Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática
Escuela Profesional de Ingeniería de Estadística e Informática

Optimization Methods

Aplicacion:

Optimización de Rutas de Vehículos: Vehicle Routing Problem (VRP) Navegando hacia la Eficiencia Logística

Estudiante: Herson Romario Condori Mamani

Profesor: Fred Torres Cruz

27 de fEBRERO de 2025

Introducción

Imaginen el desafío de una empresa de reparto que necesita entregar cientos de paquetes en una ciudad. ¿Cómo asegurar que cada paquete llegue a tiempo, con el menor costo posible y la máxima eficiencia? Aquí es donde entra en juego la Optimización de Rutas de Vehículos, o VRP.

¿Qué es VRP?

El VRP es un problema de optimización que busca la manera más eficiente de servir a un conjunto de clientes con una flota de vehículos. En términos sencillos, es como encontrar el mejor camino para que varios vehículos entreguen cosas a diferentes lugares, minimizando costos y maximizando eficiencia.

Problemas que resuelve

El VRP aborda problemas críticos en la logística, como:

- **Ineficiencia:** Rutas largas y desordenadas que consumen tiempo y combustible.
- **Altos costos:** Gastos excesivos en combustible, mantenimiento y personal.
- **Retrasos:** Entregas tardías que afectan la satisfacción del cliente.
- **Falta de control:** Dificultad para rastrear vehículos y entregas en tiempo real.
- **Impacto ambiental:** Exceso de emisiones de CO2 debido a rutas ineficientes.

¿Cómo funciona?

Nuestra aplicación utiliza algoritmos avanzados, como la Programación Lineal Entera o los algoritmos de Google OR-Tools, para analizar datos clave:

- Ubicaciones de los clientes.
- Pedidos y sus volúmenes.
- Capacidad de los vehículos.
- Restricciones de tiempo.

Con esta información, la aplicación calcula las rutas más óptimas, considerando factores como distancia, tráfico y capacidad de carga.

Beneficios clave

Al implementar VRP, las empresas pueden:

- **Reducir costos:** Ahorrar en combustible y mantenimiento.
- **Mejorar la eficiencia:** Realizar más entregas en menos tiempo.
- **Aumentar la satisfacción del cliente:** Garantizar entregas puntuales.
- **Ser más sostenibles:** Reducir la huella de carbono.
- **Obtener información valiosa:** Analizar datos para mejorar continuamente.

Aplicaciones prácticas

El VRP se aplica en diversos sectores, incluyendo:

- Empresas de reparto y mensajería.
- Distribución de alimentos y bebidas.
- Servicios de transporte público.
- Gestión de residuos.
- Servicios de emergencias.

Importancia de la Optimizacion de Rutas de Vehiculo

La optimización de rutas es un pilar fundamental en la logística y el transporte modernos, y su importancia radica en su capacidad para impactar directamente en tres factores críticos: costos, eficiencia y satisfacción del cliente.

1. Reducción de Costos

- **Combustible:** Rutas optimizadas significan menos kilómetros recorridos, lo que se traduce en un menor consumo de combustible. Dada la volatilidad de los precios del combustible, esto puede generar ahorros significativos.
- **Mantenimiento de Vehículos:** Menos kilómetros también reducen el desgaste de los vehículos, disminuyendo los costos de mantenimiento y prolongando su vida útil.
- **Tiempo de Conducción:** La optimización permite una mejor planificación, lo que minimiza el tiempo de conducción y, por ende, los costos laborales asociados.

2. Mejora de la Eficiencia

- **Entregas a Tiempo:** Rutas eficientes aseguran que las entregas se realicen a tiempo, lo que mejora la puntualidad y la confiabilidad del servicio.
- **Maximización de la Capacidad:** La optimización ayuda a aprovechar al máximo la capacidad de los vehículos, evitando viajes con carga incompleta.
- **Reducción de Tiempos de Espera:** Al planificar rutas que evitan congestiones y retrasos, se minimizan los tiempos de espera, lo que aumenta la productividad.

