Optimización de Rutas de Ambulancias en Entornos Urbanos

Universidad Pontificia Bolivariana

Técnicas de Optimización

John Fernando Vargas Buitrago

Alexandra Vasco

Roy Sandoval

2025

1 Introducción

Este proyecto desarrolla un modelo matemático de optimización basado en flujo multimercancía para determinar las rutas óptimas de ambulancias que responden a múltiples emergencias simultáneas en un área urbana de aproximadamente un kilómetro cuadrado. El modelo minimiza el costo total de operación del sistema considerando los tiempos de viaje y los costos operativos diferenciados según el tipo de ambulancia (leve, media o crítica), mientras respeta las capacidades de velocidad de las vías y garantiza tiempos de respuesta adecuados según la severidad de cada emergencia.

La implementación se realizó utilizando Python con la librería PuLP para la resolución del problema de programación lineal entera binaria, integrando datos geográficos reales obtenidos mediante OSMnx. El presente informe describe la formulación matemática del modelo, el procedimiento de implementación computacional, y los resultados obtenidos en diferentes escenarios de prueba, presentados mediante una aplicación interactiva desarrollada en Streamlit que permite visualizar las rutas óptimas en mapas.

2 Descripción del Área de Estudio y Datos Utilizados

El modelo se implementa sobre la red vial urbana de Medellín, Colombia, centrada en las coordenadas de la Clínica Medellín Occidente (6.2331°N, 75.5839°W). Se selecciona un área circular con radio configurable entre 400 y 800 metros, lo que permite analizar zonas compactas del entramado urbano donde convergen diferentes tipos de vías: avenidas principales, calles secundarias y vías residenciales.

Los datos de la red vial se obtienen de OpenStreetMap (OSM), una base de datos geográfica colaborativa de acceso libre. OSM proporciona información detallada sobre la topología de las calles, incluyendo ubicación de intersecciones, conectividad entre segmentos viales, longitudes de calles, y en algunos casos, características como tipo de vía y número de carriles.

Para acceder a estos datos se utiliza la librería OSMnx, que facilita la descarga, construcción y análisis de redes de calles desde OSM. OSMnx convierte automáticamente los datos geográficos en un grafo dirigido donde los nodos representan intersecciones y los arcos representan segmentos de calle con dirección específica.

Los datos obtenidos incluyen:

- **Nodos**: Intersecciones de calles con coordenadas geográficas (latitud, longitud)
- **Arcos dirigidos**: Segmentos de calle con información de:
  - Longitud en metros
  - Nodo origen y nodo destino
  - Geometría de la vía (coordenadas intermedias)

3 Formulación Matemática del Modelo de Optimización

El problema se modela como flujo multimercancía donde cada *commodity* representa una ambulancia con ruta propia desde la base hasta su emergencia asignada.

3.1 Definición de Conjuntos, Parámetros y Variables

3.1.1 Conjuntos:

- **N:** Conjunto de nodos de la red vial (intersecciones de calles).
- **A:** Conjunto de arcos dirigidos, donde A = {(i,j) | i ∈ N, j ∈ N}. Cada arco representa un segmento de calle transitable en dirección de i hacia j.
- **K:** Conjunto de commodities o tipos de flujo. Cada k ∈ K representa una emergencia individual definida por la tupla k = (nodo_destino, tipo_severidad), donde:
  - **nodo_destino** ∈ N es el nodo de la emergencia
  - **tipo_severidad** ∈ {Leve, Media, Crítica} es el nivel de severidad

Cada commodity es único e independiente, incluso si múltiples emergencias comparten el mismo tipo de severidad.

