Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Resolución y análisis de modelos de planificación logística

Autor: Irene Santacruz Olías

Tutor: Jesús Muñuzuri Sanz

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

Sevilla, 2023

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Resolución y análisis de modelos de planificación logística

Autor:
Irene Santacruz Olías

Tutor:
Jesús Muñuzuri Sanz

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

Sevilla, 2023

Trabajo Fin de Grado: Resolución y análisis de modelos de planificación logística

Autor: Irene Santacruz Olías
Tutor: Jesús Muñuzuri Sanz

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Resumen

En este proyecto se analiza la viabilidad de la resolución exacta de tres modelos de planificación logística utilizando las herramientas OR-Tools y Gurobi. Para ello, se aplica la misma metodología a los tres modelos. En primer lugar, se analiza el problema a resolver, así como el modelo matemático utilizado para su formulación. Una vez determinado el modelo, se establecen los pasos a seguir para la resolución exacta mediante alguna de las herramientas mencionadas. Posteriormente, se llevan a cabo experimentos computacionales utilizando datos generados de manera aleatoria. Por último, se realiza un análisis del esfuerzo computacional requerido y se reflexiona sobre la viabilidad de implementación de estas soluciones en situaciones reales.

Permite, por tanto, obtener información sobre la aplicabilidad y eficiencia de las técnicas de resolución exactas en la planificación logística.

Índice

1 Introducción y Objeto del Trabajo

En los últimos años, la búsqueda del aumento de la productividad y de la satisfacción de los clientes se ha convertido en una prioridad para todas las empresas de suministro de productos. Esto se debe al notable aumento tanto de las expectativas de los clientes como de la competencia entre las diferentes empresas.

En este contexto, el diseño de la red logística toma un papel fundamental, ya que implica la coordinación de la cadena de suministro integrando todas las etapas, desde la adquisición de materias primas hasta la entrega de productos finales a clientes. Mediante la optimización del diseño de la red logística y de las relaciones entre las diferentes etapas, las empresas pueden obtener una ventaja significativa respecto a sus competidores, asegurando tanto su capacidad para cumplir con las expectativas del mercado como la minimización de los costes para ello.

La optimización logística es, por tanto, un proceso complejo que engloba una gran variedad de decisiones para garantizar la gestión eficiente de la cadena de suministro. Estas se pueden clasificar en tres niveles diferentes: estratégicas, tácticas y operativas.

Las decisiones **estratégicas** constituyen el nivel más alto en la toma de decisiones y se centran en cuestiones fundamentales que afectan a la estructura de la cadena de suministro a largo plazo. Estas decisiones incluyen la determinación de la ubicación de las plantas de fabricación y los almacenes, así como la evaluación de la capacidad y cantidad necesaria de cada uno de ellos. Debido a que suponen un gran cambio en la estructura de la red, estas decisiones se planifican a largo plazo, teniendo un horizonte temporal de varios años.

Por otro lado, las decisiones **tácticas** se centran en aspectos relacionados con la elección de los proveedores, la asignación de los productos a las distintas plantas de fabricación y la elección del modo de distribución adecuado. Estas decisiones se toman con una frecuencia mayor y pueden ser analizadas y modificadas cada pocos meses. Sin embargo, siempre se deben tener en cuenta los acuerdos previos establecidos con los proveedores y las plantas de fabricación.

Por último, las decisiones **operativas** se refieren a la gestión diaria del flujo de materias primas y productos terminados. Estas decisiones se toman a corto plazo y están relacionadas con actividades como la programación de la producción, la gestión de inventarios, la planificación de rutas de transporte, etc.

Para la toma de estas decisiones, la investigación operativa desempeña un papel fundamental, ya que proporciona herramientas y enfoques analíticos que permiten mejorar significativamente la eficiencia de los procesos y optimizar el uso de los recursos disponibles.

A través del uso de técnicas de modelado, es posible representar con precisión los diferentes componentes de la cadena de suministro y las interacciones entre ellos. Partiendo de estos modelos, la investigación operativa emplea métodos de optimización y algoritmos avanzados para analizar diferentes escenarios y tomar decisiones fundamentadas enfocadas en maximizar la eficiencia, al mismo tiempo que se minimizan los costes logísticos.

En resumen, la capacidad de la investigación operativa para analizar datos, modelar escenarios y optimizar procesos es fundamental a la hora de tomar decisiones estratégicas, tácticas y operativas. Aprovechando estas técnicas, se pueden alcanzar ventajas competitivas y lograr una mejora general en la gestión de la cadena de suministro.

1.1 Objeto del trabajo

Debido a la importancia de las decisiones mencionadas, resulta de interés analizar varios modelos que optimicen diferentes aspectos de la red logística. Además, este enfoque a través de la investigación operativa resulta muy interesante para determinar la capacidad de resolución que tenemos de estos.

El objeto de este trabajos es estudiar varios modelos de programación lineal aplicados a la logística y a la cadena de suministro. En particular, se analizarán y resolverán los modelos propuestos por Cordeau et al. (2006), Sadjadi et al. (2009) y Kim et al. (2015). Este análisis consiste en la resolución de los modelos evaluando el tiempo de computación requerido para ello, con la finalidad de determinar tanto el tamaño de los modelos que se pueden resolver de forma exacta como la capacidad de implementación de estos modelos en situaciones reales.

En primer lugar, se resolverán diversos casos para cada modelo. Una vez resueltos, se llevará a cabo un análisis detallado del tiempo y esfuerzo computacional requeridos para la obtención de una solución exacta. Este análisis permitirá evaluar la viabilidad de la resolución exacta en función del tamaño de los modelos, así como determinar el rendimiento de las herramientas utilizadas en la resolución.

Los resultados obtenidos pueden ser de interés para tomar decisiones sobre la implementación de dichas herramientas en casos reales de gestión de la cadena de suministro.

2 Herramientas de resolución

Para la resolución de los problemas de optimización que se van a plantear en este trabajo, se ha decidido utilizar dos softwares diferentes: Gurobi y OR-Tools. Dos softwares muy conocidos y utilizados en el ámbito de la investigación operativa. Estos softwares son considerados unos de los mejores del mercado debido a su eficiencia y su capacidad para la resolución de problemas de optimización en diversas industrias y aplicaciones.

La elección final de cada uno de estos dos softwares depende de varios factores, como el tipo de problema a abordar y las características del modelo. Cada software tiene sus propias fortalezas y especialidades, lo que los hace adecuados para diferentes escenarios y requisitos.

2.1 Gurobi

Gurobi es un solver líder en el campo de la optimización matemática, siendo capaz de resolver diversos tipos de problemas: programación lineal (LP), programación lineal mixta (MILP), programación cuadrática (QP), etc. Este destaca por su capacidad para resolver modelos de optimización de gran tamaño y complejidad, lo que hace que sea usado tanto en el ámbito de la investigación como en el sector empresarial.

Una de las características por las que destaca Gurobi es por su enfoque en la eficiencia y en la velocidad de resolución de los modelos. Lo que resulta de especial interés para problemas logísticos que requieren respuestas rápidas y precisas. Además de su potencia de cálculo, Gurobi ofrece una amplia variedad de herramientas de análisis y visualización que facilitan la comprensión y análisis de los resultados.

A diferencia de otros softwares de optimización matemática, Gurobi no tiene una interfaz gráfica de usuario. En su lugar, se interactúa con él a través de lenguajes de programación como C++, Java, R, MATLAB o Python, de manera que el solver pueda interpretar y resolver los modelos. En este caso, se ha decidido utilizar Python como lenguaje de programación principal debido a la gran flexibilidad que ofrece al modelar el problema. Además, permite el uso de una amplia variedad de bibliotecas y herramientas adicionales, facilitando así la creación de los modelos de optimización.

A pesar de que Gurobi ofrece la posibilidad de utilizar el Símbolo del Sistema (CMD) para resolver modelos escritos en archivos con un formato determinado, en este caso se utilizará Spyder, un entorno de desarrollo integrado para programación en lenguaje Python debido a sus ventajas en términos de funcionalidad y facilidad de uso. Spyder ofrece muchas herramientas y características diseñadas para el desarrollo en Python. Además, su interfaz es intuitiva y fácil de usar, permitiendo escribir, depurar y ejecutar el código de manera eficiente. Lo que es especialmente beneficioso dada la complejidad de los modelos que se abordarán en este estudio.

La elección de Gurobi y Python se ha realizado considerando las necesidades específicas del estudio y las capacidades de los softwares disponibles. Se espera que esta combinación de herramientas proporcione resultados precisos y fiables, además de facilitar el análisis de los modelos de optimización en el ámbito de la gestión de la cadena de suministro.

2.2 OR-Tools

OR-Tools es un software de optimización desarrollado por Google que ofrece una amplia variedad de solvers y herramientas para resolver diversos tipos de problemas de optimización. A diferencia de Gurobi, OR-Tools es un software completamente gratuito y de código abierto, lo que lo hace accesible para cualquier persona o empresa que necesite utilizarlo sin necesidad de pagar una licencia comercial.

Una de las áreas en las que OR-Tools destaca es en la resolución de problemas de enrutamiento de vehículos (VRP). Ofrece una gran variedad de algoritmos y técnicas de resolución diseñadas específicamente para abordar este tipo de problemas, lo que permite encontrar la mejor solución posible en cada situación. Por tanto, en este estudio, usaremos OR-Tools para resolver los problemas de enrutamiento de vehículos que se presenten.

Además de los problemas VRP, OR-Tools es capaz de resolver otros tipos de problemas, incluyendo programación lineal (LP), programación lineal mixta (MILP), programación de restricciones (CP), etc. Esta versatilidad amplía el alcance de uso de OR-Tools y lo convierte en una herramienta útil para abordar una gran variedad de problemas de optimización.

Al igual que Gurobi, se puede trabajar a través de varios lenguajes de programación como C++, Java, Pyhton, etc. Siguiendo las razones mencionadas anteriormente, se ha decidido utilizar Python a través del entorno de desarrollo Spyder. Python ofrece una sintaxis clara y concisa, lo que facilita la modelización de los problemas. Spyder, por su parte, proporciona muchas herramientas que facilitan el desarrollo en Python.

En resumen, OR-Tools destaca por su amplia variedad de solvers y herramientas, su gratuidad y su enfoque en problemas de enrutamiento de vehículos.

2.3 Especificaciones del hardware

Es importante tener en cuenta que el tiempo y esfuerzo de computación necesarios para resolver los modelos puede variar significativamente según las características del hardware utilizado en las diferentes pruebas. Por lo tanto, es necesario determinar las especificaciones del hardware que se utilizará en este estudio. A continuación, se detallan las especificaciones del hardware seleccionado para llevar a cabo las pruebas:

- Procesador: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz
- RAM instalada: 12,0 GB (11,9 GB usable)
- Tipo de sistema: Sistema operativo de 64 bits, procesador basado en x64
- Disco local: SSD 500GB

3 Modelo integrado para el diseño de redes logísticas

El primer problema que se aborda en el proyecto es el descrito por Cordeau et al. (2006). En dicho artículo, se expone tanto el problema que se busca resolver como la formulación del modelo matemático.

3.1 Descripción del problema a resolver

El problema que se plantea es el diseño de la red logística de una empresa de producción que opera dentro de un único país. La estructura logística está compuesta por un conjunto de proveedores, plantas de producción y almacenes de productos terminados coordinados para gestionar de manera eficiente el suministro de materias primas, la transformación de estas en productos terminados y la distribución de dichos productos a los consumidores finales. En general, el diseño de una red logística o cadena de suministro implica la toma de decisiones sobre los siguientes aspectos:

- El número, ubicación, capacidad y tecnología de las plantas de fabricación y los almacenes
- La selección de proveedores
- La asignación de productos a las diferentes plantas de fabricación y almacenes
- La selección de los canales de distribución y los modos de transporte entre las diferentes localizaciones
- La gestión de los flujos de materias primas, productos semiacabados y acabados a lo largo de toda la red logística

Teniendo en cuenta su importancia y el horizonte de planificación necesario, estas decisiones se pueden clasificar en las tres categorías descritas en el Capítulo 1. En primer lugar, se consideran decisiones estratégicas aquellas relativas a la ubicación, la capacidad y la tecnología de las fábricas y almacenes. En segundo lugar, la selección de proveedores, la asignación de los productos y la elección del canal de distribución y del modo de transporte pertenecen al nivel táctico y deben revisarse cada pocos meses. Por último, las decisiones relacionadas con la gestión de flujos de materias primas, productos semiacabados y acabados a lo largo de toda la red pertenecen a las decisiones operativas.

El problema de diseño de la red logística (LNDP) implica tomar las decisiones anteriores con el objetivo de satisfacer la demanda de los clientes, minimizando los costes fijos y variables asociados al suministro, la producción, el almacenamiento y el transporte. Debido a su complejidad, a menudo se descompone en varios componentes que se abordan de manera independiente. Por ejemplo, es posible separar las decisiones estratégicas, tácticas y operativas o dividir la red en diferentes

partes según las categorías de productos o las consideraciones geográficas. No obstante, debido a la importancia de las interacciones entre estas decisiones, se pueden obtener importantes beneficios al tratar la red como un todo y considerar sus distintos componentes al mismo tiempo.

3.2 Formulación matemática del modelo

Aunque el objetivo principal del estudio sea el análisis de los tiempos de ejecución obtenidos en los diferentes casos, es esencial conocer el modelo que se está evaluando para poder comprender los resultados obtenidos de una manera más completa. Por tanto, se va a describir detalladamente el modelo matemático obtenido del artículo anteriormente mencionado (Cordeau et al., 2006), incluyendo información sobre los conjuntos involucrados, la función objetivo y las restricciones.

La descripción detallada del modelo permitirá tener una visión más clara de la naturaleza del problema que se está abordando, lo que a su vez facilitará la comprensión de las decisiones tomadas en el diseño del estudio y la evaluación de los resultados obtenidos.

3.2.1 Conjuntos

En primer lugar, hay que determinar los conjuntos del problema, es decir, identificar y clasificar todos los elementos que forman parte de la cadena de suministro. Este es un paso esencial ya que permitirá establecer relaciones y restricciones más adelante. A la vez que se identifican, se establece la notación que se seguirá a lo largo del documento.

Tabla 3.1 Notación de los conjuntos.

Conjunto	Descripción
F	Productos terminados f
R	Materias primas r usadas en el proceso de fabricación de los productos
S	Proveedores de la empresa s
P	Plantas disponibles p
W	Almacenes disponibles w
C	Clientes c
K=R$\cup F$	Mercancías totales
O=S$\cup P \cup W$	Posibles orígenes de las mercancías
D=P$\cup W \cup C$	Posibles destinos de las mercancías
M	Modos de transporte m

Cabe destacar que cada conjunto se representará mediante una letra mayúscula, mientas que los elementos específicos que pertenecen a cada conjunto se indicarán con una letra minúscula.

Asimismo, cabe resaltar que, en algunos casos, las mercancías o clientes pueden agruparse en familias debido a su similitud y ser tratadas como un único producto o cliente para facilitar la planificación.

3.2.2 Subconjuntos

Los conjuntos identificados anteriormente están estrechamente relacionados entre sí. Para reflejar estas relaciones, se identifican los siguientes subconjuntos:

Se va a tener en cuenta que los modos de transporte disponibles solo dependen del origen y del destino, no dependiendo de la mercancía transportada.

Tabla 3.2 Notación de los subconjuntos.

Subconjunto	Descripción
$F^r \subseteq F$	Productos terminados que necesitan materia prima r
$S^r \subseteq S$	Proveedores que pueden suministrar la materia prima r
$P^f \subseteq P$	Plantas que pueden fabricar el producto f
$W^f \subseteq W$	Almacenes que pueden guardar el producto f
$C^f \subseteq C$	Clientes que demandan el producto f
$O^k \subseteq O$	Posibles orígenes de la mercancía k
$D^k \subseteq D$	Posibles destinos de la mercancía k
$M_{od} \subseteq M$	Modos de transporte entre o y d

3.2.3 Parámetros

Una vez definidos todos los conjuntos y subconjuntos que integran la red logística, se detallan todos los parámetros que pueden variar en función de las características de cada caso.

Tabla 3.3 Notación de los parámetros.

Parámetro	Descripción
a_c^f	Demanda del cliente c del producto f
b^{rf}	Cantidad de materia prima r necesaria para fabricar el producto f
c_o	Coste fijo de seleccionar el origen o
c_o^k	Coste fijo de asignar la mercancía k al origen o
c_{od}^k	Coste fijo de llevar la mercancía k desde el origen o hasta el destino d
c_{od}^m	Coste fijo de usar el modo de transporte m entre o y d
c_{od}^{km}	Coste unitario de llevar la mercancía k desde o hasta d con el modo de transporte m
g_{od}^m	Capacidad del modo de transporte m entre o y d
g^{km}	Capacidad requerida por una unidad k en el modo de transporte m m
q_o^k	Límite superior de la cantidad de mercancía k enviada desde o
q_{od}^k	Límite superior de la cantidad de mercancía k enviada entre o y d
u_o	Capacidad del origen o en unidades equivalentes
u_o^k	Capacidad requerida por cada unidad de k en el origen o

3.2.4 Variables

Aunque en este caso no se va a estudiar el valor que toman las variables de decisión del modelo, hay que definirlas para poder comprenderlo. Estas variables pueden ser de tipo entero o binario. Las variables y sus tipos son las siguientes:

Tabla 3.4 Notación de las variables.

Variable	Tipo	Descripción
X_{od}^{km}	Entera	Cantidad de mercancía k enviada desde o hasta d en el modo de transporte m
U_o	Binaria	Toma valor 1 si el origen o es seleccionado al menos una vez
V_o^k	Binaria	Toma valor 1 si la mercancía k se asigna al origen o
Y_{od}^k	Binaria	Toma valor 1 si el origen o suministra k al destino d
Z_{od}^m	Binaria	Toma valor 1 si el modo m es seleccionado entre o y d

3.2.5 Función objetivo y restricciones

Una vez definidos todos los elementos que componen el modelo de optimización, es posible escribirlo. El modelo es un problema de minimización que cuenta con ocho restricciones.

Es importante mencionar que en el artículo de Cordeau et al. (2006) se describen varias restricciones y modificaciones que pueden ser implementadas para adaptar el modelo a otras situaciones. Entre estas modificaciones, se incluyen la fijación de límites inferiores de adquisición para cada proveedor, la satisfacción de la demanda de los clientes desde un único almacén y la imposición de límites superiores e inferiores en el número de plantas. Aunque estos cambios son interesantes y pueden ser de utilidad en determinadas situaciones, en este proyecto se tratará el modelo básico descrito a continuación.

$$\text{mín} \sum_{o\in O}\left[c_o U_o + \sum_{d\in D}\sum_{m\in M_{od}} c_{od}^m Z_{od}^m\right] + \sum_{k\in K}\sum_{o\in O^k}\left[c_o^k V_o^k + \sum_{d\in D^k}\left[c_{od}^k V_{od}^k + \sum_{m\in M_{od}^k} c_{od}^{km} X_{od}^{km}\right]\right] \tag{3.1}$$

s.a

$$\sum_{s\in S^r}\sum_{m\in M_{sp}^r} X_{sp}^{rm} - \sum_{f\in F^r}\sum_{w\in W^f}\sum_{m\in M_{pw}^f} b^{rf} X_{pw}^{fm} = 0 \qquad r\in R; p\in P \tag{3.2}$$

$$\sum_{p\in P^f}\sum_{m\in M_{pw}^f} X_{pw}^{fm} - \sum_{c\in C^f}\sum_{m\in M_{wc}^f} X_{wc}^{fm} = 0 \qquad f\in F; w\in W^f \tag{3.3}$$

$$\sum_{w\in W^f}\sum_{m\in M_{wc}^f} X_{wc}^{fm} = a_c^f \qquad f\in F; c\in C^f \tag{3.4}$$

$$\sum_{k\in K}\sum_{d\in D^k}\sum_{m\in M_{od}^k} u_o^k X_{od}^{km} - u_o U_O \leqslant 0 \qquad o\in O \tag{3.5}$$

$$\sum_{d\in D^k}\sum_{m\in M_{od}^k} X_{od}^{km} - q_o^k V_o^k \leqslant 0 \qquad k\in K;\ o\in O^k \tag{3.6}$$

$$\sum_{m\in M_{od}^k} X_{od}^{km} - q_{od}^k Y_{od}^k \leqslant 0 \qquad k\in K;\ o\in O^k;\ d\in D^k \tag{3.7}$$

$$\sum_{k\in K} g^{km} X_{od}^{km} - g_{od}^m Z_{od}^m \leqslant 0 \qquad d\in D;\ o\in O; m\in M_{od} \tag{3.8}$$

$$X_{od}^{km} \geq 0 \quad k\in K; o\in O^k;\ d\in D^k; m\in M_{od}^k \tag{3.9}$$

$$U_o \in \mathbb{B} \quad o\in O \tag{3.10}$$

$$V_o^k \in \mathbb{B} \quad k\in K; o\in O^k \tag{3.11}$$

$$Y_{od}^k \in \mathbb{B} \quad k\in K; o\in O^k; d\in D^k \tag{3.12}$$

$$Z_{od}^m \in \mathbb{B} \quad o\in O; d\in D; m\in M_{od} \tag{3.13}$$

La función objetivo, representada por la ecuación 3.1, minimiza los costes totales, tanto fijos como variables. El primer sumando refleja los costes derivados de la selección de los orígenes y de

los modos de transporte utilizados. En el segundo término se recogen los costes derivados de las mercancías específicas asignadas y transportadas.

Las restricciones establecen las siguientes relaciones entre las variables y parámetros del problema:

- Restricción 3.2: Establece que la cantidad de materia prima total enviada a cada planta para la fabricación de los productos corresponda con la cantidad de materia prima requerida para la fabricación de los productos que se hacen en esa planta. Para ello, para cada planta y materia prima, se iguala la cantidad de materia prima enviada desde cualquier proveedor a esa planta con la cantidad de materia prima necesaria para fabricar todos los productos que se envían desde esa planta.

- Restricción 3.3: Asegura que todos los productos finales que entran en un almacén también salen de él. Para cada producto y almacén, el primer sumando corresponde a la cantidad de producto que sale de cualquier planta y llega al almacén, mientras que el segundo sumando corresponde con la cantidad de producto que sale del almacén debido a la demanda de los clientes. Por tanto, la resta de ambos debe ser cero.

- Restricción 3.4: Esta restricción impone que se satisfaga la demanda de todos los clientes. Para ello, se iguala la demanda de los clientes a la cantidad de productos finales que se les envían.

- Restricción 3.5: Para cada origen (proveedores, plantas y almacenes), hace que no se incumplan los límites de capacidad. Para ello, la capacidad requerida por las unidades que pasan por ese origen debe ser menor que la capacidad total del origen en unidades equivalentes. Es decir, su resta debe ser menor o igual que cero.

- Restricción 3.6: Al igual que la restricción anterior, impone límites de capacidad, pero en este caso de la mercancía enviada desde cada origen. La cantidad de productos enviados debe ser menor que el límite superior de cantidad de producto que se puede enviar desde ese origen.

- Restricción 3.7: Impone que se cumplan los límites de la cantidad de producto k que puede ser enviada desde un origen hasta un destino.

