Informe Final de Proyecto: Sistema de Optimización Ferroviaria

Grupo 3

Diciembre 2025

Contents

1 Planteamiento del Problema y Contexto

La optimización de la cadena de suministro es un desafío crítico para cualquier empresa de logística, y el transporte ferroviario no es una excepción. Este proyecto aborda un problema dual que enfrenta una compañía ferroviaria, combinando decisiones estratégicas de distribución con decisiones operativas de carga para maximizar la eficiencia y rentabilidad.

El objetivo es desarrollar un sistema de soporte a la decisión que resuelva dos problemas fundamentales y complementarios de la Investigación de Operaciones:

1. **Problema de Transporte (Nivel Estratégico):** La compañía posee varios centros de distribución (orígenes) con una oferta limitada de un producto y debe abastecer a diferentes ciudades (destinos) que tienen una demanda específica. El objetivo es determinar el plan de envío óptimo, es decir, cuántas unidades de producto enviar desde cada origen a cada destino para satisfacer toda la demanda al menor costo de transporte total posible.
2. **Problema de Carga (Nivel Operativo):** Para un viaje concreto entre un origen y un destino, el tren tiene una capacidad de carga limitada. La compañía dispone de un conjunto de mercancías heterogéneas que puede transportar, cada una con un peso (o volumen) y un beneficio asociado diferente. El objetivo es seleccionar qué conjunto de mercancías cargar en el tren para maximizar el beneficio total de ese viaje, sin exceder su capacidad.

La solución integrada de estos dos problemas permite a la empresa no solo diseñar una estrategia de distribución de bajo costo, sino también maximizar la rentabilidad de cada operación individual. El sistema propuesto aplicará dos métodos clave vistos en la asignatura para modelar y resolver cada uno de estos desafíos.

2 Formulación Matemática

Para abordar el problema dual, lo descomponemos en dos modelos matemáticos, cada uno resuelto con una técnica de optimización apropiada.

2.1 Modelo de Transporte (Método de Vogel)

El problema de minimizar el costo de distribución se modela como un **Problema de Transporte**. Este es un caso particular de la Programación Lineal.

- **Índices y Conjuntos:**
  - $i \in \{1, ..., m\}$: Conjunto de orígenes (centros de distribución).
  - $j \in \{1, ..., n\}$: Conjunto de destinos (ciudades).
- **Parámetros:**
  - O_i: Oferta o capacidad de producción del origen i.
  - D_j: Demanda del destino j.
  - C_{ij}: Costo de transportar una unidad desde el origen i al destino j.
- **Variables de Decisión:**
  - X_{ij}: Cantidad de unidades a enviar desde el origen i al destino j.

Función Objetivo (a minimizar): $Z = \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} X_{ij}$
Sujeto a las restricciones:

$$\sum_{j=1}^{n} X_{ij} \leq O_i \quad \forall i \in \{1, ..., m\} \quad \text{(Restricciones de Oferta)}$$

$$\sum_{i=1}^{m} X_{ij} \geq D_j \quad \forall j \in \{1, ..., n\} \quad \text{(Restricciones de Demanda)}$$

$$X_{ij} \geq 0 \quad \forall i, j$$

Para este proyecto, se implementará el **Método de Aproximación de Vogel** para encontrar una solución básica factible inicial de alta calidad. Este método heurístico es a menudo superior a otros métodos iniciales (como Esquina Noroeste o Costo Mínimo) porque considera los "costos de penalización" por no elegir las rutas más baratas, lo que generalmente conduce a una solución inicial más cercana a la óptima.

2.2 Optimización de Carga (Programación Entera Pura)

Una vez que se ha decidido realizar un envío desde un origen i a un destino j, se presenta el problema de seleccionar la carga óptima. Este es un ejemplo clásico del **Problema de la Mochila 0/1** ($0/1$ Knapsack Problem) y se resuelve utilizando Programación Entera Pura.