3.1.2 Parámetros de la Red:

- **l_ij:** Longitud del arco (i,j) en kilómetros. Obtenido directamente de los datos geográficos de OSMnx.
- **c_ij**: Capacidad del arco (i,j) en km/h. Representa la velocidad máxima permitida en ese segmento. Se asigna aleatoriamente en el rango [C_min, C_max] configurable por el usuario.
- **t_ij**: Tiempo de viaje en el arco (i,j) en horas, calculado como:

$$t_{ij} = \frac{l_{ij}}{c_{ij}}$$

  Esta relación proviene de la fórmula básica de la velocidad: v = d/t

3.1.3 Parámetros de los Commodities:

- **o:** Nodo de origen de los commodities(la base de donde parten todas las ambulancias)
- **d**:El nodo de destino está implícito en la definición del commodity k = (nodo_destino, tipo_severidad). Por lo tanto, para el commodity k = (d, s), su nodo destino es d y su tipo de severidad es s.
- **r_k:** Velocidad requerida para el commodity k = (d, s) en km/h. Se genera una velocidad independiente para cada commodity según su tipo de severidad s, utilizando generación aleatoria dentro de rangos diferenciados:
    - **Crítica:** r_k ∈ [0.8·R_max, R_max] (velocidades altas: 56-70 km/h si R_max=70)
    - **Media:** r_k ∈ [R_min+0.4(R_max-R_min), 0.9·R_max] (velocidades medias: 39-63 km/h)
    - **Leve:** r_k ∈ [R_min, R_min+0.6(R_max-R_min)] (velocidades bajas: 30-48 km/h)

- **α_s:** Costo operativo en $/hora según el tipo de severidad s ∈ {Leve, Media, Crítica}.
    - **α_Leve =** $100/hora: Ambulancia de transporte simple
    - **α_Media =** $250/hora: Ambulancia de cuidados intermedios
    - **α_Crítica =** $500/hora: Ambulancia de cuidados intensivos

  Para un commodity k = (d, s), su costo operativo es α_s (basado en su severidad s).

3.1.4 Variables de Decisión:

- **x_ijk**: Variable binaria que indica si el commodity k usa el arco (i,j).

$$\boldsymbol{x_{ijk}} \in \{0,1\} \; \forall \; (i,j) \in A, \forall k \in K$$

donde:

- x_ijk = 1 si la ruta óptima del commodity k incluye el arco (i,j)
- x_ijk = 0 en caso contrario

3.2 Función Objetivo

Se busca minimizar el costo total del sistema, considerando el tiempo de viaje de cada ambulancia ponderado por su costo operativo:

$$\boldsymbol{minZ} = \sum_{\boldsymbol{k} \in \boldsymbol{K}} \left(\boldsymbol{\alpha_{sk}} * \sum_{\boldsymbol{(i,j)} \in \boldsymbol{A}} \boldsymbol{t_{ij}} * \boldsymbol{x_{ijk}} \right)$$

donde s(k) denota el tipo de severidad del commodity k = (d, s).

Interpretación:

La sumatoria externa recorre todas las emergencias individuales y para cada commodity k = (d, s):

- α_s(k) es el costo operativo por hora según su severidad

- Σ_((i,j)∈A) t_ij · x_ijk calcula el tiempo total de viaje
- El producto α_s(k) · (tiempo total) es el costo de esa ambulancia específica

La suma total agrega los costos de todas las ambulancias despachadas a todas las emergencias, permitiendo que múltiples ambulancias del mismo tipo operen simultáneamente con rutas independientes.

3.3 Restricciones

3.3.1 Conservación del Flujo:

Para cada commodity k y cada nodo i, el flujo neto debe satisfacer:

$$\sum_{j:(i,j)\in A} x_{ijk} - \sum_{j:(j,i)\in A} x_{ijk} = b_{ik}, \quad \forall i \in N, \ \forall k \in K$$

Donde el balance b_ik se define como:

$$b_{ik} \begin{cases} 1 & si\ i = o\ (nodo\ origen) \\ -1 & si\ i = d_k (nodo\ destino) \\ 0 & si\ i\ es\ algún\ otro\ nodo\ intermedio \end{cases}$$

Interpretación:

- El primer término cuenta los arcos que salen del nodo i
- El segundo término cuenta los arcos que entran al nodo i
- En el origen: sale exactamente una unidad de flujo (1 - 0 = 1)
- En el destino: entra exactamente una unidad de flujo (0 - 1 = -1)
- En nodos intermedios: lo que entra debe salir (conservación)

3.3.2 Velocidad Requerida:

Para cada arco y cada commodity:

$$r_k * x_{ijk} \leq c_{ij} \quad , \forall (i,j) \in A, \forall k \in K$$

Interpretación:

- Si $x_{ijk} = 1$ (el commodity k usa el arco), entonces la restricción se activa: $r_k \leq c_{ij}$, lo que significa que la velocidad requerida debe ser menor o igual a la capacidad del arco.
- Si $x_{ijk} = 0$ (el arco no se usa), la restricción se vuelve trivialmente verdadera: $0 \leq c_{ij}$.

