in range(0, len(Loc_prod[o])):
            Loc_total.append(Loc_prod[o][i])

    Loc_total_red = set(Loc_total)
    Loc_total_red=list(Loc_total_red)
    Loc_total_red.sort()
    Loc_total_red.insert(0,0)

    for i in range (28,len (Dist)):
      Loc_total_red.append(i)

    Red=red(Dist,Loc_total_red)
    Dist=Red

#def ejes X e Y

    book = xlrd.open_workbook("Loc_G.xlsx")
    sheet1 = book.sheet_by_index(0)
    Loc_G = []
    k = 0
    col = 0
    while k < 37:
        Loc_G.append([])
        while col < 3:
            Loc_G[k].append(int(sheet1.cell(k, col).value))
            col = col+1
        col = 0
        k = k+1
    k = 0

    # Devuelve una matriz de vectores
    Loc_x=[]
    for l in range(0,len(Loc_total_red)):
        Loc_x.append(Loc_G[Loc_total_red[l]][1])

    Loc_y=[]
    for l in range(0,len(Loc_total_red)):
        Loc_y.append(Loc_G[Loc_total_red[l]][2])


# DEFINICIÓN DE VARIABLES#

    # Z[o,t] (binaria) se activará si el carro t recoge el pedido o
    Z = {}
```

```
    for o in range(0, num_ordenes):
        for t in range(0, T):
            Z[o, t] = m.addVar(vtype=GRB.BINARY, name='Z[%d,%d]' % (o, t))

    # X[t,i,j] (binaria) se activará cuando el carro t recorra el arco (i,j)
    X = {}
    for t in range(0, T):
        for i in range(0, len(Dist)):
            for j in range(0, len(Dist)):
                if Dist [i][j] != 0.0:
                    X[t, i, j] = m.addVar(vtype=GRB.BINARY, name='X[%d,%d,%d]' % (t, i, j))

    # Y[t,i] (binaria) se activará si el nodo i es visitado por el carro t
    Y = {}
    for t in range(0, T):
        for i in range(1, len(Dist)):
            Y[t, i] = m.addVar(vtype=GRB.BINARY, name='Y[%d,%d]' % (t, i))

    # G[t,i] (entero) cantidad de arcos de salida del nodo i para el carro t
    G = {}
    for t in range(0, T):
        for i in range(0, len(Dist)):
            G[t, i] = m.addVar(vtype=GRB.INTEGER, name='G[%d,%d]' % (t, i))
    m.update()

    # E[t,i,j] (binaria) se activará cuando el carro t recorra el arco (i, j) O (j, i)
    E = {}
    for t in range(0, T):
        for i in range(0, len(Dist)):
            for j in range(0, len(Dist)):
                if Dist [i][j] != 0.0 and j < i:
                    E[t, i, j] = m.addVar(vtype=GRB.BINARY, name='E[%d,%d,%d]' % (t, i, j))

# FUNCIÓN OBJETIVO#

    m.update()
    FO = 0
    for t in range(0, T):
        for i in range(0, len(Dist)):
            for j in range(0, len(Dist)):
                if Dist[i][j] == 0.0:
                    continue
                else:
                    FO = FO+Dist[i][j]*X[t, i, j]

    m.setObjective(FO, GRB.MINIMIZE)

# DEFINICION DE RESTRICCIONES#
```

```
# |1|#
# aseguramos que ningún carro excede el num máx de cestas.

for t in range(0, T):
    R1 = 0
    for o in range(0, num_ordenes):
        R1 = R1+num_cestas[o]*Z[o, t]
    m.addConstr(R1, GRB.LESS_EQUAL, Max_CestasPorCarro)

# |2|#
# Aseguramos que un mismo pedido no se separa en distintos carros.

for o in range(0, num_ordenes):
    R2 = 0
    for t in range(0, T):
        R2 = R2+Z[o, t]
    m.addConstr(R2, GRB.EQUAL, 1)

