MOI-1: Modelo Matemático Planeación

Instituto Tecnológico de Estudios Superiores de Monterrey

Desarrollo de proyectos de ingeniería matemática (Gpo 502)
Optimización en la Industria

Damián Jacob Albino Mejía A01246716

Profesor: Rafael Muñoz

Monterrey, Nuevo León. 21 de junio de 2024

1. Descripción del Problema

Problema

Un problema común dentro de las empresas es la planeación de sus operaciones para satisfacer la demanda de un periodo de tiempo determinado mientras se maximizan las utilidades, esto se logra al determinar los niveles de capacidad, producción, subcontratación e inventario. Es así que en el presente documento se busca desarrollar un modelo general para la planeación agregada tomando en cuenta la producción en plantas, compras a proveedores, venta a cliente, inventario en plantas y periodos, costos de producción, demanda en clientes por periodos, cobertura a clientes y diferentes tipos de productos.

Los objetivos actuales son minimizar los costos operativos y maximizar las ventas, mientras se mantiene un nivel óptimo de inventario en las plantas. Adicionalmente, se busca mejorar la logística de transporte para enfrentar las crecientes complejidades del mercado global y las variaciones en la demanda.

Objetivos

1. Maximizar la rentabilidad teniendo en cuenta los costos de producción, compra, almacenamiento, transporte y las ventas.
2. Maximizar la satisfacción de las demandas de los clientes.

Restricciones

- Capacidad de producción limitada en cada planta.
- Contratos de compra mínima con proveedores.
- Demanda variable por mercado y producto.
- Requerimientos de nivel de inventario para evitar desabastecimiento.
- Capacidad de transporte limitada entre plantas y mercados.
- Costos de transporte que deben ser minimizados.

2. Aplicación del Modelo Matemático

Conjuntos

- I: Conjunto de plantas de manufactura.
- J: Conjunto de proveedores.
- K: Conjunto de clientes.

- T: Conjunto de periodos de planeación (por ejemplo, mensual).
- P: Conjunto de tipos de productos (smartphones, tablets, laptops).

Variables

- x_{ipt}: Cantidad de producto p producido en la planta i en el periodo t.
- y_{jipt}: Cantidad de producto p comprado al proveedor j y transportado directamente a la planta i en el periodo t.
- z_{ikpt}: Cantidad de producto p vendido y transportado desde la planta i al cliente k en el periodo t.
- s_{ipt}: Inventario de producto p en la planta i al final del periodo t.

Parámetros

- c_{ip}: Costo de producción de un tipo de producto p en la planta i.
- d_{jipt}: Costo neto de compra del producto p del proveedor j a la planta i en el periodo t (costo más envío).
- h_{ip}: Costo de almacenamiento de un tipo de producto p en la planta i.
- e_{ikpt}: Ingreso neto por la venta de un tipo de producto p desde la planta i al cliente k en el periodo t (ingreso menos envío).
- q_{ip}: Cantidad de tiempo requerido para producir el producto p en la planta i.
- Q_{it}: Cantidad máxima de tiempo de producción en la planta i en el período t.
- Y: Presupuesto máximo de compra a proveedores de todo el ejercicio.
- D_{kpt}: Demanda mínima del cliente k para el producto p durante el período t.

Funciones Objetivo

Maximizar rentabilidad:

$$\begin{aligned}\max Z_1 = &\sum_{k\in K}\sum_{i\in I}\sum_{p\in P}\sum_{t\in T} e_{ikpt} z_{ikpt} - \sum_{i\in I}\sum_{p\in P}\sum_{t\in T}(c_{ip}x_{ipt} + h_{ip}s_{ipt}) \\ &- \sum_{j\in J}\sum_{i\in I}\sum_{p\in P}\sum_{t\in T} d_{jipt} y_{jipt}\end{aligned}$$