3. Aumento de la Satisfacción del Cliente

- **Entregas Puntuales y Confiables:** Los clientes valoran la puntualidad y la confiabilidad en las entregas. La optimización de rutas permite cumplir con las expectativas del cliente, lo que fortalece la lealtad.
- **Comunicación Mejorada:** Las herramientas de optimización a menudo incluyen funciones de seguimiento en tiempo real, lo que permite a los clientes conocer el estado de sus envíos.
- **Flexibilidad y Adaptabilidad:** La capacidad de adaptar las rutas en tiempo real ante imprevistos, como el tráfico o cambios en los pedidos, mejora la capacidad de respuesta y la satisfacción del cliente.

Descripcion General de la Aplicacion: Optimizador de Rutas de Vehículos (VRP)

Propósito

Esta aplicación tiene como objetivo principal revolucionar la logística y el transporte mediante la optimización inteligente de rutas de vehículos. Su función central es resolver el problema del Ruteo de Vehículos (VRP), permitiendo a las empresas:

- Minimizar costos operativos: A través de la reducción del consumo de combustible, el desgaste de vehículos y los tiempos de entrega.
- Mejorar la eficiencia logística: Al maximizar el número de entregas realizadas en un período determinado, minimizando los tiempos de inactividad y optimizando la capacidad de carga.
- Aumentar la satisfacción del cliente: Garantizando entregas puntuales y confiables, respetando las ventanas de tiempo y proporcionando información precisa sobre el estado de los envíos.

En esencia, la aplicación busca transformar la planificación y ejecución de rutas de entrega en un proceso más inteligente, rentable y sostenible.

Tecnologías Utilizadas

Para lograr estos objetivos, la aplicación se construye sobre una base tecnológica sólida y eficiente:

- **Streamlit:** Se utiliza para crear una interfaz de usuario interactiva y fácil de usar. Streamlit permite a los usuarios ingresar datos de manera sencilla, visualizar los resultados de la optimización y obtener informes detallados.
- **NetworkX:** Esta biblioteca de Python se emplea para el manejo de grafos, representando las ubicaciones de los clientes y las distancias entre ellas. NetworkX facilita la creación y manipulación de la red de rutas, permitiendo el cálculo eficiente de distancias y la identificación de caminos óptimos.
- **PuLP (o Google OR-Tools):** Se utiliza para la formulación y resolución de modelos de optimización lineal entera. PuLP permite traducir el problema del VRP en un conjunto de ecuaciones matemáticas y restricciones, que luego son resueltas para encontrar la solución óptima. Opcionalmente se puede usar Google OR-Tools para obtener una mejor performance en problemas de gran escala.
- **Python:** Es el lenguaje de programación principal, que permite la integración de las librerías anteriores para crear la aplicación.
- **Matplotlib:** Una biblioteca de Python para crear gráficos y visualizaciones. Se utiliza para visualizar los resultados de la optimización y los grafos de conexiones.
- **Folium:** Una biblioteca de Python para crear mapas interactivos. Se utiliza para visualizar las rutas optimizadas en un mapa.
- **Streamlit-Folium:** Una extensión de Streamlit que permite integrar mapas interactivos de Folium en aplicaciones Streamlit.
- **Pandas:** Una biblioteca de Python para la manipulación y análisis de datos. Se utiliza para manejar y procesar los datos de entrada y los resultados de la optimización.
- **Geopy:** Una biblioteca de Python para la geocodificación. Se utiliza para obtener las coordenadas geográficas de las ubicaciones de los clientes.

Funciones Principales del Optimizador de Rutas de Vehículos (VRP)

1. Carga de Datos

Carga de Archivo CSV: La aplicación permite a los usuarios cargar datos de clientes de manera rápida y eficiente a través de archivos CSV. Este método es ideal para empresas con grandes volúmenes de datos, ya que simplifica la importación de información como:

- Nombres y direcciones de clientes.
- Coordenadas geográficas (latitud y longitud).
- Volumen o peso de los pedidos.
- Ventanas de tiempo de entrega.

La aplicación valida el formato del archivo CSV para asegurar la integridad de los datos.