- Restricción 3.8: Para cada modo de transporte que se use entre dos ubicaciones, se impone que se cumplan los límites de capacidad de este. Es decir, la capacidad requerida para la totalidad de productos transportados debe ser menor que la capacidad total de ese modo de transporte.

- Restricción 3.9: Asegura que la variable entera tome valores positivos.

El resto de restricciones determinan el dominio de las variables restantes.

3.3 Resolución mediante Gurobi

En los siguientes párrafos, se describe el procedimiento seguido para modelar y resolver el problema mediante el lenguaje de programación Python. Aunque el código completo se presenta en el Apéndice A, en esta sección se explican detalladamente algunas partes que resultan de especial interés para la comprensión del proceso seguido.

En primer lugar, es necesario importar varias librerías esenciales para la implementación y resolución del modelo. La primera de ellas es la de Gurobi, que permite hacer uso del solver así

como de diversas funciones para crear el modelo. La segunda librería que se ha importado es la librería *random*, necesaria para generar los valores de los parámetros automáticamente. Al requerir analizar múltiples casos, es mucho más eficiente generarlos de forma aleatoria. Por último, se ha importado la librería os, que es necesaria para guardar los datos relevantes de la resolución de estos casos. Con esta librería, se pueden almacenar los datos en archivos, lo que facilita el posterior análisis de la información.

En el Apéndice D se muestran las principales funciones utilizadas de la API de Gurobi para Python junto con una breve descripción de estas. Todas las funciones disponibles se encuentran en la página web de Gurobi (`https://www.gurobi.com/documentation/9.5/refman/py_python_api_details.html`) .

Después de importar las librerías necesarias, se puede empezar a definir el modelo. En primer lugar, se establecen los conjuntos que componen la cadena logística. Estos conjuntos se relacionan entre sí, dando lugar a subconjuntos. Cómo determinar, por ejemplo, que todos los proveedores pueden suministrar todas las materias primas daría lugar a un modelo irreal, se generan los siguientes números:

- Número de proveedores que pueden suministrar cada materia prima.
- Número de plantas de fabricación que pueden fabricar cada producto.
- Número de almacenes que pueden almacenar cada producto
- Número de materias primas necesarias para fabricar cada producto
- Número de clientes que tienen una demanda positiva de cada producto

Una vez definidos estos números, se pueden crear los subconjuntos de manera aleatoria usando la función *random.sample*, que devuelve una lista de n elementos extraídos sin repetición de otra lista. Cabe destacar que los orígenes y los destinos (O_k *y* D_k) no hace falta definirlos, ya que están conformados por los demás subconjuntos. Únicamente se debe establecer el destino de las materias primas, siendo este las plantas que pueden fabricar los productos que necesitan esa materia prima.

Para determinar los modos de transporte entre cada origen y destino, se asume que entre los proveedores, plantas y almacenes se utiliza un único medio de transporte. Sin embargo, para el transporte desde los almacenes a los clientes, el número de medios de transporte disponibles es mayor. Esto representa un transporte a carga completa al principio de la cadena, en el que se usa el mismo medio de transporte para todos los movimientos, y un transporte de carga compartida al final, en el que hay varias opciones de transporte.

A continuación, se establecen los valores de los parámetros y datos del modelo. Estos se generan aleatoriamente, pero están acotados entre dos valores acorde al tamaño de cada problema. Para almacenar de forma ordenada estos valores se utiliza la función *multidict*, función exclusiva de Gurobi que permite almacenar en un mismo diccionario varios valores asociados a cada clave. Además, gracias a los subconjuntos generados anteriormente, se puede tener en cuenta los posibles orígenes y destinos de cada mercancía evitando así generar datos innecesarios.

Una vez definidos todos los datos, es posible introducir las variables, las restricciones y la función objetivo del problema utilizando las funciones específicas para ello. Tras esto, se procede a optimizar el modelo definido.

Por último, se guarda tanto el modelo como los resultados obtenidos para su posterior uso. Aunque los modelos no son imprescindibles para el posterior análisis que se va a realizar, no implica ningún inconveniente guardarlos por si hicieran falta para realizar alguna comprobación. El modelo se almacena en un archivo con extensión *.lp* y los resultados obtenidos se guardan en un archivo de texto.

3.3.1 Análisis de la solución de un ejemplo simple

Para verificar la correcta implementación del modelo de optimización en Pyhton, se llevará a cabo una prueba con un problema reducido compuesto por 3 clientes, 2 proveedores, 2 plantas, 1 almacén,

2 materias primas y 2 productos. En este escenario, se tendrá un posible proveedor para cada materia prima, y cada producto podrá ser fabricado en las dos plantas y almacenado en un almacén. Cada producto se fabricará utilizando dos materias primas distintas y se espera que cada producto tenga demanda positiva de dos de los clientes. En cuanto a los medios de transporte, se determina que hay dos disponibles.

La resolución de este problema permitirá comprobar que todas las restricciones del modelo se cumplen correctamente. En la Figura 3.1 se muestra un pequeño esquema del flujo de materiales de la solución obtenida.

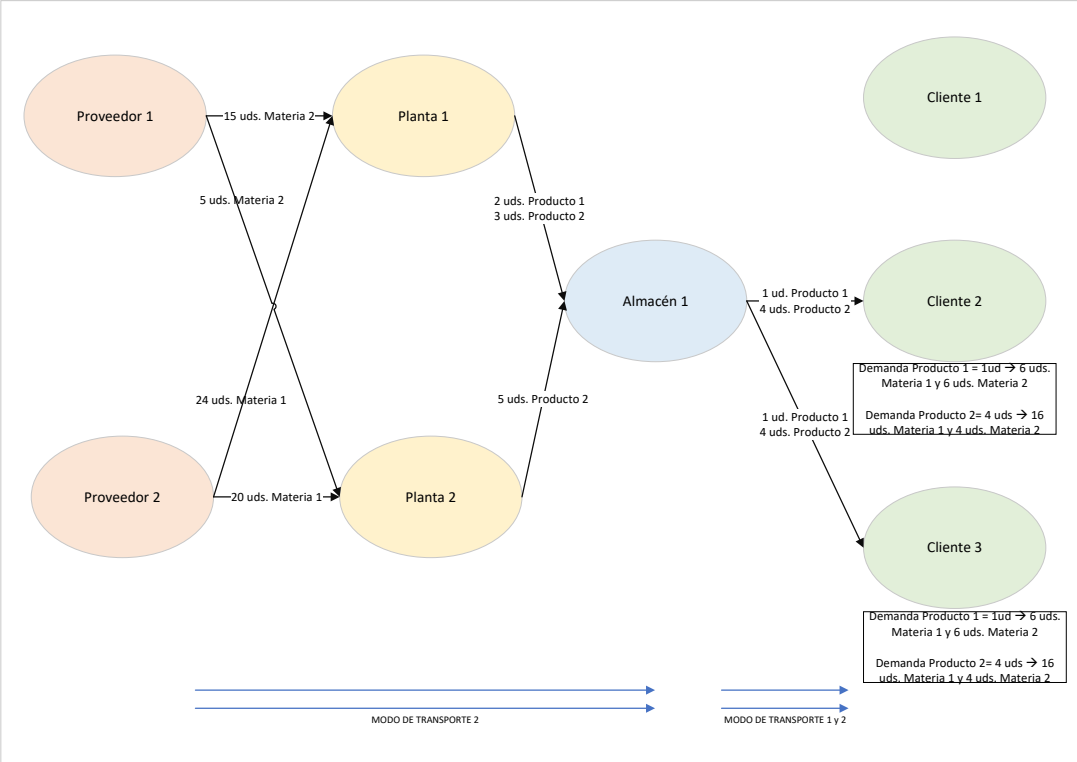

Figura 3.1 Flujo de materiales ejemplo simple.

Las variables binarias que toman valor 1 son las que se muestran en la Tabla 3.5. Comparándolo con el flujo de materiales obtenidos, se puede observar que estas variables toman los valores esperados.

Tabla 3.5 Variables binarias no nulas ejemplo simple.

U_o	V_o^k	Y_{od}^k	Z_{od}^m
U[Plant1]	V[Supplier2,Raw1]	Y[Supplier2,Plant1,Raw1]	Z[Supplier1,Plant1,Mode2]
U[Plant2]	V[Supplier1,Raw2]	Y[Supplier2,Plant2,Raw1]	Z[Supplier1,Plant2,Mode2]
U[Supplier1]	V[Plant1,Product1]	Y[Supplier1,Plant1,Raw2]	Z[Supplier2,Plant1,Mode2]
U[Supplier2]	V[Plant2,Product2]	Y[Supplier1,Plant2,Raw2]	Z[Supplier2,Plant2,Mode2]
U[Warehouse1]	V[Plant1,Product2]	Y[Plant1,Warehouse1,Product1]	Z[Plant1,Warehouse1,Mode2]
	V[Warehouse1,Product1]	Y[Plant2,Warehouse1,Product2]	Z[Plant2,Warehouse1,Mode2]
	V[Warehouse1,Product2]	Y[Plant1,Warehouse1,Product2]	Z[Warehouse1,Client2,Mode1]
		Y[Warehouse1,Client2,Product1]	Z[Warehouse1,Client2,Mode2]
		Y[Warehouse1,Client3,Product1]	Z[Warehouse1,Client3,Mode1]
		Y[Warehouse1,Client3,Product2]	
		Y[Warehouse1,Client2,Product2]	

En relación a las restricciones del modelo, tan solo observando el flujo de materias primas y

productos, se puede verificar que se cumplen las restricciones 2.2, 2.3 y 2.4, relacionadas con la demanda y la producción. En las tablas Tabla 3.6 y Tabla 3.7 , se muestran los datos relacionados con la capacidad. Con ellos se puede comprobar que se cumplen el resto de restricciones.

Tabla 3.6 Datos ejemplo simple.

g^{km}	g^{m}_{od}	u_o
('Raw1', 'Mode1'): 2	('Supplier1', 'Plant1', 'Mode2'): 103	'Plant1': 23
('Raw1', 'Mode2'): 1	('Supplier1', 'Plant2', 'Mode2'): 95	'Plant2': 20
('Raw2', 'Mode1'): 1	('Supplier2', 'Plant1', 'Mode2'): 121	'Supplier1': 141
('Raw2', 'Mode2'): 2	('Supplier2', 'Plant2', 'Mode2'): 148	'Supplier2': 129
('Product1', 'Mode1'): 1	('Plant1', 'Warehouse1', 'Mode2'): 98	'Warehouse1': 46
('Product1', 'Mode2'): 1	('Plant2', 'Warehouse1', 'Mode2'): 144	
('Product2', 'Mode1'): 1	('Warehouse1', 'Client1', 'Mode2'): 15	
('Product2', 'Mode2'): 1	('Warehouse1', 'Client1', 'Mode1'): 19	
	('Warehouse1', 'Client2', 'Mode1'): 12	
	('Warehouse1', 'Client2', 'Mode2'): 11	
	('Warehouse1', 'Client3', 'Mode1'): 13	
	('Warehouse1', 'Client3', 'Mode2'): 17	

Tabla 3.7 Datos ejemplo simple 2.

u^k_o	q^k_{od}	q^k_o
('Supplier2', 'Raw1'): 1	('Supplier2', 'Plant1', 'Raw1'): 100	('Supplier2', 'Raw1'): 73
('Supplier1', 'Raw2'): 1	('Supplier2', 'Plant2', 'Raw1'): 88	('Supplier1', 'Raw2'): 74
('Plant2', 'Product1'): 3	('Supplier1', 'Plant1', 'Raw2'): 92	('Plant2', 'Product1'): 11
('Plant1', 'Product1'): 4	('Supplier1', 'Plant2', 'Raw2'): 125	('Plant1', 'Product1'): 17
('Plant2', 'Product2'): 3	('Plant2', 'Warehouse1', 'Product1'): 18	('Plant2', 'Product2'): 10
('Plant1', 'Product2'): 4	('Plant1', 'Warehouse1', 'Product1'): 20	('Plant1', 'Product2'): 20
('Warehouse1', 'Product1'): 3	('Plant2', 'Warehouse1', 'Product2'): 10	('Warehouse1', 'Product1'): 16
('Warehouse1', 'Product2'): 3	('Plant1', 'Warehouse1', 'Product2'): 20	('Warehouse1', 'Product2'): 19
	('Warehouse1', 'Client2', 'Product1'): 18	
	('Warehouse1', 'Client3', 'Product1'): 11	
	('Warehouse1', 'Client3', 'Product2'): 16	
	('Warehouse1', 'Client2', 'Product2'): 11	

3.4 Pruebas de tiempo de computación

Para la evaluación del esfuerzo computacional necesario para resolver el modelo se han llevado a cabo experimentos computacionales en una serie de instancias generadas aleatoriamente. La Sección 3.4.1 describe el procedimiento seguido para generar estas instancias. La Sección 3.4.2 presenta las mediciones con las que se realiza el análisis de esfuerzo computacional. Por último, en la Sección 3.4.3 se presenta un resumen y un análisis detallado de los resultados computacionales obtenidos, lo que permite evaluar la eficiencia y escalabilidad del modelo planteado.

3.4.1 Generación de instancias

La generación de las instancias es un paso esencial en el proyecto, ya que las instancias son la base del análisis y de las pruebas. Se generan numerosas instancias con valores aleatorios que permiten mantener el equilibrio entre la posibilidad de ser casos reales y la facilidad de generación de estas. Lo que varía de una instancia a otra es el tamaño y los datos.

Tamaño

El tamaño de las instancias viene dado por el número de proveedores ($|S|$), el número de plantas ($|P|$), el número de almacenes ($|W|$), el número de clientes ($|C|$), el número de materias primas ($|R|$) y el número de productos terminados ($|F|$). Como en los casos reales el número de clientes suele ser mayor que todos los demás conjuntos, para una instancia con $|C| = n$, se determina que $|S| = |P| = |W| = \lceil \frac{n}{6} \rceil$ y $|R| = |F| = \lceil \frac{n}{3} \rceil$.

El número de variables y restricciones, y por tanto la complejidad del problema, también depende del número de proveedores que pueden dar cada materia prima ($|S^r|$), del número de plantas que pueden fabricar cada producto ($|P^f|$) y del número de almacenes que pueden distribuir cada producto ($|W^f|$). Estos valores se escogen aleatoriamente del conjunto $\{1,\ldots,|S|\}$. Los correspondientes proveedores, plantas y almacenes también se eligen aleatoriamente.

Otros factores que afectan al tamaño son el número de materias primas que son necesarias para fabricar cada producto ($|R^f|$) y el número de clientes que tienen demanda positiva de cada producto ($|C^f|$). El valor de $|R^f|$ varía entre 1 y $\left\lceil \frac{|R|}{4} \right\rceil$. En cambio, el valor de $|C^f|$ varía entre 1 y $\left\lceil \frac{|C|}{2} \right\rceil$.

Cabe destacar que estos números se generan al principio del código, no variando así de un proveedor a otro, un cliente a otro, etc.

Datos

Como el objetivo principal del estudio es analizar el tiempo de computación, se puede considerar que los valores del resto de parámetros (demanda, costes, capacidades…) son irrelevantes. Aun así, para lograr que los modelos sean admisibles se han tomado los valores que se describen a continuación.

En primer lugar, se determinan los valores que no dependen del tamaño del modelo. Los valores de la demanda a_c^f se seleccionan al azar de un conjunto de valores que van del 1 al 10 para cada producto y cada cliente. Asimismo, la cantidad de materia prima requerida para la producción de cada producto b^{rf}, también se elige al azar entre 1 y 10.

La estructura de los costes se ha establecido de la siguiente manera: el coste fijo de seleccionar cada origen c_o varía entre 10^4 y 10^5, el coste de asignar cada mercancía a un origen específico c_o^k está en el intervalo $[750,1250]$ para cualquier origen y mercancía, el coste c_{od}^k se elige al azar entre 1000 y 2000 y el coste c_{od}^k toma un valor aleatorio entre 2 y 10. En cuanto al coste de transporte entre cada origen y destino c_{od}^m, al haberse supuesto que se usa un único medio de transporte en los movimientos entre proveedores, plantas y almacenes y que se pueden usar varios entre almacenes y plantas, el transporte a carga completa no tiene costes fijos y los camiones con carga consolidada tienen un coste fijo comprendido entre 10^3 y 10^4.

Continuando con la capacidad requerida por cada mercancía, se determina que la capacidad requerida en cada origen u_o^k toma un valor entre 1 y 5, mientras que la capacidad requerida en cada medio de transporte *gkm* toma un valor entre 1 y 2.

Por último, determinamos los valores que sí deben cambiar dependiendo del tamaño del problema, ya que si no los modelos generados serían inadmisibles. La capacidad de cada origen u_o debe tomar un valor entre $\left[\frac{Demanda\ total\ productos/materia\ prima \cdot 4}{Número\ de\ orígenes}, \frac{Demanda\ total\ productos/materia\ prima \cdot 5}{Número\ de\ orígenes} \right]$. Este rango de valores tiene en cuenta tanto el número de orígenes como la capacidad que requiere en unidades equivalentes. Asimismo, la capacidad del modo de transporte entre o y d g_{od}^m toma un valor entre $[Demanda\ total, Demanda\ total \cdot 2]$, teniendo en cuenta que la capacidad requerida en los medios de transporte es de 1 a 2 unidades equivalentes.

Nota: Si se comienza el análisis a partir de n=10 y se sigue este método para determinar el tamaño y los datos, la mayoría de las instancias generarán un modelo admisible. Sin embargo, es importante tener en cuenta que la aleatoriedad en la selección de la mayoría de los conjuntos puede hacer que algunos de los modelos generados no sean admisibles.

3.4.2 Parámetros a analizar

Para comparar el esfuerzo computacional con el tamaño del modelo se analizarán dos parámetros: el **trabajo** y el **tiempo de ejecución**. Ambas medidas son útiles para evaluar el rendimiento de Gurobi y para comprar los diferentes tamaños en el modelo.

El tiempo de ejecución, medido en segundos, representa el tiempo real que se tarda en resolver un modelo. Comprende el tiempo desde que se ejecuta el código hasta que se obtiene el resultado, lo que significa que también incluye el tiempo utilizado por el sistema operativo en cualquier otra tarea que se esté ejecutando en segundo plano. Es importante tener en cuenta que el tiempo de ejecución de un mismo modelo puede variar incluso en el mismo hardware. Aun así, es una medida fiable ya que esta variación no suele ser muy elevada.

Por otro lado, el trabajo es una medida determinista de la cantidad de esfuerzo computacional requerido específicamente por Gurobi para resolver el modelo. Esto quiere decir que se obtendrá el mismo trabajo cada vez que se resuelva el mismo modelo en el mismo hardware. La unidad de esta medida es arbitraria.

3.4.3 Análisis de resultados

Una vez que se han generado y resuelto una gran cantidad de instancias, se pueden analizar los resultados obtenidos. Este análisis puede tener en cuenta tanto el trabajo como el tiempo de ejecución, como ya se ha mencionado. Además, estos parámetros se pueden comparar tanto con el número de variables como con el número de restricciones, ya que ambos definen el tamaño del problema.

En la Figura 3.2 se compara el número de variables con el tiempo de ejecución, mientras que en la Figura 3.3 se compara con el trabajo. Se puede observar que la tendencia que siguen los puntos es similar en ambas gráficas, lo cual sugiere que las tareas en segundo plano no han aumentado de forma significativa el tiempo de ejecución.

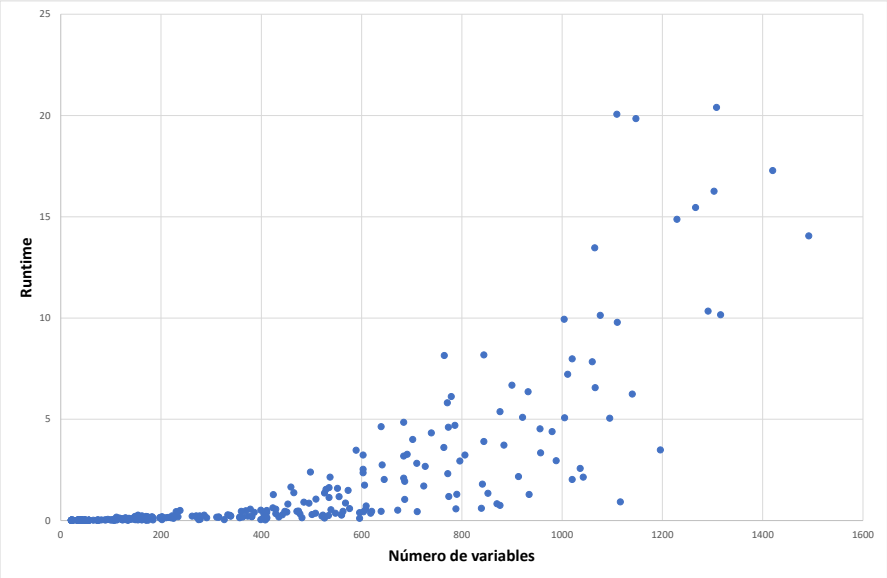

Figura 3.2 Número de variables vs Tiempo de ejecución.

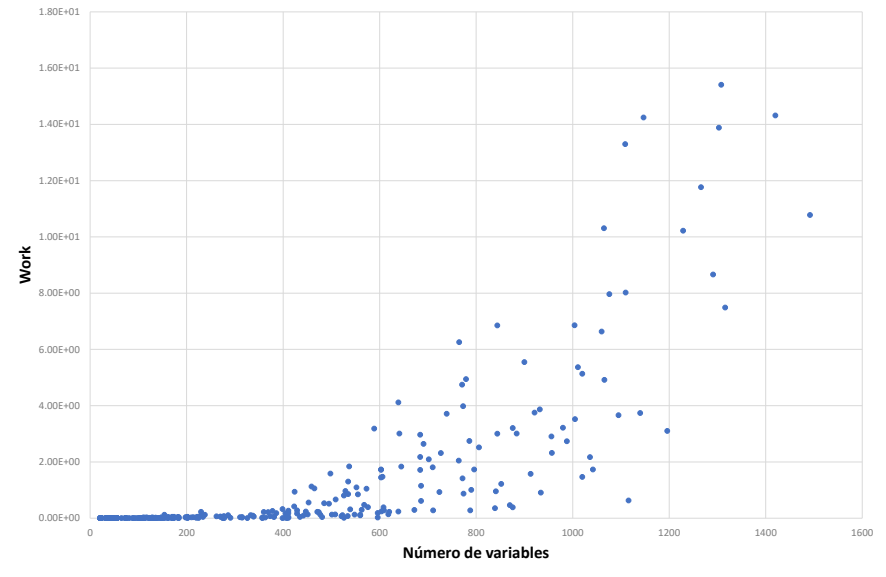

Figura 3.3 Número de variables vs Trabajo.

Si nos centramos en la Figura 3.2, se puede observar que, hasta las 400 variables, el tiempo de ejecución es prácticamente cero y no varía significativamente. Sin embargo, a partir de las 400 variables, el tiempo comienza a aumentar exponencialmente. Además, a partir de las 1200 variables, hay muy pocas pruebas debido a que ha sido necesario detener las pruebas sin alcanzar la solución final por un elevado tiempo de ejecución.

