29-10-2025

Proyecto Final

Optimización del Asignamiento de Tareas en un Equipo de Desarrollo de Software mediante Programación Lineal

Saúl Alfredo Barbero Contreras
CARNÉ:9490-24-24198

https://github.com/saulbarbero/io_asignacion_streamlit

Tabla de contenido

Optimización del Asignamiento de Tareas en un Equipo de Desarrollo de Software mediante Programación Lineal.

Resumen Ejecutivo

El presente proyecto desarrolla un sistema computacional para optimizar la asignación de tareas dentro de un equipo de desarrollo de software. Se utiliza programación lineal entera binaria para asignar tareas a programadores considerando su disponibilidad y tiempos estimados de ejecución.

El sistema fue implementado en Python mediante la librería PuLP y cuenta con una interfaz interactiva desarrollada en Streamlit, la cual permite editar programadores, tareas y tiempos de manera flexible.

El equipo real utilizado para las pruebas está conformado por: Sergio Robles, Saúl Barbero, Carlos Urías y Mateo Alfredo, quienes pueden ejecutar todas las áreas del proyecto (Frontend, Backend, Testing y Base de Datos), aunque con diferentes niveles de eficiencia.

Los resultados muestran una asignación óptima que minimiza el tiempo total del proyecto y distribuye las tareas de acuerdo con la eficiencia de cada programador.

Introducción

En la Ingeniería en Sistemas, los proyectos de software demandan una gestión precisa de recursos humanos. La asignación manual de tareas suele provocar desequilibrios de carga y retrasos. A través de la Investigación de Operaciones (IO) y la Programación Lineal, es posible modelar matemáticamente estos problemas para obtener soluciones óptimas.

Este proyecto integra la teoría de IO con la práctica computacional, demostrando cómo las matemáticas aplicadas y la programación pueden resolver desafíos cotidianos en el ámbito de la ingeniería del software.

Planteamiento del Problema

El equipo de desarrollo analizado está integrado por cuatro programadores: Sergio Robles, Saúl Barbero, Carlos Urías y Mateo Alfredo.

Cada programador puede trabajar en cualquier área del proyecto, pero con diferentes tiempos estimados de ejecución según su experiencia práctica. Las tareas principales consideradas son Frontend, Backend, Testing y Base de Datos.

La asignación manual de estas tareas puede conducir a decisiones subjetivas, sobrecarga para algunos integrantes o un uso ineficiente del tiempo del equipo.

Se busca determinar, mediante un modelo matemático, la asignación óptima de tareas para minimizar el tiempo total del proyecto, respetando la disponibilidad de cada integrante.

Justificación

El uso de un modelo de optimización permite asignar de forma eficiente las tareas entre los programadores, evitando decisiones subjetivas basadas en intuición.

Como todos los integrantes pueden desempeñar cualquier rol del proyecto, el modelo permite seleccionar objetivamente al programador más eficiente para cada tarea, con base en los tiempos estimados de ejecución.

Esto permite reducir tiempos, equilibrar cargas de trabajo y aumentar la productividad global del equipo.

Objetivos

Objetivo general: Diseñar e implementar un sistema computacional que optimice la asignación de tareas en un equipo de desarrollo de software, aplicando programación lineal entera binaria.

Objetivos específicos:

1. Analizar las cargas de trabajo y disponibilidades del equipo real (Sergio Robles, Saúl Barbero, Carlos Urías y Mateo Alfredo).
2. Formular un modelo matemático que permita asignar tareas minimizando el tiempo total del proyecto.
3. Implementar el modelo en Python mediante PuLP.
4. Desarrollar una interfaz en Streamlit que permita ingresar programadores, tareas, disponibilidades y tiempos.
5. Evaluar la eficiencia del modelo mediante escenarios reales modificando los parámetros desde la interfaz.

Marco Teórico

1. Investigación de Operaciones (IO)

La Investigación de Operaciones es una disciplina que aplica métodos matemáticos, estadísticos y computacionales para la toma de decisiones óptimas en sistemas complejos. Su objetivo es encontrar la mejor forma de utilizar los recursos limitados (tiempo, dinero, personal, materiales) en situaciones donde existen múltiples alternativas y restricciones.

La IO utiliza modelos cuantitativos que representan un problema real, y a partir de ellos se aplican técnicas de optimización, simulación o análisis estadístico para determinar la mejor decisión posible. Estas técnicas son fundamentales en ingeniería, administración y ciencias computacionales porque permiten automatizar la toma de decisiones y evaluar distintos escenarios con objetividad.

