Reto

Análisis de puesta en uso de un sistema de entrega ecommerce

INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY
Escuela de Ingeniería y Ciencias - Ingeniería en Ciencia de Datos y Matemáticas

Optimización estocástica (Gpo 101)
MA2004B.101

Ángel Azahel Ramírez Cabello - A01383328
Annette Pamela Ruiz Abreu - A01423595
Diego Garza González - A01721186
Franco Mendoza Muraira - A01383399
Jorge Raúl Rocha López - A01740816
Valeria María Serna Salazar - A01284960

Septiembre 2023

Profesores: Fernando Elizalde Ramírez, Rafael Muñoz Sánchez

Resumen

El presente informe aborda un desafío crítico en el contexto del creciente comercio electrónico en México. A medida que esta industria experimenta un auge significativo, la entrega eficiente y puntual de productos se convierte en una necesidad imperativa para garantizar la satisfacción del cliente y el éxito empresarial. Para abordar este problema, se ha desarrollado un algoritmo de simulación de logística de entrega de productos en una empresa de e-commerce. El objetivo principal es optimizar los tiempos de recorrido, reducir los costos operativos y minimizar la huella ambiental, incluyendo la emisión de contaminantes y la congestión del tráfico en entornos urbanos.

Este enfoque se basa en la integración de una simulación de Montecarlo, que permite analizar y gestionar diversos escenarios y condiciones cambiantes que afectan la cadena de suministro y las operaciones de entrega. Además, se consideran restricciones y parámetros realistas, como el tiempo de entrega, la capacidad de carga de los vehículos, las preferencias de los clientes y otros factores clave.

A través de la implementación de este algoritmo y su aplicación en escenarios de diferentes tamaños y complejidades, se busca proporcionar a la empresa de e-commerce una herramienta efectiva para la toma de decisiones informadas y estratégicas en la gestión de su cadena de suministro y entregas. Esto no solo contribuye a mejorar la eficiencia operativa, sino que también fortalece la capacidad de la empresa para mantenerse competitiva en un mercado en constante evolución y satisfacer las crecientes expectativas de los clientes.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

Gracias a la pandemia, el e-commerce en México creció 1.8 veces hasta alcanzar un valor estimado de 11,000 millones de dólares y una penetración del 5 %.(Lacayo y Estrada 2022) Y este alto crecimiento no ha parado, según Pierre Blaise, director general de AMVO, para el 2024 se espera un 12 % de crecimiento en las ventas en línea. (Carrasco 2022) Para adaptarse a esta demanda, las empresas han tenido que ajustar sus estrategias logísticas y cumplir con entregas puntuales. Para optimizar este proceso, se emplean simulaciones bajo diversos escenarios, utilizando conceptos como distribuciones de probabilidad, teoría de colas y simulación de Montecarlo. Este informe presenta un modelo que simula la entrega de productos en una empresa de e-commerce, brindando soluciones para este desafiante entorno.

1.1. Problema

El problema central al que se enfrentan las empresas de comercio electrónico es la entrega eficiente y puntual de los productos a los compradores. Este desafío se ha vuelto aún más crítico debido al crecimiento constante del comercio en línea, especialmente durante la pandemia, que ha llevado a una demanda cada vez mayor de entregas a domicilio.

Según EY-Parthenon, una empresa de consultoría de estrategia global, entre 2019 y 2020, el comercio electrónico en México creció 1.8 veces, alcanzando un valor estimado de 11,000 millones de dólares y una penetración del 5 %. Debido a este gran crecimiento y al hecho de que muchos comercios en México no contaban con la estructura adecuada, al menos el 50 % de los usuarios de e-commerce tuvo problemas con la entrega de sus productos. (Lacayo y Estrada 2022)

Los compradores ahora esperan no solo una amplia selección de productos, sino también entregas rápidas y confiables. Los retrasos en las entregas pueden tener consecuencias negativas graves, incluida la insatisfacción del cliente, la pérdida de lealtad y la disminución de la cuota de mercado. Es esencial encontrar formas de gestionar eficazmente este proceso logístico, manteniendo al mismo tiempo un equilibrio entre la eficiencia y la calidad del servicio; ya que, se prevé que el mercado de última milla en México pase de casi 400 millones de dólares en 2020 a cerca de 1,100 millones de dólares en 2025

1.2. Justificación

Garantizar la eficiencia del sistema de entrega es vital para cualquier empresa en el siglo XXI por las siguientes razones:

1. **Mantener la Reputación y Retener Clientes:** En un mercado altamente competitivo, la reputación de una empresa es un activo invaluable. Los compradores tienen opciones y pueden cambiar fácilmente a la competencia si experimentan retrasos o problemas en las entregas. Mantener una reputación sólida y retener a los clientes existentes es esencial para el crecimiento y la rentabilidad a largo plazo.

2. **Optimización de Recursos y Costos:** Las empresas deben manejar eficazmente sus recursos para mantener un margen de beneficio saludable. La gestión óptima de la logística y la distribución reduce los costos operativos, como el consumo de combustible, los gastos de mano de obra y la necesidad de almacenamiento adicional. La simulación de diferentes escenarios permite identificar formas de optimizar estos recursos y mejorar la eficiencia.

3. **Adaptación a la Evolución del Mercado:** El comercio electrónico sigue evolucionando, y las expectativas de los clientes continúan aumentando. Para mantenerse a la vanguardia y seguir siendo competitivas, las empresas deben ser ágiles y capaces de adaptarse a nuevos desafíos. La implementación de modelos de simulación ayuda a anticipar y prepararse para futuros cambios en la demanda, en las tendencias de compra y en las condiciones del mercado.

1.3. Objetivo

Crear un algoritmo que permita la simulación de la logística de entrega de productos a diversos clientes de una empresa de e-commerce, con el objetivo de optimizar los tiempos de recorrido, disminuir los costos asociados, y reducir la huella ambiental, incluyendo la emisión de contaminantes y la congestión del tráfico dentro de las ciudades. Además, se

integrará una simulación de Montecarlo para analizar y abordar distintos escenarios y condiciones variables que podrían influir en el proceso logístico, ayudando así a tomar decisiones informadas y eficaces en la planificación de las estrategias logísticas de la empresa.

Capítulo 2

Trabajos relacionados

En este capítulo se brinda información acerca de diferentes algoritmos que pueden ayudar a resolver la problemática como el TSP (Travelling Salesman Problem) o el VRP (Vehicle routing problem). Y sobre qué es una simulación de Montecarlo y cómo es que puede ayudar a modelar ciertos problemas complejos, en los que sería imposible revisar todas las combinaciones posibles.

2.1. TSP

El problema del agente viajero (TSP por sus siglas en inglés) es un problema combinatorial catalogado como NP-Completo en el área de computación e investigación de operaciones. El reto radica en encontrar el camino más corto para que un vendedor recorra, dado un punto de inicio, una cantidad de ciudades (que son los nodos) y, opcionalmente, un punto final. (Daniells s.f.)

Con base en lo anterior es posible adaptarlo a otros problemas que surgen en la vida profesional, tales como aumentar la rentabilidad de una empresa al disminuir el tiempo de traslado, así como las emisiones de carbono. (Lee s.f.)

Actualmente, las tres opciones más comunes para solucionar el problema son:

1. **El método de fuerza bruta**, también conocido como el Enfoque Naive, calcula y compara todas las posibles permutaciones de rutas o caminos para así determinar la solución más corta. (Tobin 2023)

2. **El algoritmo de ramificación y acotación** comienza creando una ruta inicial, generalmente desde el punto de inicio hasta el primer nodo en un conjunto de

ciudades. Luego, explora sistemáticamente diferentes permutaciones para extender la ruta un nodo a la vez. Cada vez que se agrega un nuevo nodo, el algoritmo calcula la longitud de la ruta actual y la compara con la ruta óptima encontrada hasta ese momento. Si la ruta actual ya es más larga que la ruta óptima, el algoritmo .ªcota.º elimina esa rama de exploración, ya que no conduciría a una solución óptima. (*Traveling Salesperson problem using branch and bound* s.f.)

3. Con el método del **Vecino más Cercano**, se empieza en un punto de inicio seleccionado al azar. Desde ahí, se encuentra el nodo no visitado más cercano y se añade a la secuencia. Luego, se avanza al siguiente nodo y se repite el proceso de encontrar el nodo no visitado más cercano hasta que todos los nodos estén incluidos en el recorrido. Finalmente, se regresa a la ciudad de inicio para completar el ciclo. Es importante mencionar que este algoritmo no suele encontrar la mejor solución posible. (Kirkpatrick 2020)

2.2. VRP

El problema de enrutamiento de vehículos, conocido como VRP por sus siglas en inglés es un modelo de optimización de rutas que adopta las características de un TSP y BPP (bin packing problem) para planear los trayectos de una flotilla de vehículos que deben cumplir con una demanda propuesta a partir de un nodo central de donde parte el suministro, esta versión del problema fue introducida por Dantzig y Ramser, quienes en 1959 propusieron una aplicación para la entrega de gasolina a estaciones de servicio. (Daza, Montoya y Narducci 2009)

Los problemas VRP de cualquier naturaleza, desde el original hasta los aplicados en la actualidad, están clasificados como problemas de optimización combinatoria, su número de rutas posibles está determinado por las n ubicaciones del problema, donde la cantidad de rutas está dado por $(n-1)!$, por lo que, una planeación para más de 20 ubicaciones necesitaría bastante tiempo de cálculo usando una búsqueda exhaustiva que tome en cuenta todas las soluciones posibles usando los equipos de cómputo actuales, por lo que, se deben aplicar modelos heurísticos para hallar propuestas de resolución cercanas a lo óptimo. (López 2021)

2.3. Simulación de Montecarlo

Es una técnica matemática, la cual es utilizada para estimar los posibles resultados de un evento incierto usando los principios de la estadística inferencial. La simulación de Monte Carlo crea un modelo con posibles resultados utilizando una distribución de probabilidades para las variables que sean inciertas y se calculan múltiples veces

utilizando números aleatorios dentro de un rango establecido, para obtener mejores resultados es necesario repetirlo miles de veces para poder tener un gran número de resultados posibles. (IBM 2018)

No existe un número corridas para en una simulación de Montecarlo que garantice que haya eficiencia. Depende de algunos factores como la precisión deseada, la complejidad del problema que se este modelando, la variabilidad de los datos y la disponibilidad de recursos computacionales que se tengan disponibles. Aunque generalmente, conforme se va aumentando el número de corridas se va mejorando la precisión de las estimaciones, existe la ley de rendimientos decreciente, la cual en este caso conforme se aumenta el número de iteraciones, va disminuyendo el aumento en la precisión.

Entre algunas de las desventajas que pueden tener las simulaciones de Montecarlo es que dependen de la información de entrada y de la elección de la distribución de probabilidad. Además que dependiendo del número de los diferentes estados que puedan ocurrir en la simulación, y las veces que se simule, puede ser muy demandante computacionalmente. (AWS s.f.)

2.4. Simulación de sistemas de entrega de productos o de ruteo de vehículos

El proceso de simulación basada en computadoras es una herramienta que permite imitar la operación de procesos del mundo real para obtener pronósticos certeros sobre los resultados de distintos parámetros en el entorno de trabajo sin necesidad de alterar de perturbarlo, solamente tomando en cuenta diferentes estados del entorno, claramente esta área de estudio depende mucho de la complejidad de los sistemas estudiados, puesto que, con un número elevado de variables se reduce la capacidad del algoritmo computacional para proveer una respuesta satisfactoria a la predicción de riesgo sobre cierto cambio en el proceso, por lo que, la hipótesis inicial de la simulación debe abarcar todas las variables a evaluar, pero debe estar lo suficientemente simplificada para ser procesada por el sistema de cómputo, por consiguiente, la modelación generalmente requiere que los sucesos del proceso ocurran durante intervalos de tiempo discretos, los cuales no son fácilmente trasladables a un entorno real.

Por lo anterior se recomienda seguir la siguiente metodología al momento de diseñar y ejecutar una simulación de sistemas de entrega o ruteo de vehículos:

1. Formulación del problema
2. Establecer los objetivos

3. Objetivo general
4. Objetivos específicos
5. Definición y conceptualización del sistema a modelar
6. Recolección de datos
7. Codificación del modelo
8. Verificación del modelo
9. Validación del modelo

(Giraldo-Picon, Giraldo-GarcÃa y Valderrama-Ortega 2018)

2.5. Algoritmos de búsqueda

Un espacio de búsqueda es un conjunto de soluciones posibles a un problema concreto. En informática, se utiliza para referirse al dominio de la función a ser optimizada en algoritmos de búsqueda. En el campo de la inteligencia artificial, la búsqueda en el espacio de estados es un proceso que considera sucesivos estados de una instancia con el objetivo de encontrar un estado final con las características deseadas. Los problemas se modelan a menudo como un espacio de estados, un conjunto de estados que contiene el problema. El conjunto de estados forma un grafo donde dos estados están conectados si hay una operación que se pueda llevar a cabo para transformar el primer estado en el segundo. La búsqueda en el espacio de estados difiere de los métodos de búsqueda tradicionales porque el espacio de estados está implícito: el grafo del espacio de estados típico es demasiado grande para generarlo y guardarlo en memoria. En su lugar, los nodos se generan en el momento en que se exploran y generalmente son descartados después. (documentation s. f.)