En la Figura 3.4 y en la Figura 3.5 se compara el número de restricciones con el tiempo de ejecución y el trabajo, respectivamente. En ambas gráficas se puede notar la misma tendencia observada en las gráficas anteriores. En estos casos, es a partir de las 800 restricciones cuando el esfuerzo computacional aumenta considerablemente, dificultando la realización de pruebas.

Por tanto, se podría determinar que este método para la resolución exacta puede ser aplicado sin ninguna dificultad en modelos que contengan hasta 1200 variables, lo cual generalmente equivale a unas 800 restricciones, dado que el número de variables y restricciones están estrechamente relacionados.

Debido a la aleatoriedad seguida en la creación de las instancias, resulta difícil determinar con precisión el número de clientes n que corresponde con estos valores máximos. Esto se debe a que el número de variables y restricciones no depende únicamente del tamaño n, es decir, del número de clientes, proveedores, almacenes, productos terminados y materias primas. La cantidad de variables y restricciones también esta influenciada por la forma en que se relacionan estos conjuntos, como por ejemplo, el número de proveedores que suministran cada materia prima, el número de plantas que pueden fabricar cada producto, la cantidad de clientes que demandan cada producto, entre otros factores.

En la generación de instancias que se ha llevado a cabo, estos números se han seleccionado aleatoriamente dentro de un rango determinado. En la Figura 3.6 y en la Figura 3.7, se aprecia que los valores discutidos corresponden a un rango de valores del número de clientes n entre 20 y 25. Aun así, es notable la gran variedad de variables y restricciones posibles para un mismo valor del número de clientes n, lo que reafirma la naturaleza aleatoria de la generación de las instancias.

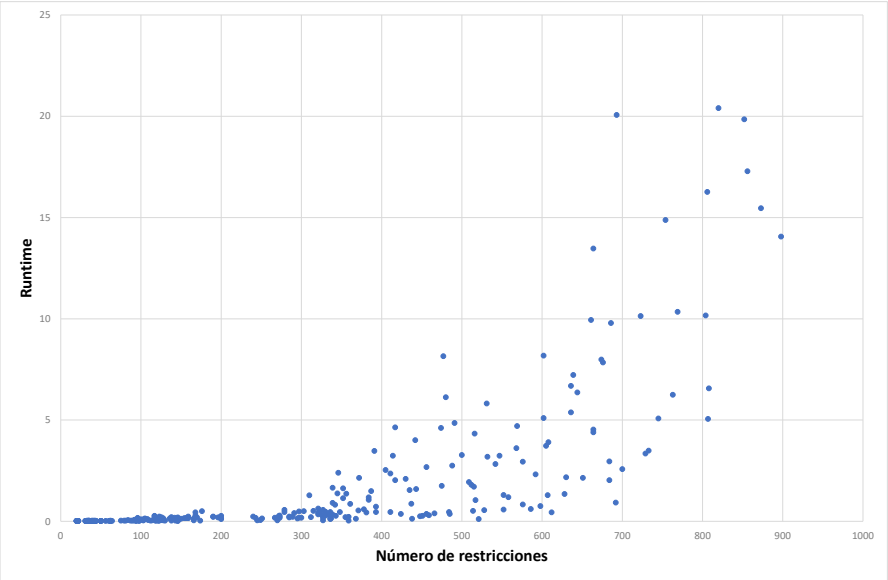

Figura 3.4 Número de restricciones vs Tiempo de ejecución.

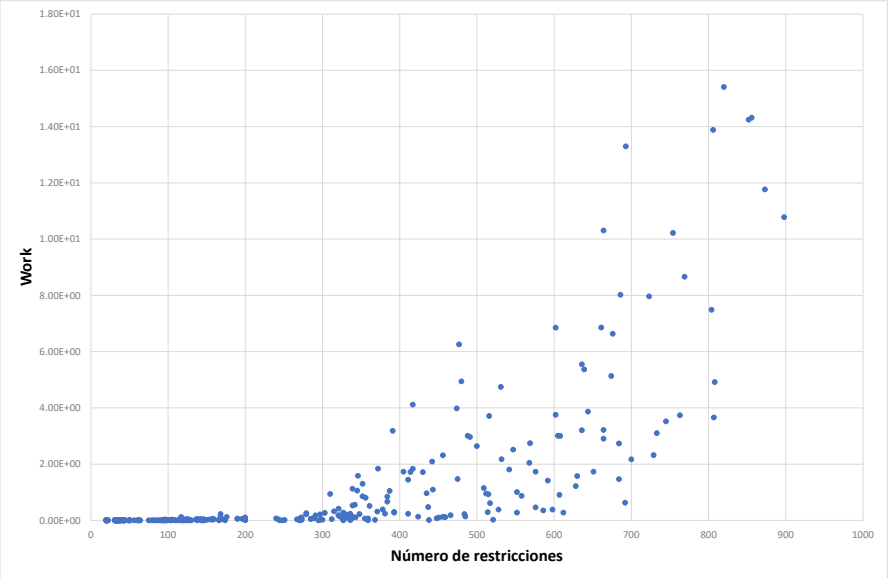

Figura 3.5 Número de restricciones vs Trabajo.

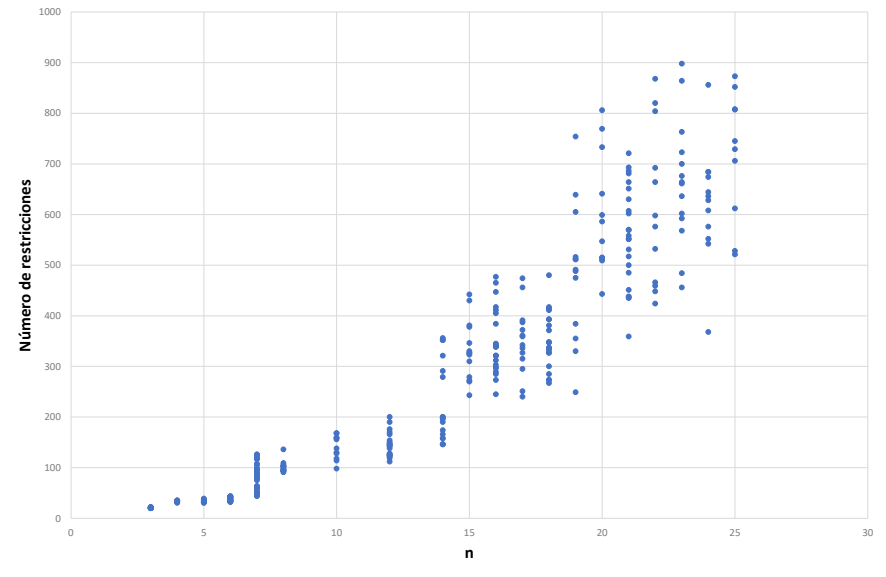

Figura 3.6 Número de clientes *n* vs Número de restricciones.

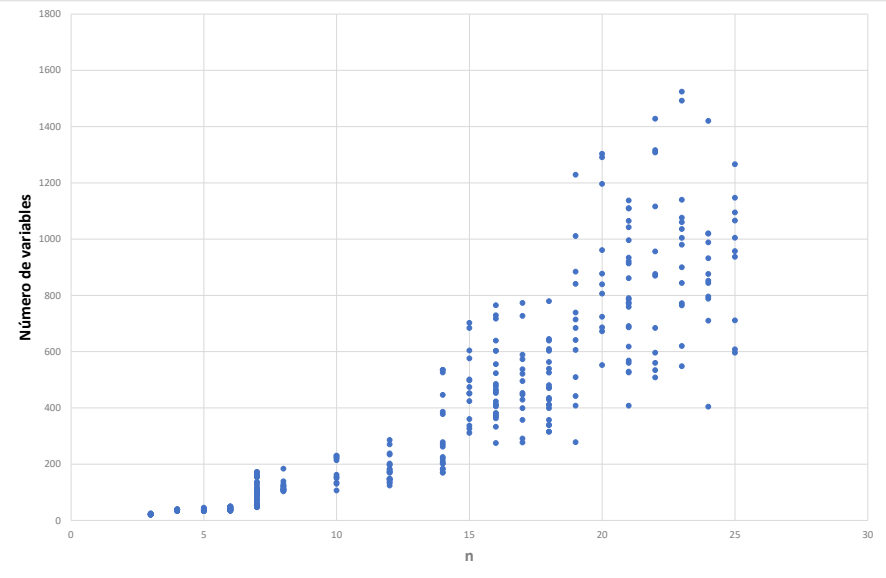

Figura 3.7 Número de clientes *n* vs Número de variables.

4 Modelo para un problema *Milk Run*

El segundo problema que se va a abordar es el descrito por Sadjadi et al. (2009). Dicho artículo incluye la descripción del problema a resolver y el modelo propuesto.

El término *Milk Run* se refiere a un método de transporte de mercancías utilizado en la logística, especialmente en la industria manufacturera. Este método toma su nombre de la manera en la que los antiguos vendedores de leche entregaban los productos a sus clientes. El vendedor seguía una ruta determinada, llevando las botellas de leche a las puertas de las casas de los clientes y recogiendo las botellas vacías.

En la actualidad, el sistema se aplica en diversas industrias, incluyendo la fabricación de automóviles. Este sistema se encarga de establecer la ruta de los camiones y la cantidad y tipo de piezas que correspondientes a cada proveedor, basándose en las necesidades de material de la empresa. El supuesto principal de este método reside en que todos los camiones deben devolver los palés vacíos al centro de demanda, por ejemplo, al fabricante de automóviles que adquiere las diferentes piezas.

El principal objetivo del uso del método *Milk Run* es mejorar la eficiencia y la productividad de la cadena de suministro al reducir los gastos de transporte, los tiempos de espera y los costes asociados al inventario. Para lograrlo, se busca reducir el número de pedidos, aumentar la frecuencia de entregas al centro demandante y asegurar que la entrega de materiales sea fluida.

Las ventajas de este sistema se maximizan en situaciones de envíos con alta frecuencia de entrega, distancias cortas y productos de alto valor. Estos factores están relacionados ya que, las piezas de gran tamaño y coste requieren entregas más frecuentes para evitar el aumento de los costes de almacenamiento. A medida que aumenta la distancia entre los puntos de entrega y recogida, las ventajas de este sistema disminuyen, pudiendo ser más favorable el uso de otros sistemas. Sin embargo, al aumentar factores como la cantidad de proveedores y su cercanía, así como el valor de los productos y los costes de almacenamiento, las ventajas del método aumentan considerablemente.

La ventaja principal de la aplicación de este tipo de sistema es la entrada fluida de materias primas en la línea de producción. Esto permite realizar entregas más frecuentes y con menores cantidades de materiales, reduciendo así el inventario total en la cadena de suministro, los costes de almacenamiento y el capital invertido en inventario. Además, aumenta la confianza en las entregas de productos, reduciendo de nuevo el nivel de inventario acumulado por las empresas. Al reducir el inventario total, se necesita menos espacio para almacenar los materiales, permitiendo la utilización de instalaciones más pequeñas y económicas. También se puede lograr una mejor utilización del espacio en los vehículos de transporte, reduciendo así estos costes.

Sin embargo, el *Milk Run* también presenta algunas desventajas importantes. Por ejemplo, si un punto de entrega o de recogida no está listo en la ventana horaria planeada puede llegar a retrasar todo el proceso y afectar la programación del resto de la ruta. Por tanto, este método requiere una buena coordinación entre los diferentes participantes en la cadena de suministro, incluyendo

proveedores, transportistas y clientes. Esto puede ser un desafío y puede requerir una inversión significativa en sistemas y tecnología para asegurar un buen flujo de información.

En la Figura 4.1 se muestra esquemáticamente la diferencia entre el flujo de materiales desde los proveedores a la empresa en el sistema de envío directo y en el sistema *Milk Run*.

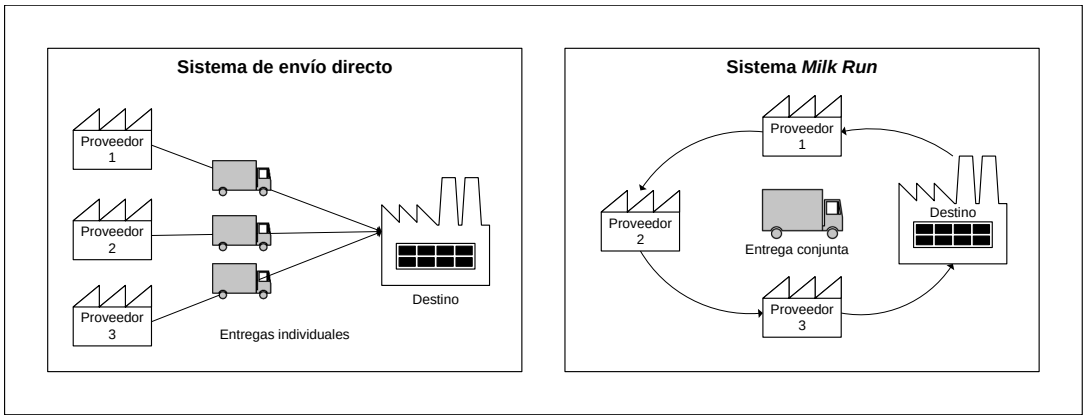

Figura 4.1 Diferencia Milk Run y envío directo.

4.1 Descripción del problema a resolver

En este documento se analiza el caso particular de un problema *Milk Run* propuesto por Sadjadi et al. (2009). En su artículo se aborda un caso particular de implementación de un sistema *Milk Run* adaptado a una compañía de fabricación de automóviles. Para ello, se consideran ciertos requisitos que permiten su correcta implementación.

En primer lugar, deben existir distintos proveedores que suministran el mismo producto. Permitiendo así a la empresa supervisar el nivel de servicio ofrecido por los proveedores y, en función de la calidad de sus servicios, tomar decisiones para ajustar la cantidad de productos que se adquieren de cada proveedor.

Así mismo, resulta fundamental que el modelo determine con precisión el momento en que los proveedores entregan la mercancía para poder preprogramar el plan de producción. Esto evitará esperas innecesarias y aumentará la eficiencia del proceso productivo.

Otro aspecto importante es la flexibilidad que se le otorga a los camiones de transporte para trabajar en diferentes rutas siempre y cuando sea en diferentes franjas horaria. Esto permitirá una mejor utilización de los recursos de transporte disponibles y la reducción de costos en la cadena de suministro.

4.2 Formulación matemática del modelo

Al igual que se comentó en el modelo anterior, pese a no ser el principal objetivo del proyecto, es necesario describir el modelo matemático que se va a analizar para poder comprender los resultados obtenidos de una manera más completa. Por tanto, se va a describir de forma detallada el modelo matemático expuesto en el artículo anteriormente mencionado (Sadjadi et al., 2009).

4.2.1 Conjuntos

En primer lugar, es necesario determinar los conjuntos que tienen relevancia en el problema para, posteriormente, analizar sus relaciones. A la vez que se definen, se determina la notación que se usará en el documento completo.

Tabla 4.1 Notación de los conjuntos.

Conjunto	Descripción
S	Proveedores j
K	Vehículos de transporte k
P	Productos p
T	Fechas o momento de entrega establecido t

Los proveedores son las ubicaciones donde se recogen los productos para su posterior distribución. En el contexto de una fábrica de automóviles, estos puntos pueden ser los almacenes de los proveedores de materias primas o piezas terminadas que se utilizan en la planta de fabricación. En esta formulación en particular, es importante destacar que el almacén de la empresa productora también se considera como un proveedor.

Respecto a los vehículos, estos son utilizados tanto para recolectar como para entregar los productos desde los proveedores hasta el centro de entrega.

Es importante comentar que los productos se transportan en palés, tanto en su recolección como en su entrega. Como resultado, muchos de los parámetros y datos relevantes están referidos a los palés de un producto en concreto. Por lo tanto, es necesario considerar los palés de los diferentes productos de forma individual.

4.2.2 Parámetros

Para poder determinar las características de estos conjuntos en los diferentes casos, se detallan los parámetros fundamentales que los caracterizan en la Tabla 4.2.

Tabla 4.2 Notación de los parámetros.

Parámetro	Descripción
V_k	Capacidad del medio de transporte k
V_p^{PL}	Capacidad máxima de un palé de producto p
U_{tp}	Consumo medio de producto p en el tiempo t
H_p	Coste de inventario de un palé que contiene el producto p por t que pase en el almacén
C_{kij}	Coste de transporte del camión k cuando va desde el proveedor i al proveedor j
γ_{pj}	Cantidad de palés de producto p asignada al proveedor j
c_p^{min}	Mínimo nivel de inventario del producto p
$\hat{C}$	Coste fijo de cargar un vehículo en cada proveedor

La cantidad asignada a cada proveedor debe ser determinada por la empresa, basándose en factores como el nivel de servicio de cada proveedor y la calidad de los productos que ofrece.

Es importante tener en cuenta que el tiempo t no se va a considerar en horas. Se va a suponer que el horizonte de planificación se divide en varios intervalos de tiempo. La empresa de fabricación es la que debe determinar a qué corresponden estos intervalos de tiempo en la situación real.

4.2.3 Variables

A continuación, en la Tabla 4.3, se detallan las variables de decisión, que son aquellas que se deben determinar para encontrar una solución óptima al problema. También es importante determinar el tipo de estas variables.

Tabla 4.3 Notación de las variables.

Variable	Tipo	Descripción
x_{tkpj}	Entera	Número de palés de producto p transportados por el vehículo k desde el proveedor j con fecha de entrega t
y_{tkij}	Binaria	Toma valor 1 en caso de que un camión k recoja cualquier producto del proveedor j en ese tiempo t $y_{tkij} = \begin{cases} 1 & \text{si } x_{tkpj} > 0 \\ 0 & \text{si } x_{tkpj} = 0 \end{cases}$
x_{tp}^{M}	Entera	Cantidad de producto p restante al final del tiempo t

La primera variable permite determinar la cantidad de palés suministrados por cada proveedor en cada camión. La segunda, en cambio, es útil para determinar si un vehículo realiza una parada en un proveedor específico, así como para determinar su ruta. En cuanto al valor de x_{tp}^{M}, dependerá de la cantidad inicial de producto y del consumo del producto en el tiempo.

Relacionando adecuadamente estas variables con los parámetros definidos anteriormente, se puede lograr un modelo matemático que maximice los beneficios y la eficiencia de este sistema de aprovisionamiento de mercancía.

4.2.4 Función objetivo y restricciones

Una vez se han establecido todos los parámetros y variables que conforman el modelo, se puede proceder a su definición. Resulta interesante describir detalladamente tanto la función objetivo como las once restricciones que garantizan la viabilidad y coherencia del modelo.

$$\text{mín} \sum_{t}\sum_{k}\sum_{i}\sum_{j}\left[C_{kij}+\hat{C}\right] y_{tkij} + \sum_{t}\sum_{p} H_p \times x_{tp}^{M} \tag{4.1}$$

s.a

$$\sum_{p}\sum_{j} x_{tkpj} \times V_p^{PL} \leq V_k \qquad \forall (t,k) \tag{4.2}$$

$$\sum_{t}\sum_{k} x_{tkpj} = \gamma_{pj} \qquad \forall (p,j) \tag{4.3}$$

$$x_{tp}^{M} \geq c_p^{min} \qquad \forall (t,p) \tag{4.4}$$

$$x_{tp}^{M} = \sum_{k}\sum_{j} x_{tkpj} + x_{(t-1)p}^{M} - U_{tp} \qquad \forall (t,p) \tag{4.5}$$

$$\sum_{k}\sum_{j} y_{tkij} \leq 1 \qquad \forall i \geq 2, \forall t \tag{4.6}$$

$$\sum_{k}\sum_{i} y_{tkij} \leq 1 \qquad \forall j \geq 2, \forall t \tag{4.7}$$

$$\sum_{j} y_{tk1j} \leq 1 \qquad \forall (t,k) \tag{4.8}$$

$$\sum_{i} y_{tki1} \leq 1 \qquad \forall (t,k) \tag{4.9}$$

$$\sum_i y_{tkiq} = \sum_j y_{tkqj} \qquad \forall (t,k), \forall q > 1 \tag{4.10}$$

$$\sum_{i \in S} \sum_{j \in S} y_{tkij} \leq |S| - 1 \;\; S \subseteq \{2,3,...,KT\} \qquad \forall (t,k) \tag{4.11}$$

(KT es el número total de proveedores)

$$x_{tkpj} \leq M \times \sum_i y_{tkIj} \qquad \forall (t,k,p,j) \tag{4.12}$$

(M es un número grande)

$$x_{tkpj} \geq 0 \tag{4.13}$$

$$y_{tkij} \in \mathbb{B} \tag{4.14}$$

$$x_{tp}^{M} \geq 0 \tag{4.15}$$

La función objetivo, definida por la ecuación 4.1, tiene como meta principal la minimización de los gastos de transporte y del nivel de inventario en el almacén, reduciendo a su vez los gastos de inventario.

El primer término se enfoca en el coste total del transporte, teniendo en cuenta tanto los costes de transporte variables, cuando un vehículo se desplaza desde un proveedor a otro, como los costes fijos de carga. Esta consideración resulta imprescindible para evitar que los costes de transporte aumenten debido a cargas ineficientes o incompletas y a movimientos innecesarios entre proveedores.

El segundo término esta enfocado a la minimización de los costes asociados al inventario. Para ello, se busca minimizar la cantidad de inventario restante al finalizar cada período, teniendo en cuenta el coste de almacenamiento del producto en cuestión.

En cuanto a las restricciones, establecen las siguientes relaciones entre parámetros y variables:

- Restricción 4.2: Esta restricción asegura, para cada tiempo y medio de transporte, que la cantidad total de productos transportada por ese vehículo no supere la capacidad del vehículo.

- Restricción 4.3: Iguala la cantidad de cada producto asignada a cada proveedor por parte de la empresa con la cantidad suministrada por los mismos, lo que permite garantizar que cada proveedor entregue la cantidad acordada en el contrato. Esta restricción es necesaria porque un mismo producto puede ser suministrado por distintos proveedores. Cabe destacar que la variable γ_{pj} representa la cantidad total asignada a cada proveedor a lo largo del horizonte de planificación total. En cada división de tiempo, el proveedor entregará una fracción de la cantidad total asignada, cumpliendo con los suministros requeridos para esa división.

- Restricción 4.4: Asegura que se mantenga el mínimo nivel de inventario establecido para cada producto. Para ello, la cantidad de producto restante al final de cada periodo debe ser mayor que el mínimo nivel de inventario necesario. La cantidad mínima se suele establecer a partir de las necesidades de dos o tres días laborables.

- Restricción 4.5: La cantidad total de un determinado producto es igual al número de total de piezas recibidas en el tiempo t, más el inventario que se tenía disponible al final del tiempo

t-1 menos el número de piezas utilizadas en el transcurso del tiempo *t*. Esto permite estudiar la cantidad de piezas disponibles en el almacén en cada momento para evitar la escasez de estas.

- Restricción 4.6 y 4.7: En este problema se ha decidido que no se pueden usar dos vehículos simultáneamente en la misma ruta. En el mismo horario sí se puede usar más de un vehículo siempre y cuando no visiten a los mismos proveedores.
  La primera restricción de este par determina que de cada proveedor *i* solo puede salir un vehículo, mientras que la segunda determina que a cada proveedor *j* solo puede llegar un vehículo.
  Además, es importante tener en cuenta que en este modelo se determina que el almacén de la empresa de fabricación automóviles toma el valor *i=1*. Al considerarse como un proveedor, se deben separar las restricciones que se aplican a dicho almacén. En primer lugar, se han tomado los casos con $i,j \geq 2$ y en las siguientes restricciones se analiza el caso $i,j = 2$.