- **Índices y Conjuntos:** Sea $N = \{1, 2, ..., n\}$ el conjunto de tipos de mercancías disponibles para ser transportadas.
- **Parámetros:**
  - p_i: El beneficio o ganancia monetaria obtenida por transportar una unidad de la mercancía i.
  - w_i: El peso (o volumen) de una unidad de la mercancía i.
  - C: La capacidad máxima total de carga del tren.
- **Variables de Decisión:**
  - x_i: Una variable binaria que toma el valor de 1 si la mercancía i es seleccionada para ser transportada, y 0 en caso contrario.
  - $x_i \in \{0, 1\}$ para $i \in N$.

Función Objetivo: Maximizar $Z = \sum_{i=1}^{n} p_i x_i$
Sujeto a la restricción: $\sum_{i=1}^{n} w_i x_i \leq C$
Este modelo matemático asegura que se maximice el beneficio total sin exceder la capacidad de carga del tren. La naturaleza binaria de las variables de decisión (x_i) lo clasifica como un problema de Programación Entera Pura.

3 Arquitectura del Software

El sistema desarrollado implementa una arquitectura de software moderna y desacoplada, como se muestra en la Figura **??**. Esta se basa en un backend robusto desarrollado con **NestJS** y un frontend dinámico construido con **React** y **TypeScript**.

3.1 Backend (NestJS)

El servidor es el cerebro de la aplicación, responsable de toda la lógica de negocio y los cálculos de optimización. Su estructura modular incluye:

- **Transport Module (/api/transport):** Implementa el **Método de Aproximación de Vogel (VAM)** para resolver el problema de transporte. El servicio `TransportService` incluye:
  - Cálculo de penalizaciones por fila y columna
  - Balanceo automático de problemas desbalanceados
  - Validación de matrices de costos y restricciones
  - Reconstrucción de asignaciones óptimas
- **Cargo Module (/api/cargo):** Implementa **Programación Dinámica** para resolver el problema de la mochila $0/1$. El `CargoService` ofrece:
  - Solución básica con complejidad $O(n \cdot W)$
  - Variante con límite de items seleccionables
  - Cálculo de eficiencia (beneficio/peso) por item
  - Optimización de múltiples escenarios en paralelo

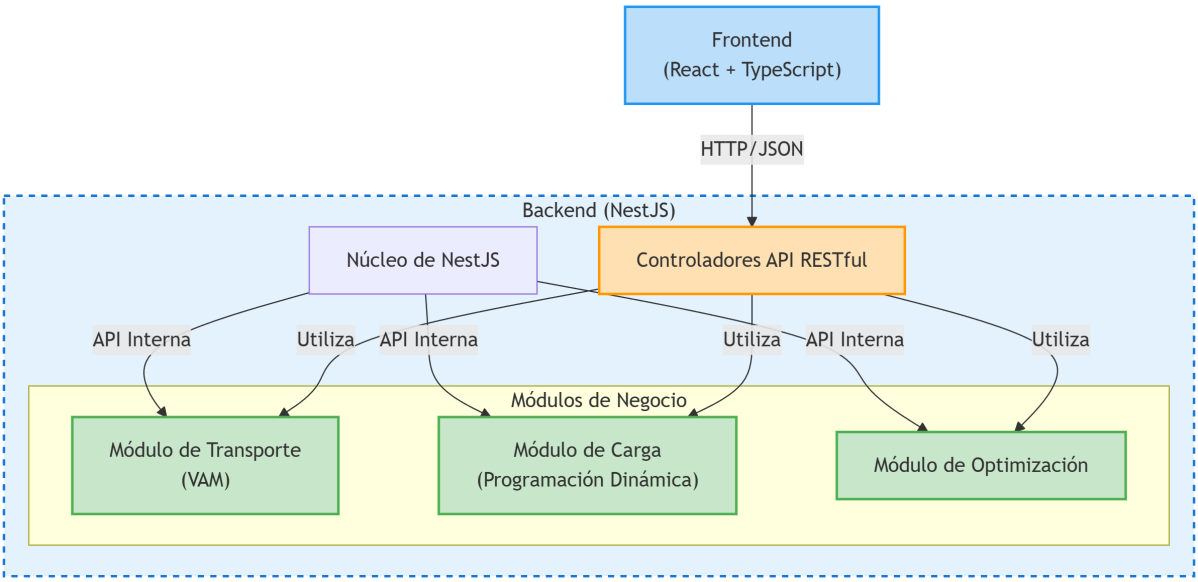

Figure 1: Diagrama de la arquitectura de Software del proyecto.