2.5.1. Búsqueda no informada

Los algoritmos de búsqueda ciega o no informada son aquellos que no dependen de información específica del problema para resolverlo, sino que proporcionan métodos generales para recorrer los árboles de búsqueda asociados a la representación del problema. Estos algoritmos se basan en la estructura del espacio de estados y determinan estrategias sistemáticas para su exploración; es decir, que siguen una estrategia fija a la hora de visitar los nodos que representan los estados del problema. Además, son algoritmos exhaustivos; es decir, en el peor de los casos pueden acabar recorriendo todos los nodos del problema para hallar la solución.

Existen dos estrategias básicas para recorrer un espacio de búsqueda: en anchura y en profundidad. La búsqueda en anchura consiste en explorar todos los nodos de un nivel antes de pasar al siguiente nivel, mientras que la búsqueda en profundidad consiste en explorar todos los nodos de una rama antes de pasar a la siguiente rama. En la búsqueda en anchura, se expanden primero los nodos más cercanos al nodo inicial, mientras que en la búsqueda en profundidad se expanden primero los nodos más profundos del árbol. (Caparrini 2016b)

2.5.2. Búsqueda informada

La búsqueda informada es una estrategia de búsqueda que utiliza conocimiento específico del problema para encontrar soluciones de manera más eficiente. A diferencia de la búsqueda no informada, que evalúa el siguiente estado sin conocer a priori si es mejor o peor que el anterior, la búsqueda informada utiliza una función heurística que estima el costo de llegar a la solución deseada. Esta función heurística se utiliza para guiar la búsqueda hacia las soluciones más prometedoras.

Entre los algoritmos de búsqueda informada se encuentran el algoritmo de búsqueda Greedy, que minimiza el costo estimado hasta la meta, y el algoritmo A*, que combina la función heurística con el costo real del camino recorrido para encontrar la solución óptima. La búsqueda informada es especialmente útil en problemas en los que el tamaño del espacio de búsqueda es muy grande y la búsqueda no informada sería ineficiente. (Caparrini 2016a)

Capítulo 3

Definición del Problema

3.1. Problema a Resolver

Suponiendo ser una empresa de e-commerce, se buscará satisfacer la demanda diaria de paquetes de tal empresa, optimizando costos de viaje, tales como gasolina y tiempo dedicado a las entregas de estos paquetes. Esto se realizará usando demandas estimadas, fechas y horas requeridas para la entrega de paquetes. Además, se tomarán en cuenta los camiones de carga disponibles y sus capacidades de carga, siempre buscando la combinación óptima que se debe utilizar para satisfacer la demanda diaria y minimizar costos.

3.2. Variables y Parámetros

Variables:

- Cantidad de repartidores/camiones.
- Cantidad de rampas de carga.
- Tiempo de recorrido resultante.
- Gasolina utilizada.
- Paradas para cargas gasolina.

Parámetros:

- Paquetes a entregar en el día.
- Lugares de entrega.

- Costo de traslado (distancia*gasolina).
- Límite de tiempo de recorrido.
- Cantidad de camiones.
- Velocidad promedio de vehículo.
- Capacidad de gasolina de vehículo.

3.3. Función Objetivo y Restricciones

La función objetivo de este problema sería minimizar la cantidad de tiempo que toma repartir los paquetes (trasladarse de x_i a x_j), al igual que minimizar la gasolina usada en los trayectos de entrega. Con el objetivo de satisfacer las demandas diarias propuestas por la compañía. Por último, también se puede agregar de parámetro la cantidad de vehículos disponibles para la entrega de los paquetes, los cuales se necesitan usar, buscando minimizarlos, usando menos gasolina y mano de obra.
Las restricciones del problema planteado serían la cantidad de clientes a los que se les va a entregar en el día, la cantidad de paquetes que serán entregados, en total y por cliente, las ubicaciones a dónde serán entregados los paquetes, la fecha a la que se deben entregar los paquetes, la velocidad promedio del repartidor con el automóvil, y el costo de la gasolina dónde se está rellenando la gasolina, este siendo una variable fija dependiendo del costo actual.

3.3.1. Formulación TSP

Analizando el problema y resolviéndolo con el método de agente viajero, la formulación sería la siguiente:

$$\text{Minimizar:} \quad \sum_{i=1}^{n} \sum_{j=1} c_{ij} x_{ij}$$

Sujeto a:

$$\sum_{j \neq i} x_{ij} = 1, \quad \forall i \neq j$$

$$\sum_{i \neq j} x_{ij} = 1, \quad \forall j \neq i$$

$$u_i - u_j + n x_{ij} \leq n - 1, \quad 2 \leq i \neq j \leq n$$

$$x_{ij} \in \{0, 1\}, \quad i, j = 1, \ldots, n$$

$$u_i \geq 0, \quad i = 1, \ldots, n$$

Donde:

- n es el número total de ubicaciones o lugares de entrega (nodos).
- c_{ij} es el costo de viajar de la ciudad i a la ciudad j. El costo puede ser tiempo o gasolina.
- x_{ij} es una variable binaria que toma el valor 1 si se viaja del nodo i al nodo j, y 0 en caso contrario.
- u_i es una variable auxiliar que se utiliza para eliminar subciclos en la solución. Es una variable continua que tiene un valor asociado con cada nodo i.
- Las restricciones de suma aseguran que desde cada lugar de entrega i, se debe viajar a exactamente una ubicación diferente.
- La restricción de eliminación de subciclos evita que se formen ciclos pequeños no deseados en la solución.

3.3.2. Formulación VRP

Analizando el problema y resolviéndolo con el método VRP; es decir, si tenemos muchos vehículos y distintas ubicaciones, la formulación sería la siguiente:

Dado:

- Conjunto de ubicaciones de entrega: $C = \{1, 2, \ldots, n\}$
- Ubicación del depósito: O
- Demanda de cada ubicación: $q_i, \quad i \in C$

- Capacidad del vehículo: Q
- Distancia entre ubicaciones: $d_{ij}, \quad i, j \in C$
- Número de vehículos: K

Variables de Decisión:

- Variable binaria x_{ij}:

$$x_{ij} = \begin{cases} 1, & \text{si el vehículo viaja directamente desde el cliente } i \text{ al cliente } j \\ 0, & \text{en otro caso} \end{cases}$$

- Variable entera u_i: Demanda acumulada en el cliente i

Objetivo:

$$\text{mín} \sum_{i \in C} \sum_{j \in C, j \neq i} d_{ij} \cdot x_{ij}$$

Sujeto a:

$$\sum_{j \in C, j \neq i} x_{ij} = 1, \quad \forall i \in C$$
$$\sum_{i \in C, i \neq j} x_{ij} = 1, \quad \forall j \in C$$
$$u_i + q_i \leq u_j + Q \cdot (1 - x_{ij}), \quad \forall i \in C, j \in C, j \neq 0$$
$$u_i \geq q_i, \quad \forall i \in C$$
$$u_i \geq 0, \quad \forall i \in C$$
$$x_{ij} \in \{0, 1\}, \quad \forall i \in C, j \in C$$
$$u_i \in \mathbb{Z}^+, \quad \forall i \in C$$

3.4. Datos

3.4.1. Vehículos de transporte

Tipo Unidad	Modelo	Capacidad Vol m3	Capacidad Carga kg
514 SEMI LONG S 300	2019	8.539054	2470.0
716 LONG SERIE 300	2019	8.539054	2470.0
CHASIS CABINA	2015	6.325899	1480.0
	2020	8.479113	1206.0
	2021	7.363343	2880.0
ELF 300	2013	8.705302	2500.0
	2015	8.705302	2500.0
MANAGER FURGON L2H2	2021	11.500000	1850.0

Tipo Unidad	Modelo	Consumo Gasolina km/L	Capacidad tanque L
514 SEMI LONG S 300	2019	8.20	70.0
716 LONG SERIE 300	2019	8.20	70.0
CHASIS CABINA	2015	11.19	60.0
	2020	11.68	80.0
	2021	11.80	80.0
ELF 300	2013	11.00	75.0
	2015	9.40	75.0
MANAGER FURGON L2H2	2021	13.50	125.0

Cuadro 3.1: Descripción de vehículos de transporte
(Nissan Mexicana 2015-2021; Hino 2019; Norte 2013-2015; Mexicana 2020)

3.4.2. Información de pedidos

El archivo de pedidos tiene 31654 registros con el código del producto, las unidades que se pidieron y la factura (el cliente). Es importante resaltar que si un cliente pidió más de un producto diferente, se registra como una compra separada.

Al agrupar los datos por unidades compradas y contar la cantidad de registros, se obtuvo la siguiente tabla que nos indica la frecuencia con la que se compra cierta cantidad de unidades.

	Producto	Unidades	Factura
count	31654.000000	31654.000000	31654.000000
mean	29598.876572	1.106432	580401.154862
std	12588.167286	0.613263	214301.475443
min	23.000000	1.000000	379.000000
25 %	18267.000000	1.000000	448228.000000
50 %	34248.000000	1.000000	654167.000000
75 %	38167.000000	1.000000	734489.250000
max	53475.000000	20.000000	997165.000000

Cuadro 3.2: Descripción de tabla de pedidos

Unidades	Frecuencia
1	29635
2	1510
3	203
4	174
5	31
6	41
7	6
8	5
9	3
10	34
12	1
15	1
16	3
18	1
20	6

Cuadro 3.3: Descripción de frecuencia de unidades

3.4.3. Información de productos

El archivo de productos tiene 53632 registros con el código del producto y el volumen en metros cúbicos.

	Producto	Volumen mtro3
count	53632.000000	5.363200e+04
mean	26816.500000	1.117353e-01
std	15482.369155	3.267456e-01
min	1.000000	1.000000e-08
25 %	13408.750000	9.568000e-04
50 %	26816.500000	4.511000e-03
75 %	40224.250000	5.044900e-02
max	53632.000000	6.272098e+00

Cuadro 3.4: Descripción de la tabla de productos

Para obtener la tabla que detalla la frecuencia con la cual cada producto es solicitado en los pedidos, se realizó una operación de unión entre el archivo que contiene la información de los productos y los datos relacionados con los pedidos en sí. Mediante esta unión de datos, se logró agrupar de manera coherente y eficiente la información con base en los códigos únicos asignados a cada producto. A continuación, se procedió a llevar a cabo un proceso de conteo de los registros individuales correspondientes a cada producto, lo que permitió tener una visión clara y cuantitativa de la frecuencia de solicitudes para cada artículo en cuestión. Una vez que se completaron estas etapas de procesamiento de datos, se obtuvo una tabla resultante que refleja de manera detallada la relación entre los productos y la frecuencia de sus solicitudes en los pedidos.

La utilidad y relevancia de esta tabla resultante se reflejaron de manera efectiva al generar una gráfica representativa que visualiza de manera gráfica y clara la distribución de los datos que luego analizaremos en profundidad. Dicha gráfica brinda una comprensión visual instantánea de cómo se distribuye la frecuencia de solicitud entre los diferentes productos, lo que puede servir como base para tomar decisiones informadas.

	Producto	Frecuencia	Volumen
count	4761.000000	4761.000000	4.761000e+03
mean	27429.013233	6.648603	1.034189e-01
std	15038.019317	14.435972	3.033956e-01
min	23.000000	1.000000	1.000000e-08
25 %	17238.000000	1.000000	9.568000e-04
50 %	28005.000000	2.000000	4.509000e-03
75 %	40445.000000	5.000000	4.556950e-02
max	53475.000000	282.000000	3.850000e+00

Cuadro 3.5: Descripción de la tabla de la frecuencia de los productos

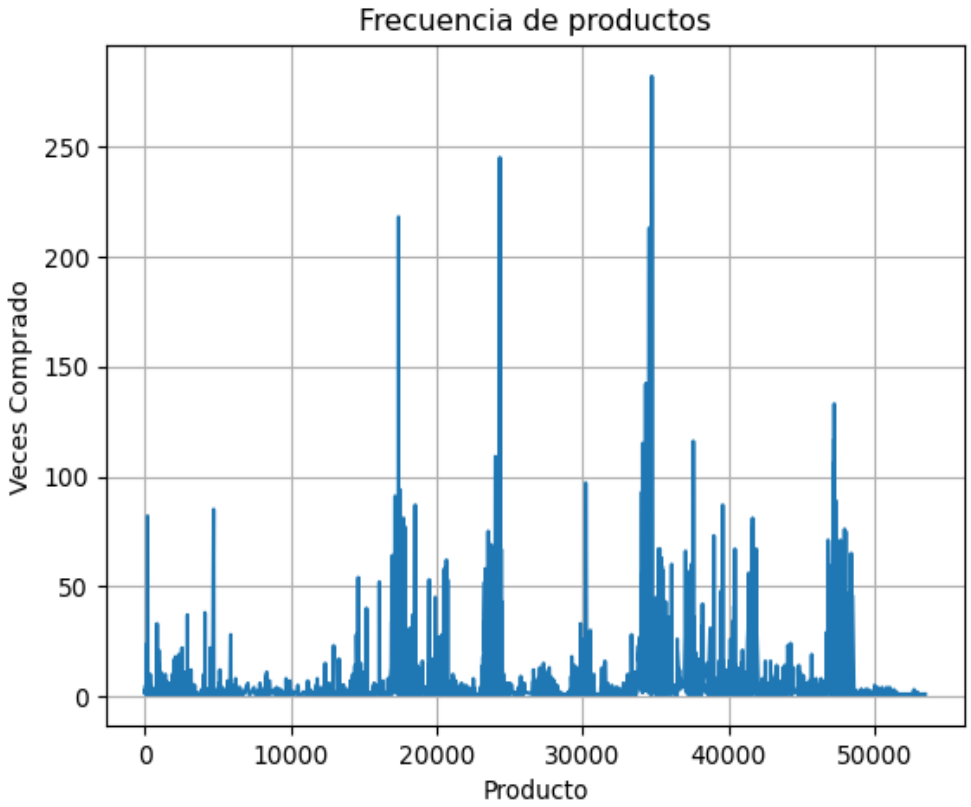

Figura 3.1: Frecuencia de productos comprados

3.4.4. Información de clientes

Al agrupar los datos por facturas y contar la cantidad de registros, se obtuvo la siguiente tabla que nos indica que hay 24803 clientes que en promedio solo realizan una compra.