# |3|#
# Aseguramos que el carro que recoge un pedido O, visitara al menos una vez el nodo que contiene cada producto.

for o in range(0, num_ordenes):
    for t in range(0, T):
        for i in range(0, len(Loc_prod[o])):
            R3 = 0
            for j in range(0, len(Dist)):
                if Dist[Loc_total_red.index(Loc_prod[o][i])][j] == 0.0:
                    continue
                else:
                    R3 = R3+X[t, Loc_total_red.index(Loc_prod[o][i]), j]
            m.addConstr(R3, GRB.GREATER_EQUAL, Z[o, t])

# |4|#
# Los carros atraviesan lo nodos i es decir entran y salen
for t in range(0, T):
    for i in range(0, len(Dist)):
        R41 = 0
        R42 = 0
        for j in range(0, len(Dist)):
            if Dist[i][j] == 0.0:
                continue
            else:
                R41 = R41+X[t, i, j]
        for j in range (0,len(Dist)):
            if Dist[j][i] == 0.0:
```

```
                continue
            else:
                R42 = R42+X[t, j, i]
        m.addConstr(R41, GRB.EQUAL, R42)

    # |5|#
    # Si un carro recoge un pedido, el carro parte siempre del origen
    for t in range(0, T):
        for o in range(0, num_ordenes):
            R5 = 0
            for j in range(1, len(Dist)):
                if Dist[0][j] == 0.0:
                    continue
                else:
                    R5 = R5+X[t, 0, j]
            m.addConstr(R5, GRB.GREATER_EQUAL, Z[o, t])

    # |6|#
    # G[t][i] es el conjunto de arcos de salida para cada nodo i dividdps por carros t.

    for i in range(0, len(Dist)):
        for t in range(0, T):
            R6 = 0
            for j in range(0, len(Dist)):
                if Dist[i][j] == 0.0:
                    continue
                else:
                    R6 = R6+X[t, i, j]
            m.addConstr(R6, GRB.EQUAL, G[t, i])

    # |7|#
    # Si existe un arco (i,j) recorrido por t, el carro visita el nodo i.

    for t in range(0, T):
        for i in range(1, len(Dist)):
            for j in range(0, len(Dist)):
                if Dist[i][j] == 0.0:
                    continue
                else:
                    m.addConstr(Y[t, i], GRB.GREATER_EQUAL, X[t, i, j])

#    """ 8
#     Para que exista una sub-ruta al menos un nodo debe tener 2 arcos de salida para un mismo carro t,
#      La suma de los arcos de salida de los nodos debe ser mayor o igual al nº de arcos (j,k) visitados
#    {0,1} para cada nodo estudiado.
```

```
L=list(powerset(range(1,len(Dist))))

for t in range(0, T):
    for l in range (0,len(L)):
        for i in L[l]:
            R81 = 0
            R82 = 0
            for j in L[l]:
                R81 = R81+G[t, j]
                for k in L[l]:
                    if Dist[j][k] == 0.0:
                        continue
                    else:
                        R82 = R82+X[t, j, k]
            m.addConstr(R81, GRB.GREATER_EQUAL, Y[t, i]+R82)

#Añadidas#
for t in range(0, T):
    for i in range(0, len(Dist)):
        for j in range(0, len(Dist)):
            if Dist [i][j] == 0.0 or j > i:
                continue
            else:
                m.addConstr(E[t, i, j], GRB.GREATER_EQUAL, X[t, i, j])
                m.addConstr(E[t, i, j], GRB.GREATER_EQUAL, X[t, j, i])
                m.addConstr(E[t, i, j], GRB.LESS_EQUAL, X[t, i, j] + X[t, j, i])