- Restricción 4.8 y 4.9: Como se ha comentado, estas restricciones son similares a las anteriores con la diferencia de que estas están relacionadas exclusivamente con el almacén de la compañía de fabricación de automóviles. La primera determina, para todas las combinaciones de vehículos y tiempos, que del almacén de la fábrica solo puede salir un camión, mientras que la segunda determina que solo puede entrar un camión.

- Restricción 4.10: Esta restricción está diseñada para establecer una secuencia lógica de rutas para los vehículos de transporte. En concreto, se establece que un vehículo que llega a un determinado nodo también debe salir de él, lo que garantiza que ningún vehículo se detenga en la ubicación de un proveedor. Como sí deben volver al almacén y detenerse ahí una vez finalizado el recorrido, la restricción se establece para $q > 1$.

- Restricción 4.11: Esta restricción establece que cada vehículo sale del almacén de la planta de producción y su destino es el mismo almacén. Esta restricción es necesaria para evitar la creación de bucles en las rutas de transporte, determinando que un vehículo no puede volver a un proveedor que ya ha visitado en esa ruta.

- Restricción 4.12: Establece que se pueden traer piezas del proveedor *j* siempre que el vehículo de transporte pueda ir desde un proveedor a este proveedor. Esta restricción es muy importante ya que es la que establece la relación que se muestra en la Tabla 4.3 entre x_{tkpj} y y_{tkij}.

  El resto de restricciones determinan el dominio de las diferentes variables de decisión.

4.3 Resolución mediante Gurobi

De manera similar a como se realizó para el primer modelo, se detallan los pasos seguidos para modelar y resolver el problema *Milk Run* descrito por Sadjadi et al. (2009) utilizando la biblioteca de optimización de Gurobi en combinación con el lenguaje de programación Python. El código completo está disponible en el Apéndice B, pero en este apartado se describirán algunas partes importantes para facilitar la comprensión de este.

Al igual que en la mayoría de los códigos escritos en Python, el primer paso consiste en importar las librerías que se van a utilizar. En este caso, se utilizan las mismas librerías que en el Capítulo 3, pero a continuación se recuerda brevemente cuáles son y su utilidad. Las librerías importadas son *gurobipy*, *random* y *os*.

La primera es fundamental para poder utilizar tanto el solver de Gurobi como las funciones disponibles para crear el modelo. Las principales funciones utilizadas de esta librería se encuentran en el Apéndice D, donde se proporciona una breve descripción de cada una de ellas. La segunda librería importada sirve para generar los datos de forma aleatoria y automática. La última librería importada es fundamental, ya que es la que permite guardar los datos de la resolución de los modelos en archivos de texto.

Una vez que se han importado las librerías utilizadas, se procede a la definición del modelo. En primer lugar, es imprescindible determinar los conjuntos que componen dicho modelo: proveedores, camiones, productos y tiempo. Dado que el modelo matemático establece que el almacén de la fábrica se representa como un proveedor (*i=1*), se ha establecido que el primer elemento de la lista de proveedores corresponda al almacén de la fábrica.

En relación con la consideración de los tiempos, se ha tomado la decisión de representarlos como 'Tiempo 1', 'Tiempo 2'… Es importante destacar que la interpretación exacta de estos tiempos puede variar dependiendo del caso particular. Será la propia empresa la que tenga la capacidad de determinar si esos tiempos son días, divisiones dentro de los días u otras unidades de tiempo relevantes en su situación.

A continuación, se determinan los datos y parámetros que definen el modelo. Estos datos son generados de forma aleatoria, pero están acotados entre dos valores acorde al tamaño de cada caso particular. Con el fin de mantener un orden, se establecen primero los parámetros que dependen de un solo conjunto, es decir, aquellos que no relacionan varios conjuntos para su definición. En segundo lugar, se establecen los parámetros que relacionan varios conjuntos entre sí. Estos incluyen el consumo medio de cada producto, el coste de transporte entre los distintos proveedores y la cantidad de producto asignada a cada proveedor para satisfacer la demanda total.

En cuanto a la cantidad de producto asignada a cada proveedor, al haber definido el almacén en la lista S, es necesario determinar que la asignación comience a partir del segundo elemento de la lista, garantizando así que el almacén no reciba asignación de producto. Además, puesto que no es realista esperar que todos los proveedores sean capaces de suministrar todos los productos, se establece que cada proveedor solamente puede suministrar un número determinado de productos diferentes. Para ello, al crear el conjunto γ_{pj}, se asignan diferentes productos por proveedor.

Después de haber definido todos los datos necesarios, se procede a la inicialización y definición del modelo. Esto incluye la creación de las variables, que representan las decisiones a tomar dentro del modelo, y la creación de las restricciones que deben ser satisfechas en el modelo. Después de añadir todas las variables y restricciones del modelo, se define tanto la función objetivo como su sentido, en este caso de minimización.

Las restricciones que tienen alguna peculiaridad en su implementación son la Restricción 4.5 y la 4.11. Para formular la Restricción 4.5, hay que tener en cuenta el estado del inventario al comienzo del horizonte de planificación, representado por x^M_{0p}. Por tanto, es necesario inicializar estos valores, y se ha decidido que tomen el valor correspondiente al inventario mínimo para cada producto. La implementación de la Restricción 4.11, es bastante compleja, ya que hay que generar todos los subconjuntos de proveedores. Para ello se importa la librería *itertools* y se utiliza la función *combinations* para generar todas las combinaciones posibles de subconjuntos de S. El bucle '`for r in range(1, len(S[1:])+1)`'permite iterar sobre los diferentes tamaños de subconjuntos, desde 1 hasta el tamaño máximo. Para cada uno de estos subconjuntos se crea una restricción.

Con el modelo debidamente configurado, se procede a la optimización de este. Una vez que la optimización ha finalizado, se guardan en variables los resultados que son de interés para el análisis: número de variables, número de restricciones, tiempo de ejecución, trabajo, etc. Posteriormente, los valores de estas variables se almacenan en un archivo de texto para facilitar el análisis una vez se hayan optimizado varias situaciones diferentes. Además, se almacena el modelo en un archivo con extensión *.lp*. Esto, aunque no es necesario, se hace como medida de precaución para poder realizar posibles comprobaciones o futuros análisis.

4.3.1 Análisis de la solución de un ejemplo simple

Al igual que se ha hecho en la Subsección 3.3.1, se lleva a cabo una prueba con un problema de tamaño reducido. El propósito de esta prueba es verificar que todas las restricciones del modelo se cumplen y que la implementación con Gurobi se ha realizado de manera correcta. De este modo, se busca asegurar la calidad y fiabilidad de la implementación del modelo antes de hacer el análisis del esfuerzo computacional.

En este ejemplo sencillo del problema *Milk Run* se consideran 4 proveedores, sin tener en cuenta el almacén de la planta de fabricación de automóviles, 3 camiones y 2 productos diferentes. Además, se divide la entrega de productos de los proveedores al almacén principal en 2 rondas o etapas distintas.

En la Figura 4.2 se muestran todos los datos del problemas excepto los costes.

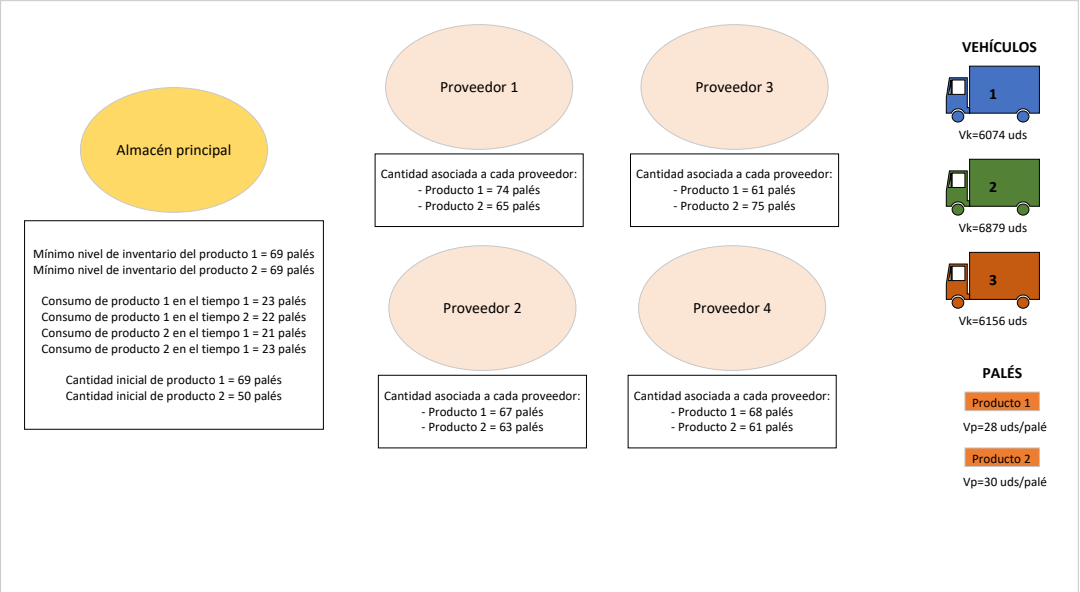

Figura 4.2 Datos ejemplo simple.

En la Figura 4.3 se presenta un pequeño esquema del flujo de materiales del problema, desde los distintos proveedores hasta el almacén principal. Las flechas indican la ruta de cada camión, mientras que los datos ubicados debajo de cada proveedor muestran la cantidad de materiales recogidos por el camión en cuestión en ese proveedor. Además, en la Tabla 4.4 se presentan todas las variables que toman un valor distinto de cero, junto con su correspondiente valor.

Ahora se procede a verificar si se cumplen las restricciones del modelo. A partir de la capacidad de los camiones en unidades, la capacidad de cada palé en unidades y el número de palés transportados en cada camión, se puede comprobar la restricción 4.2. Es importante tener en cuenta que, en cada recorrido, el camión debe transportar la mercancía de todos los proveedores a los que visita. Se observa que el camión uno cuenta con capacidad sobrante, mientras que los camiones dos y tres no podrían transportar ningún palé adicional. Esta podría ser la razón por la cual se necesitan dos camiones diferentes en ese momento.

En la Figura 4.3 también se puede observar que los proveedores entregan la cantidad exacta de palés asignados por la empresa y que el número de palés al final de cada periodo siempre es mayor que el inventario mínimo. Además, partiendo de los datos de inventario inicial y el consumo en cada periodo, se puede comprobar que el cálculo del inventario al final de cada periodo es correcto.

Por último, observando la Tabla 4.4 se puede comprobar el cumplimiento de todas las restricciones relacionadas con y_{tkij}, es decir, de la Restricción 4.6 a la 4.10.

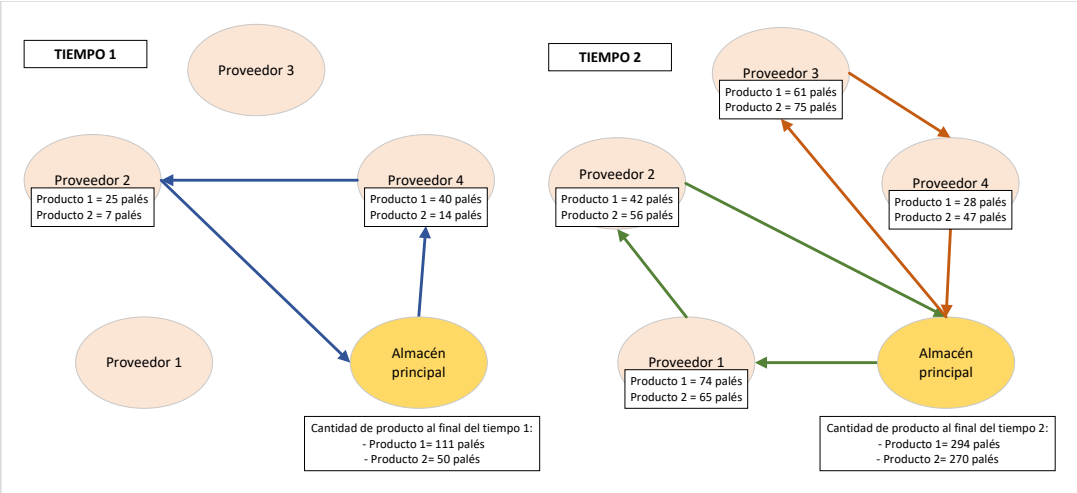

Figura 4.3 Resultados ejemplo simple.

Tabla 4.4 Variables no nulas ejemplo simple.

x_{tkpj}	x^M_{tp}	y_{tkij}
x[Time1,Truck1,Part1,Supplier2]: 25	x[Time0,Part1]: 69	y[Time1,Truck1,Warehouse,Supplier4]: 1
x[Time1,Truck1,Part1,Supplier4]: 40	x[Time0,Part2]: 50	y[Time1,Truck1,Supplier2,Warehouse]: 1
x[Time1,Truck1,Part2,Supplier2]: 7	x[Time1,Part1]: 111	y[Time1,Truck1,Supplier4,Supplier2]: 1
x[Time1,Truck1,Part2,Supplier4]: 14	x[Time1,Part2]: 50	y[Time2,Truck2,Warehouse,Supplier1]: 1
x[Time2,Truck2,Part1,Supplier1]: 74	x[Time2,Part1]: 294	y[Time2,Truck2,Supplier1,Supplier2]: 1
x[Time2,Truck2,Part1,Supplier2]: 42	x[Time2,Part2]: 270	y[Time2,Truck2,Supplier2,Warehouse]: 1
x[Time2,Truck2,Part2,Supplier1]: 65		y[Time2,Truck3,Warehouse,Supplier3]: 1
x[Time2,Truck2,Part2,Supplier2]: 56		y[Time2,Truck3,Supplier3,Supplier4]: 1
x[Time2,Truck3,Part1,Supplier3]: 61		y[Time2,Truck3,Supplier4,Warehouse]: 1
x[Time2,Truck3,Part1,Supplier4]: 28		
x[Time2,Truck3,Part2,Supplier3]: 75		
x[Time2,Truck3,Part2,Supplier4]: 47		

4.4 Pruebas de tiempo de computación

Con el fin de evaluar el esfuerzo computacional requerido para resolver el modelo, se han realizado pruebas utilizando diversas instancias generadas de manera aleatoria. En la Sección 4.4.1 se describe el proceso seguido para generar estas instancias. La Sección 4.4.2 presenta las mediciones que se utilizan para llevar a cabo el análisis del esfuerzo computacional. Por último, en la Sección 4.4.3 se expone un análisis detallado de los resultados computacionales obtenidos, lo cual permite evaluar la eficiencia y escalabilidad del modelo propuesto.

4.4.1 Generación de instancias

Al igual que en el Capítulo 3, es fundamental destacar la importancia de la generación de las instancias, dado que estas son la base sobre la cual se realiza el análisis. Se generan diversas instancias, cada una de ellas caracterizada por sus propios valores aleatorios.

Estos valores aleatorios permiten mantener el equilibrio entre la posibilidad de ser casos reales y la facilidad de generación de estas. Además, al incorporar valores aleatorios en la generación de instancias, se garantiza una mayor diversidad y representatividad de los escenarios a estudiar.

Tamaño

El tamaño de cada una de las instancias viene determinado por el número de proveedores ($|S|$), el número de camiones ($|K|$), el número de productos ($|P|$) y el número de divisiones de tiempo ($|T|$).

Las dimensión de estos conjuntos se determina en función de un valor dado n y se utilizan las siguientes relaciones:

- Número de proveedores: El número de proveedores vendrá dado por un número n: $|S| = n$. El resto de tamaños dependerán de ese número.
- Número de camiones: Se establece el número de camiones como una fracción del número de proveedores. Habrá 1 camión por cada 3 proveedores: $|K| = \frac{n}{3}$. Como el número de camiones debe ser un número entero y el número de proveedores no tiene por qué ser un múltiplo de 3, se establece que el número de vehículos es el entero superior de esa fracción: $|K| = \lceil \frac{n}{3} \rceil$.
- Número de productos: Al igual que el anterior, se establece el número de productos como una fracción del número de proveedores. Se decide que existan 2 productos por cada 3 proveedores: $|P| = \frac{2n}{3}$. Como también debe ser un número entero, se redondea al entero superior: $|P| = \lceil \frac{2n}{3} \rceil$. Además, se establece que cada proveedor solo va a poder suministrar un número determinado de productos diferentes, comprendido entre 1 y el número de productos existentes.
- Número de divisiones de tiempo: El número de divisiones de tiempo se mantendrá constante para todas las instancias, tomando un valor de $|T| = 4$

Al establecer el tamaño de los conjuntos a partir del número de proveedores n, se pretende garantizar una generación de instancias aleatorias que refleje la naturaleza del problema y permita obtener situaciones lo más realistas posible.

Cabe destacar que el número de almacenes principales de la fábrica de automóviles permanece constante en todas las instancias, es decir, siempre se considera la existencia de un único almacén para la empresa de producción.

Datos

El número de variables y restricciones que tiene un modelo no se ve afectado por los valores de los parámetros, como las capacidades, el coste, el consumo o la demanda. Por lo tanto, estos valores no tienen un impacto significativo en el tiempo de cálculo del modelo.

Sin embargo, estos valores son determinantes para establecer si el modelo es admisible o no. En el análisis a realizar, es importante que todas las instancias sean admisibles. Por lo tanto, es necesario asignar valores apropiados a dichos parámetros de modo que se garantice la admisibilidad de las instancias.

En primer lugar, se definen los valores que no están condicionados por el tamaño del modelo.

El mínimo nivel de inventario de cada producto en el almacén principal c_p^{min} se elige de manera aleatoria dentro de un rango de valores que oscila entre 150 y 200 para cada producto. Asimismo, el consumo medio de los productos U_{tp} se elige al azar entre 50 y 75 para cada producto en cada periodo de tiempo. Por último, se determina que la cantidad de cada producto asignada a cada proveedor γ_{pj} oscila entre 100 y 150. La capacidad máxima de un palé V_p depende del tipo de producto que contenga, y puede variar entre 10 y 20.

En lo que respecta a la estructura de los costes, se determina que el coste de cargar un vehículo $\hat{C}$ no depende del proveedor al que se visite ni del vehículo que se utilice y toma un valor contenido en el intervalo [30,50]. En cambio, el coste de transporte del camión C_{kij} sí depende de los proveedores que visite y del camión utilizado. Este toma un valor comprendido entre 50 y 100. Por último, el coste de inventario de cada producto H_p oscila entre 30 y 50.

Por último, se determina la capacidad de cada medio de transporte V_k, la cual es un parámetro fundamental para determinar si el modelo es admisible o no. La capacidad de los vehículos debe estar vinculada al número de proveedores, al número de productos y al número de vehículos disponibles.

Sin embargo, dado que estos tres valores se relacionan a través del parámetro n, es posible determinar el valor de V_k en función del número de proveedores, representado por n. Se decide que el valor de V_k se encuentre en el intervalo $[3500 \cdot n, 4000 \cdot n]$.

4.4.2 Parámetros a analizar

Los parámetros que se van a utilizar para obtener valores cuantitativos del esfuerzo computacional en la resolución de las diferentes instancias del modelo son los mismos que se utilizaron en el Capítulo 3: **trabajo** y **tiempo de ejecución**. La definición detallada de cada uno de ellos y de sus diferencias se encuentra en la Sección 3.4.2 de dicho capítulo.

No obstante, se hace un breve recordatorio de cada uno de ellos. El tiempo de ejecución se refiere al tiempo real que se tarda en resolver el modelo y se mide en segundos. Cabe destacar que este tiempo no siempre es constante en el mismo hardware. Por otro lado, el trabajo es una medida que no tiene unidades y que permanece constante para una misma instancia cuando se utiliza el mismo hardware.

Ambos parámetros son fundamentales para comprender el esfuerzo computacional requerido y para evaluar la eficiencia del modelo en diferentes condiciones y escenarios.

4.4.3 Análisis de los resultados

En este capítulo, se lleva a cabo una evaluación de los datos recopilados durante la resolución de las diferentes instancias del modelo. Además del tiempo de ejecución y el trabajo para cada instancia, se han recopilado el número de variables, restricciones y proveedores presentes en cada instancia. Sin embargo, no se han guardado los datos referentes al tamaño del resto de conjuntos, ya que estos están relacionados con el número de proveedores, representado por el parámetro n.

Teniendo todos estos datos recopilados, resulta interesante realizar una comparación entre el número de variables y su influencia tanto en el tiempo de ejecución como en el trabajo necesario. También es interesante realizar esas comparaciones con el número de restricciones, ya que tanto el número de variables como el número de restricciones definen la magnitud del problema. Por último, se analiza la relación entre el número de proveedores n y el número de variables y restricciones.

En la Figura 4.4 se compara el número de variables con el tiempo de ejecución, mientras que en la Figura 4.5 se compara con el trabajo. La tendencia seguida por los puntos es similar en ambas gráficas, lo que implica que la ejecución de las tareas en segundo plano no ha supuesto un aumento significativo en el tiempo de ejecución.

Si analizamos la Figura 4.4, se puede observar que el tiempo de ejecución se mantiene en un valor muy bajo y sin cambios significativos hasta alcanzar las 400 variables. Sin embargo, a partir de ese punto, el tiempo comienza a aumentar de manera exponencial. Además, a partir de las 1435 variables, no se han podido realizar más pruebas debido a su alto tiempo de ejecución sin alcanzar el óptimo. También se observa que no hay instancias que tengan entre 700 y 1200 variables, esto se discutirá más adelante. La Figura 4.5 sigue la misma tendencia.

En la Figura 4.6 y la Figura 4.7 se compara el número de restricciones con el tiempo de ejecución y el trabajo, respectivamente. En ambas gráficas se puede apreciar la misma tendencia observada en las gráficas anteriores. En estos casos, se ha notado que a partir de las 800 restricciones, el esfuerzo computacional comienza a aumentar significativamente. Además, a partir de las 2135 restricciones no se han podido llevar a cabo más pruebas.

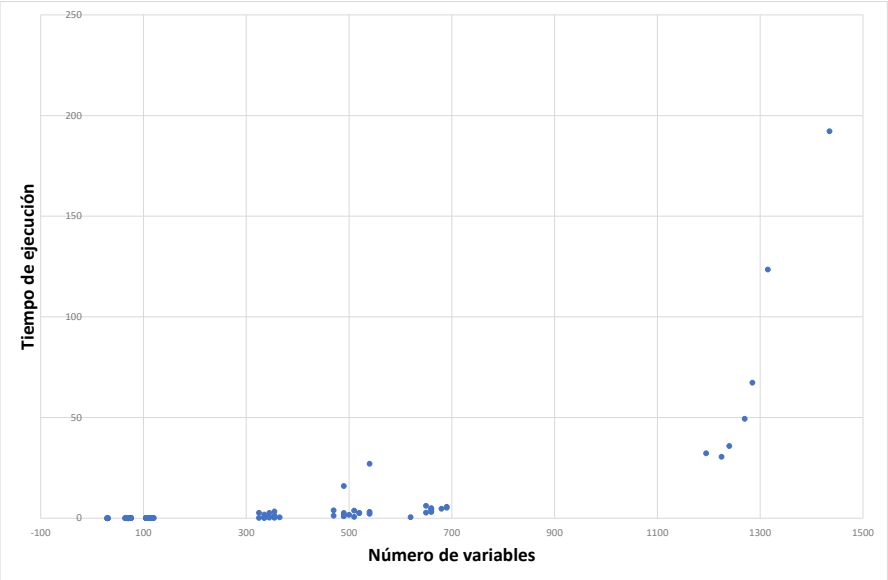

Figura 4.4 Número de variables vs Tiempo de ejecución.