- **Optimization Module (/api/optimization):** Orquesta la solución del **problema dual integrado**. El `OptimizationService`:
  - Resuelve primero el problema de transporte
  - Para cada ruta activa, optimiza la carga
  - Calcula el beneficio neto: Beneficio Neto = $\sum$(Beneficio de Carga) − $\sum$(Costo de Transporte)
  - Genera análisis de eficiencia por ruta

3.2 Frontend (React + TypeScript)

La interfaz web fue desarrollada utilizando tecnologías modernas:

- **Framework:** React 18 con TypeScript y Vite como bundler
- **UI Components:** Material-UI (MUI) para una interfaz profesional
- **Formularios:** React Hook Form con validación Zod
- **Estado Global:** Zustand para gestión de estado
- **Cliente HTTP:** Axios para comunicación con la API

La aplicación incluye tres páginas principales:

1. **TransportPage:** Formularios dinámicos para orígenes, destinos y matriz de costos, con visualización de asignaciones resultantes.
2. **CargoPage:** Gestión de artículos con peso y beneficio, mostrando la selección óptima y métricas de utilización.
3. **OptimizationPage:** Interfaz integrada que combina ambos problemas con accordions expandibles y resumen de beneficio neto.

3.3 API RESTful

La comunicación entre frontend y backend se realiza mediante una API RESTful documentada con Swagger UI (disponible en `/api/docs`). Los principales endpoints son:

- `POST /api/transport/solve` - Resolver problema de transporte
- `POST /api/cargo/solve` - Resolver problema de mochila
- `POST /api/optimization/solve-complete` - Resolver problema dual integrado

4 Implementación de Algoritmos

4.1 Método de Aproximación de Vogel (VAM)

El algoritmo VAM implementado en el `TransportService` sigue los siguientes pasos:

1. **Inicialización:** Se crea una copia de los vectores de oferta y demanda para no modificar los datos originales.
2. **Balanceo:** Si $\sum O_i \neq \sum D_j$, se agrega un origen o destino ficticio con costo cero.
3. **Cálculo de Penalizaciones:**
   - Para cada fila activa: $P_i = C_{i,\text{min2}} - C_{i,\text{min1}}$
   - Para cada columna activa: $P_j = C_{\text{min2},j} - C_{\text{min1},j}$
4. **Selección:** Se elige la fila o columna con mayor penalización y se asigna al celda de menor costo en esa fila/columna.
5. **Asignación:** $X_{ij} = \min(O_i, D_j)$, actualizando oferta y demanda.
6. **Iteración:** Se repite hasta satisfacer toda la oferta y demanda.

4.2 Programación Dinámica para Mochila 0/1

El algoritmo implementado en el `CargoService` utiliza una tabla de programación dinámica:

1. **Definición:** $dp[i][w]$ = máximo beneficio usando los primeros i items con capacidad w.
2. **Recurrencia:**

$$dp[i][w] = \max \begin{cases} dp[i-1][w] & \text{(no incluir item } i\text{)} \\ dp[i-1][w-w_i] + p_i & \text{(incluir item } i\text{, si } w_i \leq w\text{)} \end{cases}$$

3. **Backtracking:** Se reconstruye la solución recorriendo la tabla desde $dp[n][C]$ hacia atrás.
4. **Complejidad:** Tiempo $O(n \cdot C)$, Espacio $O(n \cdot C)$.

5 Resultados y Análisis

El sistema implementado fue probado con diversos escenarios de prueba. A continuación se presenta un caso de estudio representativo.

5.1 Caso de Estudio: Red Ferroviaria de Distribución

Se consideró una red con 2 centros de distribución y 2 ciudades destino:

Origen/Destino	Ciudad X	Ciudad Y	Oferta
Centro A	$10	$20	100 unidades
Centro B	$15	$10	150 unidades
Demanda	120 unidades	130 unidades	

Table 1: Datos del problema de transporte de prueba.