# 9|10|11|12 #
for t in range(0, T):
    for i in range(0, len(Dist)):
        for j in range(0, len(Dist)):
            if Dist[i][j] == 0.0:
                continue
            else:
                m.addConstr(X[t, i, j],GRB.GREATER_EQUAL,0)
                m.addConstr(X[t, i, j],GRB.LESS_EQUAL,1)

for o in range(0, num_ordenes):
    for t in range(0, T):
        m.addConstr(Z[o, t], GRB.GREATER_EQUAL, 0)

for i in range(0, len(Dist)):
    for t in range(0, T):
        m.addConstr(G[t, i], GRB.GREATER_EQUAL, 0)

for i in range(1, len(Dist)):
    for t in range(0, T):
```

```
            m.addConstr(Y[t, i], GRB.LESS_EQUAL, 1)

    m.update()

    #23# Symmetry breaking constraints:

    for o in range (0, T):
        R23=0
        for t in range(0, num_ordenes):
            try:
                R23=R23 + Z[o, t]
            except:
                pass
        m.addConstr(R23,GRB.GREATER_EQUAL,1)

#RESOLUCIÓN DEL MODELO
    m.update()
    m.optimize()
    if m.status == GRB.Status.OPTIMAL:
        for v in m.getVars():
            if v.x != 0:
                print'%s%g' % (v.varName, v.x)
        print'Obj:%g' % m.objVal

## DIBUJAR RESULTADOS ##
 # nodos #
# Gráfica Y Etiquetar los puntos



    ## DIBUJAR RESULTADOS ##
    for t in range(0,T):
        colors=(0,0,0)
        area = np.pi*3
     # nodos #
        x = Loc_x # valores x
        y = Loc_y # valores y
    # Gráfica Y Etiquetar los puntos
        area = np.pi*3
        x_labels=Loc_total_red
        plt.figure(figsize=(10,8))
        fig, ax = plt.subplots(figsize=(10,8))
        ax.scatter(x, y)
        for i, txt in enumerate(x_labels):
            ax.annotate(txt, (x[i], y[i]),(x[i]+0.15, y[i]))
        plt.scatter(x, y, s=area, c=colors, alpha=1)

    # vectores #
        for i in range(0, len(Dist)):
```

```
        for j in range(0, len(Dist)):
            if Dist[i][j]!=0:
                ax.annotate("",
                xy=(x[i], y[i]), xycoords='data',
                xytext=(x[j], y[j]), textcoords='data',
                arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3"),
                )

    # solución #

        for i in range(0, len(Dist)):
            for j in range(0, len(Dist)):
                if Dist [i][j] != 0.0 and j < i:
                    if E[t,i,j].x == 1.0:

                        plt.plot((x[i],x[j]),(y[i],y[j]),c='r',linewidth=5)
                else:
                    pass

    plt.show()
except GurobiError:
    print('Error reported')
```

Funciones auxilliares de elaboración propia:

```
### Distancias1 ###

def Dist1 (book,sheet):
    Dist=[]
    nodoi=0
    nodoj=0
    while nodoi<37:
        Dist.append([])
        while nodoj<37:
            Dist[nodoi].append(0)
            nodoj=nodoj+1
        nodoj=0
        nodoi=nodoi+1
    nodoi=0
    # rellena la matriz con datos:
    nodoi=0
    nodoj=0
    while nodoi<37:
        while nodoj<37:
            Dist[nodoi][nodoj]=sheet.cell(nodoi,nodoj).value
```

```
            nodoj=nodoj+1
        nodoj=0
        nodoi=nodoi+1
    nodoi=0
    return Dist


### Localización productos ###

def Loc_prod (num_ordenes,productos_orden,Loc):
    aux=[]
    for l in range(0,num_ordenes):
        aux.append([])
        for k in range(0,len(productos_orden[l])):
            for j in range(0,1650):
                try:
                    aux[l].append(Loc.index([productos_orden[l][k],j]))
                except:
                    pass

    # Devuelve una matriz de vectores
    Loc_prod=[]
    for l in range(0,num_ordenes):
        Loc_prod.append([])
        for k in range(0,len(productos_orden[l])):
            Loc_prod[l].append(Loc[aux[l][k]][1])
    return Loc_prod
```