Para el problema, esto significa que las ambulancias con velocidades requeridas más altas, solo van a poder usar calles que tengan capacidades lo suficientemente altas.

4. Procedimiento de Implementación Computacional

La implementación se desarrolló en Python utilizando una arquitectura modular que separa responsabilidades en tres componentes principales:

4.1 Modulo de Visualización

Este módulo encapsula toda la funcionalidad relacionada con datos geográficos y visualización cartográfica.

4.1.1 NetworkManager (network.py):

Esta clase es responsable de la extracción, procesamiento y gestión de la red vial. Sus principales funcionalidades incluyen:

- **Descarga de redes viales**: Utiliza OSMnx para obtener grafos dirigidos desde OpenStreetMap. Soporta dos métodos de extracción: circular (radio desde un punto central) y rectangular (bounding box). Filtra por tipo de red drive para incluir únicamente vías transitables por vehículos.
- **Procesamiento topológico**: Extrae la componente fuertemente conexa más grande del grafo para garantizar que todos los nodos sean alcanzables desde cualquier origen. Esto es crítico para que el modelo de optimización encuentre rutas factibles.

- **Asignación de capacidades**: Genera aleatoriamente capacidades (velocidades máximas) para cada arco dentro del rango [C_min, C_max] configurable. Estas capacidades se almacenan como atributo capacity en los arcos del grafo.
- **Selección de nodos**: Implementa un algoritmo de selección aleatoria de nodos origen y destino mediante análisis de descendientes en el grafo dirigido.
- **Sistema de caché**: Serializa redes descargadas en formato pickle para evitar descargas repetidas de OSM para reducir el tiempo de carga en ejecuciones posteriores.

4.1.2 MapVisualizer (map_display.py):

Esta clase genera visualizaciones interactivas de la red y las rutas óptimas utilizando la librería Folium, que crea mapas HTML embebibles basados en Leaflet.js. Sus funcionalidades principales incluyen:

- **Renderizado de red base:** Dibuja todos los arcos de la red vial como líneas en el mapa, con opacidad reducida para servir como contexto visual sin dominar la visualización.
- **Visualización de rutas optimizadas:** Superpone las rutas calculadas por el modelo de optimización con colores diferenciados según severidad (azul para Leve, naranja para Media, rojo para Crítica) y mayor grosor para destacarlas visualmente.
- **Marcadores geográficos:** Añade marcadores con iconografía diferenciada para el origen (ícono de casa en verde) y los destinos (íconos según severidad). Cada marcador incluye popups informativos con datos de la emergencia.
- **Leyenda interactiva:** Genera una leyenda HTML embebida en el mapa que explica la simbología utilizada, mejorando la interpretabilidad de la visualización.

- **Integración con Streamlit:** Retorna objetos Folium compatibles con el componente st_folium para visualización embebida en la aplicación web.

4.2 Modulo de Optimización

Este módulo implementa el modelo matemático de flujo multimercancía y su resolución.

4.2.1 OptimizationData (data_interface.py):

Esta clase actúa como estructura de datos intermedia entre el módulo de visualización y el modelo de optimización. Encapsula toda la información necesaria para construir el problema de optimización e implementa:

- **Extracción de datos del grafo**: Método from_network() que recibe un grafo de OSMnx y extrae listas de nodos, arcos (con claves para multigrafos), y diccionarios de atributos de arcos incluyendo longitud, capacidad y tiempo de viaje.
- **Estructuración de emergencias**: Almacena tuplas (nodo_id, severidad) para cada emergencia, manteniendo la información necesaria para construir los commodities del modelo.
- **Generación de parámetros**: Método auxiliar get_required_speeds() que genera velocidades requeridas aleatorias según distribuciones diferenciadas por severidad, siguiendo los rangos definidos en la formulación matemática.