	Factura
count	24803.000000
mean	1.276217
std	0.828117
min	1.000000
25 %	1.000000
50 %	1.000000
75 %	1.000000
max	23.000000

Cuadro 3.6: Descripción de los clientes

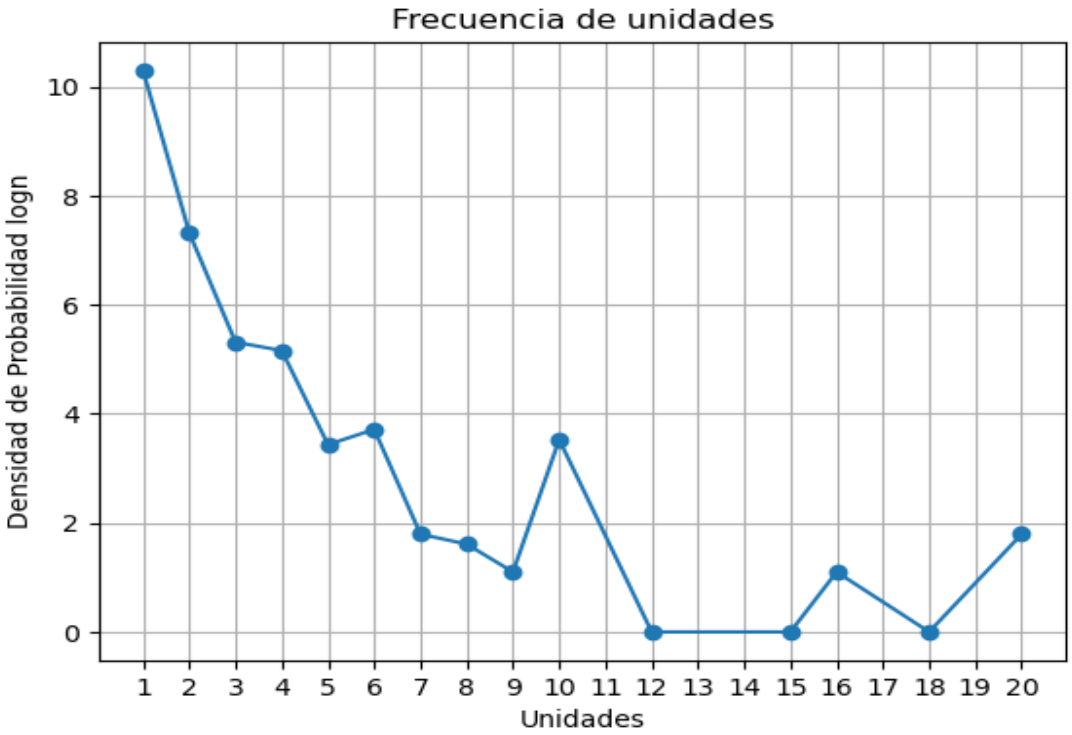

Figura 3.2: Frecuencia con logaritmo natural de las unidades compradas

3.4.5. Ubicaciones de entrega

El archivo con las ubicaciones de entrega contiene 344 puntos de entrega diferentes. Cada fila de la tabla representa un nodo y la tabla tiene 4 columnas: colonia, código postal, calle, número de casa. Esta tabla es valiosa para identificar y rastrear la geolocalización de cada colonia y proporciona información esencial para la gestión logística.

Al procesar los datos y utlizar la librería geopy.geocoders de Python, se obtuvieron las latitudes y longitudes de cada dirección.

	direccion	latitud	longitud
0	4026, JESUS M GARZA, Monterrey, Nuevo León, 64...	25.686482	-100.273079
1	404, PEDRO ZORRI, Monterrey, Nuevo León, 64140...	25.658769	-100.307752
2	5300, AV FUNDADORES, Monterrey, Nuevo León, 64...	25.625050	-100.301162
3	4001, AV ABRAHAM LIN, Monterrey, Nuevo León, 6...	25.632088	-100.286045
4	2923, PINO SILVESTRE, Monterrey, Nuevo León, 6...	25.636429	-100.288347
...	...	...	...
341	8616, VILLA ALEGRE, Monterrey, Nuevo León, 641...	25.744850	-100.366996
342	2503, AV LINCOLN, Monterrey, Nuevo León, 64270...	25.714393	-100.349825
343	6845, ABASOLO, Monterrey, Nuevo León, 64260, M...	25.740099	-100.333635
344	26001, AV VICENTE GUE, Monterrey, Nuevo León, ...	25.727254	-100.191688

Cuadro 3.7: Ejemplo de la tabla de direcciones
(Monterrey 2023)

3.4.6. Matriz de distancias

Con la latitud y longitud calculamos la distancia geográfica entre cada ubicación utilizando la fórmula de Haversine, una técnica matemática ampliamente reconocida para estimar distancias en la superficie de una esfera, como la Tierra. Esta fórmula es particularmente útil cuando se trata de medir distancias en coordenadas geográficas, ya que tiene en cuenta la curvatura de la Tierra. (Miguel 2011)

	0	1	2	3	...344
0	0.000000	4.644359	7.388132	6.186319	9.330111
1	4.644359	0.000000	3.807212	3.679257	13.901149
2	7.388132	3.807212	0.000000	1.705719	15.796092
3	6.186319	3.679257	1.705719	0.000000	14.191158
4	5.772193	3.155105	1.803155	0.535003	13.993604
5	5.036002	1.826380	2.406617	1.860719	13.804768
6	9.330111	13.901149	15.796092	14.191158	0.000000
7	16.140844	15.799689	19.259426	19.478355	21.788652
8	9.442485	9.060083	12.676565	12.730885	16.381549
9	5.372941	2.548120	6.096034	6.215290	14.573262
10	5.218111	1.065231	2.746283	2.720032	14.281375
...	...	...	...	...	...
344	9.330111	13.901149	15.796092	14.191158	0.000000

Cuadro 3.8: Ejemplo de la matriz de direcciones

Capítulo 4

Implementación

La implementación se compone de varias secciones clave que se centran en diferentes aspectos de la logística y la optimización. La primera sección se dedica a la distribución, en la cual se diseñarán estrategias con el propósito de asignar de manera óptima los productos a los distintos clientes, considerando variables como la ubicación geográfica y la demanda individual de cada cliente. A continuación, se aborda la simulación, permitiendo la creación de escenarios virtuales que replican el proceso logístico en un entorno controlado. Esta simulación posibilita realizar análisis exhaustivos y pruebas sin afectar las operaciones del mundo real.

Un componente crucial de la implementación es la simulación de Montecarlo, que añade una capa adicional de solidez al algoritmo. Al considerar diversas condiciones variables que podrían influir en el proceso logístico, esta simulación permite tomar decisiones informadas y estratégicas. La integración de esta simulación amplía la capacidad del algoritmo para adaptarse a escenarios cambiantes y tomar medidas proactivas.

4.1. Distribuciones

Para poder realizar simulaciones efectivas, debemos estimar el tipo de distribución de ciertos datos como la cantidad de unidades que se piden y el producto que se pide en cada compra.

Al graficar las frecuencias de las unidades que se piden, se estimó que estos datos tenían una distribución exponencial con un valor de lambda muy alto como se observa en la siguiente figura.

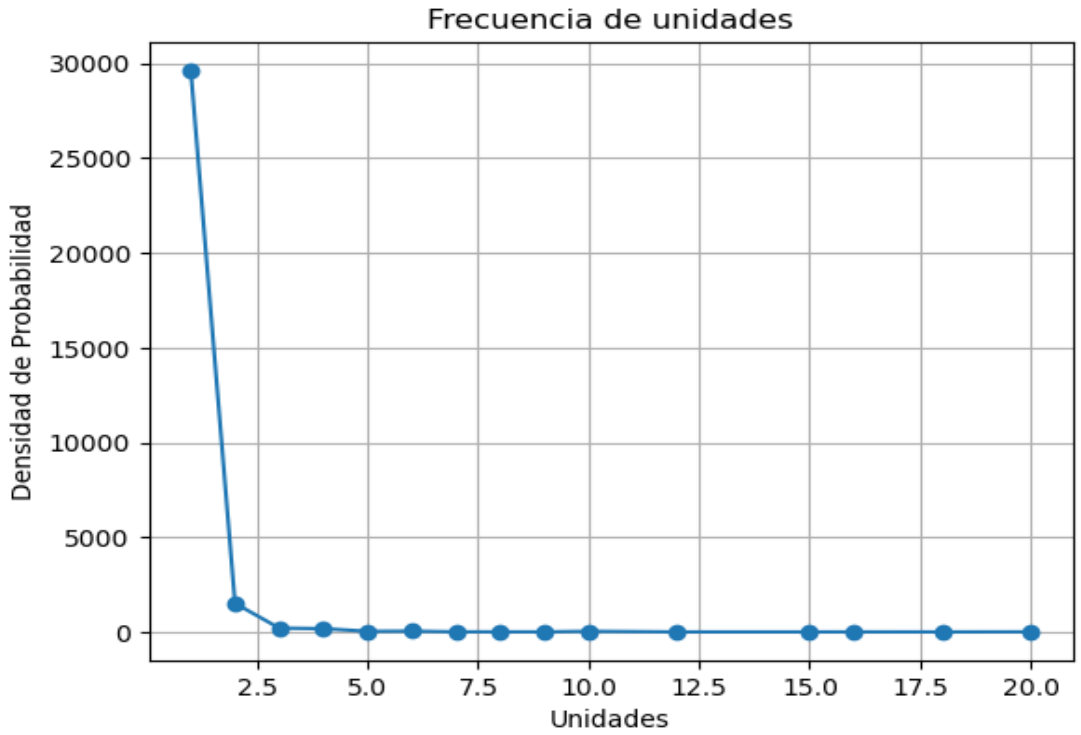

Figura 4.1: Frecuencia de las unidades compradas

Sin embargo, al analizar la figura 4.2 en donde se le aplicó un logaritmo natural a la frecuencia de cada unidad comprada, se observa en los picos irregulares que no tiene una distribución exponencial. Además, esta hipótesis se demostró estimando el parámetro lambda, realizando una simulación con las probabilidades exponenciales y comparando el promedio de esta simulación con el promedio real. El proceso de la simulación se explicará en la siguiente sección con la simulación que proporcionará los datos correctos.

El parámetro estimado lambda fue: $\lambda = 0{,}325006$

Promedio obtenido con la simulación con distribución exponencial: 2,77721

Promedio real: 1,106432

Comparando el promedio real y el simulado se puede comprobar que los datos no tienen una distribución exponencial.