Ingreso Manual de Datos: Para usuarios con un número reducido de clientes o para pruebas rápidas, la aplicación ofrece la opción de ingresar datos manualmente a través de la interfaz. Los usuarios pueden ingresar la información de cada cliente directamente en formularios interactivos, lo que proporciona flexibilidad y facilidad de uso.

2. Configuración de Vehículos

Definición del Número de Vehículos: Los usuarios pueden especificar el número de vehículos disponibles en su flota. Este parámetro es crucial para la optimización, ya que el algoritmo debe distribuir las entregas entre los vehículos disponibles de manera eficiente.

Definición de Capacidades de Vehículos: La aplicación permite a los usuarios definir la capacidad de carga de cada vehículo (en peso o volumen). Este parámetro es esencial para garantizar que ningún vehículo exceda su capacidad máxima, lo que podría resultar en retrasos o problemas logísticos. La correcta definición de estos parámetros permite que el algoritmo pueda crear rutas viables.

Importancia en la Optimización: La correcta configuración de los vehículos es esencial para que el algoritmo pueda generar una solución viable y óptima.

3. Restricciones de Tiempo (Ventanas de Tiempo)

Establecimiento de Ventanas de Tiempo: Los usuarios pueden definir ventanas de tiempo específicas para cada cliente, indicando los horarios en los que las entregas deben realizarse. Esto permite a las empresas cumplir con los compromisos de entrega y adaptarse a las necesidades de los clientes.

Aseguramiento de la Satisfacción del Cliente: Las restricciones de tiempo juegan un papel fundamental en la satisfacción del cliente. Al respetar las ventanas de tiempo, las empresas demuestran profesionalismo y confiabilidad, lo que fortalece la relación con los clientes. El algoritmo de optimización tratará de cumplir con todas las ventanas de tiempo, y si no es posible, mostrará un mensaje al usuario.

Proceso de Optimización del Optimizador de Rutas de Vehículos (VRP)

1. Función Objetivo

Función Objetivo Ponderada: La aplicación utiliza una función objetivo ponderada para equilibrar dos metas principales: minimizar el tiempo total de viaje y maximizar la satisfacción del cliente. Esta función se puede expresar matemáticamente como:

Minimizar: (Peso_Tiempo×Tiempo_Total)+(Peso_Satisfacción×(1−Cumplimiento_Ventanas_Tiempo))

Donde:

- Tiempo_Total es el tiempo total de viaje de todos los vehículos.
- Cumplimiento_Ventanas_Tiempo es una medida del cumplimiento de las ventanas de tiempo de los clientes.
- Peso_Tiempo y Peso_Satisfacción son los pesos asignados por el usuario para priorizar el tiempo de viaje o la satisfacción del cliente, respectivamente.

Ajuste de Pesos por el Usuario: La aplicación permite a los usuarios ajustar los pesos (Peso_Tiempo y Peso_Satisfacción) según sus prioridades. Por ejemplo:

- Si la empresa prioriza la rapidez de las entregas, puede aumentar el Peso_Tiempo.
- Si la empresa prioriza el cumplimiento de las ventanas de tiempo y la satisfacción del cliente, puede aumentar el Peso_Satisfacción.

Esta flexibilidad permite a las empresas adaptar la optimización a sus necesidades específicas.

2. Restricciones

Restricciones de Entrada y Salida: Cada vehículo debe iniciar y finalizar su ruta en el depósito central. Esto asegura que todos los vehículos regresen al punto de partida después de completar sus entregas.

Restricciones de Capacidad: La carga total de cada vehículo en cualquier momento de su ruta no debe exceder su capacidad máxima. Esto garantiza que los vehículos no se sobrecarguen, lo que podría provocar retrasos o problemas de seguridad.

Restricciones de Tiempo: Cada entrega debe realizarse dentro de la ventana de tiempo especificada por el cliente. Esto asegura el cumplimiento de los compromisos de entrega y la satisfacción del cliente. Si no es posible cumplir todas las ventanas de tiempo, el algoritmo informará al usuario.

Restricciones de Ruta: Cada cliente debe ser visitado por un solo vehículo. Se evita que un mismo pedido sea entregado por varios vehículos.