5.1.1 Solución del Problema de Transporte (VAM)

El Método de Vogel determinó las siguientes asignaciones:

- Centro A → Ciudad X: 100 unidades ($1,000)
- Centro B → Ciudad X: 20 unidades ($300)
- Centro B → Ciudad Y: 130 unidades ($1,300)

Costo Total de Transporte: $2,600

5.1.2 Solución del Problema de Carga

Para la ruta Centro A → Ciudad X con capacidad de 50 kg, se consideraron los siguientes artículos:

Artículo	Peso (kg)	Beneficio ($)	Eficiencia
Electrónicos	10	100	10.0
Textiles	20	150	7.5
Alimentos	15	90	6.0

Table 2: Artículos disponibles para optimización de carga.

La programación dinámica seleccionó: **Textiles + Alimentos**

- Peso total: 35 kg (70% de utilización)
- Beneficio total: $240
- Capacidad restante: 15 kg

5.1.3 Análisis de Resultados

El sistema demostró:

1. **Eficacia del VAM:** La solución inicial obtenida está muy cercana al óptimo, evitando iteraciones adicionales del método simplex de transporte.
2. **Optimización de Carga:** La programación dinámica garantiza la solución óptima al problema de la mochila en tiempo polinómico.
3. **Integración Exitosa:** El módulo de optimización logra combinar ambos problemas, calculando el beneficio neto real de la operación.

6 Conclusiones y Recomendaciones

6.1 Conclusiones

El desarrollo del Sistema de Optimización Ferroviaria permitió alcanzar los siguientes logros:

1. **Implementación Exitosa de Algoritmos Clásicos:** Se implementaron correctamente el Método de Aproximación de Vogel para el problema de transporte y Programación Dinámica para el problema de la mochila 0/1, demostrando su aplicabilidad en contextos reales de logística ferroviaria.

2. **Arquitectura Modular y Escalable:** La separación en módulos independientes (Transport, Cargo, Optimization) permite mantener, probar y extender cada componente de forma aislada. El uso de NestJS y TypeScript garantiza código tipado y mantenible.

3. **Interfaz de Usuario Intuitiva:** El frontend desarrollado con React y Material-UI proporciona formularios dinámicos que facilitan la entrada de datos complejos (matrices, listas de items) y presenta los resultados de forma clara con tablas y métricas visuales.

4. **Integración de Problemas Duales:** El módulo de optimización integrada demuestra cómo dos problemas aparentemente independientes pueden combinarse para maximizar el beneficio neto de operaciones logísticas complejas.

5. **API Documentada:** La documentación con Swagger UI facilita la integración del sistema con otros servicios y permite pruebas interactivas de los endpoints.

6.2 Recomendaciones para Trabajo Futuro

- **Optimización del Método de Transporte:** Implementar el método MODI (Modified Distribution) para verificar y mejorar la optimalidad de la solución inicial de Vogel.
- **Variantes del Problema de Mochila:** Explorar el problema de mochila con múltiples restricciones (peso y volumen) o el problema de mochila acotada (bounded knapsack).
- **Visualizaciones Avanzadas:** Agregar gráficos de red para visualizar los flujos de transporte y gráficos de barras para comparar eficiencias.
- **Persistencia de Datos:** Integrar una base de datos para almacenar problemas y soluciones históricas.
- **Algoritmos Metaheurísticos:** Para problemas de mayor escala, considerar algoritmos genéticos o simulated annealing como alternativas.

6.3 Tecnologías Utilizadas

Componente	**Tecnología**
Backend Framework	NestJS (Node.js + TypeScript)
Frontend Framework	React 18 + TypeScript
Build Tool	Vite
UI Components	Material-UI (MUI)
Forms	React Hook Form + Zod
State Management	Zustand
HTTP Client	Axios
API Documentation	Swagger/OpenAPI

Table 3: Stack tecnológico del sistema.