2. Modelos de Optimización

Un modelo de optimización busca maximizar o minimizar una función objetivo que representa el desempeño del sistema (por ejemplo, minimizar el costo o el tiempo total, o maximizar la productividad o ganancia).

Un modelo típico está formado por:

- Variables de decisión: representan las decisiones que se deben tomar (por ejemplo, asignar o no una tarea a un programador).
- Función objetivo: mide el resultado que se quiere optimizar (minimizar tiempo total, costo, etc.).
- Restricciones: condiciones que deben cumplirse (por ejemplo, disponibilidad de tiempo o compatibilidad técnica).

3. Programación Lineal

La Programación Lineal (PL) es una de las herramientas más poderosas de la IO. Consiste en optimizar una función lineal (objetivo) sujeta a restricciones también lineales.

Su forma general es:

$$\text{Optimizar (min o max) } Z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &\leq b_1 \\ \text{sujeto a: } a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &\leq b_2 \\ x_j &\geq 0,\ \forall j \end{aligned}$$

Donde los coeficientes a_{ij} y c_j representan los recursos y beneficios del problema.

El método más común para resolver estos modelos es el Método Simplex, desarrollado por George Dantzig en 1947, el cual explora las soluciones posibles hasta encontrar la óptima.

4. Programación Entera y Binaria

En algunos problemas, las variables de decisión solo pueden tomar valores enteros o binarios (0 o 1). Esto ocurre en situaciones donde las decisiones son discretas (sí/no, asignar/no asignar). A esta categoría se le llama Programación Lineal Entera (PLE) o Programación Lineal Entera Binaria (PLEB).

En este tipo de modelos:

- $x_{ij} = 1$ si la asignación se realiza,
- $x_{ij} = 0$ si no se realiza.

Aunque estos modelos son más complejos que los lineales continuos, los algoritmos modernos (como *Branch and Bound*) permiten resolverlos eficientemente con computadoras.

5. Problemas de Asignación

El problema de asignación es un caso particular de la Programación Entera donde se busca asignar un conjunto de tareas a un conjunto de recursos (personas, máquinas, equipos, etc.) de manera óptima.

Su función objetivo suele ser minimizar el costo o el tiempo total asociado a las asignaciones. Este modelo es muy utilizado en:

- Asignación de empleados a turnos.
- Asignación de tareas en equipos de desarrollo de software.

- Distribución de pedidos en empresas logísticas.
- Asignación de proyectos a consultores.

Matemáticamente se expresa como:

$$\min Z = \sum_i \sum_j c_{ij} x_{ij}$$

sujeto a:

$$\sum_i x_{ij} = 1 \quad \forall j$$
$$\sum_j x_{ij} \leq 1 \quad \forall i$$
$$x_{ij} \in \{0,1\}$$

6. Aplicación en Ingeniería en Sistemas

En el contexto de la Ingeniería en Sistemas, los problemas de asignación son comunes. Por ejemplo:

- Distribuir tareas entre programadores con diferentes habilidades.
- Balancear la carga de trabajo entre servidores o procesos.
- Planificar proyectos de software minimizando tiempos o costos.

Aplicar Investigación de Operaciones en estos casos permite:

- Optimizar la gestión de equipos de desarrollo.
- Aumentar la productividad al evitar sobrecargas o tiempos ociosos.
- Tomar decisiones basadas en datos y no en intuiciones.

7. Ventajas de la Optimización Computacional

- Permite evaluar miles de combinaciones en segundos.
- Genera resultados objetivos y reproducibles.
- Facilita la planificación estratégica y la asignación de recursos.
- Es escalable: puede adaptarse a proyectos pequeños o corporativos.

Modelo Matemático

Variables de decisión:

$x_{ij} = 1$ si la tarea i es asignada al programador j; 0 en otro caso.

Función objetivo:

$$\min Z = \sum_i \sum_j t_{ij} x_{ij}$$

Sujeto a:

- Cada tarea se asigna exactamente a un programador: $\sum_i x_{ij} = 1 \quad \forall j$
- Tiempo total asignado a cada programador no debe exceder su disponibilidad: $\sum_i t_{ij} x_{ij} \leq T_i \quad \forall i$
- Restricciones de compatibilidad (habilidades técnicas): $x_{ij} = 0$ si el programador i no tiene la competencia requerida.