Garantía de Solución Factible y Eficiente: Estas restricciones garantizan que la solución generada por el algoritmo sea factible (es decir, que cumpla con todos los requisitos) y eficiente (es decir, que minimice la función objetivo). Al considerar todas estas restricciones, la aplicación proporciona rutas optimizadas que son realistas y prácticas para las operaciones logísticas.

Resultados y Visualización del Optimizador de Rutas de Vehículos (VRP)

1. Mapa Interactivo

Visualización de Rutas Optimizadas: La aplicación muestra las rutas optimizadas en un mapa interactivo, permitiendo a los usuarios visualizar claramente el recorrido de cada vehículo. Se utilizan marcadores para representar las ubicaciones de los clientes y el depósito central. Las rutas se muestran como líneas que conectan los marcadores, indicando la secuencia de entregas para cada vehículo.

Diferenciación de Rutas por Color: Para facilitar la identificación de las rutas de diferentes vehículos, se utilizan colores distintos para cada ruta. Esto permite a los usuarios distinguir rápidamente las rutas asignadas a cada vehículo y comprender la distribución de las entregas. La interactividad del mapa permite hacer zoom, desplazarse y obtener información detallada sobre cada punto de entrega al hacer clic en los marcadores.

Integración con API de Mapas: Se utiliza una API de mapas (como Leaflet o Mapbox) para mostrar el mapa interactivo, asegurando una visualización precisa y detallada de las rutas.

2. Grafo de Conexiones

Presentación del Grafo: La aplicación genera un grafo que muestra las conexiones entre los nodos (ubicaciones de clientes y depósito). Los nodos representan las ubicaciones, y las aristas representan las conexiones entre ellas, indicando la secuencia de visitas.

Comprensión de la Estructura de la Red: El grafo permite a los usuarios comprender la estructura de la red de entregas y la secuencia de visitas de cada vehículo. Se puede visualizar la secuencia de visitas y la relación entre los nodos. Esto es útil para identificar patrones y evaluar la eficiencia de la solución.

3. Análisis de Costos y Distancias

Gráficos de Barras: La aplicación genera gráficos de barras que resumen los costos y distancias de las rutas. Estos gráficos muestran:

- La distancia total recorrida por cada vehículo.
- El costo total de combustible por vehículo.
- El tiempo total de entrega por vehículo.

Evaluación de la Eficiencia: Los gráficos de barras permiten a los usuarios evaluar fácilmente la eficiencia de la solución. Se pueden comparar los costos y distancias de diferentes rutas para identificar áreas de mejora. Se pueden identificar rutas con costos o distancias excesivas, lo que indica posibles ineficiencias. Al analizar estos gráficos, los usuarios pueden tomar decisiones informadas para optimizar aún más sus operaciones logísticas.

Conclusión: Optimizando el Futuro de la Logística con VRP

Resumen de Beneficios

En resumen, nuestra aplicación de optimización de rutas de vehículos (VRP) ofrece una solución integral para los desafíos logísticos modernos. Al implementar esta herramienta, las empresas pueden lograr beneficios significativos:

- Minimización de costos: Reducción drástica del consumo de combustible, costos de mantenimiento y tiempos de entrega, lo que se traduce en ahorros sustanciales.
- Mejora de la eficiencia: Optimización de rutas para maximizar el número de entregas, minimizar los tiempos de inactividad y aprovechar al máximo la capacidad de los vehículos.
- Aumento de la satisfacción del cliente: Entregas puntuales y confiables, respeto de las ventanas de tiempo y comunicación transparente, fortaleciendo la lealtad del cliente.
- Sostenibilidad: Reducción de la huella de carbono al optimizar las rutas y reducir las emisiones.
- Visibilidad y control: Seguimiento en tiempo real y análisis detallado del rendimiento de las rutas.

En un entorno empresarial cada vez más competitivo, nuestra aplicación VRP proporciona una ventaja estratégica al permitir a las empresas operar de manera más inteligente, rentable y sostenible.