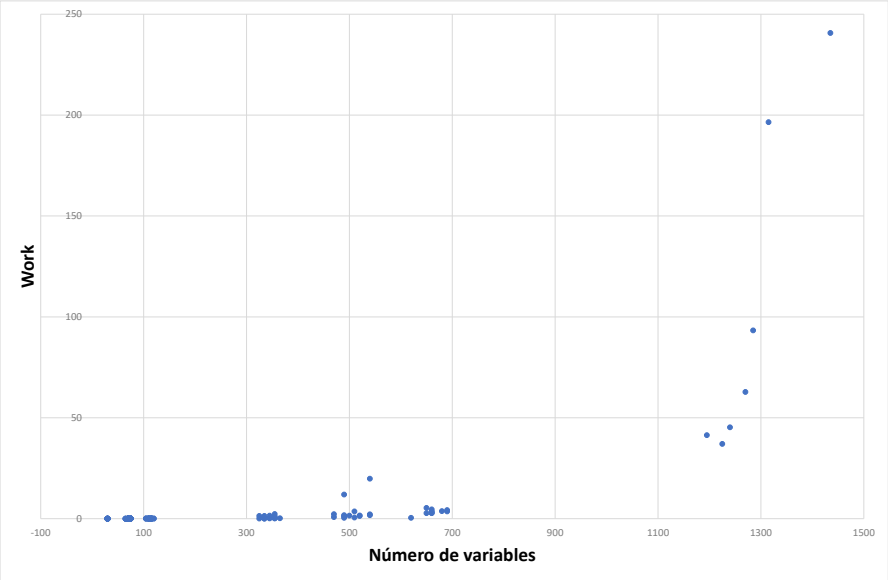

Figura 4.5 Número de variables vs Trabajo.

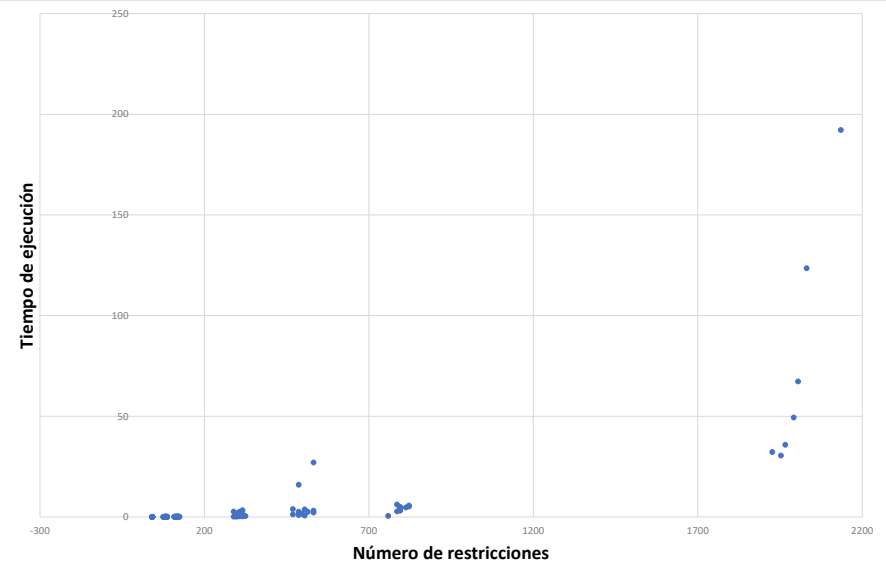

Figura 4.6 Número de restricciones vs Tiempo de ejecución.

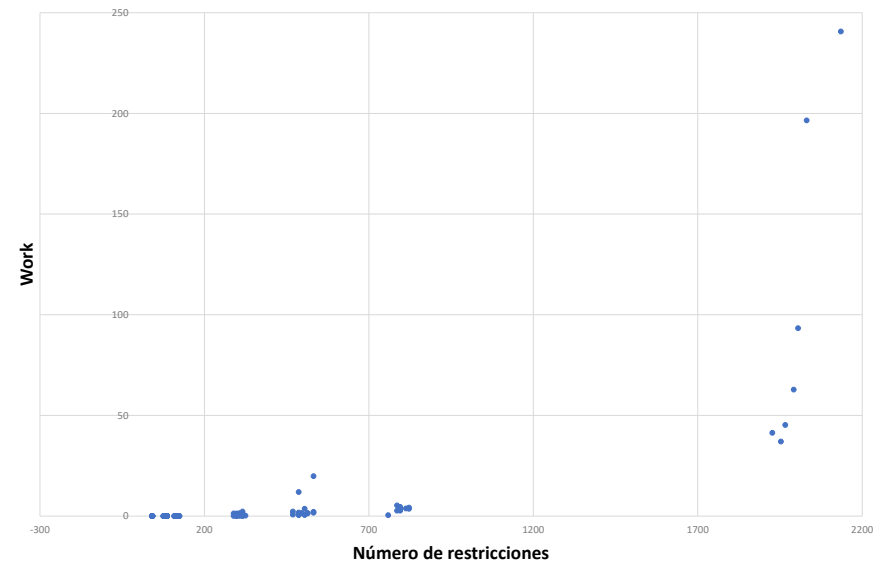

Figura 4.7 Número de restricciones vs Trabajo.

En cuanto a la aleatoriedad en el número de restricciones y variables, a diferencia de lo que ocurría en el modelo del Capítulo 3, en el que el número de variables y restricciones variaba significativamente para un mismo valor de número de clientes n, en este caso el número de restricciones y

variables apenas varía para un mismo valor del número de proveedores n.

Esto se debe a que, para un mismo número de proveedores, el resto de conjuntos se mantienen iguales en todas las instancias, cambiando tan solo la variedad de productos que puede suministrar cada proveedor. Al cambiar la variedad de productos suministrada por cada proveedor, tan solo varía el número de variables de tipo x_{tkpj}, debido a que es el único tipo de variable que depende tanto del proveedor como del producto, y el número de restricciones de tipo 4.2, 4.3, 4.5 y 4.12, debido a que son las únicas restricciones que relacionan x_{tkpj} con otras variables. Como se puede observar en la Figura 4.8 y la Figura 4.9, estas variaciones no suponen un cambio significativo en el número de variables y de restricciones.

En estas mismas figuras también se observa el aumento exponencial en el número de restricciones y variables al aumentar el número de proveedores n. Este hecho es el que hace que, como se ha comentado anteriormente, el trabajo y el tiempo de ejecución aumente exponencialmente al aumentar el número de proveedores n.

Estas tendencias exponenciales que se comentan, se deben, entre otras cosas, a la Restricción 4.11. Para imponer que no se generen bucles se crean un gran número de restricciones, ya que, para cada tiempo y vehículo, hay que hacer una restricción para cada subconjunto de S.

Por tanto, se podría determinar que la resolución exacta de este modelo a través de Gurobi puede ser aplicada a modelos que contengan hasta 7 proveedores, lo que genera un modelo de unas 2000 restricciones y 1300 variables.

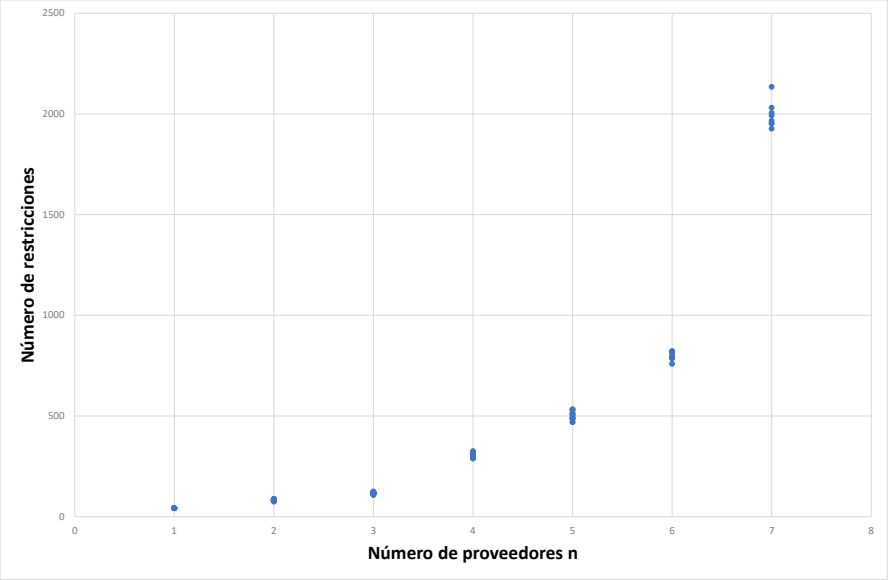

Figura 4.8 Número de proveedores n vs Número de restricciones.

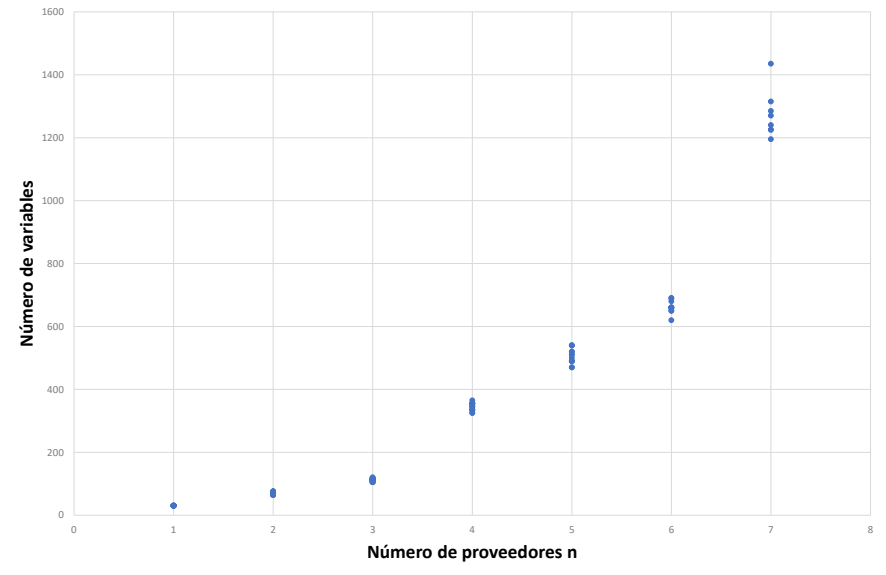

Figura 4.9 Número de proveedores *n* vs Número de variables.

5 Problema de enrutamiento de vehículos

El problema de Enrutamiento de Vehículos (VRP, por sus siglas en ingés **V**ehicle **R**outing **P**roblem) es una ampliación del Problema del Agente Viajero (TSP, por sus siglas en inglés **T**ravelling **S**alesman **P**roblem). El objetivo del Problema del Agente Viajero es encontrar la trayectoria más corta para un vendedor que necesita visitar a varios clientes en diferentes localizaciones, mientras que el objetivo del problema de Enrutamiento de Vehículos es encontrar rutas óptimas para varios vehículos que visitan una serie de ubicaciones. Además, los problemas VRP pueden incluir restricciones adicionales, como la capacidad de los vehículos y la ventana de tiempo en la cual cada ubicación debe ser visitada. Por lo tanto, si se toma un problema VRP con un solo vehículo y sin restricciones adicionales, se tiene un problema TSP.

Centrándonos en el problema VRP, es importante aclarar el concepto de rutas óptimas, el cual varía según el objetivo específico de cada caso particular. Si el objetivo principal es minimizar el coste total de transporte, las rutas óptimas son aquellas con la menor distancia total acumulada. En cambio, si se busca realizar todas las entregas en el menor tiempo posible, las rutas óptimas son aquellas que minimizan la longitud de la ruta más larga.

Este problema se puede aplicar en diversos ámbitos de la gestión de la cadena de suministro y de la logística, ya que permite optimizar las operaciones de transporte y distribución. Entre los diferentes usos en la gestión de la cadena de suministro destacan:

- Planificación de rutas de distribución de mercancías: Implica la entrega eficiente de productos desde un centro de distribución a múltiples destinos o clientes.
- Optimización del uso de flotas de vehículos: La información obtenida puede ayudar a determinar la cantidad y capacidad de los vehículos requeridos.
- Gestión de almacenes: Optimiza las operaciones de carga y descarga de vehículos, determinando el orden a seguir por estos.
- Planificación de rutas de recolección en logística inversa: Permite definir las rutas a seguir para recolectar productos en diversas localizaciones y llevarlas a un almacén central.

Además, es posible implementar este sistema en entornos urbanos ya que, en los últimos años, el crecimiento demográfico y la expansión del transporte a nivel global ha generado nuevos retos en materia de transporte y logística dentro de las ciudades. Cabe destacar que, si bien el VRP tiene múltiples aplicaciones en las ciudades, no se debe pasar por alto que los clientes ubicados en las ciudades suelen ser el último eslabón de la cadena de suministro, siendo la entrega de productos un aspecto clave en la logística de la ciudad.

Además de su aplicación en la distribución de mercancías, el modelo VRP se puede usar en muchas otras situaciones en las ciudades. Por ejemplo, se puede utilizar para la gestión de rutas y

horarios del transporte público. Así mismo, puede ser implementado en iniciativas de movilidad compartida, para facilitar la organización de los recorridos. Estas aplicaciones contribuyen a su vez a la disminución de la congestión del tráfico.

Dado el creciente aumento en la necesidad de implementar soluciones eficientes de VRP tanto en las ciudades como en la gestión de la cadena de suministro, es fundamental comprender y evaluar la capacidad de resolución que tenemos de los modelos más simples. Al adquirir una comprensión más profunda de la capacidad de resolución de estos modelos simples, podremos avanzar hacia implementaciones más complejas y adaptadas a las necesidades actuales. Además, el conocimiento del modelo simple nos dará una perspectiva más clara acerca de las limitaciones y posibilidades de aplicar técnicas de enrutamiento de vehículos en este entorno.

En el artículo de Konstantakopoulos et al. (2020), se hace una revisión bibliográfica y una clasificación de las diferentes variantes de los problemas de VRP. La Figura 5.1 muestra el crecimiento continuo en la cantidad de artículos publicados sobre los problemas de VRP y todas sus variantes a lo largo del tiempo. Estos resultados demuestran un constante interés en la resolución de problemas relacionados con VRP por parte de los investigadores.

La clasificación realizada en el artículo identifica un total de 16 variantes de VRP. Sin embargo, se concluye que el CVRP (**C**apacitated **V**ehicle **R**outing **P**roblem) es la variante más estudiada. Esto se debe a que esta variante es la más sencilla, lo que facilita su combinación con otras variantes y su utilización como base en investigaciones posteriores. La Figura 5.2 muestra la evolución en el número de artículos publicados específicamente sobre CVRP, corroborando así su importancia.

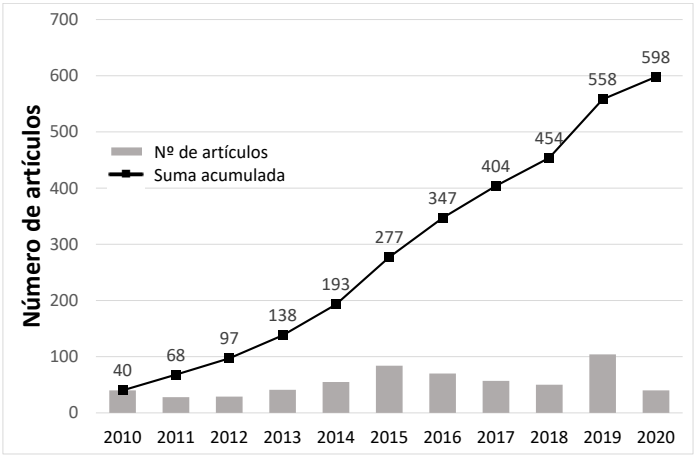

Figura 5.1 Crecimiento de las publicaciones de *VRP* (Konstantakopoulos et al., 2020).

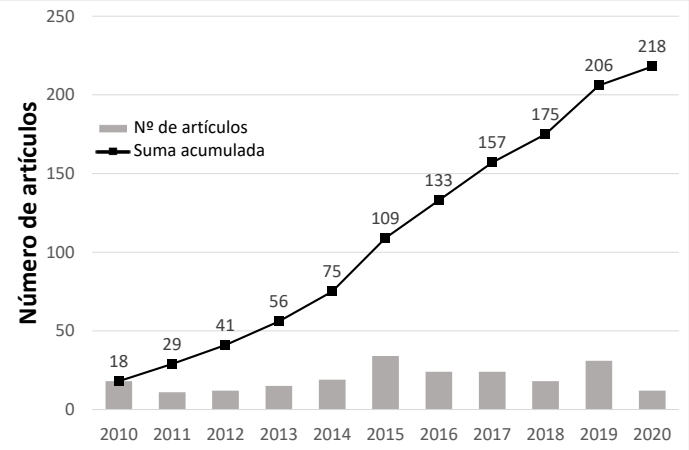

Figura 5.2 Crecimiento de las publicaciones de *CVRP* (Konstantakopoulos et al., 2020).

5.1 Descripción del problema a resolver

Dado lo expuesto anteriormente, se va a resolver un problema simple de enrutamiento de vehículos CVRP. En este problema, el objetivo principal es establecer las rutas óptimas para los vehículos involucrados, con el propósito de minimizar el coste total, o alternativamente, reducir al máximo el coste asociado a la ruta más larga entre todos los vehículos. El coste asociado a cada arco que conecta dos ubicaciones puede ser medido en términos monetarios, distancias u otros indicadores específicos, según las necesidades y criterios establecidos para cada problema.

Es fundamental asegurar que cada destino sea visitado únicamente una vez y por un solo vehículo. Además, se debe tener en cuenta que los vehículos deben partir desde un depósito o nodo inicial y regresar a él una vez hayan finalizado su ruta.

También es importante tener en cuenta que los vehículos cuentan con una capacidad limitada, la cual no puede ser excedida durante el proceso. En caso de que no existieran restricciones de capacidad, la solución óptima consistiría en asignar un solo vehículo para visitar todas las ubicaciones, lo cual transformaría el problema en un problema del Agente Viajero TSP.

5.2 Formulación matemática del modelo

A continuación, se describe la formulación matemática del modelo propuesto por Kim et al. (2015), un modelo básico de enrutamiento de vehículos con capacidades. Este modelo puede ser de gran utilidad para abordar diversos problemas de mayor complejidad.

5.2.1 Conjuntos

Dado que este problema busca minimizar los costes en una red que conecta distintas localizaciones, el problema se puede modelar utilizando un grafo, donde los nodos representan las localizaciones y los arcos las rutas de transporte entre ellos. Por tanto, para construir el grafo, se necesitan dos conjuntos principales: el conjunto de nodos y el conjunto de arcos.

El conjunto de nodos se divide en dos. El primer nodo representa el almacén central, desde donde comienzan y terminan todas las rutas, mientras que el resto de los nodos representan las localizaciones a visitar. Por otra parte, el conjunto de arcos se compone por todas las posibles rutas

Tabla 5.1 Notación de los conjuntos.

Conjunto	Descripción
$V = \{0,1,...,n\}$	Nodos. 0: almacén y 1 $\sim$ n: clientes
$A = \{(i,j) \mid i,j \in V, i \neq j\}$	Arcos
$G = (V,A)$	Grafo completo

que puede seguir un vehículo para conectar los diferentes nodos. Es decir, cada arco conecta dos nodos diferentes del grafo.

5.2.2 Parámetros

A continuación, se describen los parámetros clave que varían en función de cada caso particular. Uno de los parámetros más importantes es el coste de transporte, que puede depender de diversos factores como la distancia recorrida, el tiempo que se tarda en recorrer esa distancia, etc. También es fundamental considerar el número de vehículos del que se dispone para realizar las entregas y su capacidad. Esto dependerá de la inversión que haya realizado la empresa en su flota de transporte.

Dado que se está tratando un problema muy simple, se supone que todos los vehículos disponibles son idénticos y tienen la misma capacidad.

Tabla 5.2 Notación de los parámetros.

Parámetro	Descripción
c_{ij}	Coste asociado al arco (i,j)
K	Número de vehículos idénticos disponibles
C	Capacidad de los vehículos

5.2.3 Variables

Este problema tiene como objetivo determinar la ruta óptima que deben seguir los camiones. Para lograr esto, se debe determinar qué arcos se deben recorrer en cada ruta, pero no es necesario especificar el orden en que se recorren dichos arcos. Por lo tanto, solo hace falta una variable de decisión, que será binaria y tomará el valor 1 en caso de que un determinado arco pertenezca a la ruta óptima, y 0 en caso contrario.

Tabla 5.3 Notación de las variables.

Notación	Tipo	Descripción
x_{ij}	Binaria	Toma el valor uno si el arco (i,j) pertenece a la ruta

5.2.4 Consideraciones sobre capacidad y demanda

En este subapartado se abordan aspectos esenciales relacionados con la capacidad y la demanda siguiendo el artículo de Guancha et al. (2015). Esto permite comprender las restricciones del siguiente apartado.

Cada cliente tiene asociada una demanda positiva conocida, representada por d_i, que debe ser satisfecha. Por otro lado, el nodo almacén tiene una demanda ficticia $d_0 = 0$. Considerando un

conjunto de vértices $S \subseteq V$, la demanda total de dicho conjunto se define como la suma de las demandas individuales de cada nodo perteneciente a ese conjunto:

$$d(S) = \sum_{i \in S} di$$

Para asegurar que el modelo sea admisible, es fundamental que la demanda de cada cliente sea menor o igual que la capacidad de los vehículos ($d_i \leq C$). Si la demanda de un cliente superara la capacidad de los vehículos, se crearía un problema inadmisible donde ningún vehículo sería capaz de satisfacer completamente esa demanda. Esto implicaría que dos vehículos tendrían que visitar el mismo vértice, lo cual no está permitido en el contexto de este problema. Además, se determina que el número de vehículos K es mayor o igual que el número mínimo de vehículos necesarios.

Para cada conjunto $S \subseteq V$, $r(S)$ es el mínimo número de vehículos necesario para atender a todos los clientes del conjunto S.

5.2.5 Función objetivo y restricciones

Después de haber definidos los parámetros y variables que componen el problema, se procede a la formulación del modelo, que incluye tanto la función objetivo como las restricciones.

$$\text{mín} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{5.1}$$

s.a

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \setminus \{0\} \tag{5.2}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \setminus \{0\} \tag{5.3}$$

$$\sum_{i \in V} x_{i0} = K \tag{5.4}$$

$$\sum_{j \in V} x_{0j} = K \tag{5.5}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S) \qquad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset \tag{5.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V \tag{5.7}$$

La función objetivo, definida por la ecuación 5.1, minimiza el coste global asociado a todos los arcos que componen las rutas que deben seguir los vehículos involucrados. Su propósito es, por tanto, encontrar la asignación óptima de las rutas reduciendo al máximo el coste total.