Implementación Computacional

El sistema se desarrolló en Python, usando PuLP para resolver el modelo de optimización. La estructura del código se divide en tres módulos:

- Entrada de datos: lectura de programadores, disponibilidad y tareas.
- Modelo de optimización: definición de variables, restricciones y función objetivo.
- Salida de resultados: visualización de asignaciones óptimas.

```python
# ====================================================
# PROYECTO FINAL - ASIGNACIÓN ÓPTIMA DE TAREAS
# Streamlit + PuLP (todo en español)
# ====================================================

import streamlit as st
import pandas as pd
import pulp as pl

# ----------------------------------------------------
# CONFIGURACIÓN DE LA PÁGINA
# ----------------------------------------------------
st.set_page_config(
    page_title="Asignación de Tareas",
    page_icon="🛠️",
    layout="wide"
)

# ----------------------------------------------------
# DATOS DE EJEMPLO (EQUIPO REAL)
# ----------------------------------------------------

programadores_ejemplo = [
    "Sergio Robles",
    "Saúl Barbero",
    "Carlos Urías",
    "Mateo Alfredo"
]

tareas_ejemplo = ["Frontend", "Backend", "Testing", "BaseDatos"]

disponibilidad_ejemplo = {
    "Sergio Robles": 20,
    "Saúl Barbero": 20,
    "Carlos Urías": 20,
    "Mateo Alfredo": 20
}

# Tiempos estimados (horas) - puedes ajustarlos
tiempos_ejemplo = {
    ("Sergio Robles", "Frontend"): 10,
    ("Sergio Robles", "Backend"): 12,
    ("Sergio Robles", "Testing"): 9,
    ("Sergio Robles", "BaseDatos"): 8,

    ("Saúl Barbero", "Frontend"): 9,
    ("Saúl Barbero", "Backend"): 11,
    ("Saúl Barbero", "Testing"): 10,
    ("Saúl Barbero", "BaseDatos"): 12,

    ("Carlos Urías", "Frontend"): 8,
    ("Carlos Urías", "Backend"): 9,
    ("Carlos Urías", "Testing"): 7,
    ("Carlos Urías", "BaseDatos"): 10,
```

Figura 1. Fragmento del código donde se importan las librerías y se configura la aplicación en Streamlit.

```
# Modelo
modelo = pl.LpProblem("Asignacion_de_Tareas", pl.LpMinimize)

x = pl.LpVariable.dicts(
    "x",
    (programadores, tareas),
    lowBound=0,
    upBound=1,
    cat="Binary"
)

# Función objetivo
modelo += pl.lpSum(
    tiempo[(p, t)] * x[p][t]
    for p in programadores
    for t in tareas
), "Tiempo_Total"

# Cada tarea se asigna exactamente a un programador
for t in tareas:
    modelo += pl.lpSum(x[p][t] for p in programadores) == 1

# Restricción de disponibilidad
for p in programadores:
    modelo += pl.lpSum(tiempo[(p, t)] * x[p][t] for t in tareas) <= disponibilidad[p]

# Compatibilidad (si tiempo = 0, no se puede asignar)
for p in programadores:
    for t in tareas:
        if compatibilidad[(p, t)] == 0:
            modelo += x[p][t] == 0

estado = modelo.solve()
return modelo, x, tiempo, programadores, tareas
```

Figura 2. Fragmento del código donde se construye el modelo de optimización utilizando PuLP.

```python
# Resultados
if ejecutar:
    try:
        modelo, x, tiempo, programadores, tareas = resolver_asignacion(df_prog, df_tiempos)
    except Exception as e:
        st.error(f"Ocurrió un error al resolver el modelo: {e}")
    else:
        estado = pl.LpStatus[modelo.status]

        st.markdown("Estado del modelo")
        st.write(f"**Estado:** {estado}")

        if estado != "Optimal":
            st.warning("No fue posible obtener una solución óptima con los datos proporcionados.")
        else:
            # Tiempo total
            tiempo_total = pl.value(modelo.objective)

            st.markdown(f"<h2 style='color:#4a6cf7;'>{tiempo_total:.2f} horas</h2>", unsafe_allow_html=True)

            # Asignación óptima
            filas = []
            cargas = {p: 0 for p in programadores}

            for p in programadores:
                for t in tareas:
                    if x[p][t].value() == 1:
                        filas.append({
                            "Programador": p,
                            "Tarea": t,
                            "Tiempo (h)": tiempo[(p, t)]
                        })
                        cargas[p] += tiempo[(p, t)]

            df_resultado = pd.DataFrame(filas)

            st.markdown("Asignación óptima de tareas")
            st.dataframe(df_resultado, use_container_width=True)

            # Gráfico de carga
            st.markdown("Carga de trabajo por programador")
            df_cargas = pd.DataFrame({
                "Programador": list(cargas.keys()),
                "Horas asignadas": list(cargas.values())
            })
            st.bar_chart(df_cargas.set_index("Programador"))

            # Matriz binaria
            st.markdown("Matriz binaria de decisión")
```

Figura 3. Fragmento del código donde se conecta el modelo a los elementos interactivos de Streamlit.