4.2.2 AmbulanceRoutingModel (model.py):

Esta es la clase central que implementa el modelo matemático completo siguiendo la formulación de la Sección 3. Su arquitectura está organizada en las siguientes etapas:

- **Inicialización y mapeo de commodities**: El constructor recibe un objeto OptimizationData y crea un commodity único para cada emergencia individual, definido como tupla k = (nodo_destino, tipo_severidad).
- **Configuración de parámetros**: Método set_parameters() que define costos operativos por tipo de severidad (α_Leve, α_Media, α_Crítica) y genera velocidades requeridas individuales para cada commodity mediante distribuciones aleatorias diferenciadas.
- **Construcción del modelo PuLP**: Método build_model() que ejecuta la siguiente secuencia:

    a. Crea el problema de optimización con sentido de minimización
    b. Genera variables binarias x_ijk para cada combinación de arco y commodity
    c. Define la función objetivo como suma ponderada de costos operativos por tiempos de viaje
    d. Añade restricciones de conservación de flujo para cada nodo y commodity
    e. Añade restricciones de velocidad requerida para cada arco y commodity

- **Resolución del problema**: Se implementa el método solve() de Pulp.
- **Extracción de resultados**: Conjunto de métodos que procesan la solución óptima:

    - _extract_solution(): Identifica qué variables x_ijk tienen valor 1
    - get_routes_as_paths(): Reconstruye secuencias ordenadas de nodos desde origen hasta cada destino
    - get_solution_summary(): Calcula métricas agregadas (distancia, tiempo, costo) por cada ruta
    - print_solution(): Genera reportes textuales legibles

4.3 Interfaz de Usuario

La aplicación web se implementó utilizando Streamlit, un framework Python para desarrollo rápido de aplicaciones de datos interactivas.

4.3.1 Estructura de la Interfaz

- **Panel de configuración lateral:** Contiene controles para todos los parámetros del modelo organizados en secciones:
    - Configuración geográfica (coordenadas, forma del área, distancia)
    - Parámetros de velocidad (R_min, R_max, C_min, C_max)
    - Cantidad de emergencias a generar
    - Costos operativos por tipo de ambulancia
    - Botones de recálculo de flujos y capacidades
    -
- **Pestañas principales:** La interfaz se organiza en tres pestañas:
    - **Map Visualization:** Carga de red, visualización del mapa con rutas optimizadas, y estadísticas de la red
    - **Results:** Ejecución del modelo de optimización, métricas agregadas, detalles por emergencia, y exportación de resultados
    - **About:** Documentación del proyecto, instrucciones de uso, y detalles técnicos

4.3.2 Flujo de Ejecución

El proceso completo sigue esta secuencia:

1. Usuario configura parámetros en el panel lateral.
2. Se hace clic en "Load Network": Esto hace que NetworkManager descargue/cargue la red vial.

3. NetworkManager asigna capacidades aleatorias y selecciona nodos origen/destino.
4. OptimizationData estructura los datos para el modelo.
5. En pestaña Results, al hacer clic en "Run Optimization": Esto hace que AmbulanceRoutingModel construya y resuelva el problema de optimización.
6. MapVisualizer dibuja rutas óptimas en el mapa con colores diferenciados.
7. Interfaz despliega métricas agregadas y detalles por emergencia.
8. Usuario puede exportar solución en formato JSON.

Los botones "Recalculate Flows" y "Recalculate Capacities" permiten explorar múltiples escenarios.

4.4 Tecnologías y Bibliotecas

- **Python 3.10+:** Lenguaje base para toda la implementación.
- **PuLP:** Librería de modelado de optimización lineal.
- **OSMnx:** Librería para descargar, modelar y analizar redes de calles desde OpenStreetMap.
- **NetworkX:** Librería para manipulación y análisis de grafos.
- **Folium:** Librería para crear mapas interactivos basados en Leaflet.js.
- **Streamlit:** Framework para construcción de aplicaciones web interactivas.

4 Escenarios de Prueba y Resultados

Se diseñaron tres escenarios con el fin de validar el modelo bajo distintas condiciones operativas: ideal, moderada y crítica. Cada escenario evalúa la capacidad del modelo para optimizar el enrutamiento de ambulancias considerando restricciones de velocidad, capacidad vial y severidad de emergencias.

4.1 Escenario 1: Escenario Ideal

Descripción:

Baja demanda de emergencias y alta capacidad vial. Representa una situación ideal con recursos abundantes y tráfico fluido.