Código Fuente

app.py

```
import streamlit as st
import networkx as nx
import matplotlib.pyplot as plt
import folium
import random
import pandas as pd
from streamlit_folium import st_folium
from solver import resolver_vrp_multiobjetivo
from graph import crear_grafo, cargar_datos_csv
from folium.plugins import AntPath
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut, GeocoderServiceError

# Inicializar el geocodificador
geolocator = Nominatim(user_agent="Optimizaci n de Ruteo de
  Veh culos")

st.set_page_config(page_title="Optimizaci n de Ruteo de
  Veh culos", layout="wide")

st.title("     Optimizaci n de Ruteo de Veh culos")
st.markdown("---")

# Inicializar el estado de sesi n
if 'resultados' not in st.session_state:
st.session_state.resultados = None

def mostrar_guia_usuario():
st.subheader("     Gu a del Usuario")
st.markdown("""
**Bienvenido a la aplicaci n de Optimizaci n de Ruteo de
  Veh culos.**

**1. Cargar Datos:**
- Utiliza el bot n en la barra lateral para cargar un archivo
  CSV con los datos de los clientes.
- El archivo debe contener columnas para el origen, destino y
  costo.

**2. Ingresar Clientes Manualmente:**
- Puedes ingresar clientes manualmente usando los campos en la
  barra lateral.
- Proporciona el origen, destino y costo para cada cliente.

**3. Seleccionar Veh culos:**
- Define el n mero de veh culos y sus capacidades en la barra
  lateral.

**4. Restricciones de Tiempo:**
- Establece ventanas de tiempo para cada nodo (excepto el
  dep sito).

**5. Ejecutar la Optimizaci n:**
```

```
- Una vez que todos los datos est n ingresados, la aplicaci n calcular  las rutas  ptimas .
- Los resultados se mostrar n en el mapa y en el grafo de conexiones.

**6. Mensajes de Error:**
- Si encuentras alg n error, revisa los mensajes de error para obtener m s informaci n.
""")

def mostrar_formulario_feedback():
st.subheader("     Formulario de Feedback")
feedback = st.text_area("Por favor, proporciona tu feedback sobre la aplicaci n:")

if st.button("Enviar Feedback"):
if feedback.strip():
guardar_feedback(feedback)
st.success("  Gracias  por tu feedback!")
else:
st.warning("Por favor, ingresa alg n comentario antes de enviar.")

def guardar_feedback(feedback):
with open("feedback.txt", "a") as file:
file.write(feedback + "\n")

# Bot n para mostrar la gu a del usuario
if st.sidebar.button("     Mostrar Gu a del Usuario"):
mostrar_guia_usuario()

# Bot n para mostrar el formulario de feedback
if st.sidebar.button("     Proporcionar Feedback"):
mostrar_formulario_feedback()

# Cargar datos desde archivo CSV
archivo_csv = st.sidebar.file_uploader("     Cargar archivo CSV de clientes", type=["csv"])

# Configuraci n de clientes y costos en la barra lateral
st.sidebar.header("Datos de Clientes y Costos")

# N mero de clientes
num_clientes = st.sidebar.number_input("N mero de clientes", min_value=1, value=1)

# Ingresar datos de clientes
client_data = []
for i in range(num_clientes):
with st.sidebar.expander(f"Cliente {i+1}"):
st.write(f"### Detalles del Cliente {i+1}")
origen = st.text_input(f"Origen (Cliente {i+1})", f"Origen_{i+1}")
destino = st.text_input(f"Destino (Cliente {i+1})", f"Destino_{i+1}")
costo = st.number_input(f"Costo (Cliente {i+1})", min_value=0, value=100)
```

```
93
94  try:
95  # Obtener coordenadas para el origen y destino
96  location_origen = geolocator.geocode(origen)
97  location_destino = geolocator.geocode(destino)
98  x_origen, y_origen = location_origen.latitude, location_origen.
      longitude
99  x_destino, y_destino = location_destino.latitude,
      location_destino.longitude
100
101 except GeocoderTimedOut:
102 st.sidebar.error("Error: El servicio de geocodificaci n est
      tardando demasiado. Int ntalo de nuevo.")
103 continue
104 except GeocoderServiceError as e:
105 st.sidebar.error(f"Error de servicio de geocodificaci n: {e}.
      Por favor, int ntalo de nuevo m s tarde.")
106 continue
107
108 if st.sidebar.button(f"Agregar Cliente {i+1}"):
109 client_data.append((origen, destino, costo, (x_origen, y_origen),
       (x_destino, y_destino)))
110
111 # Configuraci n de veh culos en la barra lateral
112 st.sidebar.header("Configuraci n de Veh culos")
113 num_vehiculos = st.sidebar.number_input("N mero de veh culos",
      min_value=1, value=1)
114 capacidades = [st.sidebar.number_input(f"Capacidad del veh culo
      {i+1}", min_value=1, value=100) for i in range(num_vehiculos)]
115
116 # Selecci n de ciudades para restricciones de tiempo
117 st.sidebar.header("Selecci n de Ciudades para Restricciones de
      Tiempo")
118 ciudades = ["Juliaca", "Puno", "Arequipa", "Cusco", "Tacna"]
119 ciudades_seleccionadas = st.sidebar.multiselect("Selecciona
      ciudades para restricciones de tiempo:", ciudades)
120
121 # Inicializaci n de ventanas_tiempo con valores predeterminados
122 ventanas_tiempo = {ciudad: (pd.to_datetime("09:00").time(), pd.
      to_datetime("17:00").time()) for ciudad in ciudades}
123
124 # Configuraci n de restricciones de tiempo para ciudades
      seleccionadas
125 for ciudad in ciudades_seleccionadas:
126 with st.sidebar.expander(f"{ciudad}"):