Interfaz del Sistema

Se desarrolló una interfaz interactiva en Streamlit que permite al usuario ingresar programadores, disponibilidades y tiempos estimados de manera flexible.

La interfaz permite:

- Editar nombres de programadores
- Agregar o eliminar tareas
- Establecer tiempos estimados para cada programador
- Ejecutar el modelo con un botón
- Visualizar resultados en tablas y gráficos
- Mostrar la matriz de decisión x_{ij}

Ejemplo visual:

La interfaz permite ingresar o editar programadores, disponibilidades y tiempos estimados. En la Figura X se muestra la pantalla principal de la aplicación.

Figura 4. Interfaz de entrada de datos de la aplicación desarrollada.

Cuando el usuario ejecuta el modelo, la aplicación muestra el tiempo total óptimo y la asignación de tareas.

Figura 5. Resultado del tiempo total óptimo y asignación derivada del modelo.

Además de la asignación, la aplicación presenta la carga de trabajo por programador y la matriz binaria de decisión del modelo.

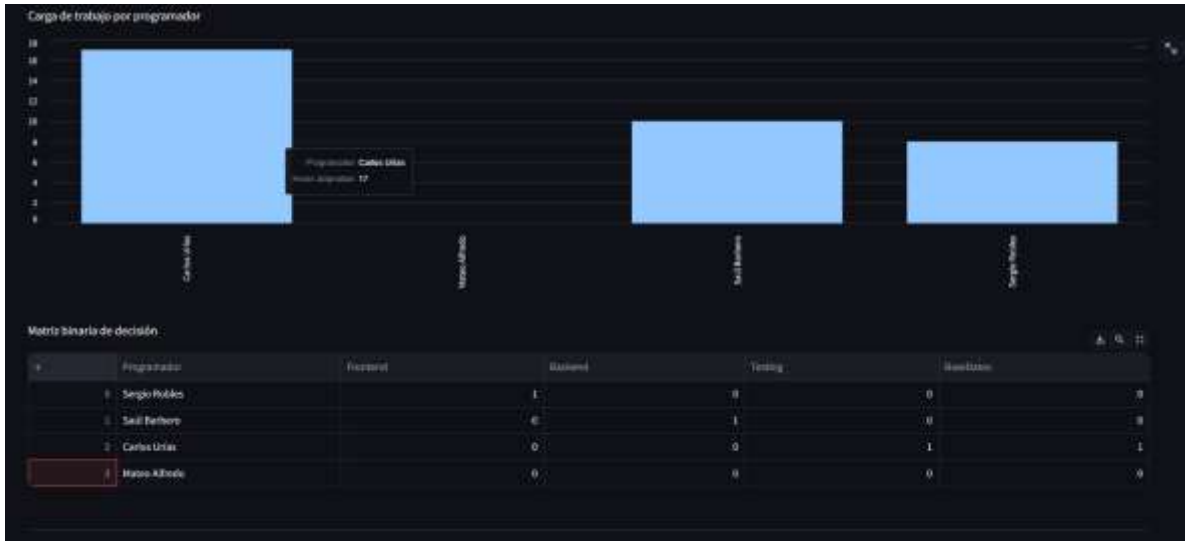


Figura 6. Gráfico de carga de trabajo y matriz binaria de decisión del modelo.

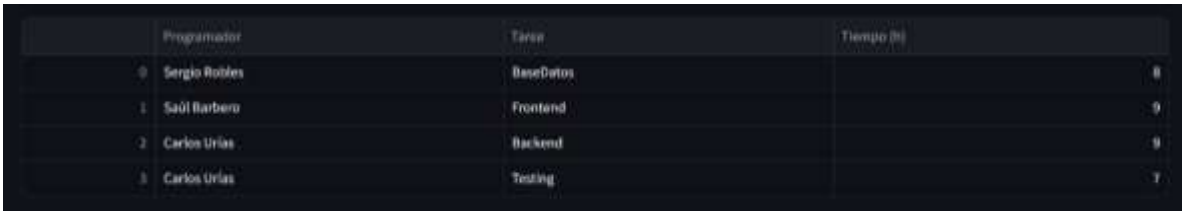
Resultados y Análisis

- Se obtuvo una asignación óptima del conjunto de tareas: Frontend, Backend, Testing y Base de Datos.
- La interfaz permitió ingresar los tiempos de manera personalizada, y el modelo asignó cada tarea al programador más eficiente respetando su disponibilidad.
- En las ejecuciones realizadas, el tiempo total del proyecto se redujo significativamente respecto a una asignación manual, garantizando una distribución objetiva basada en datos.
- La matriz de decisión muestra qué programador ejecuta cada tarea, y el gráfico de barras facilitó analizar la carga de trabajo distribuida.