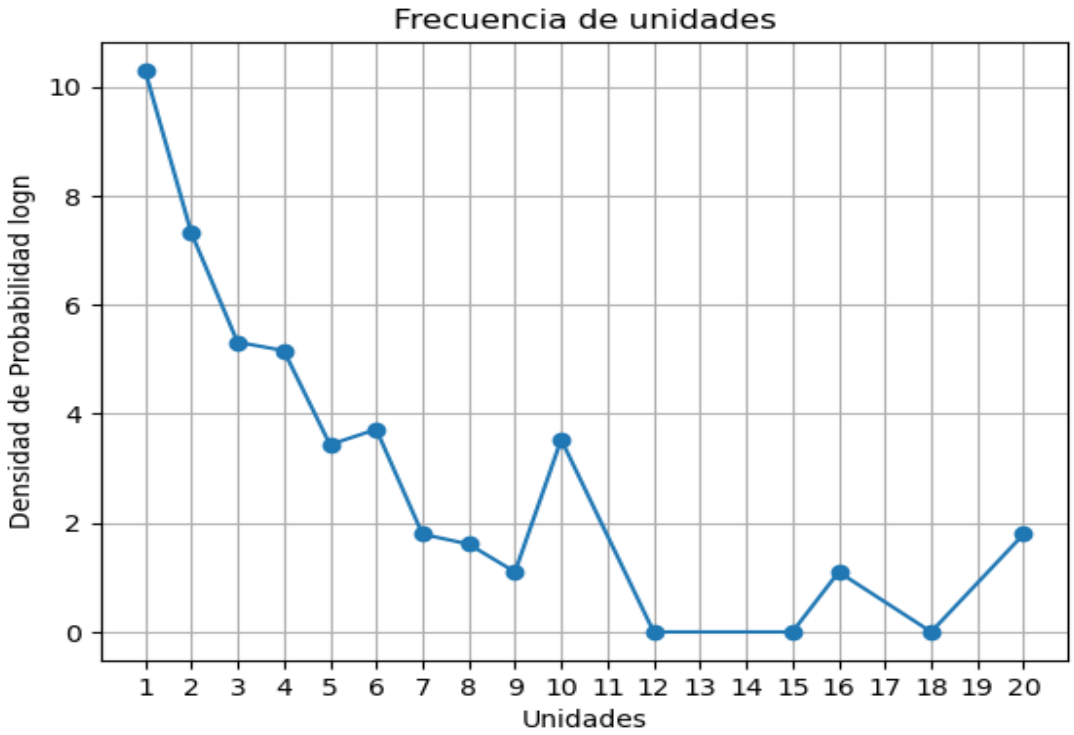

Figura 4.2: Frecuencia con logaritmo natural de las unidades compradas

Al representar gráficamente las frecuencias de los productos solicitados, se generó la figura 4.3, cuya observación revela que los datos exhiben una ausencia de conformidad con cualquier distribución predefinida. Esta representación gráfica ofrece una perspicacia valiosa acerca de la naturaleza de los datos, indicando la necesidad de utilizar las frecuencias relativas para realizar la simulación.

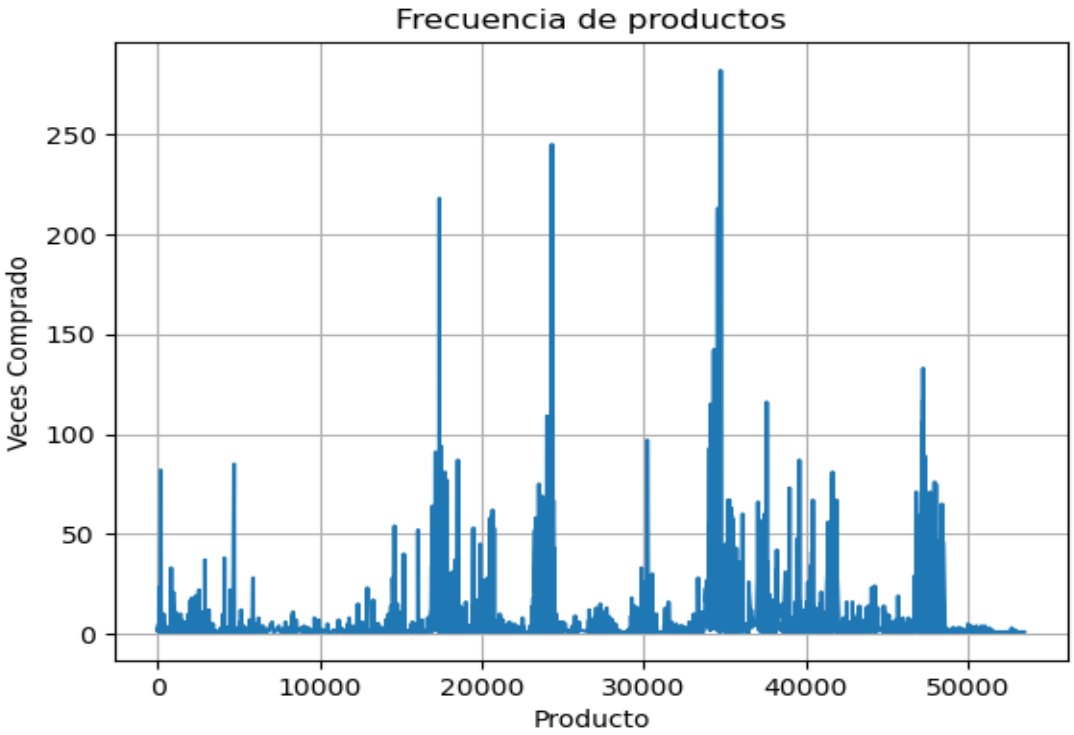

Figura 4.3: Frecuencia de los productos comprados

4.2. Simulación

Para abordar la problemática planteada realizamos una simulación Montecarlo para simular la cantidad de unidades que pedirá cada cliente y el tipo de producto.

Para la simulación de las unidades se calculó la frecuencia relativa de las veces que se pidió cada cantidad. Con esto y la frecuencia acumulada, se estableció un límite inferior y uno superior para posteriormente realizar 10000 experimentos para la simulación.

Al realizar cada experimento se generó un número aleatorio entre el 0 y el 1. Luego se buscó la fila en donde el número aleatorio estuviera entre el límite inferior y superior. Después se agregó la cantidad de unidades correspondiente a esa fila a una lista para guardar los resultados. Finalmente, después de todos los experimentos se obtuvo la siguiente tabla con la simulación de las unidades pedidas.

Unidad	Veces
1	9327
2	490
3	88
4	53
6	15
5	12
7	3
8	1
10	9
16	1
20	1

Cuadro 4.1: Simulación de la cantidad de unidades pedidas

Para la simulación de los productos pedidos se siguió la misma metodología y se obtuvieron los siguientes resultados.

Producto	Cantidad
24335	94
34742	85
34568	66
17387	65
34369	59
47259	49
37584	38
17481	37
4723	37
34324	36
34068	36
34112	36
34483	35
24333	35
37583	34
...	...

Cuadro 4.2: Simulación de los quince productos más vendidos

4.3. VRP

En esta sección del informe, nos adentramos en la implementación y metodología utilizada para abordar el Problema de Ruta de Vehículos (VRP). Habiendo establecido previamente una comprensión profunda del VRP y su formulación matemática en la sección de antecedentes, esta parte del informe se centra en la aplicación práctica de dichos conceptos. Aquí, describimos en detalle cómo hemos adaptado y desarrollado soluciones para resolver diferentes variantes del VRP, destacando los enfoques específicos que empleamos para optimizar la asignación de vehículos a rutas y la distribución eficiente de recursos.

4.4. Pseudocódigo

1. Importación de librerías: Lo primero que se hace en el programa fue importar algunas herramientas de la librería 'simpleai' en python, en la cual se incluyen diferentes algoritmos de búsqueda y algunos visualizadores de los algoritmos.

2. Se creó una clase denominada Tour, la cual se deriva de SearchProblem y representa el problema de la entrega de productos a diferentes lugares.

3. Se creó el constructor de la clase el cual inicializa el estado inicial del nodo de origen (bodega) y otras variables como la capacidad de volumen, pedidos entregados y tiempo restante.

4. Se creó un método en la clase denominado actions, en el cual se definen las acciones posibles a partir del estado actual considerando las restricciones de volumen y tiempo y retorna una lista de posibles destinos.

5. Se creó un método en la clase, result, el cual calcula el nuevo estado después de ejecutar una acción y actualiza el volumen y el tiempo.

6. Se creó un método en la clase, is_goal, el cual verifica si el estado actual cumple con las condiciones para ser considerado como estado objetivo.

7. El método cost, calcula el costo de moverse de un estado a otro, basado en la distancia entre las ciudades.

8. El método heuristic, proporciona una estimación de la distancia desde el estado actual hasta el objetivo, tomando en cuenta cuanta distancia en línea recta desde la base le falta para acabar de entregar todos los paquetes.

9. Se tiene una función de visualización, display, la cual muestra los resultados obtenidos después de resolver el problema y recibe el resultado de la búsqueda como

entrada. Muestra cada acción realizada y el estado resultante en forma de lista.

10. Se definen los datos de las demandas iniciales, las demandas objetivo y las entregas realizadas para diferentes ciudades.

11. Se resuelve el problema utilizando un algoritmo Greedy. Se instancia la clase Tour con la ciudad de origen y se pasa al algoritmo de búsqueda 'greedy'. Se utiliza el parámetro 'graph_search=True' para activar la búsqueda de grafo.

4.5. Supuestos

Para reducir la complejidad del programa y hacer que el espacio de búsqueda del programa sea menor, se tuvo establecer una serie de reglas o asunciones a la hora de resolver el problema.

1 - Camiones: Todos los camiones son el mismo modelo y tienen el mismo volumen, 7.6 m^3 (Chassis Nissan).

2 - Cada camión se tarda 1 hora en cargar +1 hora por cada camión delante de él.

3 - Un trabajador solo puede hacer 1 ruta al día.

4 - No se está considerando un tiempo de entrega; es decir en cuanto el camión pasa por la ubicación del cliente lo entrega inmediatamente y se va de ahí por el siguiente.

El programa define un problema de entregas y rutas (VRP) creando una clase llamadada 'Tour' y luego lo resuelve utilizando un algoritmo greedy, haciendo uso de la librería 'simpleai'. Las acciones posibles dependen de las restricciones de volumen y tiempo, y se busca alcanzar el estado objetivo en donde todas las demandas estén satisfechas. Los detalles de los nodos, así como las demandas y restricciones específicas se utilizan para calcular las acciones y los estados en el proceso de búsqueda.

Capítulo 5

Resultados

En la sección dedicada a los resultados, se presentan los hallazgos derivados de la resolución del Problema de Ruta de Vehículos (VRP) en distintas configuraciones de nodos. Las instancias del VRP examinadas abarcan conjuntos de 5, 20, 50, 100 y 150 nodos, permitiendo así un análisis exhaustivo de cómo varía la complejidad y la eficiencia de las rutas en función del tamaño. Para cada una de estas configuraciones, se exponen con detalle las rutas óptimas que se han identificado, poniendo en relieve tanto las distancias recorridas como los esquemas de asignación de vehículos. Además, se presentan métricas cuantitativas como los tiempos de ejecución de los algoritmos empleados y se realiza una comparativa de las soluciones obtenidas en términos de optimización de recursos y tiempo. Tomando en cuenta que todos los camiones se tratan del mismo modelo CHASIS Cabina con volumen 7.363342728

5.1. VRP con 5 nodos

El número de R representa la cantidad de rutas necesarias para cumplir con la demanda ['R1', 'C1', 'Dia: 1', 'Sobrante m3: 7.3287288', 'Sobrante hr: 6.5492572253874926', [0, 1, 4, 2, 3, 5]] Distancia recorrida: 65.0 km Tiempo de ejecución: 0.0s

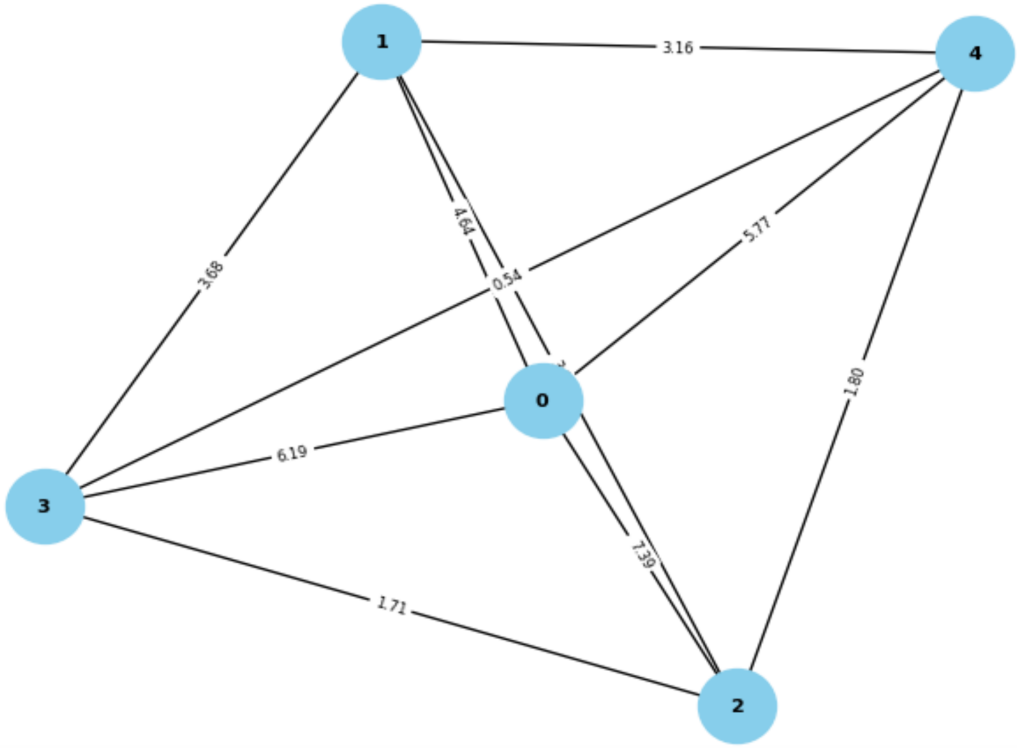

Figura 5.1: Representación del grafo con cinco nodos

5.2. VRP con 20 nodos

El número de R representa la cantidad de rutas necesarias para cumplir con la demanda ['R1', 'C1', 'Dia: 1', 'Sobrante m3: 5.14437866', 'Sobrante hr: 5.268020404038376', [0, 5, 6, 4, 3, 16, 18, 20, 7, 1, 8, 2, 11, 13, 14, 19, 12, 15, 17, 10, 9]] Distancia recorrida: 229.0 km Tiempo de ejecución: 0.1s