En cuanto a las restricciones, se definen las siguientes relaciones entre los parámetros y la variable:

- Restricción 5.2: Esta restricción sirve para garantizar que a cada nodo llegue únicamente un vehículo procedente de cualquiera de los otros nodos. Es decir, que en cada nodo solo haya un flujo de entrada. Por lo tanto, hace falta una restricción de este tipo para cada nodo excepto para el nodo almacén, ya que este es el punto de partida y llegada de todas las rutas.

- Restricción 5.3: Continuando con el razonamiento de la restricción anterior, se establece que de cada nodo cliente solo puede salir un arco que lo conecte con otro nodo. De nuevo, esta restricción no aplica para el nodo almacén.

- Restricción 5.4 y 5.5: Como el nodo almacén no se ha incluido en las restricciones anteriores, es necesario establecer los requisitos específicos que debe seguir este nodo. Estas restricciones tienen como objetivo asegurar que todos los camiones comiencen y terminen en el nodo almacén. Además, el hecho de que el número de arcos que salen y entran sea igual al número de camiones disponibles, permite garantizar que no haya más rutas que camiones disponibles ni que se utilice más de un camión en la misma ruta.

- Restricción 5.6: Se trata de una restricción de capacidad-corte. que garantiza que cada conjunto de clientes S forme un corte que sea cruzado por al menos $r(S)$ arcos. En otras palabras, para cualquier subconjunto S de nodos, la suma de las variables asociadas a los arcos que salen de los nodos y que no pertenecen a S y llegan a nodos que sí pertenecen a S, debe ser mayor o igual que el número mínimo de vehículos necesario para atender la demanda de dicho subconjunto S.
  Esta restricción se aplica a todos los subconjuntos no vacíos de nodos, excluyendo el almacén. Por lo tanto, la cantidad de restricciones de la familia 5.6 aumenta de forma exponencial a medida que aumenta el número de vértices.

- Restricción 5.7: Impone que la variable de decisión sea binaria.

5.3 Resolución mediante OR-Tools

Dado que OR-Tools cuenta con una librería específica para la resolución de problemas de enrutamiento de vehículos, se ha decidido utilizarla para la resolución del modelo. No obstante, existe una limitación, ya que no siempre se alcanza el óptimo, sino que se utilizan heurísticas y metaheurísticas para la resolución. Por lo tanto, se ha decido resolver el problema también con la librería general de programación lineal de OR-Tools, lo que permite obtener la solución óptima en todo momento. En el análisis de las soluciones se compararán estas dos opciones.

Al igual que lo realizado en los dos modelos anteriores, se detallan a continuación los pasos seguidos para modelar y resolver el problema CVRP utilizando el solver de optimización OR-Tools a través del lenguaje de programación Python.

El código completo se encuentra disponible en el Apéndice C, pero en esta sección se describen algunas partes que son relevantes para facilitar su comprensión. En la Sección C.2 se encuentra el código correspondiente a la resolución exacta del modelo, mientras que la Sección C.3 contiene el código de la resolución mediante heurísticas. Por último, en la Sección C.1 se encuentra el código para la generación de datos, común para ambas opciones. Además, en el Apéndice E se describen algunas de las funciones empleadas y se hacen aclaraciones sobre cómo modelar el problema y utilizar el solver.

5.3.1 Generación de instancias

Para la generación de los datos de las diferentes instancias se crea una función llamada ***create_data_model()***. Esta función genera los datos a partir de tres parámetros: el número de nodos n, el número de vehículos k y la capacidad de estos vehículos c. A partir de estos parámetros, se crea un diccionario que almacena toda la información necesaria con las siguientes claves: “distance_matrix”, “demands”, “vehicle_capacities”, “num_vehicles” y “depot”.

La clave “distance_matrix” está asociada a una matriz que contiene las distancias entre los nodos. Esta se genera de forma aleatoria, teniendo en cuenta que la distancia de i a j debe ser la misma que de j a i. Se considera, por tanto, que es un grafo de arcos no dirigidos.

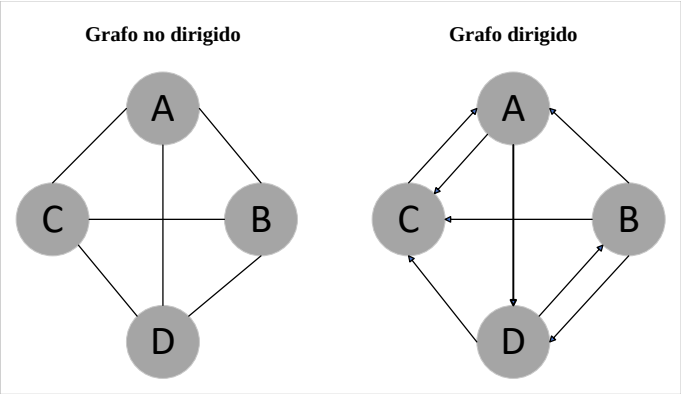

Figura 5.3 Grafo no dirigido vs Grafo dirigido.

La clave "demands" contiene la demanda de cada nodo, teniendo en cuenta que la demanda del nodo almacén es siempre cero. También se determina que el nodo correspondiente al almacén es el nodo 0 y que todos los vehículos tienen la misma capacidad.

Por último, con ayuda de las librerías *os* y *json*, se almacenan estos datos en un archivo *.json* para utilizarlos posteriormente.

5.3.2 Resolución exacta

La explicación del método seguido para la resolución exacta es breve, ya que sigue una estructura muy similar a la empleada con Gurobi, pero aplicando los métodos proporcionados por librería de OR-Tools. Por lo tanto, resulta más interesante centrar la explicación en el método de resolución mediante heurísticas y metaheurísticas.

Antes de empezar, hay que aclarar que, a pesar de que el modelo matemático describe variables que dependen únicamente de i y de j, en la implementación estas variables también deben depender de k para poder aplicar las restricciones antibucle.

En cuanto al modelado y la resolución, se comienza importando la librería *pywraplp* para poder hace uso del solver y de las herramientas de modelado. A continuación, se leen los datos del archivo *.json* y se hace una llamada al solver que se va a utilizar. Luego se añaden todas las variables y las restricciones utilizando los métodos *BoolVar* y *Add*, respectivamente. Una vez definidas las restricciones, se establece la función objetivo, completando así la construcción del modelo.

Finalmente, una vez que el modelo está completamente definido, se resuelve y se muestra la solución en la pantalla.

5.3.3 Resolución mediante heurísticas y metaheurísticas

Dado que OR-Tools proporciona ejemplos detallados como sobre utilizar su librería de enrutamiento de vehículos en su página web, el código utilizado se basa en el que se encuentra disponible en la web Google OR-Tools (`https://developers.google.com/optimization/routing/cvrp?hl=es-419#complete_programs`), aunque se han realizado algunas modificaciones. .

Es importante volver a destacar que este código no siempre proporciona la solución óptima. En su lugar, se emplea una combinación de heurísticas y metaheurísticas para la resolución. La primera solución se obtiene con una heurística y, a partir de ella, se utiliza una metaheurística de búsqueda local guiada. Si bien estas técnicas pueden alcanzar muy buenas soluciones, no se garantiza que sean óptimas en todos los casos.

Aunque esta librería no garantice soluciones óptimas en todos los casos, se ha decidido utilizarla debido a que estas soluciones aproximadas suelen ser lo suficientemente buenas en la práctica y pueden servir como una base sólida para la toma de decisiones y el análisis de situaciones reales. Además, permite aprovechar herramientas y algoritmos desarrollados por expertos y aprender otra

posibilidad en el ámbito de la resolución de problemas de optimización mediante el uso de lenguajes de programación.

Una vez comentadas estas pequeñas consideraciones, se procede a la explicación del código.

Para empezar, se importan las librerías necesarias de OR-Tools. Dado que son distintas a las utilizadas en los modelos anteriores, se explica brevemente para qué sirve cada una de ellas:

- *from ortools.constraint_solver import routing_enums_pb2*: Este módulo contiene funciones utilizadas para configurar y personalizar el comportamiento del modelo de enrutamiento de OR-Tools. Es decir, sirve para especificar las políticas de búsqueda y otros parámetros relacionados con el enrutamiento.
- *from ortools.constraint_solver import pywrapcp*: Este módulo permite crear variables, establecer restricciones y definir el objetivo.

Además, se importan las librerías *random*, *time*, *os* para la generación de datos, el cálculo del tiempo de resolución y almacenamiento de los resultados, respectivamente.

Después de esto, se define la función principal ***main()***. A continuación se detallan los pasos más relevantes que se siguen en esta función. En primer lugar, se leen los datos del archivo *.json*. Después, se crea el objeto *manager*. Este objeto se utiliza para gestionar los índices de los nodos, ya que se asigna un índice único a cada nodo del problema que es usado internamente por OR-Tools para realizar cálculos y resolver el problema. El objeto *manager* proporciona métodos y funciones que permiten y simplifican la interacción y conversión entre índices de nodos y nodos. También se crea el objeto *routing*, que se utiliza para crear y configurar el modelo. Proporciona métodos y funciones para añadir las restricciones, definir el objetivo, determinar la estrategia de búsqueda y buscar soluciones. Estos métodos se utilizan más adelante.

A continuación, se crea una función interna llamada ***distance_callback*** que se utiliza para calcular la distancia entre los nodos. A través de los métodos de *manager* se obtienen los nodos reales correspondientes con los índices dados para poder acceder a la matriz de distancias con ellos. Esta función se registra en el objeto *routing* con el método *RegisterTransitCallback*, lo que le indica al modelo que debe utilizar la función *distance_callback* para calcular las distancias durante la búsqueda de soluciones. Una vez registrada, se utiliza el método *SetArcCostEvaluatorOfAllVehicles* para establecer el evaluador del coste de los arcos, en este caso la distancia entre nodos.

Dado que se trata de un problema CVRP, es necesario añadir la restricción de capacidad. Para ello, se crea otra función de "callback" para la demanda. Una vez creada, se registra en el objeto *routing* para indicarle al modelo que debe utilizarla para acceder a la demanda de cada nodo durante el proceso de optimización. A continuación, se procede a agregar la restricción de capacidad utilizando el método *AddDimensionWithVehicleCapacity*. Esto implica añadir una dimensión al modelo. Las dimensiones sirven para realizar un seguimiento de las cantidades acumuladas durante la ruta de cada vehículo. En este caso, se calcula la carga acumulada que transporta un vehículo y se le aplica la restricción de capacidad, garantizando que la capacidad de cada vehículo no se exceda en ningún momento.

En cuanto a la función objetivo, no es necesario proporcionarla explícitamente, ya que el solver utiliza por defecto la función objetivo definida en el modelo de enrutamiento. En este caso, minimiza la distancia total recorrida.

En la siguiente parte del código, se configuran los parámetros de búsqueda para la resolución del problema. Se crea un objeto llamado *search_parameters* que almacena los parámetros que se quieren utilizar, como la elección de la heurística y la metaheurística.

Para determinar la primera solución se elige la heurística denominada *Cheapest Insertion*, que establece que se active el arco, dentro de los posibles, que suponga un menor aumento de la longitud de la ruta. La calidad de esta primera solución puede influir en los resultados posteriores, por lo que la elección de la heurística a seguir es importante. En la web `https://developers.google.com/`

`optimization/routing/routing_options?hl=es-419#first_solution_strategy` se encuentran las diferentes heurísticas que se pueden utilizar. En cuanto a la metaheurística, se establece la Búsqueda Local Guiada para mejorar la solución inicial obtenida. Las otras metaheurísticas que se pueden utilizar se encuentran en la web `https://developers.google.com/optimization/routing/routing_options?hl=es-419#local_search_options`.

Por último se invoca al solver y se imprime por pantalla la solución obtenida. La solución incluye la ruta seguida por cada vehículo, la distancia recorrida y la carga. Además se utiliza la misma técnica que en los modelos anteriores para guardar la información relevante.

5.3.4 Análisis de la solución de un ejemplo simple

Al igual que en los modelos anteriores, se realiza un pequeño ejemplo para comprobar la correcta implementación del modelo. En este caso se va a realizar un ejemplo con 8 nodos y 2 vehículos.

En las Tablas 5.4 y 5.5 se muestran los datos del problema: la demanda de cada nodo, la distancia entre ellos y la capacidad de los vehículos.

Tabla 5.4 Datos ejemplo simple.

Nodos	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Demanda (unidades)	0	2	2	5	1	3	4	1
Capacidad de los vehículos (unidades): 10								

Tabla 5.5 Datos ejemplo simple 2.

Distancia	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	47	4	20	25	74	91	55
1	47	0	42	7	9	89	56	30
2	4	42	0	1	23	87	24	95
3	20	7	1	0	72	36	69	99
4	25	9	23	72	0	41	93	8
5	74	89	87	36	41	0	27	84
6	91	56	24	69	93	27	0	25
7	55	30	95	99	8	84	25	0

La resolución de este ejemplo se ha llevado a cabo con las dos opciones de optimización comentadas, obteniendo el mismo resultado en ambas.

La solución obtenida es la representada en la Figura 5.4. En ella se representan las rutas de los camiones y se comprueba que se cumplen todas las restricciones comentadas anteriormente.

Es importante resaltar que la representación gráfica no es realista en lo que a las distancias se refiere. Es decir, la longitud de los arcos no guarda proporción alguna con la distancia entre los nodos.

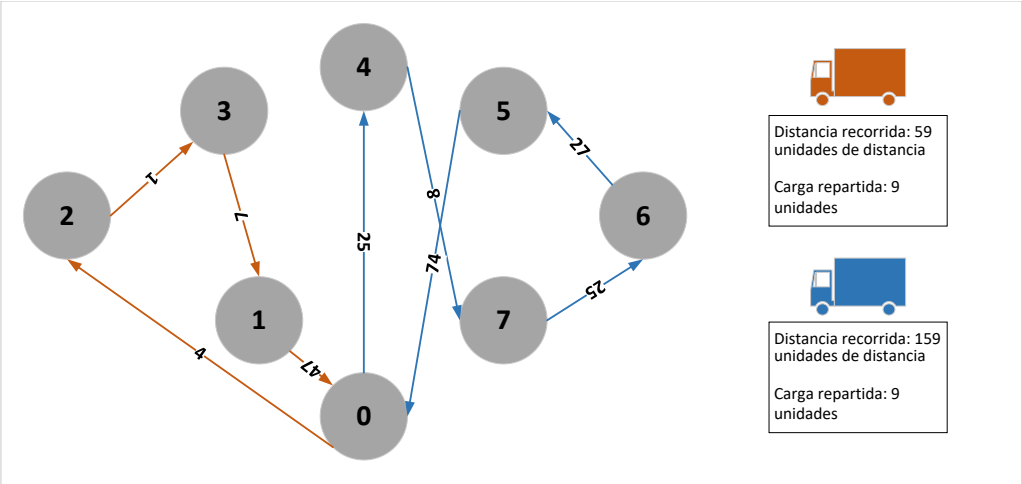

Figura 5.4 Solución ejemplo simple.

5.4 Pruebas de tiempo de computación

Al igual que en los apartados anteriores, se han llevado a cabo diversas pruebas utilizando instancias generadas aleatoriamente. En la Sección 5.4.1 se describe el proceso que se ha seguido para generar las instancias y en la Sección 5.4.2 se describen los parámetros que se van a analizar. Por último, en la Sección 5.4.3 se analizan los resultados obtenidos a partir de estas pruebas.

Es importante destacar que el análisis difiere ligeramente del seguido en los modelos anteriores. Esto se debe a la peculiaridad de contar con un modelo definido por OR-Tools que lo resuelve de forma no exacta.

5.4.1 Generación de instancias

Siguiendo con el enfoque aplicado en los modelos anteriores, se define la metodología empleada para generar las diferentes instancias. Esta metodología busca el equilibrio entre la facilidad de generación de las instancias y la diversidad y el realismo de estas.

Tamaño

En este caso, el tamaño de las instancias está determinado únicamente por el número de nodos, representado por n, y el número de vehículos, representado por k. Dado que el número de nodos suele ser mayor que el número de vehículos, para una instancia con n nodos, se establece que el número de vehículos k sea $\lceil \frac{n}{4} \rceil$. El número de nodos n incluye tanto a los clientes como al almacén principal.

Datos

Aunque los valores que tomen los datos no tienen un impacto significativo en el tiempo de computación, es importante asegurar que los datos generados den lugar a un modelo admisible.

En este modelo, la admisibilidad está principalmente determinada por la relación entre la capacidad de los vehículos y la demanda de los nodos. En cambio, la distancia entre nodos no afecta ni a la admisibilidad ni a la complejidad del modelo. Por lo tanto, se puede determinar que la distancia entre cada par de nodos tome un valor aleatorio comprendido en el intervalo [25,75]. No es relevante especificar en qué unidades se mide esta distancia.

En cuanto a la demanda de los nodos, se establece que tome valores aleatorios comprendidos entre 1 y 5 para todos los nodos, excepto para el nodo almacén. Para determinar la capacidad de los vehículos, es importante considerar que la capacidad de los vehículos siempre debe ser mayor que la demanda máxima entre los nodos. Esto garantiza que un solo vehículo pueda atender la demanda

de cualquier nodo, evitando la necesidad de que varios vehículos visiten el mismo nodo, lo cual no es admisible en este tipo de problemas.

Teniendo en cuenta los valores que toma la demanda y que se dispone de $\lceil \frac{n}{4} \rceil$ vehículos, se establece que la capacidad de los vehículos sea de 16 unidades, teniendo todos los vehículos la misma capacidad. Al igual que con la distancia, no es relevante especificar en qué unidades se miden la demanda y la capacidad, tan solo hay que tener en cuenta que ambas están medidas en las mismas unidades.

5.4.2 Parámetros a analizar

En este caso se va a analizar un único parámetro para obtener valores cuantitativos del esfuerzo computacional: **el tiempo de ejecución**. A diferencia de Gurobi, OR-Tools no dispone de ningún parámetro específico que determine el esfuerzo computacional requerido por el programa. De todas formas, esto no supone ningún problema, ya que con los dos modelos anteriores se ha demostrado que el trabajo sigue la misma tendencia que el tiempo de ejecución.

5.4.3 Análisis de los resultados

Análisis resolución exacta

En esta sección se lleva a cabo el análisis de los datos obtenidos durante la resolución del modelo CVRP de manera exacta. Los datos almacenados en las diferentes pruebas son el número de nodos n, el número de variables, el número de restricciones y el tiempo de ejecución.

Para comprender los demás resultados, es imprescindible empezar analizando la Figura 5.5. En esta figura se observa cómo aumenta el número de restricciones a medida que se incrementa el número de nodos. Se puede observar una tendencia exponencial, con un crecimiento cada vez mayor. La gran cantidad de restricciones que se genera se debe a las restricciones antibucle, ya que iteran sobre los subconjuntos de nodos.

También se observa que, a diferencia de lo que ocurre en el Capítulo 3 y en el Capítulo 4, el número de restricciones no varía para un mismo valor del número de nodos n. Esto se debe a que la creación de restricciones no depende de ningún valor que se genere aleatoriamente.

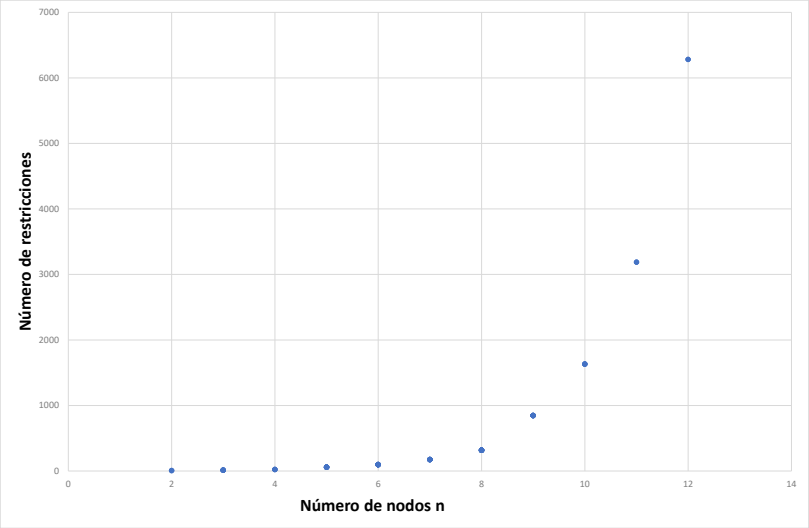

Figura 5.5 Número nodos n vs Número de restricciones.

Debido a este considerable aumento en el número de restricciones, se puede observar en la Figura 5.6 que, al aumentar el número de nodos n, también aumenta el tiempo de ejecución de manera exponencial. También se observa que para un mismo valor del número de nodos n el tiempo de ejecución puede variar debido a la generación aleatoria de la demanda. Aun así se ve claramente el aumento en el tiempo de ejecución.

Cabe destacar que se ha intentado llevar a cabo pruebas con 13 nodos, pero, a pesar de esperar un tiempo prolongado, no ha sido posible llegar a ninguna solución.

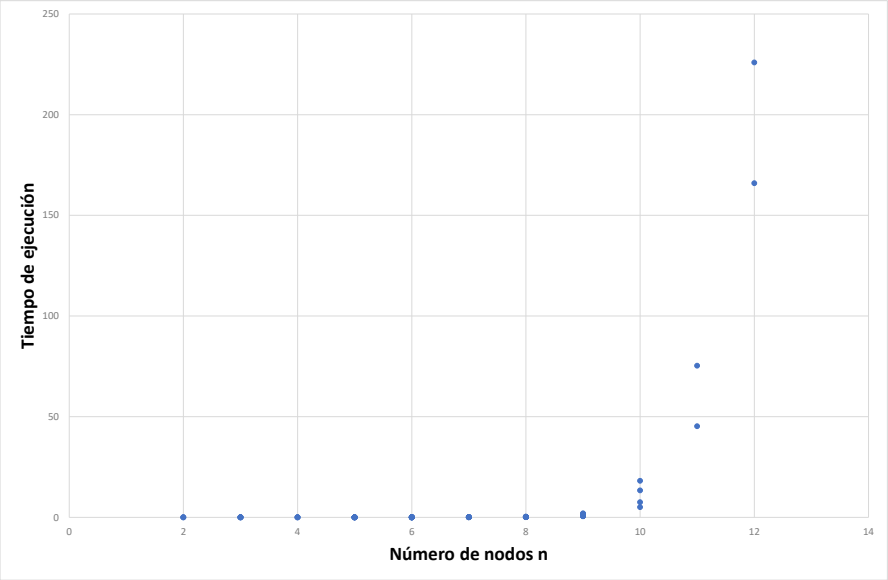

Figura 5.6 Número nodos n vs Tiempo de ejecución.

Por último se compara el número de restricciones y el número de variables con el tiempo de ejecución. En la Figura 5.7 y 5.8 se vuelve a observar el aumento considerable del tiempo de ejecución. Esto se debe al gran aumento del número de restricciones que se comentaba anteriormente.

Después de realizar todas estas comparaciones, se recomienda utilizar la resolución exacta de este modelo CVRP en casos que involucren hasta 10 nodos. A partir de ese punto, el tiempo de ejecución requerido se vuelve excesivamente largo, lo que hace que sea poco viable continuar utilizando este enfoque.