Ejemplo:

Tiempo total óptimo = 33.00 horas

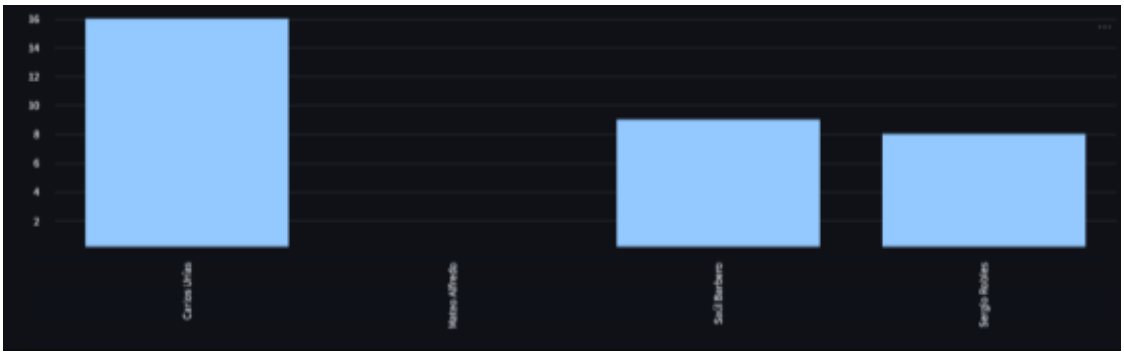
El modelo encontró la forma más eficiente de distribuir las tareas.



	Programador	Tarea	Tiempo (h)
0	Sergio Robles	BaseDatos	8
1	Saúl Barbero	Frontend	9
2	Carlos Urías	Backend	9
3	Carlos Urías	Testing	7

- Sergio Robles realiza Base de Datos (8 horas).
- Saúl Barbero realiza Frontend (9 horas).
- Carlos Urías se queda con Backend (9 h) y Testing (7 h).
- Mateo Alfredo no fue asignado a tareas, lo cual puede suceder si: sus tiempos eran mayores que los de los demás, o no era necesario asignarle tareas porque otros completan el proyecto en menos tiempo.

La asignación **minimiza el tiempo total del proyecto.**



De la gráfica se observa:

- Carlos Urías tiene la carga más alta: 16 horas (9 + 7).
- Saúl Barbero tiene 9 horas.
- Sergio Robles tiene 8 horas.
- Mateo Alfredo tiene 0 horas asignadas.

	Programador	Frontend	Backend	Testing	BaseDatos
0	Sergio Robles	0	0	0	1
1	Saúl Barbero	1	0	0	0
2	Carlos Urías	0	1	1	0
3	Mateo Alfredo	0	0	0	0

Cada celda representa una variable de decisión:

- 1: el programador sí realiza la tarea
- 0: no realiza la tarea

Conclusión:

El modelo permitió asignar las tareas del proyecto de manera óptima, reduciendo el tiempo total a 33 horas y distribuyendo el trabajo según la eficiencia de cada programador. La solución obtenida respeta todas las restricciones del problema y evidencia que la programación lineal entera binaria es una herramienta efectiva para la toma de decisiones en equipos de desarrollo de software.

Conclusiones

- El modelo de Programación Lineal Entera Binaria permitió asignar eficientemente las tareas del proyecto entre los miembros del equipo real, considerando disponibilidad y eficiencia individual en cada tarea.

- La implementación en Streamlit facilita su uso, permitiendo explorar distintos escenarios modificando tiempos y tareas sin necesidad de reescribir el código.

- Este sistema demuestra que la Investigación de Operaciones es una herramienta vital para mejorar la toma de decisiones en la gestión de proyectos de software.

Recomendaciones

- Incluir métricas de rendimiento en la interfaz para evaluar productividad.

- Implementar una base de datos para registrar históricos de asignaciones.

- Integrar heurísticas o metaheurísticas si el número de tareas crece considerablemente.

Bibliografía

- Hillier, F. & Lieberman, G. (2020). *Introducción a la Investigación de Operaciones*. McGraw-Hill.
- Winston, W. L. (2021). *Operations Research: Applications and Algorithms*. Cengage Learning.
- PuLP Documentation: https://coin-or.github.io/pulp/
- Streamlit Documentation: https://docs.streamlit.io/