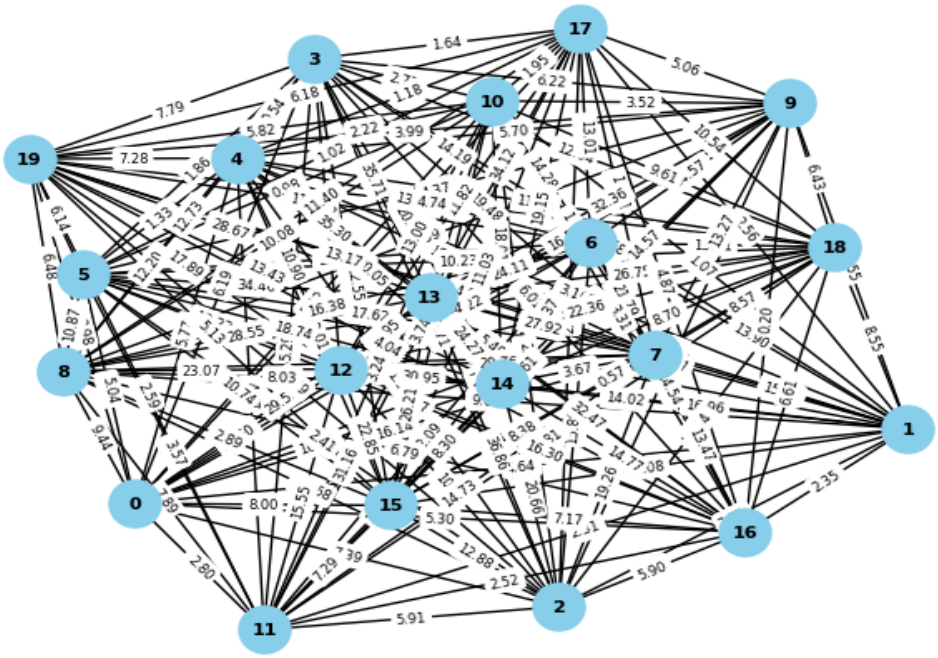

Figura 5.2: Representación del grafo con veinte nodos

5.3. VRP con 50 nodos

El número de R representa la cantidad de rutas necesarias para cumplir con la demanda ['R1', 'C1', 'Dia: 1', 'Sobrante m3: 2.0464583699999994', 'Sobrante hr: 3.0341071009164127', [0, 34, 40, 30, 31, 29, 21, 7, 14, 44, 4, 26, 2, 11, 25, 27, 36, 41, 20, 42, 37, 1, 9, 3, 12, 24, 32, 35, 38, 13, 17, 22, 43, 45, 46, 47, 39, 49, 8, 28, 15, 16, 23, 33, 5, 48, 6, 50, 10, 19, 18]] Distancia recorrida: 344.0 km Tiempo de ejecución: 8.1s

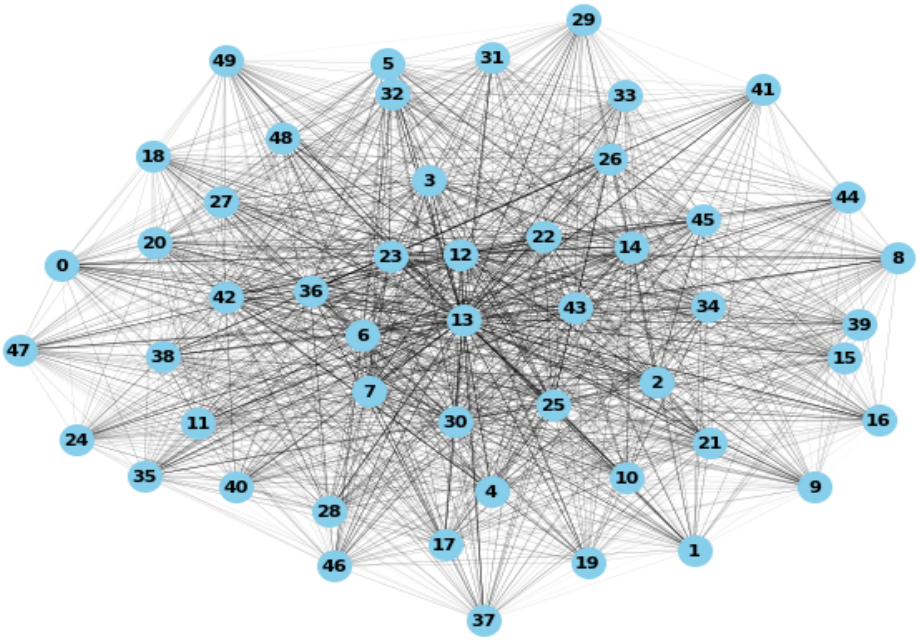

Figura 5.3: Representación del grafo con cincuenta nodos

5.4. VRP con 100 nodos

El número de R representa la cantidad de rutas necesarias para cumplir con la demanda ['R1', 'C1', 'Dia: 1', 'Sobrante m3: 5.934472540000001', 'Sobrante hr: 0.05943491314005305', [0, 47, 35, 72, 100, 2, 95, 71, 83, 52, 19], 'R2', 'C1', 'Dia: 2', 'Sobrante m3: 0.07365849000000074', 'Sobrante hr: 3.227139429499074', [0, 9, 82, 70, 4, 14, 42, 64, 51, 1, 59, 6, 81, 20, 30, 74, 75, 96, 94, 13, 53, 3, 55, 61, 62, 78, 79, 21, 41, 66, 39, 32, 15, 36, 46, 23, 76, 77, 80, 84, 89, 90, 91, 92, 97, 98, 99, 5, 7, 10, 11, 12, 16, 17, 22, 24, 25, 33, 29, 44, 49, 54, 56, 63, 67, 68, 31, 34, 73, 40, 8, 87, 50, 60, 65, 58, 85, 27, 43, 26, 48, 18, 28, 86, 69, 57, 88, 93], 'R3', 'C1', 'Dia: 3', 'Sobrante m3: 3.8112888600000003', 'Sobrante hr: 6.718452848005869', [0, 37, 38, 45]] Distancia recorrida: 1145.0 km Tiempo de ejecución: 40.2s

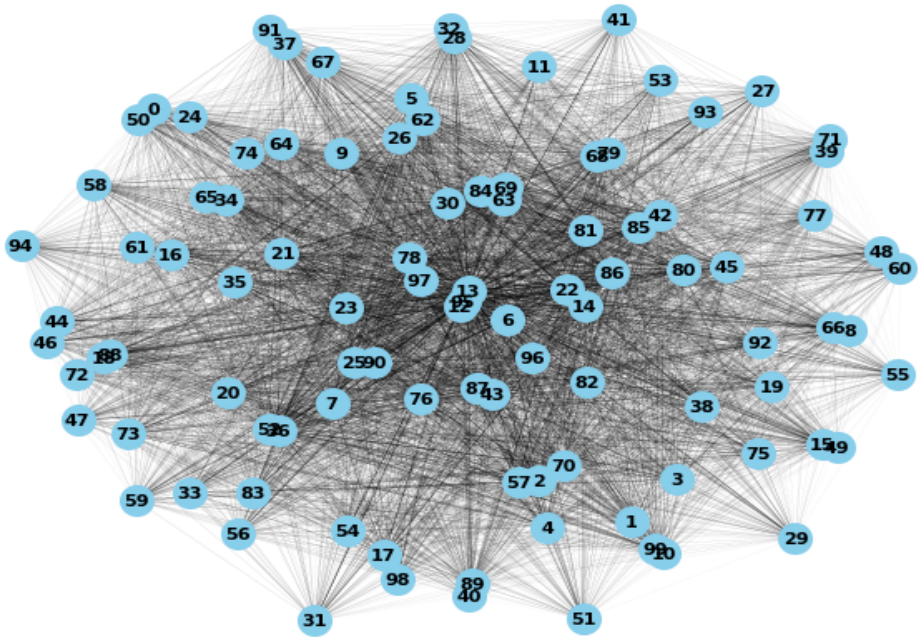

Figura 5.4: Representación del grafo con cien nodos

5.5. VRP con 150 nodos

El número de R representa la cantidad de rutas necesarias para cumplir con la demanda ['R1', 'C1', 'Dia: 1', 'Sobrante m3: 5.934472540000001', 'Sobrante hr: 0.05943491314005305', [0, 47, 35, 72, 100, 2, 95, 71, 83, 52, 19], 'R2', 'C1', 'Dia: 2', 'Sobrante m3: 0.07365849000000074', 'Sobrante hr: 3.227139429499074', [0, 9, 82, 70, 4, 14, 42, 64, 51, 1, 59, 6, 81, 20, 30, 74, 75, 96, 94, 13, 53, 3, 55, 61, 62, 78, 79, 21, 41, 66, 39, 32, 15, 36, 46, 23, 76, 77, 80, 84, 89, 90, 91, 92, 97, 98, 99, 5, 7, 10, 11, 12, 16, 17, 22, 24, 25, 33, 29, 44, 49, 54, 56, 63, 67, 68, 31, 34, 73, 40, 8, 87, 50, 60, 65, 58, 85, 27, 43, 26, 48, 18, 28, 86, 69, 57, 88, 93], 'R3', 'C1', 'Dia: 3', 'Sobrante m3: 3.8112888600000003', 'Sobrante hr: 6.718452848005869', [0, 37, 38, 45]] Distancia recorrida: 1412.0 km Tiempo de ejecución: 3m 20.2s

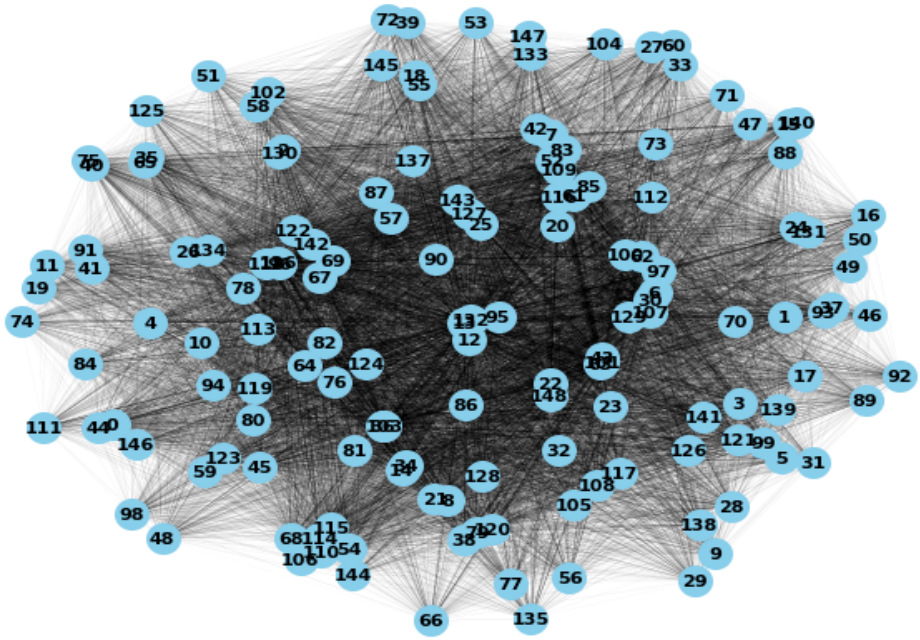

Figura 5.5: Representación del grafo con ciento cincuenta nodos

5.6. Experimentos usando distintas combinaciones de número de camiones y rampas de carga

Como se desea encontrar una combinación cercana a lo óptima de recursos para la entrega de los paquetes en la menor cantidad de tiempo, se realizarán distintas pruebas con el algoritmo de VRP para encontrar cuál cantidad de camiones y rampas de carga sería la ideal tomando en cuenta el contexto del problema. que solamente permite 8 horas de trabajo diarias, requiere de 1 hora para cargar los camiones con los paquetes, establece que los trabajadores solo pueden hacer una ruta al día, porque no es factible regresar a la base y volver a cargar el camión y que no se toma en cuenta que el camión requiere cargar de combustible y el tiempo que en teoría se debería tardar entregando cada paquete, es decir se toma como 0 s.