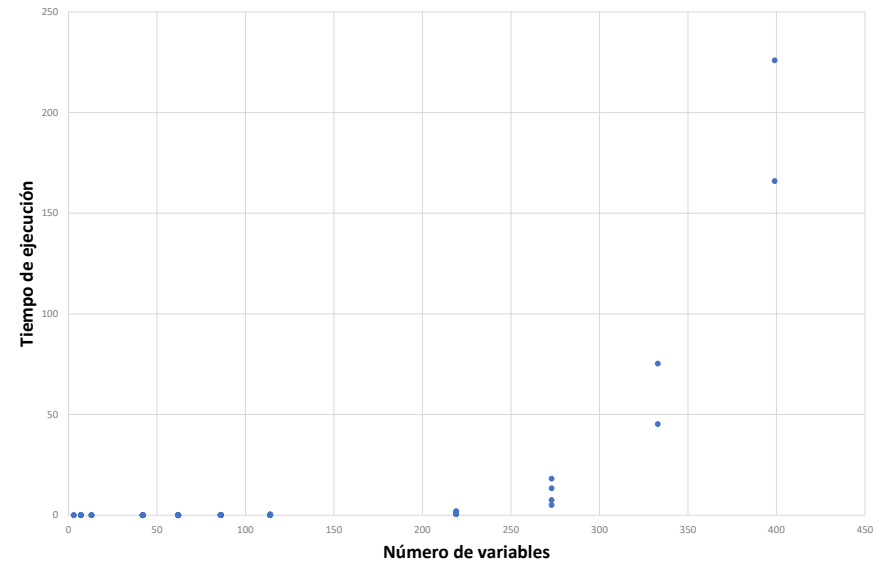

Figura 5.7 Número de variables vs Tiempo de ejecución.

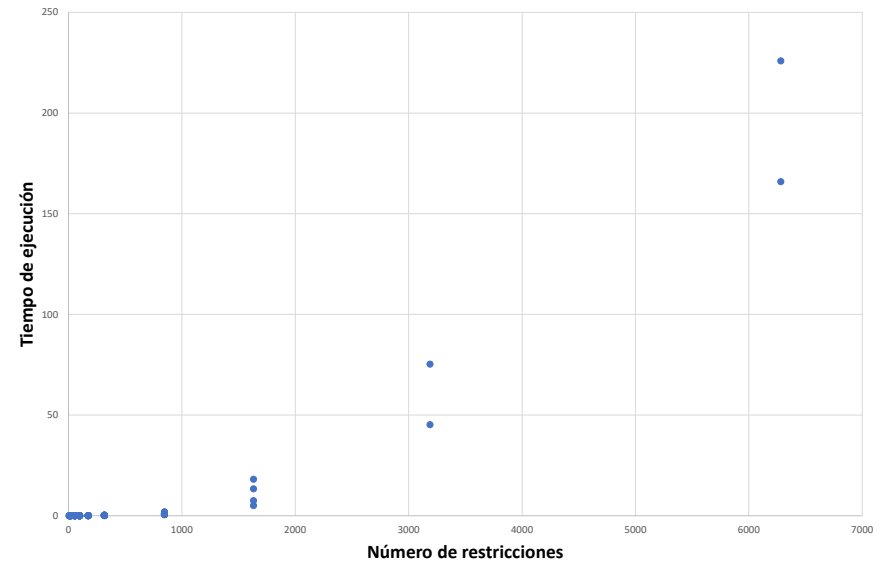

Figura 5.8 Número de restricciones vs Tiempo de ejecución.

Comparación

Dado que se tiene el mismo modelo pero implementado con la librería específica de OR-Tools, resulta interesante realizar una comparación entre ambos métodos de resolución.

En primer lugar, se resolvieron tres instancias para cada valor del número de nodos n, comprendido

entre 2 y 12, mediante los dos métodos de resolución. Dado que la resolución con heurísticas hace uso de la Búsqueda Local Guiada, hay que establecer un tiempo límite para esa búsqueda. Se establece un límite de un segundo para esta búsqueda en todas las instancias

Ambos métodos obtuvieron los mismos resultados para todas esas instancias. Para las instancias de 1 a 8 nodos, el método exacto resolvió el problema en menos de 1 segundo, mientras que se estableció un límite de tiempo de 1 segundo para el método aproximado, ya que solo se pueden introducir números enteros.

Para el resto de instancias, el tiempo de ejecución del método exacto aumentó considerablemente. En cambio, el tiempo límite del método aproximado se mantuvo en un segundo y, aun así, se obtuvieron las mismas soluciones.

Debido a la imposibilidad de realizar más pruebas con el método exacto a partir de 12 nodos, se decidió realizar otro tipo de análisis para el método aproximado. Para cada instancia, se le dan diferentes tiempos de resolución al método aproximado con el fin de observar si, al aumentar el tiempo de ejecución, la solución obtenida cambia. Esto nos puede servir para verificar que la primera solución obtenida no era óptima, pero no nos permite determinar cuál es la solución óptima.

En la Tabla 5.6 y 5.7 se observa que a partir de 15 nodos, las soluciones obtenidas varían dependiendo del tiempo de ejecución. Por lo tanto, se puede afirmar que algunas de las soluciones obtenidas seguro que no son óptimas. Sin embargo, no se puede determinar que la solución con menor valor objetivo sea óptima. Este análisis nos permite concluir que, aunque con este método no se puede garantizar la obtención de soluciones óptimas, con hasta 12 nodos suele determinar la solución óptima, pero a partir de ese punto no. Es importante destacar que esto no es una regla absoluta y puede haber excepciones.

En conclusión, generalmente el método aproximado es mejor para cualquiera de las situaciones, ya que logra obtener una buena solución en un tiempo muy corto cuando el número de nodos es pequeño. Para un número elevado de nodos, la calidad de la solución depende del tiempo empleado en la resolución. Sin embargo, dado que el método exacto no es viable para resolver estas instancias, el método aproximado se convierte en la única opción disponible. Además, cuando el único objetivo del problema sea determinar una solución admisible, el método aproximado la proporciona en un tiempo muy bajo.

Tabla 5.6 Pruebas método de resolución heurístico.

15 nodos		20 nodos		25 nodos		30 nodos	
Solución	**Tiempo**	**Solución**	**Tiempo**	**Solución**	**Tiempo**	**Solución**	**Tiempo**
365	5	396	5	502	5	796	5
347	10	375	10	493	10	766	10
347	15	375	15	471	15	799	15
347	20	375	20	470	20	784	20
347	25	375	25	471	25	785	25
347	30	375	30	471	30	784	30
347	35	375	35	471	35	787	35
347	40	375	40	471	40	766	40
347	45	375	45	470	45	758	45
347	50	375	50	470	50	775	50

Tabla 5.7 Pruebas método de resolución heurístico.

35 nodos		40 nodos		50 nodos	
Solución	**Tiempo**	**Solución**	**Tiempo**	**Solución**	**Tiempo**
821	5	885	5	1335	5
738	10	834	10	1367	10
719	15	797	15	1236	15
769	20	794	20	1270	20
716	25	794	25	1220	25
762	30	776	30	1229	30
727	35	765	35	1240	35
721	40	776	40	1216	40
710	45	783	45	1216	45
703	50	798	50	1199	50

6 Conclusiones

Partiendo del aumento de la importancia y la complejidad de la gestión de la cadena de suministro, así como de la necesidad de optimizar los recursos y mejorar la eficiencia, el análisis realizado en este trabajo puede dar lugar a las siguientes conclusiones.

En primer lugar, el uso de modelos de programación lineal ha demostrado ser una opción viable para abordar diversas situaciones que pueden surgir en la gestión de la cadena de suministro. Estos modelos permiten representar de manera precisa las relaciones existentes entre los componentes de la cadena de suministro, así como las restricciones y los objetivos de esta.

Además, se ha observado que es posible resolver estos modelos de manera exacta en diversas situaciones, lo que puede ser muy beneficioso para aumentar la eficiencia y la competitividad.

Sin embargo, se ha identificado una limitación importante en relación al tamaño de los modelos a resolver. A medida que aumenta la complejidad de la cadena de suministro, el tiempo de computación necesario para resolver los modelos puede volverse demasiado largo, resultando inviable su resolución. En consecuencia, en casos donde los modelos sean muy grandes, puede ser necesario explorar alternativas que permitan obtener soluciones aceptables en tiempos más cortos.

En resumen, se reconoce la gran utilidad de la investigación operativa y los modelos de programación lineal en la gestión de la cadena de suministro. Estas herramientas permiten abordar de manera efectiva los desafíos asociados a la cadena de suministro, optimizando los recursos y mejorando la eficiencia. No obstante, es importante considerar las limitaciones del tamaño de los modelos a resolver y explorar otras técnicas cuando sea necesario.

Apéndice A
Código Capítulo 3

```
#Librerías necesarias
from gurobipy import *
import random
from os import path
import math

#1º. Definición de los conjuntos
n=12
n_10=math.ceil(n/6)
n_5=math.ceil(n/3)

R = [f"Raw{i+1}" for i in range(n_5)]
F = [f"Product{i+1}" for i in range(n_5)]
S = [f"Supplier{i+1}" for i in range(n_10)]
P = [f"Plant{i+1}" for i in range(n_10)]
W = [f"Warehouse{i+1}" for i in range(n_10)]
C = [f"Client{i+1}" for i in range(n)]
M = [f"Mode{i+1}" for i in range (3)]


#2º. Definición de los subconjuntos
num_suppliers_per_raw_material = random.randint(1,len(S))
num_plants_per_product = random.randint(1,len(S))
num_warehouses_per_product = random.randint(1,len(S))
num_raw_materials_per_product = random.randint(1,math.ceil(len(R)/4))
num_customers_with_positive_demand = random.randint(1,math.ceil(len(C
    )/2))

Sr = {r:random.sample(S, num_suppliers_per_raw_material) for r in R}
Pf = {f:random.sample(P, num_plants_per_product) for f in F}
Wf = {f:random.sample(W, num_warehouses_per_product) for f in F}
Rf = {f:random.sample(R, num_raw_materials_per_product) for f in F}
Cf = {f:random.sample(C, num_customers_with_positive_demand) for f in
     F}
Fr = {r:[f for f in F if r in Rf[f]] for r in R}
```

```
34
35 Or = {r:Sr[r] for r in R}
36 Dr = {r:list(set([p for f in Fr[r] for p in Pf[f]])) for r in R}
37
38 #3º. Definición de los modos de transporte
39 truck=random.sample(M,1)
40 full_truck_load = { (o,d) : truck for o in S for d in P }
41 full_truck_load.update( { (o,d) : truck for o in P for d in W } )
42 less_truck_load = { (o,d) : random.sample(M,2) for o in W for d in C
       }
43
44 #-------------------------DATOS-----------------------------
45 # Demanda de cada cliente
46 demand = {(c, f):random.randint(1,5) for f in F for c in Cf[f]}
47 total_demand = sum(demand[c, f] for f in F for c in Cf[f])
48
49 # Cantidad de materia prima necesaria
50 brf = {(r, f):random.randint(1,10) for f in F for r in Rf[f]}
51 total_raw = sum(demand[c, f] * brf[r, f] for f in F for c in Cf[f]
       for r in Rf[f])
52
53 # Origen, costes y capacidad
54 origin = {o:[random.randint(10000, 100000), random.randint(total_
       demand*4//len(P),total_demand*5//len(P))] for o in P}
55 origin.update({(o):[random.randint(10000, 100000), random.randint(
       total_raw*4//len(S),total_raw*5//len(S))] for o in S})
56 origin.update({(o):[random.randint(10000, 100000), random.randint(
       total_demand*4//len((W)),total_demand*5//len(W))] for o in W})
57 origin, fcost_o, capacity_o = multidict(origin)
58
59 # Origen-Mercancía: coste fijo por asignar la mercancía k al origen o
       , límite superior de la cantidad de k envíada por el origen o,
       capacidad requerida por una unidad de k en el origen o
60 aux1 = {(s, r):[random.randint(750, 1250), random.randint(total_raw,
       total_raw*2), random.randint(1,5)] for r in R for s in Sr[r]}
61 aux1.update({(p, f):[random.randint(750, 1250), random.randint(total_
       demand,total_demand*2), random.randint(1,5)] for f in F for p in
       Pf[f]})
62 aux1.update({(w, f):[random.randint(750, 1250), random.randint(total_
       demand,total_demand*2), random.randint(1,5)] for f in F for w in
       Wf[f]})
63 ok, fcost_ok, q_ok, u_ok = multidict(aux1)
64
65 # Origen-Destino-Mercancía: coste fijo de dar k al destino d desde el
        origen o, límite superior de la cantidad de k envíada desde o
       hasta d
66 aux2 = {(s, p, r): [random.randint(1000,2000), random.randint(total_
       raw, total_raw*2)] for r in R for s in Sr[r] for p in Dr[r]}
```

```
67 aux2.update({(p, w, f): [random.randint(1000,2000), random.randint(
       total_demand,total_demand*2)] for f in F for p in Pf[f] for w in
       Wf[f]})
68 aux2.update({(w, c, f): [random.randint(1000,2000), random.randint(
       total_demand,total_demand*2)] for f in F for w in Wf[f] for c in
       Cf[f]})
69 odk, fcost_odk, q_odk = multidict(aux2)
70
71 # Origen-Destino-Modo de transporte:coste fijo de usar modo de
       transporte m entre o y d, capacidad del modo m entre o y d
72 aux3 = {(o,d,m): [0, random.randint((total_demand+total_raw)*1,2*(
       total_demand+total_raw))] for o,d in full_truck_load.keys() for m
        in full_truck_load[o,d]}
73 aux3.update({(o,d,m): [random.randint(1000,10000), random.randint(
       total_demand,2*total_demand)] for o,d in less_truck_load.keys()
       for m in less_truck_load[o,d]})
74 odm,fcost_odm,g_odm=multidict(aux3)
75
76 # Origen-Destino-Modo-Mercancía: coste unitario de llevar una mercanc
       ía desde el origen hasta el destino con el modo de transporte m
77 aux4 = {(o,d,r,m): [random.randint(2,10)] for r in R for o in Sr[r]
       for d in Dr[r] for m in full_truck_load[o,d]}
78 aux4.update({(o,d,f,m): [random.randint(2,10)] for f in F for o in Pf
       [f] for d in Wf[f] for m in full_truck_load[o,d]})
79 aux4.update({(o,d,f,m): [random.randint(2,10)] for f in F for o in Wf
       [f] for d in Cf[f] for m in less_truck_load[o,d]})
80 odkm,fcost_odmk=multidict(aux4)
81
82 # Mercancí a-Modo de transporte, cantidad de capacidad requerida por
       una unidad de k en el modo m
83 gkm = {(r,m): random.randint(1,2) for r in R for m in M}
84 gkm.update({(f,m): random.randint(1,2) for f in F for m in M})
85
86 #-------------------------VARIABLES-----------------------------
87 mod = Model('cordeau')
88 Uo = mod.addVars(origin, vtype=GRB.BINARY, name='Uo')
89 Vok = mod.addVars(ok, vtype=GRB.BINARY, name='Vok')
90 Yodk = mod.addVars(odk, vtype=GRB.BINARY, name='Yodk')
91 Zodm = mod.addVars(odm, vtype=GRB.BINARY, name='Zodm')
92 Xodkm = mod.addVars(odkm, lb=0, vtype=GRB.INTEGER, name='Xodmk')
93 mod.update()
94
95 #----------------------RESTRICCIONES----------------------------
96 # Restricción 1
97 for r in R:
98     for p in Dr[r]:
99             mod.addLConstr((quicksum(Xodkm[s,p,r,m] for s in Sr[r] for
                  m in full_truck_load[s,p]) - quicksum(Xodkm[p,w,f,m]*
                 brf[r,f] for f in Fr[r] for w in Wf[f] for m in full_
```

```
                truck_load[p,w] if p in Pf[f]) == 0),'R1[%s,%s]' % (r,
                p))
100
101 # Restricción 2
102 for f in F:
103     for w in Wf[f]:
104         mod.addLConstr((quicksum(Xodkm[p,w,f,m] for p in Pf[f] for m
                in full_truck_load[p,w]) - quicksum(Xodkm[w,c,f,m] for c
                in Cf[f] for m in less_truck_load[w,c]) == 0), 'R2[%s,%s]'
                 %(f,w))
105
106 # Restricción 3
107 for f in F:
108     for c in Cf[f]:
109         mod.addLConstr((quicksum(Xodkm[w,c,f,m] for w in Wf[f] for m
                in less_truck_load[w,c]) == demand[c,f]),'R3[%s,%s]' %(f,c
                ))
110
111 # Restricción 4
112 for o in S:
113     mod.addLConstr((quicksum(Xodkm[o,d,r,m] * u_ok[o,r] for r in R if
             o in Sr[r] for d in Dr[r] for m in full_truck_load[o,d]) -
            capacity_o[o]*Uo[o]<=0),'R4[%s]' %o)
114 for o in P:
115     mod.addLConstr((quicksum(Xodkm[o,d,f,m] * u_ok[o,f] for f in F if
             o in Pf[f] for d in Wf[f] for m in full_truck_load[o,d])-
            capacity_o[o]*Uo[o]<=0),'R4[%s]' %o)
116 for o in W:
117     mod.addLConstr((quicksum(Xodkm[o,d,f,m] * u_ok[o,f] for f in F if
             o in Wf[f] for d in Cf[f] for m in less_truck_load[o,d])-
            capacity_o[o]*Uo[o]<=0),'R4[%s]' %o)
118
119 # Restricción 5
120 for r in R:
121     for o in Sr[r]:
122         mod.addLConstr((quicksum(Xodkm[o,d,r,m] for d in Dr[r] for m
                in full_truck_load[o,d]) - q_ok[o,r]*Vok[o,r] <=0),'R5[%s
                ,%s]' %(r,o))
123 for f in F:
124     for o in Pf[f]:
125         mod.addLConstr(((quicksum(Xodkm[o,d,f,m] for d in Wf[f] for m
                in full_truck_load[o,d]) - q_ok[o,f]*Vok[o,f] <=0)),'R5[%s
                ,%s]' %(f,o))
126 for f in F:
127     for o in Wf[f]:
128         mod.addLConstr(((quicksum(Xodkm[o,d,f,m] for d in Cf[f] for m
                in less_truck_load[o,d]) - q_ok[o,f]*Vok[o,f] <=0)),'R5[%s
                ,%s]' %(f,o))
129
130 # Restricción 6.
```

```
131 for r in R:
132     for o in Sr[r]:
133         for d in Dr[r]:
134             mod.addLConstr((quicksum(Xodkm[o,d,r,m] for m in full_
                    truck_load[o,d]) - q_odk[o,d,r]*Yodk[o,d,r] <=0),'R6[%
                    s,%s,%s]' %(o,d,r) )
135 for f in F:
136     for o in Pf[f]:
137         for d in Wf[f]:
138             mod.addLConstr((quicksum(Xodkm[o,d,f,m] for m in full_
                    truck_load[o,d]) - q_odk[o,d,f]*Yodk[o,d,f] <=0),'R6[%
                    s,%s,%s]' %(o,d,f) )
139 for f in F:
140     for o in Wf[f]:
141         for d in Cf[f]:
142             mod.addLConstr((quicksum(Xodkm[o,d,f,m] for m in less_
                    truck_load[o,d]) - q_odk[o,d,f]*Yodk[o,d,f] <=0),'R6[%
                    s,%s,%s]' %(o,d,f) )
143
144 # Restricción 7
145 for o in S:
146     for d in P:
147         for m in full_truck_load[o,d]:
148             mod.addLConstr((quicksum(Xodkm[o,d,r,m]*gkm[r,m] for r in
                    R if o in Sr[r] if d in Dr[r])-g_odm[o,d,m]*Zodm[o,d,m
                    ]<=0 ),'R7[%s,%s,%s]' %(o,d,m))
149
150 for o in P:
151     for d in W:
152         for m in full_truck_load[o,d]:
153             mod.addLConstr((quicksum(Xodkm[o,d,f,m]*gkm[f,m] for f in
                    F if o in Pf[f] if d in Wf[f])-g_odm[o,d,m]*Zodm[o,d,m
                    ]<=0 ),'R7[%s,%s,%s]' %(o,d,m))
154 for o in W:
155     for d in C:
156         for m in less_truck_load[o,d]:
157             mod.addLConstr((quicksum(Xodkm[o,d,f,m]*gkm[f,m] for f in
                    F if o in Wf[f] if d in Cf[f])-g_odm[o,d,m]*Zodm[o,d,m
                    ]<=0 ),'R7[%s,%s,%s]' %(o,d,m))
158
159 #-----------------------FUNCIÓN OBJETIVO----------------------------
160 mod.setObjective(quicksum(Uo[o]*fcost_o[o] + quicksum(Zodm[o,d,m]*
        fcost_odm[o,d,m] for d in P for m in full_truck_load[o,d] ) for o
         in S)
161                 +quicksum(Uo[o]*fcost_o[o] + quicksum(Zodm[o,d,m]*
                        fcost_odm[o,d,m] for d in W for m in full_truck_
                        load[o,d] ) for o in P)
162                 +quicksum(Uo[o]*fcost_o[o] + quicksum(Zodm[o,d,m]*
                        fcost_odm[o,d,m] for d in C for m in less_truck_
                        load[o,d] ) for o in W)
```

```
163                 +quicksum(Vok[o,k]*fcost_ok[o,k] +quicksum(Yodk[o,d,k
                        ]*fcost_odk[o,d,k] + quicksum(Xodkm[o,d,k,m]*
                        fcost_odmk[o,d,k,m] for m in full_truck_load[o,d
                        ]) for d in Dr[k]) for k in R for o in Or[k])
164                 +quicksum(Vok[o,k]*fcost_ok[o,k] +quicksum(Yodk[o,d,k
                        ]*fcost_odk[o,d,k] + quicksum(Xodkm[o,d,k,m]*
                        fcost_odmk[o,d,k,m] for m in full_truck_load[o,d
                        ]) for d in Wf[k]) for k in F for o in Pf[k])
165                 +quicksum(Vok[o,k]*fcost_ok[o,k] +quicksum(Yodk[o,d,k
                        ]*fcost_odk[o,d,k] + quicksum(Xodkm[o,d,k,m]*
                        fcost_odmk[o,d,k,m] for m in less_truck_load[o,d
                        ]) for d in Cf[k]) for k in F for o in Wf[k]),GRB.
                        MINIMIZE)
166
167
168 #-------------------------RESOLUCIÓN MODELO-----------------
169 mod.optimize()
170 print("\nValor de la función objetivo: ",mod.objVal)
171
172 # Guardar el modelo
173 num_vars=mod.getAttr(GRB.Attr.NumVars)
174 filename=f"cordeau_num_vars{num_vars}.lp"
175 mod.write(filename)
176
177 # Guardar los datos computacionales
178 num_variables = mod.getAttr(GRB.Attr.NumVars)
179 num_constr = mod.getAttr(GRB.Attr.NumConstrs)
180 runtime = mod.getAttr(GRB.Attr.Runtime)
181 work =mod.getAttr(GRB.Attr.Work)
182 datos = {
183     "num_variables": num_variables,
184     "runtime": runtime,
185     "work": work
186         }
187
188 if path.exists(str('archivo_resultados.txt')):
189     with open('archivo_resultados.txt','a') as outfile:
190         outfile.write(filename+ ' ' + str(num_variables)+ ' ' + str(
                num_constr) + ' ' + str(n) + ' ' + str(runtime) + ' ' +
                str(work) +'\n' )
191 else:
192     with open('archivo_resultados.txt','a') as outfile:
193         outfile.write("Nombre Variables Restricciones n Runtime Work\n
                ")
194         outfile.write(filename+ ' ' + str(num_variables) + ' ' + str(
                num_constr) + ' ' + str(n) + ' ' + str(runtime) + ' ' +
                str(work) +'\n' )
```