Parámetros:

- Área: Radio 560 m
- Velocidades requeridas: R_min = 15 km/h, R_max = 35 km/h
- Capacidades viales: C_min = 50 km/h, C_max = 90 km/h
- Número de emergencias: K = 2
- Costos operativos:
    - Leve: $100/h
    - Media: $250/h
    - Crítica: $500/h

Resultados:

- **Estado:** Factible – Solución óptima encontrada
- **Métricas globales:**
    - Costo total: **$13.70**
    - Distancia total: **2.39 km**
    - Tiempo total: **4.78 min**
    - Emergencias atendidas: **2/2 (100%)**

Detalles por emergencia:

- **Emergencia 1 (Leve):**

  Velocidad: 15.8 km/h · Distancia: 1.24 km · Tiempo: 2.48 min · Costo: $4.14
- **Emergencia 2 (Media):**

  Velocidad: 28.6 km/h · Distancia: 1.15 km · Tiempo: 2.29 min · Costo: $9.56

4.2 Escenario 2: Escenario Moderado

Descripción:

Demanda moderada y capacidades viales medias. Representa condiciones operativas típicas de una ciudad.

Parámetros:

- Área: Radio 560 m
- Velocidades requeridas: R_min = 18 km/h, R_max = 40 km/h
- Capacidades viales: C_min = 40 km/h, C_max = 80 km/h
- Número de emergencias: K = 4
- Costos operativos:
    - Leve: $100/h
    - Media: $250/h
    - Crítica: $500/h

Resultados:

- **Estado:** ✓ Factible – Solución óptima encontrada
- **Métricas globales:**
    - Costo total: **$30.52**
    - Distancia total: **3.06 km**
    - Tiempo total: **6.12 min**
    - Emergencias atendidas: **4/4 (100%)**

Detalles por emergencia:

- **Emergencia 1 (Leve):** 29.6 km/h · 0.91 km · 1.81 min · $3.02
- **Emergencia 2 (Media):** 32.3 km/h · 0.11 km · 0.21 min · $0.89
- **Emergencia 3 (Crítica):** 34.8 km/h · 1.15 km · 2.30 min · $19.13
- **Emergencia 4 (Media):** 33.2 km/h · 0.90 km · 1.79 min · $7.48

4.3 Escenario 3: Escenario Crítico

Descripción:

Alta demanda de emergencias críticas y capacidad vial limitada. Simula condiciones extremas, como congestión severa o eventos masivos.

Parámetros:

- Área: Radio 560 m
- Velocidades requeridas: R_min = 25 km/h, R_max = 50 km/h
- Capacidades viales: C_min = 40 km/h, C_max = 80 km/h
- Número de emergencias: K = 6 (3 críticas, 2 medias, 1 leve)
- Costos operativos:
    - Leve: $100/h
    - Media: $250/h
    - Crítica: $500/h

Resultados:

- **Estado:** ✓ Factible – Solución óptima encontrada
- **Métricas globales:**
    - Costo total: **$73.38**
    - Distancia total: **6.47 km**
    - Tiempo total: **12.94 min**
    - Emergencias atendidas: **6/6 (100%)**

Detalles por emergencia:

- **Emergencia 1 (Crítica):** 49.5 km/h · 1.36 km · 2.72 min · $22.65
- **Emergencia 2 (Crítica):** 45.7 km/h · 0.47 km · 0.94 min · $7.80
- **Emergencia 3 (Media):** 38.9 km/h · 0.68 km · 1.36 min · $5.65
- **Emergencia 4 (Crítica):** 45.2 km/h · 1.31 km · 2.62 min · $21.86
- **Emergencia 5 (Media):** 41.7 km/h · 1.31 km · 2.62 min · $10.93
- **Emergencia 6 (Leve):** 28.6 km/h · 1.34 km · 2.69 min · $4.48

4.4 Resumen Comparativo

Escenario	Emergencias	Estado	Costo total	Tiempo total	Distancia total
Ideal	2	Factible	13.70	4.78 min	2.39km
Moderado	4	Factible	30.52	6.12 min	3.06km
Crítico	6	Factible	73.38	12.94 min	6.47km