Características de la PC

- Modelo de la PC
  - Matebook D 15
- Sistema Operativo

    - Windows 10 Home Single Language versión 22H2
- Capacidad de disco duro
    - 110
- Memoria Ram y capacidad
    - 16 GB
- Tipo de procesador
    - Intel core i5
- Número de núcleos
    - CPU de 8 núcleos
- Software utilizado y versión
    - Python · 3.11.3

Tamaño del problema

- Número de nodos: 100
- Número de variables consideradas: 6

5.6.1. 1 camión y 1 rampa de carga

Después de realizar las simulaciones, se vieron los siguientes resultados:

- Un promedio de distancia de 776.486 km (Función objetivo)
- Un promedio sobrante de volumen (por ruta) de 2.1790 m3
- Un promedio de tiempo sobrante (por ruta) de 3.6683 horas
- Un total de 1 hora perdida por día en la fila de espera
- Un promedio de días para entregar todos los paquetes de 2.486 días
- Un promedio de 2.486 de rutas para completar con los pedidos
- El tiempo de ejecución fue de 21,530 segundos = 5.98 horas

5.6.2. 2 camiones y 1 rampa de carga

Después de realizar las simulaciones, se vieron los siguientes resultados:

- Un promedio de distancia de 801.23 km (Función objetivo)
- Un promedio sobrante de volumen (por ruta) de 2.09 m3
- Un promedio de tiempo sobrante (por ruta) de 3.5184 horas
- Un total de 3 horas perdida por día en la fila de espera
- Un promedio de días para entregar todos los paquetes de 1.4 días
- Un promedio de 2.44 de rutas para completar con los pedidos
- El tiempo de ejecución fue de 20,553 segundos = 5.7 horas

5.6.3. 2 camiones y 2 rampas de carga

Después de realizar las 500 simulaciones, los resultados fueron:

- Un promedio de distancia de 776.092 km (Función objetivo)
- Un promedio sobrante de volumen (por ruta) de 2.0965 m3
- Un promedio de tiempo sobrante (por ruta) de 3.7087 horas
- Un total de 2 horas perdidas por día en la fila de espera
- Un promedio de días para entregar todos los paquetes de 1.492 días
- Un promedio de 2.552 de rutas para completar con los pedidos
- El tiempo de ejecución fue de 15,760 segundos = 4.3 horas

5.6.4. 3 camiones y 1 rampa

Después de realizar las 500 simulaciones, los resultados fueron:

- Un promedio de distancia de 782.14 km (Función objetivo)
- Un promedio sobrante de volumen (por ruta) de 2.116 m3
- Un promedio de tiempo sobrante (por ruta) de 3.232 horas
- Un total de 6 horas perdidas por día en la fila de espera
- Un promedio de días para entregar todos los paquetes de 1.07 días
- Un promedio de 2.532 de rutas para completar con los pedidos
- El tiempo de ejecución fue de 18,364 segundos = 5.1 horas

5.6.5. 3 camiones y 3 rampas

Después de realizar las 500 simulaciones, los resultados fueron:

- Un promedio de distancia de 809.24 km (Función objetivo)
- Un promedio sobrante de volumen (por ruta) de 2.0965 m3
- Un promedio de tiempo sobrante (por ruta) de 3.725 horas
- Un total de 3 horas perdidas por día en la fila de espera
- Un promedio de días para entregar todos los paquetes de 1.1 días
- Un promedio de 2.64 de rutas para completar con los pedidos
- El tiempo de ejecución fue de 19,753 segundos = 5.4 horas

5.6.6. 4 camiones y 1 rampa

Después de realizar las 500 simulaciones, los resultados fueron:

- Un promedio de distancia de 797.2 km (Función objetivo)
- Un promedio sobrante de volumen (por ruta) de 2.2563 m3
- Un promedio de tiempo sobrante (por ruta) de 3.2377 horas
- Un total de 10 horas perdidas por día en la fila de espera
- Un promedio de días para entregar todos los paquetes de 1.04 días
- Un promedio de 2.62 de rutas para completar con los pedidos
- El tiempo de ejecución fue de 17,829 segundos = 4.9 horas

5.6.7. 4 camiones y 2 rampas

Después de realizar las 500 simulaciones, los resultados fueron:

- Un promedio de distancia de 747.34 km (Función objetivo)
- Un promedio sobrante de volumen (por ruta) de 2.067 m3
- Un promedio de tiempo sobrante (por ruta) de 3.63 horas
- Un total de 6 horas perdidas por día en la fila de espera
- Un promedio de días para entregar todos los paquetes de 1.0 día

- Un promedio de 2.46 de rutas para completar con los pedidos
- El tiempo de ejecución fue de 13,134 segundos = 3.6 horas

5.6.8. Tabla de resultados

Camiones	**Rampas**	**Distancia (km)**	**Volumen sobrante (m3)**	**Horas sobrante**
1	1	776.486	2.179	3.6683
2	1	801.23	2.09	3.5184
2	2	776.092	2.0965	3.7087
3	1	782.14	2.116	3.232
3	3	809.24	2.0965	3.725
4	1	797.2	2.2563	3.237
4	2	747.34	2.067	3.63

Camiones	**Rampas**	**Horas en fila**	**Días entrega**	**Rutas**	**Horas de ejecución**
1	1	1	2.486	2.486	5.98
2	1	3	1.4	2.44	5.7
2	2	2	1.492	2.552	4.3
3	1	6	1.07	2.532	5.1
3	3	3	1.1	2.64	5.4
4	1	10	1.04	2.62	4.9
4	2	6	1	2.46	3.6

Cuadro 5.1: Tabla de resultados de las 500 simulaciones

Capítulo 6

Conclusiones

6.1. Recomendaciones

Con base en los resultados obtenidos, la mejor combinación de instalación para una empresa repartidora depende de las necesidades de la empresa. Si la empresa quiere asegurar la entrega de los paquetes en un mismo día, la mejor combinación es la de 3 camiones y 1 rampa; ya que esta combinación es la que tiene menos camiones y rampas, pero que a la vez asegura que se entregue todo el 1 día. Sin embargo, el tiempo que se pierde en la fila de espera es grande. Si se deseara sacrificar días de entrega por menos camiones; por ende, menor costo, se podría utilizar la combinación de 2 camiones y 1 rampa. Además, esta combinación reduce el volumen sobrante y el tiempo perdido en la fila de espera.

Estas configuraciones significan una optimización de los recursos de la empresa y una disminución de los costos operativo, son altamente adaptables y pueden ser ajustadas según las necesidades cambiantes de la empresa, lo que asegura su capacidad para enfrentar futuros desafíos logísticos y mantener una entrega eficiente y confiable en un mercado en constante evolución.

6.2. Futuras Mejoras

- **Mejorar la heurística:** La optimización de la heurística es una mejora esencial para el proyecto. Actualmente, se utiliza el algoritmo Greedy para encontrar las rutas más eficientes. Sin embargo, una heurística más precisa y adaptable puede llevar a una optimización aún mayor. Esto implica desarrollar una función heurística que evalúe con mayor precisión el costo esperado de llegar a un destino y, al mismo tiempo, sea capaz de adaptarse a las condiciones cambiantes en

tiempo real, como el tráfico y las condiciones climáticas. La mejora de la heurística garantizará que se pueda utilizar el algoritmo A* para tomar decisiones más informadas y, en consecuencia, mejorar la eficiencia en la planificación de rutas.

- **Mejorar el ejecutable:** La mejora del ejecutable del software es clave para su funcionalidad y eficiencia. Esto incluye la optimización del código para garantizar que el programa se ejecute de manera rápida y eficiente, lo que reducirá los tiempos de procesamiento y aumentará la capacidad de respuesta. Además, se podría desarrollar una interfaz más amigable o una aplicación para poder correr las simulaciones y utilizar el algoritmo desarrollado
- **Colaboración con externos:** La colaboración con otras empresas, como proveedores de datos de tráfico, empresas de tecnología de mapas y organizaciones de sostenibilidad, puede enriquecer la simulación con datos adicionales y recursos que mejorarían la toma de decisiones.
- **Agregar más restricciones:** Para hacer simulaciones más reales se podrían considerar más restricciones y parámetros como el tiempo que se tarda un repartidor en entregar un paquete, las ventanas de tiempo y los horarios específicos de entrega.

Capítulo 7

Anexo

7.1. Enlace de descarga

https://drive.google.com/file/d/1U-83_BRcrTfCpxRuQmhk2BKTy1PTqD0d/view?usp=sharing

7.2. Código Montecarlo

```
1
2  import numpy as np
3  import math
4  import random
5  import pandas as pd
6  import seaborn as sns
7  import matplotlib.pyplot as plt
8  from scipy import stats
9  from collections import Counter
10 from geopy.geocoders import Nominatim
11
12
13 info_productos = pd.read_csv("Datos_reto/info_productos.csv")
14 info_compra = pd.read_excel("Datos_reto/informacion_compra.xlsx")
15 unidades_transporte = pd.read_excel("Datos_reto/Unidades_transporte.xlsx")
16 info_ubicaciones = pd.read_excel("Datos_reto/direcciones.xlsx")
17
18 # Unir la informacin de la compra con la informacin del producto
19 info_compras_completas = info_compra.merge(info_productos, how="left", validate=" ↩
       many_to_many")
20
21 dic = {key:value for key, value in zip(info_compras_completas.columns, ["Producto", ↩
       "Unidades", "Cliente", "Volumen"])}
```

```
22 info_compras_completas.rename(mapper=dic, axis=1, inplace=True)
23
24 info_compras_unidades = info_compras_completas.groupby("Unidades")[["Unidades"]]. ↩
       count()
25
26 info_compras_unidades.rename(mapper={"Unidades":"Veces"}, axis=1, inplace=True)
27 info_compras_unidades.reset_index(inplace=True)
28
29 plt.plot(info_compras_unidades["Unidades"], info_compras_unidades["Veces"], marker=' ↩
       o')
30 plt.xlabel('Unidades')
31 plt.ylabel('Densidad de Probabilidad')
32 plt.title('Frecuencia de unidades')
33 plt.grid(True)
34 plt.show()
35
36 plt.plot(info_compras_unidades["Unidades"], np.log(info_compras_unidades["Veces"]), ↩
       marker='o')
37 plt.xlabel('Unidades')
38 plt.ylabel('Densidad de Probabilidad logn')
39 plt.title('Frecuencia de unidades')
40 plt.xticks([x for x in range(1, 21)])
41 plt.grid(True)
42 plt.show()
43
44 #  Aqu  podemos ver que no sigue una distribucin exponencial
45
46 info_veces_productos = info_compras_completas.groupby("Producto").agg({"Producto":" ↩
       count", "Volumen":"mean"})
47 info_veces_productos.rename(mapper={"Producto":"Veces"}, axis=1, inplace=True)
48
49 info_veces_productos.reset_index(inplace=True)
50
51 plt.plot(info_veces_productos["Producto"], info_veces_productos["Veces"])
52 plt.xlabel('Producto')
53 plt.ylabel('Veces Comprado')
54 plt.title('Frecuencia de productos')
55 plt.grid(True)
56 plt.show()
57
58 for i, j in enumerate(info_ubicaciones["Calle"]):
59     j = str(j)
60     if "AVE " in j.upper():
61         info_ubicaciones.loc[i, "Calle"] = "AV "+j[4:]
62     if "LIC " in j.upper():
63         info_ubicaciones.loc[i, "Calle"] = j[4:]
64
```

```
info_ubicaciones

def concat_dir(df):
    concat = []
    for i in df.values:
        s = str(i[3])+", "+str(i[2])+", Monterrey, Nuevo Len, "+ str(i[1])+ ", Mxico"
        concat.append(s)
    return concat


info_ubicaciones["direccion"] = concat_dir(info_ubicaciones)
info_ubicaciones

def probar_dif_versiones(texto, geolocator):
    # Sin  cdigo  postal
    indice = len(texto)
    for i in range(2):
        indice = texto[:indice-1].rfind(",")
    location = geolocator.geocode(texto[:indice])
    if location!=None:
        return location.latitude, location.longitude

    # Sin calle
    indice1 = texto.find(",")
    indice2 = texto[indice1+1:].find(",")+indice1+1

    t = texto[:indice1]+texto[indice2:]
    location = geolocator.geocode(t)
    if location!=None:
        return location.latitude, location.longitude

    return None, None


def get_lat_long(lista):
    lat = []
    lon = []
    geolocator = Nominatim(user_agent="my_geocoder", timeout=1000) # Replace with  ↩
        your user agent
    for i in lista:
        location = geolocator.geocode(i)
        if location:
            a = location.latitude
            b = location.longitude
        else:
            a, b = probar_dif_versiones(i, geolocator)
        lat.append(a)
```

```
        lon.append(b)
    return lat, lon

lat, lon = get_lat_long(info_ubicaciones["direccion"])
info_ubicaciones["latitud"] = lat
info_ubicaciones["longitud"] = lon


def haversine_distance(ubi1, ubi2):
    lat1 = info_ubicaciones["latitud"][ubi1]
    lon1 = info_ubicaciones["longitud"][ubi1]
    lat2 = info_ubicaciones["latitud"][ubi2]
    lon2 = info_ubicaciones["longitud"][ubi2]

    # Radio de la Tierra en kilmetros
    earth_radius = 6371.0

    # Convertir las latitudes y longitudes a radianes
    lat1_rad = math.radians(lat1)
    lon1_rad = math.radians(lon1)
    lat2_rad = math.radians(lat2)
    lon2_rad = math.radians(lon2)

    # Diferencias de latitud y longitud
    delta_lat = lat2_rad - lat1_rad
    delta_lon = lon2_rad - lon1_rad

    # Calcular el haversine
    a = math.sin(delta_lat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math. ↩
        sin(delta_lon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    distance = earth_radius * c

    # resultado en km
    return distance


matriz = np.zeros((info_ubicaciones.shape[0], info_ubicaciones.shape[0]))

for fila, i in enumerate(matriz):
    for columna in range(len(i)):
        matriz[fila][columna] = haversine_distance(fila, columna)

matriz_distancias = pd.DataFrame(matriz)

repeticiones = 10000