127 inicio = st.time_input(f"Hora de inicio para {ciudad}", value=
      ventanas_tiempo[ciudad][0])
128 fin = st.time_input(f"Hora de fin para {ciudad}", value=
      ventanas_tiempo[ciudad][1])
129 ventanas_tiempo[ciudad] = (inicio, fin)
130
131 # Pesos de los objetivos en la barra lateral
132 tiempo_viaje_ponderado = st.sidebar.number_input("Peso para
      minimizar tiempo de viaje", min_value=0.0, max_value=1.0, value
      =0.7)
133 satisfaccion_cliente_ponderado = st.sidebar.number_input("Peso
      para maximizar satisfacci n del cliente", min_value=0.0,
```

```
    max_value=1.0, value=0.3)
134
135 # Bot n de optimizaci n
136 if st.sidebar.button("Optimizar"):
137 # Procesar datos de clientes
138 nodes = {}
139 edges = {}
140
141 if archivo_csv is not None:
142 nodes, edges = cargar_datos_csv(archivo_csv)
143 graph = nx.DiGraph()
144 for node, coord in nodes.items():
145 graph.add_node(node, pos=coord)
146 for edge, cost in edges.items():
147 graph.add_edge(edge[0], edge[1], weight=cost)
148 else:
149 graph, nodes, edges = crear_grafo()
150
151 # Agregar datos de clientes ingresados manualmente
152 for data in client_data:
153 origen, destino, costo, coord_origen, coord_destino = data
154 if origen in ciudades_seleccionadas and destino in
    ciudades_seleccionadas:
155 nodes[origen] = coord_origen
156 nodes[destino] = coord_destino
157 edges[(origen, destino)] = costo
158 graph.add_node(origen, pos=coord_origen)
159 graph.add_node(destino, pos=coord_destino)
160 graph.add_edge(origen, destino, weight=costo)
161
162 # Definir la demanda para cada nodo
163 demanda = {node: 10 for node in nodes}
164
165 # Ejemplo de satisfacci n del cliente (puede ser una funci n de
     la puntualidad)
166 satisfaccion = {edge: random.uniform(0, 1) for edge in edges}
167
168 # Antes de llamar a la funci n de resoluci n
169 print("Ventanas de tiempo:", ventanas_tiempo)
170
171 # Llamar a la funci n de resoluci n
172 model, x = resolver_vrp_multiobjetivo(nodes, edges, num_vehiculos
    , capacidades, ventanas_tiempo, demanda, satisfaccion,
    tiempo_viaje_ponderado, satisfaccion_cliente_ponderado)
173
174 # Guardar resultados en el estado de sesi n
175 st.session_state.resultados = {
176   "model": model,
177   "x": x,
178   "nodes": nodes,
179   "edges": edges,
180   "graph": graph
181 }
182
183 # Mostrar resultados si est n disponibles
184 if st.session_state.resultados:
185 resultados = st.session_state.resultados
```

```
model = resultados["model"]
x = resultados["x"]
nodes = resultados["nodes"]
edges = resultados["edges"]
graph = resultados["graph"]

# Mostrar resultados de la optimizaci n
st.subheader("Resultados de Optimizaci n")
st.write("### Estado de optimizaci n:", model.status)
for edge in edges:
if x[edge].varValue == 1:
st.write(f"    Ruta seleccionada: {edge} con costo {edges[edge]}")

# Mapa interactivo con Folium centrado en Juliaca
st.subheader("    Mapa de Rutas Optimizadas")
mapa = folium.Map(location=[-15.500060871723104, -70.12876822242285], zoom_start=10)

# Colores para diferentes veh culos
colores = ['blue', 'green', 'purple', 'orange', 'darkred', 'lightred', 'beige', 'darkblue', 'darkgreen', 'cadetblue']

# Agregar marcadores de nodos
for node, coord in nodes.items():
folium.Marker([coord[0], coord[1]], popup=node).add_to(mapa)

# Agregar rutas animadas
for v in range(num_vehiculos):
ruta = [edge for edge in edges if x[edge].varValue == 1 and edge[0] == f"vehiculo_{v+1}"]
if ruta:
puntos = [nodes[edge[0]] for edge in ruta] + [nodes[ruta[-1][1]]]
AntPath(locations=[list(reversed(punto)) for punto in puntos], color=colores[v % len(colores)]).add_to(mapa)

st_folium(mapa, width=700, height=500)

# Visualizaci n del grafo
st.subheader("    Grafo de Conexiones")
fig, ax = plt.subplots()
pos = {node: coord for node, coord in nodes.items()}

nx.draw(graph, pos, with_labels=True, node_size=500, node_color='lightblue', ax=ax)
edge_labels = {(i, j): f"{cost}" for (i, j), cost in edges.items()}
nx.draw_networkx_edge_labels(graph, pos, edge_labels=edge_labels, ax=ax)

st.pyplot(fig)

# Gr ficos de barras para costos y distancias
st.subheader("    An lisis de Costos y Distancias")
costos = [edges[edge] for edge in edges if x[edge].varValue == 1]
distancias = [graph[edge[0]][edge[1]]['weight'] for edge in edges if x[edge].varValue == 1]
```

```
234
235  fig, ax = plt.subplots(1, 2, figsize=(14, 6))
236  ax[0].bar(range(len(costos)), costos, color='skyblue')
237  ax[0].set_title('Costos de las Rutas')
238  ax[0].set_xlabel('Rutas')
239  ax[0].set_ylabel('Costo')
240
241  ax[1].bar(range(len(distancias)), distancias, color='lightgreen')
242  ax[1].set_title('Distancias de las Rutas')
243  ax[1].set_xlabel('Rutas')
244  ax[1].set_ylabel('Distancia')
245
246  st.pyplot(fig)
247
248  # Tabla interactiva de resultados
249  st.subheader("        Tabla de Resultados")
250  resultados_tabla = pd.DataFrame({
251    "Ruta": [f"{edge[0]} -> {edge[1]}" for edge in edges if x[edge
     ].varValue == 1],
252    "Costo": costos,
253    "Distancia": distancias
254  })
255  st.dataframe(resultados_tabla)
```