Apéndice B
Código Capítulo 4

```
1  from gurobipy import *
2  import random
3  from os import path
4  import math
5  import itertools
6
7  #1º. Definición de los conjuntos
8  n=4
9  n_3=math.ceil(n/3)
10 n_23=math.ceil(2*n/3)
11
12 S = ['Warehouse'] + [f"Supplier{i+1}" for i in range(n)]
13 K = [f"Truck{i+1}" for i in range(n_3)]
14 P = [f"Part{i+1}" for i in range(n_23)]
15 T = ['Time0'] + [f"Time{i+1}" for i in range(4)]
16
17 #-----------------------DATOS--------------------
18 # Capacidad del medio de transporte k
19 Vk = {(k): random.randint(3500*n,4000*n) for k in K}
20
21 # Coste de cargar un vehículo.
22 C = random.randint(30,50)
23
24 # Capacidad máxima de un palé de producto p, coste de inventario de
       un palé de producto p,
25 #mínimo nivel de inventario de producto p
26 part = {(p): [random.randint(10, 20),random.randint(30,50), random.
       randint(100,150)] for p in P}
27 part, Vp, Hp, cpmin = multidict(part)
28
29 # Consumo medio de cada producto en cada tiempo
30 Utp = {(t,p): random.randint(25,50) for t in T[1:] for p in P}
31
32 # Coste de transporte del camión k por ir de i a j
```

```
33 Ckij = {(k,i,j): random.randint(50, 100) if i != j else 0 for k in K
       for i in S for j in S}
34
35 # Cantidad de producto p correspondiente al proveedor j. Primero se
       inicializa en 0 y luego se introducen los datos.
36 Sp = {p: [] for p in P} #Proveedores que pueden suministrar cada
       materia prima.
37 gamma = {(p, j): 0 for p in P for j in S}
38 for j in S:
39     if j != 'Warehouse':
40         products = random.sample(P,random.randint(1,n_23))
41         for p in products:
42             gamma[p, j] = random.randint(150, 200)
43             Sp[p].append(j)
44
45 #----------------VARIABLES-----------------
46 mod=Model('sadjadi')
47
48 xtkpj = {}
49
50 for p in P:
51     for s in Sp[p]:
52         for t in T:
53             for k in K:
54                 xtkpj[(t, k, p, s)] = mod.addVar(vtype=GRB.INTEGER,
                       name=f"xtkpj_{t}_{k}_{p}_{s}")
55
56
57 xMtp = mod.addVars(T, P, vtype=GRB.INTEGER, name="xMtp")
58 ytkij = mod.addVars(T, K, S, S, vtype=GRB.BINARY, name="ytkij")
59 mod.update()
60
61 #-----------------RESTRICCIONES--------------
62 # Restricción 1.
63 for t in T[1:]:
64     for k in K:
65         mod.addLConstr((quicksum(xtkpj[t,k,p,j]*Vp[p] for p in P for j
              in Sp[p]) <= Vk[k]),'R1[%s,%s]' % (t,k))
66
67 # Restricción 2
68 for p in P:
69     for j in Sp[p]:
70         mod.addLConstr((quicksum(xtkpj[t,k,p,j] for t in T[1:] for k
              in K) == gamma[p,j]), 'R2[%s,%s]' %(p,j))
71
72 # Restricción 3
73 for t in T[1:]:
74     for p in P:
75         mod.addLConstr((xMtp[t,p] >= cpmin[p]),'R3[%s,%s]' %(t,p))
76
```

```
# Restricción 4
for t in T[1:]:
    for p in P:
        mod.addLConstr(xMtp[t, p] == quicksum(xtkpj[t, k, p, j] for k
            in K for j in Sp[p]) + xMtp[T[T.index(t) - 1], p] - Utp[t,
             p], 'R4[%s,%s]' % (t, p))
# Establezco que en 'Tiempo 0' el inventario es el mínimo.
for p in P:
    mod.addLConstr(xMtp['Time0',p]==cpmin[p])


# Restricción 5
for t in T[1:]:
    for i in S[1:]:
        mod.addLConstr((quicksum(ytkij[t,k,i,j] for k in K for j in S)
             <= 1), 'R5[%s,%s]' %(t,i))

# Restricción 6
for t in T[1:]:
    for j in S[1:]:
        mod.addLConstr((quicksum(ytkij[t,k,i,j] for k in K for i in S)
             <=1),'R6[%s,%s]' %(t,j))

# Restricción 7
for t in T[1:]:
    for k in K:
        mod.addLConstr((quicksum(ytkij[t,k,'Warehouse',j] for j in S
            [1:]) <=1),'R7[%s,%s]' %(t,k))

# Restricción 8
for t in T[1:]:
    for k in K:
        mod.addLConstr((quicksum(ytkij[t,k,i, 'Warehouse'] for i in S
            [1:]) <=1),'R8[%s,%s]' %(t,k))

# Restricción 9
for t in T[1:]:
    for k in K:
        for q in S[1:]:
            mod.addLConstr((quicksum(ytkij[t,k,i,q] for i in S if i!=q
                ) == quicksum(ytkij[t,k,q,j] for j in S if j!=q)), 'R
                9[%s,%s,%s]' %(t,k,q))

# Restricción 10
for t in T[1:]:
    for k in K:
        for r in range(1, len(S[1:])+1):
            for subset in itertools.combinations(S[1:], r):
```

```
117                 mod.addConstr(quicksum(ytkij[t, k, i, j] for i in
                        subset for j in subset) <= len(subset) - 1, f'R10[{
                        t},{k},{subset}]')
118
119 # Restricción 11
120 M = 1000000
121 for t in T[1:]:
122     for k in K:
123         for p in P:
124             for j in Sp[p]:
125                 mod.addLConstr((xtkpj[t,k,p,j] <= M * quicksum(ytkij[t,
                        k,i,j] for i in S if i!=j)), 'R11[%s,%s,%s,%s]' %(t
                        ,k,p,j))
126
127
128
129 mod.update()
130 #---------------FUNCIÓN OBJETIVO-------------
131 mod.setObjective(quicksum((Ckij[k,i,j]+C)*ytkij[t,k,i,j] for t in T
        [1:] for k in K for i in S for j in S if j!=i) +
132                 quicksum(Hp[p]*xMtp[t,p] for t in T[1:] for p in P)
133                 ,GRB.MINIMIZE)
134
135 #---------------RESOLUCIÓN MODELO---------------
136 mod.optimize()
137 print("\nValor de la función objetivo: ",mod.objVal)
138
139 # Guardar el modelo
140 num_vars = mod.getAttr(GRB.Attr.NumVars)
141 index = 1
142 filename = f"sadjadi_num_vars{num_vars}_{index}.lp"
143 while path.exists(filename):
144     index += 1
145     filename = f"sadjadi_num_vars{num_vars}_{index}.lp"
146 mod.write(filename)
147
148 # Guardar los datos computacionales
149 num_variables = mod.getAttr(GRB.Attr.NumVars)
150 num_constr = mod.getAttr(GRB.Attr.NumConstrs)
151 runtime = mod.getAttr(GRB.Attr.Runtime)
152 work =mod.getAttr(GRB.Attr.Work)
153
154 if path.exists(str('archivo_resultados_s.txt')):
155     with open('archivo_resultados_s.txt','a') as outfile:
156         outfile.write(filename+ ' ' + str(num_variables)+ ' ' + str(
                num_constr) + ' ' + str(n) + ' ' + str(runtime) + ' ' +
                str(work) +'\n' )
157 else:
158     with open('archivo_resultados_s.txt','a') as outfile:
```

```
159         outfile.write("Nombre Variables Restricciones n Runtime Work\n
                ")
160         outfile.write(filename+ ' ' + str(num_variables) + ' ' + str(
                num_constr) + ' ' + str(n) + ' ' + str(runtime) + ' ' +
                str(work) +'\n' )
```

Apéndice C
Codigo Capítulo 5

C.1 Generación de instancias

```
import json
import os
import random

def create_data_model(n, k, c):

    # Se generan los datos de forma aleatoria
    data = {}
    data['distance_matrix'] = [[0] * n for _ in range(n)]
    for i in range(n):
        for j in range(i + 1, n):
            distance = random.randint(25,75)
            data['distance_matrix'][i][j] = distance
            data['distance_matrix'][j][i] = distance
    data['demands'] = [0] + [random.randint(1,5) for _ in range(n-1)]
    data['vehicle_capacities'] = [c] * k
    data['num_vehicles'] = k
    data['depot'] = 0


    # Se guardan en un archivo .json
    file_path = os.path.join(os.getcwd(), 'data.json')
    with open(file_path, 'w') as f:
        json.dump(data, f)


    return data
```

C.2 Resolución exacta

```
from ortools.linear_solver import pywraplp
import itertools
import os
import json
from os import path
import time

# Imprime las variabes no negativas de la solución óptima
def print_solution(data, x, num_vehicles):
    print("Valores de las variables:")
    for i in range(len(data['distance_matrix'])):
        for j in range(len(data['distance_matrix'])):
            if j!=i:
                for k in range(num_vehicles):
                    if x[i, j, k].solution_value() == 1:
                        print(f"x[{i}, {j}, {k}] = {x[i, j, k].solution
                            _value()}")

# Función principal
def main():

    # 1º. Se leen los datos del archivo
    file_path = os.path.join(os.getcwd(), 'data.json')
    if os.path.exists(file_path):
        with open(file_path, 'r') as f:
            data = json.load(f)


    # 2º. Se declara el solver
    solver = pywraplp.Solver.CreateSolver('SAT')
    if not solver:
        return


    # 3º. Variables
    num_nodes = len(data['distance_matrix'])
    num_vehicles = data['num_vehicles']

    x = {}
    for i in range(num_nodes):
        for j in range (num_nodes):
            if j != i:
                for k in range(num_vehicles):
                    x[(i,j,k)] = solver.BoolVar('x[%i,%i, %i]' %(i,j,k)
                        )

    for k in range(num_vehicles):
```

```
46          x[(0,0,k)] = solver.BoolVar('x[%i,%i, %i]' %(i,j,k))
47
48
49      # 4º. Restricciones
50      # Restricción 1 y 2
51      for j in range(1, num_nodes):
52          solver.Add(sum(x[i,j,k] for i in range(num_nodes) for k in
                range(num_vehicles) if i != j) == 1)
53      for i in range(1, num_nodes):
54          solver.Add(sum(x[i,j,k] for j in range(num_nodes) for k in
                range(num_vehicles) if i != j) == 1)
55
56
57       # Restricción 3 y 4
58      for k in range(num_vehicles):
59          solver.Add(sum(x[i,0,k] for i in range(num_nodes)) == 1)
60      for k in range(num_vehicles):
61          solver.Add(sum(x[0,j,k] for j in range(num_nodes)) == 1)
62
63
64      # Restricciones antibucle
65      for size in range(2, num_nodes):
66          subsets = list(itertools.combinations(range(1, num_nodes),
                size))
67          for subset in subsets:
68              for k in range(num_vehicles):
69                  solver.Add(sum(x[i, j, k] for i in subset for j in
                        subset if i != j) <= size - 1)
70
71      # No se vuelve al nodo anterior
72      for i in range(1, num_nodes):
73          for j in range(1, num_nodes):
74              if i != j:
75                  solver.Add(sum(x[i, j, k] + x[j, i, k] for k in range(
                        num_vehicles)) <= 1)
76
77      # En cada ruta solo se puede utilizar un camión
78      for k in range(num_vehicles):
79          for j in range(1, num_nodes):
80              solver.Add(sum(x[i, j, k] for i in range(num_nodes) if i
                    != j) == sum(x[j, i, k] for i in range(num_nodes) if i
                     != j))
81
82      # Capacidad
83      for k in range(num_vehicles):
84          solver.Add(sum(data['demands'][i] * x[i,j,k] for i in range(
                num_nodes) for j in range(num_nodes) if i != j) <= data['
                vehicle_capacities'][0])
85
86
```

```
87      # 5º. Se define la función objetivo
88      solver.Minimize(solver.Sum([data['distance_matrix'][i][j] * x[i,j,
            k] for i in range(num_nodes) for j in range (num_nodes) for k
            in range(num_vehicles) if i != j]))
89
90
91      # 6º. Se llama al solver y se imprime la solución
92      start_time = time.time()
93
94      status = solver.Solve()
95
96      end_time = time.time()
97
98      if status == pywraplp.Solver.OPTIMAL:
99          print('Solución óptima encontrada')
100         print('Distancia total recorrida:', solver.Objective().Value()
                )
101         print_solution(data, x, num_vehicles)
102
103     if status == pywraplp.Solver.INFEASIBLE:
104         print('Modelo inadmisible')
105
106
107     # Se guardan los parámetros para el análisis
108     runtime = end_time-start_time
109     num_var = solver.NumVariables()
110     num_constr= solver.NumConstraints()
111     print('Tiempo',runtime)
112     print('Variables', num_var)
113     print('Restricciones', num_constr)
114
115     if path.exists(str('archivo_resultados_k.txt')):
116         with open('archivo_resultados_k.txt','a') as outfile:
117             outfile.write(str(num_nodes)+ ' ' + str(num_var) + ' ' +
                    str(num_constr) + ' ' + str(runtime) +'\n')
118     else:
119         with open('archivo_resultados_k.txt','a') as outfile:
120             outfile.write("N Variables Restricciones Runtime\n")
121             outfile.write(str(num_nodes)+ ' ' + str(num_var) + ' ' +
                    str(num_constr) + ' ' + str(runtime) +'\n' )
122
123 if __name__ == '__main__':
124     main()
```

C.3 Resolución con heurísticas y metaheurísticas

```
1 from ortools.constraint_solver import routing_enums_pb2
2 from ortools.constraint_solver import pywrapcp
```

```
3  import os
4  import json
5
6  # Función para imprimir la solución obtenida
7  def print_solution(data, manager, routing, solution):
8      total_distance = 0
9      total_load = 0
10     for vehicle_id in range(data['num_vehicles']):
11         index = routing.Start(vehicle_id)
12         plan_output = 'Ruta del vehículo {}:\n'.format(vehicle_id)
13         route_distance = 0
14         route_load = 0
15         while not routing.IsEnd(index):
16             node_index = manager.IndexToNode(index)
17             route_load += data['demands'][node_index]
18             plan_output += ' {0} Carga({1}) -> '.format(node_index,
                   route_load)
19             previous_index = index
20             index = solution.Value(routing.NextVar(index))
21             route_distance += routing.GetArcCostForVehicle(
22                 previous_index, index, vehicle_id)
23         plan_output += ' {0} Carga({1})\n'.format(manager.IndexToNode(
               index),
24                                                   route_load)
25         plan_output += 'Distancia de la ruta: {}m\n'.format(route_
               distance)
26         plan_output += 'Carga total de la ruta: {}\n'.format(route_
               load)
27         print(plan_output)
28         total_distance += route_distance
29         total_load += route_load
30     print(f'Valor de la función objetivo: {solution.ObjectiveValue()
           }')
31
32
33 # Función principal
34 def main():
35
36     # 1º. Se leen los datos del archivo
37     file_path = os.path.join(os.getcwd(), 'data.json')
38     if os.path.exists(file_path):
39         with open(file_path, 'r') as f:
40             data = json.load(f)
41
42
43     # 2º. Se crean los objetos principales
44     manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix
           ']), data['num_vehicles'], data['depot'])
45     routing = pywrapcp.RoutingModel(manager)
46
```

```
47
48      # 3º. Función de distancias
49      def distance_callback(from_index, to_index):
50          from_node = manager.IndexToNode(from_index)
51          to_node = manager.IndexToNode(to_index)
52          return data['distance_matrix'][from_node][to_node]
53
54
55      # 4º. Se determina el 'coste' de los arcos
56      transit_callback_index = routing.RegisterTransitCallback(distance
            _callback)
57      routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
58
59
60      # 5º. Restricciones de capacidad
61      def demand_callback(from_index):
62          from_node = manager.IndexToNode(from_index)
63          return data['demands'][from_node]
64
65      demand_callback_index = routing.RegisterUnaryTransitCallback(
            demand_callback)
66      routing.AddDimensionWithVehicleCapacity(
67          demand_callback_index,
68          0,
69          data['vehicle_capacities'],
70          True,
71          'Capacity')
72
73
74      # 6º. Se establece la heurística y la metaheurística
75      search_parameters = pywrapcp.DefaultRoutingSearchParameters()
76      search_parameters.first_solution_strategy = (
77          routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
78      search_parameters.local_search_metaheuristic = (
79          routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH
                )
80      search_parameters.time_limit.FromSeconds(3)
81
82
83      # 7º. Se llama al solver y se imprime la solución
84      solution = routing.SolveWithParameters(search_parameters)
85
86      if solution:
87          print_solution(data, manager, routing, solution)
88          wall_time = routing.solver().WallTime()
89          print(f"Tiempo de ejecución de OR-Tools: {wall_time}
                milisegundos")
90
91  if __name__ == '__main__':
92      main()
```

Apéndice D
Funciones Gurobi

La API de de Gurobi para Pyhton proporciona una gran variedad de funciones que permiten resolver los problemas de optimización. En este apéndice se recogen las funciones más relevantes de la API utilizadas en este proyecto. Puede consultarse la dirección `https://www.gurobi.com/documentation/9.5/refman/py_python_api_details.html` para conocer el resto de funciones disponibles.

D.1 Creación del modelo

En la Tabla D.1 se muestran las funciones utilizadas para la creación del modelo.

Tabla D.1 Funciones para la creación del modelo.

Función	Descripción
Model()	Inicializa el modelo y le asigna un nombre
Model.addVars()	Añade varias variables de decisión
Model.addLConstr()	Añade una restricción lineal
Model.setObjective()	Establece la función objetivo del modelo

Es importante destacar que la función *Model.addLConstr()* solo permite introducir restricciones lineales. Sin embargo, este método es más rápido que *Model.addConstr*, que permite introducir restricciones de todo tipo. Como los modelos a resolver tienen todas las restricciones lineales y el objetivo final es analizar los tiempos de computación, resulta interesante usar la función más rápida.

D.2 Resolución del modelo

Las funciones de la Tabla D.2 son útiles tanto para la optimización como para el análisis de los resultados.

Solo si la optimización con *Model.optimize()* se completa con éxito, se pueden consultar los atributos relacionados con la solución del modelo. La función *Model.getAttr* es útil para obtener información de la solución del modelo. Se puede usar para consultar una gran variedad de atributos disponibles en `https://www.gurobi.com/documentation/9.5/refman/attributes.html#sec:Attributes`, como el número de variables, el tiempo de computación y el trabajo realizado.

Tabla D.2 Funciones para la resolución del modelo.

Función	Descripción
Model.optimize()	Optimiza el modelo
Model.getAttr()	Consulta el valor de un atributo
Model.write()	Crea un archivo y escribe los datos del modelo en él
Model.objVal()	Devuelve el valor de la función objetivo en el punto óptimo

Dado que los modelos se construyen mediante la asignación de valores aleatorios a los parámetros, es recomendable guardarlos con la función *Model.write()* por si fuera necesario volver a analizarlos en un futuro.

D.3 Auxiliares

Las funciones de la Tabla D.3 permiten una mayor eficiencia a la hora de crear los datos, las restricciones y la función objetivo del modelo.

Tabla D.3 Funciones auxiliares.

Función	Descripción
quicksum()	Una versión de la función *sum* de Python más rápida
multidict()	Divide un solo diccionario en varios diccionarios

La función *quicksum()* aporta rapidez y eficiencia al modelado cuando se escriben ecuaciones complejas. Por otra parte, la función *multidict()* resulta muy interesante ya que permite asignar una lista de claves compartidas a diferentes diccionarios.

Apéndice E
Funciones OR-Tools

OR-Tools ofrece una gran cantidad de funciones y métodos que permiten modelar y resolver problemas de optimización. En este apéndice, se incluyen las funciones más importantes utilizadas en este proyecto. Puede consultarse la dirección `https://developers.google.com/optimization/reference/python/index_python` para conocer el resto de funciones disponibles.

E.1 Librería general

En la Tabla E.1 se muestran las funciones utilizadas. Cabe destacar que la primera de las funciones sirve para crear un objeto *solver* y, una vez creado, se utilizan diferentes métodos para crear el modelo de optimización.

Tabla E.1 Funciones librería general.

Función	Descripción
Solver.CreateSolver()	Crea un objeto de la clase *solver*
solver.BoolVar()	Crea una variable booleana
solver.Add()	Añade una restricción al modelo
solver.Minimize()	Establece la función objetivo del modelo y su sentido
solver.Solve()	Resuelve el modelo de optimización

E.2 Librería específica enrutamiento

En la Tabla E.2 se describen las funciones de enrutamiento utilizadas. Al igual que en el apartado anterior, las dos primeras funciones definen los objetos y el resto de funciones son métodos de esos objetos.

Tabla E.2 Funciones librería general.

Función	Descripción
RoutingIndexManager()	Crea el administrador de índices
RoutingModel()	Crea el modelo de enrutamiento
manager.IndexToNode()	Determina el nodo correspondiente a cada índice interno
routing.SetArcCostEvaluatorOfAllVehicles()	Determina el coste asociado a cada arco
routing.AddDimensionWithVehicleCapacity	Añade las restricciones de capacidad
routing.SolveWithParameters	Resuelve el modelo según los parámetros establecidos

Índice de Figuras

Índice de Tablas

Bibliografía

Google developers: Optimización de rutas. https://developers.google.com/optimization/routing?hl=es-419.

Google developers: Restricciones de capacidad. https://developers.google.com/optimization/routing/cvrp?hl=es-419.

Python api details - gurobi optimization. https://www.gurobi.com/documentation/9.5/refman/py_python_api_details.html.

Cordeau, J. F., Pasin, F., & Solomon, M. M. (2006). An integrated model for logistics network design. *Annals of operations research*, *144*(1), 59–82.

Guancha, L. F. G., Ocampo, E. M. T., & Zuluaga, A. E. (2015). Solución del problema de ruteo capacitado considerando efectos ambientales mediante una técnica híbrida. *Scientia et technica*, *20*(3), 207–216.

Kim, G., Ong, Y. S., Heng, C. K., Tan, P. S., & Zhang, N. A. (2015). City vehicle routing problem (city VRP): A review. *IEEE Transactions on Intelligent Transportation Systems*, *16*(4), 1654–1666.

Konstantakopoulos, G. D., Gayialis, S. P., & Kechagias, E. P. (2020). Vehicle routing problem and related algorithms for logistics distribution: a literature review and classification. *Operational research*, 1–30.

Sadjadi, S. J., Jafari, M., & Amini, T. (2009). A new mathematical modeling and a genetic algorithm search for milk run problem (an auto industry supply chain case study). *The International Journal of Advanced Manufacturing Technology*, *44*, 194–200.