```

```
157 def simulacion(df, repeticiones):
158     unidades = []
159     for i in range(repeticiones):
160         rand = random.random()
161         for j in df.index:
162             if (rand>=df.iloc[j]["Lmite inferior"]) and (rand<df.iloc[j]["Lmite
                    superior"]):
163                 unidades.append(df.iloc[j]["Unidades"])
164                 break
165     return unidades
166 
167 info_compras_unidades["Frecuencia Relativa"] = info_compras_unidades["Veces"]/
        info_compras_unidades["Veces"].sum()
168 info_compras_unidades["FR Acumulada"] = info_compras_unidades["Frecuencia Relativa"
        ].cumsum()
169 info_compras_unidades["Lmite inferior"] = info_compras_unidades["FR Acumulada"] -
        info_compras_unidades["Frecuencia Relativa"]
170 info_compras_unidades["Lmite superior"] = info_compras_unidades["FR Acumulada"]
171 
172 info_compras_unidades
173 unidades_fr = simulacion(info_compras_unidades, repeticiones)
174 
175 print("Promedio simulacin frecuencia relativa:", sum(unidades_fr)/repeticiones)
176 print("Promedio real:", np.mean(lista_unidades))
177 
178 simulacion_unidades_df = pd.DataFrame(unidades_fr)
179 simulacion_unidades_df.head()
180 
181 
182 info_veces_productos["Frecuencia Relativa"] = info_veces_productos["Veces"]/
        info_veces_productos["Veces"].sum()
183 info_veces_productos["FR Acumulada"] = info_veces_productos["Frecuencia Relativa"].
        cumsum()
184 info_veces_productos["Lmite inferior"] = info_veces_productos["FR Acumulada"] -
        info_veces_productos["Frecuencia Relativa"]
185 info_veces_productos["Lmite superior"] = info_veces_productos["FR Acumulada"]
186 
187 
188 lista_productos = []
189 for i in info_veces_productos.index:
190     for j in range(int(info_veces_productos.iloc[i]["Veces"])):
191         lista_productos.append(info_veces_productos.iloc[i]["Producto"])
192 
193 def simulacion_producto(df, repeticiones):
194     productos = []
195     for i in range(repeticiones):
196         rand = random.random()
```

```
197         for j in df.index:
198             if (rand>=df.iloc[j]["Lmite inferior"]) and (rand<df.iloc[j]["Lmite
                    superior"]):
199                 productos.append(df.iloc[j]["Producto"])
200                 break
201     return productos
202 
203 productos = simulacion_producto(info_veces_productos, repeticiones)
204 print("Promedio simulacin frecuencia relativa:", sum(productos)/repeticiones)
205 print("Promedio real:", np.mean(lista_productos))
206 simulacion_productos_df = pd.DataFrame(productos)
207 
208 simulacion_productos_df.to_excel("simulacion_productos.xlsx", index=False)
209 
210 
211 tabla_simulacion = pd.DataFrame()
212 tabla_simulacion["Cliente"] = [i for i in range(len(simulacion_productos_df))]
213 tabla_simulacion["Unidades"] = simulacion_unidades_df
214 tabla_simulacion["Producto"] = simulacion_productos_df.copy()
215 
216 info_productos.set_index("Producto", inplace=True)
217 
218 tabla_simulacion = pd.read_excel("Datos_reto/simulacion_final_completa.xlsx")
219 
220 tabla_simulacion["Volumen"] = tabla_simulacion["Unidades"]*(info_productos.loc[
        tabla_simulacion["Producto"], " Volumen mtro3"].values)
221 
222 
223 tabla_simulacion.to_excel("simulacion_final_completa.xlsx", index=False)
```

7.3. Código VRP

```
1 
2 import pandas as pd
3 import numpy as np
4 
5 # Parmetros para realizar las pruebas
6 nClientes= 100
7 nTiempo= 8
8 Camiones=1
9 nRampas=1
10 VCarro=7.36
11 
12 numSimulaciones =2
13 
14 
15 def simulacion():
```

```
16  #Cargar la matriz de distancias en lnea recta
17  df= pd.read_excel('distancias_extras.xlsx',index_col='Unnamed: 0')
18  df = df.astype(float)
19
20  #Guardar fila 0
21  fila0=pd.DataFrame(df.loc[0])
22  fila0=fila0.T
23
24  #Borrar fila 0
25  df = df.drop(index=0)
26
27  #Escoger una muestra de 100 nodos aleatorios
28
29  # Concatenar la fila de 0 siempre y escoger 100 clientes aleatorios
30  dF=df.sample(n=nClientes)
31  dF = pd.concat([fila0, dF], ignore_index=False)
32
33  # Dejar solamente las columnas correspondientes a las filas
34  nodos=list(dF.index)
35  nodoss=[(x) for x in nodos]
36  dF=dF[nodoss]
37
38  #Cambiar el nombre de las columnas y filasde 0 a 100
39  dF=dF.reset_index(drop=True)
40  dF.columns
41  new_column_names = {old_name: i for i, old_name in enumerate(dF.columns)}
42  dF.rename(columns=new_column_names, inplace=True)
43
44  dict_from_df = dF.to_dict(orient='index')
45
46  #Evitar que el nodo viaje a s mismo
47  for i in dict_from_df.keys():
48      dict_from_df[i].pop(i)
49
50  #Cargar matriz de distancias en auto
51  dr= pd.read_excel('rutas_auto.xlsx',index_col='Unnamed: 0')
52  dr=dr.reset_index(drop=True)
53  new_column_names = {old_name: i for i, old_name in enumerate(dr.columns)}
54  dr.rename(columns=new_column_names, inplace=True)
55
56  dR=dr.loc[nodos]
57  # Dejar solamente las columnas correspondientes a las filas
58  nodoS=list(dR.index)
59  dR=dR[nodoS]
60  #Cambiar el nombre de las columnas y filas de 0 a 100
61  dR=dR.reset_index(drop=True)
62  dR.columns
```

```
63    new_column_names = {old_name: i for i, old_name in enumerate(dR.columns)}
64    dR.rename(columns=new_column_names, inplace=True)
65
66    dict_from_dr = dR.to_dict(orient='index')
67
68    #Evitar que el nodo viaje a s mismo
69    for i in dict_from_dr.keys():
70        dict_from_dr[i].pop(i)
71    dict_from_dr
72
73    #Cargar matriz de tiempo en carro
74    dt= pd.read_excel('matriz_tiempo.xlsx',index_col='Unnamed: 0')
75    dt=dt.reset_index(drop=True)
76    new_column_names = {old_name: i for i, old_name in enumerate(dt.columns)}
77    dt.rename(columns=new_column_names, inplace=True)
78
79    dT=dt.loc[nodos]
80    # Dejar solamente las columnas correspondientes a las filas
81    nodoS=list(dT.index)
82    dT=dT[nodoS]
83    #Cambiar el nombre de las columnas y filas de 0 a 100
84    dT=dT.reset_index(drop=True)
85    dT.columns
86    new_column_names = {old_name: i for i, old_name in enumerate(dT.columns)}
87    dT.rename(columns=new_column_names, inplace=True)
88
89    dict_from_dt = dT.to_dict(orient='index')
90
91    #Evitar que el nodo viaje a s mismo
92    for i in dict_from_dt.keys():
93        dict_from_dt[i].pop(i)
94    dict_from_dt
95
96
97    #Usar tabla con la descripcin de los productos para calcular el volumen de la ↩
          entrega
98    dP= pd.read_csv('info_productos.csv',index_col='Producto')
99    dictProductos = dP.to_dict(orient='index')
100   #Cargar tabla con 10,000 simulaciones de montecarlo con la distribucin de los ↩
          pedidos de productos
101   compras=pd.read_excel("simulacion_final_completa.xlsx",index_col='Cliente')
102
103
104
105   # Obtener 100 muestras random
106   CC = compras.sample(n=nClientes+1)
107   CC = CC.reset_index(drop=True)
```

```
108 dCompra=CC.to_dict(orient='index')
109 dCompra[0]=VCarro
110
111 #Copiar dCompra para tener el estado inicial
112 import copy
113 E=copy.deepcopy(dCompra)
114 clients=list(E.keys())
115 clients.remove(0)
116 for i in clients:
117     E[i]['Unidades']=0
118
119 # Funcin  para determinar el volumen de la compra
120 def calcularVolumen(cliente):
121         totalV=0
122         #Si se regresa a la bodega vaciar el camin y dejarlo disponible para ↩
              cargar todo el volumen disponible
123         if cliente == 0:
124                 return -VCarro
125                 exit()
126
127
128         cant=dCompra[cliente]['Unidades']
129         totalV+=dictProductos[dCompra[cliente]['Producto']][' Volumen mtro3']* ↩
              cant
130
131         return totalV
132
133 def tRes(a,state,res):
134     if a == 0:
135         if res ==0:
136             return -(nTiempo-1)
137             exit()
138     return dict_from_dt[state[-1][0]][a]
139
140 #-------------------------------------------------------------------------------------------------
141 #    Esqueleto de PSA para el problema VRP
142 #-------------------------------------------------------------------------------------------------
143
144 from simpleai.search import SearchProblem, depth_first, breadth_first, ↩
      uniform_cost, greedy, astar
145 from simpleai.search.viewers import BaseViewer, ConsoleViewer, WebViewer
146
147 import time
148 import copy
149
```

```
150 #-------------------------------------------------------------------------------------------------------------
151 #   Definicin  del problema
152 #-------------------------------------------------------------------------------------------------------------
153
154 class Tour(SearchProblem):
155     """
156         Clase que es usada para definir el problema de VRP. Los estados son  ↩
                representados
157         con los nmeros de nodos.
158     """
159
160     def __init__(self, origen):
161         """ Constructor de la clase. Inicializa el problema de acuerdo a ...
162
163             origen: No se ha entregado ningn paquete
164             destino: Ya se entregaron todos los paquetes y se regresa a la base
165         """
166
167         #Restricciones
168         self.capacidad=VCarro
169         self.tiempo=nTiempo-1
170         self.idCamion=1
171         self.dia=1
172         self.rampas=nRampas
173         #Pedido entregado
174         pedidos=0
175         #Tiempo restante en los camiones
176         self.statuscamiones = [nTiempo] * Camiones
177
178         #Asignar n las horas dependiendo de la cantidad de rampas
179         if self.rampas==2:
180             for i in range(0, len(self.statuscamiones), self.rampas):
181                 self.statuscamiones[i] = self.statuscamiones[i + 1] = (self. ↩
                        statuscamiones[i]-(i+1))
182
183         elif self.rampas==3:
184             for i in range(0, len(self.statuscamiones), self.rampas):
185                 self.statuscamiones[i] = self.statuscamiones[i + 1] = self. ↩
                        statuscamiones[i + 2] = (self.statuscamiones[i]-(i+1))
186
187         elif self.rampas ==1:
188             for i in range(len(self.statuscamiones)):
189                 self.statuscamiones[i]-=(i+1)
190
191         self.statuscamiones=tuple(self.statuscamiones)
```

```
192
193         origen=[(0,0,self.capacidad,self.tiempo,self.statuscamiones,self.idCamion ↩
                ,self.dia)]
194         origen=tuple(origen)
195         SearchProblem.__init__(self, origen)
196
197
198     def actions(self, state):
199         """
200             Este mtodo regresa una lista con las acciones posibles que pueden ser ↩
                    ejecutadas de
201             acuerdo con el estado especificado.
202
203             state: Numero del nodo actual
204
205         """
206         posible = list(dict_from_df[state[-1][0]].keys())
207         posible = posible[:nClientes]
208         posible = [int(element) for element in posible]
209         p=copy.deepcopy(posible)
210
211         #Evitar pasarse del volumen
212         listt=[]
213         for j in p:
214
215             if((state[-1][2]-calcularVolumen(j))<0):
216                 listt.append(j)
217
218         for i in listt:
219             if i in posible:
220                 posible.remove(i)
221
222
223         #Evitar pasarse del tiempo
224         listt=[]
225         for j in p:
226
227
228             if(state[-1][0]!=0):
229                 if(j!=0):
230                     if((state[-1][3]-dict_from_dt[state[-1][0]][j])<=dict_from_dt[ ↩
                            j][0]):
231
232                         listt.append(j)
233
234         for i in listt:
235             if i in posible:
```

```
                posible.remove(i)

        #Remover los nodos ya visitados
        lista=[]
        for i in state:
            if i[0]!=0:
                lista.append(i[0])

        for i in lista:
            if i in posible:
                posible.remove(i)



        posible = [int(element) for element in posible]









        return posible #Regresar lista con el nombre de nodos posibles



    def result(self, state, action):
        """
            Este mtodo regresa el nuevo estado obtenido despues de ejecutar la accin.

            state: Nodo origen (el actual).
            action: Nodo especificado por action
        """



        #Copiar el estado actual
        s=list(state)
        s=s.copy()
        dia=state[-1][6]
        if(action==0):
            dia+=1
            #print('algoo')
            demand=dCompra[action]
        if(action !=0):
            demand=tuple(dCompra[action].items())