solver.py

```
1  import pulp as lp
2  import random
3
4  def convertir_a_minutos(tiempo):
5  return tiempo.hour * 60 + tiempo.minute
6
7  def resolver_vrp_multiobjetivo(nodes, edges, num_vehiculos,
     capacidades, ventanas_tiempo, demanda, satisfaccion,
     tiempo_viaje_ponderado, satisfaccion_cliente_ponderado):
8  model = lp.LpProblem("Vehicle_Objetive_VRP", lp.LpMinimize)
9  x = lp.LpVariable.dicts("Route", edges, cat=lp.LpBinary)
10
11 # Variables de tiempo para cada nodo
12 tiempo_llegada = lp.LpVariable.dicts("TiempoLlegada", nodes,
     lowBound=0, cat='Continuous')
13
14 # Funci n objetivo ponderada
15 model += lp.lpSum(tiempo_viaje_ponderado * x[edge] * cost for
     edge, cost in edges.items()) - \
16 lp.lpSum(satisfaccion_cliente_ponderado * satisfaccion[edge] for
     edge in edges)
17
18 # Restricciones de entrada y salida
19 for node in nodes:
20 if node != "depot":
21 model += lp.lpSum(x[(i, node)] for i in nodes if (i, node) in
     edges) == 1
22 model += lp.lpSum(x[(node, j)] for j in nodes if (node, j) in
     edges) == 1
23
```

```
    # Restricciones de capacidad
    for v in range(num_vehiculos):
    model += lp.lpSum(x[(i, j)] * demanda[j] for (i, j) in edges if i
        == f"vehiculo_{v+1}") <= capacidades[v]

    # Restricciones de tiempo
    for node in nodes:
    if node != "depot":
    inicio, fin = ventanas_tiempo[node]
    inicio_minutos = convertir_a_minutos(inicio)
    fin_minutos = convertir_a_minutos(fin)
    model += tiempo_llegada[node] >= inicio_minutos
    model += tiempo_llegada[node] <= fin_minutos

    # Restricciones de tiempo de viaje (simplificado)
    for (i, j) in edges:
    if i != "depot" and j != "depot":
    model += tiempo_llegada[j] >= tiempo_llegada[i] + x[(i, j)] *
      edges[(i, j)] - (1 - x[(i, j)]) * 100000

    model.solve()
    return model, x
```

graph.py

```
    import streamlit as st
    import networkx as nx
    import pandas as pd

    def crear_grafo():
    graph = nx.DiGraph()

    # Definir nodos con coordenadas reales del sur de Per , centrado
        en Juliaca
    nodes = {
      "Juliaca": (-15.500060871723104, -70.12876822242285),  # Centro
      "Puno": (-15.84206715081413, -70.02148324949029),
      "Arequipa": (-16.409047, -71.537451),
      "Cusco": (-13.532, -71.967),
      "Tacna": (-18.0066, -70.2463)
    }

    # Definir conexiones entre nodos con costos ficticios (distancias
        aproximadas en km)
    edges = {
      ("Juliaca", "Puno"): 60,
      ("Juliaca", "Arequipa"): 280,
      ("Juliaca", "Cusco"): 350,
      ("Juliaca", "Tacna"): 600,
      ("Puno", "Cusco"): 380,
      ("Arequipa", "Tacna"): 380,
      ("Cusco", "Tacna"): 800
    }

    # Agregar nodos al grafo
    for node, coord in nodes.items():
```

```
    graph.add_node(node, pos=coord)

    # Agregar conexiones entre nodos (edges)
    for edge, cost in edges.items():
    graph.add_edge(edge[0], edge[1], weight=cost)

    return graph, nodes, edges

    def cargar_datos_csv(archivo_csv):
    df = pd.read_csv(archivo_csv)

    # Leer nodos (clientes) con sus coordenadas
    nodes = {row['Cliente']: (row['X'], row['Y']) for _, row in df.
      iterrows()}
    print("Nodos cargados:", nodes)  # Mensaje de depuraci n
    # Leer las conexiones con costos
    edges = {(row['Origen'], row['Destino']): row['Costo'] for _, row
       in df.iterrows()}
    print("Conexiones cargadas:", edges)  # Mensaje de depuraci n

    # Verificar que todos los nodos en las conexiones tengan
      coordenadas
    for edge in edges:
    origen, destino = edge
    if origen not in nodes:
    print(f"Advertencia: El nodo {origen} no tiene coordenadas
      asignadas.")
    if destino not in nodes:
    print(f"Advertencia: El nodo {destino} no tiene coordenadas
      asignadas.")

    return nodes, edges
```

requirements.txt

```
streamlit
networkx
matplotlib
pulp
folium
streamlit_folium
pandas
geolocator
```

Enlaces

- Aplicación en Streamlit
- Repositorio en GitHub