```

```
282             #Calcular el volumen del pedido
283             volumenTrasladado=s[-1][2]-calcularVolumen(action)
284             restriccion=0
285             if volumenTrasladado>=VCarro:
286                 volumenTrasladado=VCarro
287                 restriccion=1
288
289
290
291             #Calcular el tiempo que va a tomar hacer la accin
292             tiempoRestante=s[-1][3]-(tRes(action,state,restriccion))
293             if tiempoRestante>=(nTiempo):
294                 tiempoRestante=(nTiempo)
295
296             #Cambiar el tiempo restante a los camiones
297             tiempos=state[-1][4]
298             tiempos=list(tiempos)
299
300             tiempos[(state[-1][5])-1]=tiempoRestante
301             restriccion=0
302
303             numCamion=state[-1][5]
304             if action ==0:
305                 numCamion+=1
306
307                 if numCamion>Camiones:
308                     numCamion=1
309
310             #Si solamente hay un camin cambiar de da cada que complete una ruta
311             if Camiones==1:
312                     if dia != state[-1][6]:
313                         if self.rampas==1:
314                             for i in range(len(tiempos)):
315                                 tiempos[i]=nTiempo
316                                 tiempos[i]-=(i+1)
317                                 tiempoRestante=nTiempo-1
318                         if self.rampas==2:
319                             for i in range(0, len(tiempos), self.rampas):
320                                 tiempos[i] = tiempos[i + 1] = (tiempos[i]-(i+1))
321
322
323             #Si hay dos camiones o ms cambiar de da hasta que todos hayan completado ↩
                    una ruta
324             if Camiones != state[-1][5]:
325
326                 tiempoRestante=state[-1][4][numCamion-1]
327                 for i in range(len(tiempos)):
```

```
                tiempos[numCamion-1]=tiempoRestante
                tiempos[i]=nTiempo-1
            if numCamion==1:
                dia+=1
                if self.rampas==1:
                    for i in range(len(tiempos)):
                        tiempos[i]=nTiempo
                        tiempos[i]-=(i+1)
                        tiempoRestante=nTiempo-1
                if self.rampas==2:
                    for i in range(0, len(tiempos), self.rampas):
                        tiempos[i] = tiempos[i + 1] = (tiempos[i]-(i+1))


        tiempos=tuple(tiempos)









        #Lista para guardar toda la informacin del resultado
        traslado=list()
        traslado.append(action)
        traslado.append(demand)
        traslado.append(volumenTrasladado)
        traslado.append(tiempoRestante)
        traslado.append(tiempos)
        traslado.append(numCamion)
        traslado.append(dia)

        traslado=tuple(traslado)
        s.append(traslado)

        state=tuple(s)









        return state

    def is_goal(self, state):
        """
            Determinar si el estado es el estado meta.
```

```
375
376             state: Lista con los pedidos entregados.
377         """
378
379
380         entregado=copy.deepcopy(E)
381
382         #Llenar la lista con los pedidos entregados
383         for i in state:
384             if(i[0]!=0):
385                 entregado[i[0]]["Unidades"]+=i[1][0][1]
386
387
388         #Si la lista de pedidos entregados est completa, entonces buscar la forma ↩
                de regresar a la base
389         if   entregado==dCompra:
390             if state[-1][0]==0:
391                 return True
392         else:
393
394             return False
395
396     def cost(self, state, action, state2):
397         """
398             Este mtodo recibe el estado y una accin, y regresa el costo de
399             aplicar la accin al estado
400
401             state: nodo actual
402             action: nodo a donde me muevo
403         """
404         return dict_from_dr[state[-1][0]][action] #Regresar el costo de la accin ↩
                de acuerdo con la distancia en carro real
405
406     def heuristic(self, state):
407         """
408             Este mtodo regresa un estimado de la distancia desde el estado a la ↩
                    meta.
409
410             Lo logra calculando la distancia en lnea recta desde la base hasta ↩
                    cada uno de los nodos que faltan por entregarles su paquete
411         """
412
413         listo=[]
414         for i in state:
415             if i[0]!=0:
416                 listo.append((i[0]))
417         L=list(dict_from_df[0].keys())
```

```
418             L=L[:nClientes]
419
420             for i in listo:
421                 if i in L:
422                     L.remove(i)
423             suma=0
424             for i in L:
425                 suma+=dict_from_df[0][i]
426
427
428             return suma
429
430
431
432
433
434
435
436     # Despliega los resultados
437     def display(result):
438
439         if result is not None:
440             for i, (action, state) in enumerate(result.path()):
441                 if action == None:
442                     print('Configuracin inicial')
443                 elif i == len(result.path()) - 1:
444                     print(i,'- Despus de moverse a', action)
445                     print('Meta lograda con costo =', result.cost,'!')
446                 else:
447                     print(i,'- Despus de moverse a', action)
448
449                 print(' ', state)
450                 Rutas=state
451             return Rutas
452         else:
453             print('Mala configuracin del problema')
454             return 0
455
456     #-------------------------------------------------------------------------------------------------------------
457     #   Programa
458     #-------------------------------------------------------------------------------------------------------------
459
460     #-------------------------------------------------------------------------------------------------------------
461     #   Programa
```

```
462 #-------------------------------------------------------------------------------------------------
463
464
465
466 my_viewer = None
467 my_viewer = BaseViewer()    # Solo estadsticas
468 #my_viewer = ConsoleViewer() # Texto en la consola
469 #my_viewer = WebViewer()     # Abrir en un browser en la liga http://localhost ↩
        :8000
470
471 # Crea un PSA y lo resuelve con la bsqueda greedy
472 result = greedy(Tour('Paris'), graph_search=True, viewer=my_viewer)
473
474
475
476 if my_viewer != None:
477     print('Stats:')
478     print(my_viewer.stats)
479
480 print()
481
482 print('>> Bsqueda Greedy <<')
483 Rutas=display(result)
484
485
486
487 #-------------------------------------------------------------------------------------------------
488 #   Fin del archivo
489 #-------------------------------------------------------------------------------------------------
490
491 #Regresar resultados
492 Trayecto=[]
493 RutaCompleta=[]
494 contadorR=0
495 n=0
496 sumVol=0
497 sumTiempo=0
498 sumdias=0
499 for i in range(len(Rutas)):
500     if Rutas[i][0] == 0 and n>0 :
501         contadorR+=1
502         numRuta=('R'+str(contadorR))
503         numCamion=('C'+str(Rutas[i-1][5]))
504         dia=("Dia: "+str(Rutas[i-1][6]))
```

```
            sobVol=("Sobrante m3: "+str(Rutas[i-1][2]))
            sobTiempo=("Sobrante hr: "+str(Rutas[i-1][3]))
            sumVol+=Rutas[i-1][2]
            sumTiempo+=Rutas[i-1][3]
            sumdias=Rutas[i-1][6]
            RutaCompleta.append(numRuta)
            RutaCompleta.append(numCamion)
            RutaCompleta.append(dia)
            RutaCompleta.append(sobVol)
            RutaCompleta.append(sobTiempo)
            RutaCompleta.append(Trayecto)
            Trayecto=[]


        Trayecto.append(Rutas[i][0])
        n+=1

    promVol=sumVol/contadorR
    promTiempo=sumTiempo/contadorR

    print(RutaCompleta)
    print(result.cost)
    print(promVol)
    print(promTiempo)
    print(sumdias)
    print(contadorR)
    return result.cost,promVol,promTiempo,sumdias,contadorR

sumaC=0
sumaV=0
sumaT=0
sumaD=0
sumaR=0
conteo=0
for i in range(numSimulaciones):
    C,V,T,D,R=simulacion()
    sumaC+=C
    sumaV+=V
    sumaT+=T
    sumaD+=D
    sumaR+=R
    conteo+=1
    print("Sim #",conteo)
promedioC=sumaC/conteo
promedioV=sumaV/conteo
promedioT=sumaT/conteo
promedioD=sumaD/conteo
```

```
552 promedioR=sumaR/conteo
553 print("Promedio Costo: ",promedioC)
554 print("Promedio Volumen sobrante: ",promedioV)
555 print("Promedio Tiempo sobrante: ",promedioT)
556 print("Promedio Dias: ",promedioD)
557 print("Promedio Num Rutas: ",promedioR)
```

Bibliografía

AWS (s.f.). *¿Qué es la simulación de Monte Carlo? - Explicación de la simulación de Monte Carlo - AWS*. URL: https://aws.amazon.com/es/what-is/monte-carlo-simulation/.

Caparrini, F. S. (2016a). *Búsquedas informadas*. URL: https://www.cs.us.es/~fsancho/?e=62.

– (2016b). *Búsquedas no informadas*. URL: https://www.cs.us.es/~fsancho/?e=95.

Carrasco, D. (dic. de 2022). *Para 2024 las ventas en eCommerce alcanzarán los 610 MDP en México (AMVO)*. URL: https://marketing4ecommerce.mx/para-2024-las-ventas-en-ecommerce-alcanzaran-los-610-mdp-en-mexico-amvo/ (visitado 24-08-2023).

Daniells, L. (s.f.). *The travelling salesman problem*. URL: https://www.lancaster.ac.uk/stor-i-student-sites/libby-daniells/2020/04/21/the-travelling-salesman-problem/.

Daza, Julio Mario, Jairo R. Montoya y Francesco Narducci (dic. de 2009). «RESOLUCIÓN DEL PROBLEMA DE ENRUTAMIENTO DE VEHÍCULOS CON LIMITACIONES DE CAPACIDAD UTILIZANDO UN PROCEDIMIENTO METAHEURÍSTICO DE DOS FASES». En: *Revista EIA* 12, Scielo. URL: http://www.scielo.org.co/pdf/eia/n12/n12a03.pdf.

documentation, IBM (s. f.). *Espacio de búsqueda para modelos de planificación*. URL: https://www.ibm.com/docs/es/icos/12.9.0?topic=optimizer-search-space-scheduling-models.

Giraldo-Picon, Edgar L., Jaime A. Giraldo-GarcÃa y Jorge A. Valderrama-Ortega (dic. de 2018). «Modelo de SimulaciÃde un Sistema LogÃstico de DistribuciÃcomo Plataforma Virtual para el Aprendizaje Basado en Problemas». es. En: *InformaciÃtecnolÃ* 29, págs. 185-198. ISSN: 0718-0764. URL: http://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-07642018000600185&nrm=iso.

Hino (2019). *Ficha técnica de los modelos 514 y 616 de la serie 300 [PDF]*. URL: https://hinotoluca.mx/public/fichas/Serie300_Modelos514-616.pdf.

IBM (2018). *¿Qué es la simulación Monte Carlo? — IBM*. URL: https://www.ibm.com/mx-es/topics/monte-carlo-simulation.

Kirkpatrick, I. S. (2020). *A nearest neighbor solution in go to the traveling salesman problem*. URL: https://levelup.gitconnected.com/a-nearest-neighbor-solution-in-go-to-the-traveling-salesman-problem-d4d56125b571.

Lacayo, J. y A. Estrada (2022). «Comercio electrónico pospandemia: el mejor momento para entrar al sector de última milla en México». En: *EY-Parthenon*. URL: https://www.ey.com/es_mx/strategy/ultima-milla#:~:text=Durante%20a%C3%B1os%2C%20el%20comercio%20electr%C3%B3nico,millones%20de%20d%C3%B3lares%20en%202020..

Lee, J. (s.f.). *A Travelling Salesman Problem With Carbon Emission Reduction in the Last Mile DeliveryJia*. URL: https://ieeexplore.ieee.org/Xplore/guesthome.jsp.

López, Bryan Salazar (ago. de 2021). «Problema de enrutamiento de vehículos (VRP) con Google OR-Tools». En: *Ingenieria Industrial Online*. URL: https://www.ingenieriaindustrialonline.com/investigacion-de-operaciones/problema-de-enrutamiento-de-vehiculos-vrp-con-google-or-tools/.

Mexicana, Nissan (2020). *Catálogo Nissan NP300 2020 [PDF]*. URL: https://www.nissan.com.mx/content/dam/Nissan/mexico/brochures/np300/MY20/np300_2020_catalogo.pdf.

Miguel, J. (2011). «¿Cómo calcular la distancia entre dos puntos geográficos en C? (Fórmula de Haversine)». En: *Genbeta*. URL: https://www.genbeta.com/desarrollo/como-calcular-la-distancia-entre-dos-puntos-geograficos-en-c-formula-de-haversine.

Monterrey, Tecnológico de (2023). *Módulos*. URL: https://experiencia21.tec.mx/courses/405971/modules.

Nissan Mexicana, S.A. de C.V. (2015-2021). *Catálogo Nissan NP300 2015-2021 [PDF]*. URL: https://www.nissan.com.mx/content/dam/Nissan/mexico/vehicles/NP300/my21/preventa/FT_NP300.pdf.

Norte, ISUZU (2013-2015). *ISUZU ELF 300 E, 3 TON*. URL: http://isuzunorte.alden.mx/Modelo/120731.html.

Tobin, B. (2023). *The travelling salesman problem (TSP)*. URL: https://smartroutes.io/blogs/the-travelling-salesman-problem/.

Traveling Salesperson problem using branch and bound (s.f.). URL: https://www.javatpoint.com/traveling-salesperson-problem-using-branch-and-bound.