Maximizar demanda satisfecha:

$$\max Z_2 = \sum_{k\in K}\sum_{i\in I}\sum_{p\in P}\sum_{t\in T} z_{ikpt}$$

Restricciones

$$\begin{aligned}
\text{Balance de inventario:} &\quad s_{ipt} = s_{i,p,t-1} + x_{ipt} + \sum_{j\in J} y_{jipt} - \sum_{k\in K} z_{ikpt} \quad \forall i \in I, p \in P, t \in T \\
\text{Capacidad de producción:} &\quad \sum_{p\in P} q_{ip} x_{ipt} \le Q_{it} \quad \forall i \in I, t \in T \\
\text{Demanda de clientes:} &\quad \sum_{i\in I} z_{ikpt} \le D_{kpt} \quad \forall k \in K, p \in P, t \in T \\
\text{Oferta de plantas:} &\quad z_{ikpt} \le s_{i,p,t-1} + x_{ipt} + \sum_{j\in J} y_{jipt} \quad \forall i \in I, k \in K, p \in P, t \in T \\
\text{Presupuesto de compra:} &\quad \sum_{i\in I}\sum_{j\in J}\sum_{p\in P}\sum_{t\in T} d_{jipt} y_{jipt} \le Y
\end{aligned}$$

3. Implementación del Modelo

Para la implementación del modelo se utilizó Python como lenguaje de programación en conjunto con la herramienta de OR-Tools desarrollada por Google para la resolución de problemas de optimización. A continuación se muestra el código realizado:

```python
import os
import sys
import pandas as pd
from ortools.linear_solver import pywraplp

os.chdir(os.getcwd())
```

Listing 1: Importación de librerías utilizadas.

```python
def load_data(file_path):
    """ Carga los datos desde un archivo Excel especificado. """

    sheets = ["Plants", "Suppliers", "Clients", "Periods", "Products",
              "Production_costs", "Purchase_costs", "Storage_costs",
              "Sales_revenue", "Production_time", "Production_capacity",
              "Client_demand", "Initial_inventory", "Max_purchase_budget"]

    data = {sheet: pd.read_excel(file_path, sheet_name=sheet) for sheet in sheets}
    return data
```

Listing 2: Función para cargar los datos a utilizar (parámetros del modelo) desde un archivo de Excel.

```python
def create_dict_from_df(df, index_cols, value_col):
    """ Crea un diccionario a partir de columnas especificadas en un DataFrame. """
    return dict(zip(df[index_cols].apply(tuple, axis=1), df[value_col]))
```

Listing 3: Función para transformar las tablas en el formato que requiere el código

```python
def initialize_solver():
    """ Inicializa y devuelve un objeto Solver de OR-Tools. """
    return pywraplp.Solver.CreateSolver('SCIP')
```

Listing 4: Función para inicializar el solver

```python
def build_model(data, solver):
    # Definición de conjuntos y parámetros
    I = data['Plants']['Plant'].unique().tolist()
    J = data['Suppliers']['Supplier'].unique().tolist()
    K = data['Clients']['Client'].unique().tolist()
    T = ['Periodo 0'] + data['Periods']['Period'].unique().tolist()  # Añadimos Periodo 0
    P = data['Products']['Product'].unique().tolist()

    # Diccionarios de costos y otros parámetros
    c = create_dict_from_df(data['Production_costs'], ['Plant', 'Product'], 'Cost')
    d = create_dict_from_df(data['Purchase_costs'], ['Supplier', 'Plant', 'Product', 'Period'], 'Cost')
    h = create_dict_from_df(data['Storage_costs'], ['Plant', 'Product'], 'Cost')
    e = create_dict_from_df(data['Sales_revenue'], ['Plant', 'Client', 'Product', 'Period'], 'Revenue')
    q = create_dict_from_df(data['Production_time'], ['Plant', 'Product'], 'Time')
    Q = create_dict_from_df(data['Production_capacity'], ['Plant', 'Period'], 'Capacity')
    Y = float(data['Max_purchase_budget']['Y'].values[0])
    D = create_dict_from_df(data['Client_demand'], ['Client', 'Product', 'Period'], 'Demand')
    initial_inventory = create_dict_from_df(data['Initial_inventory'], ['Plant', 'Product'], 'Inventory')

    # Variables de decisión
    x, y, z, s = {}, {}, {}, {}
    for i in I:
        for p in P:
             # Obtener el valor del inventario inicial para la planta i y el producto p
            initial_inv = initial_inventory.get((i, p), 0)  # Usamos .get() para manejar casos sin inventario inicial
            # Fijar el inventario inicial en el Periodo 0 a ese valor específico
            s[(i, p, T[0])] = solver.IntVar(initial_inv, initial_inv, f's_{i}_{p}_{T[0]}')
            for t in T[1:]:  # Empezamos en 1 para saltar 'Periodo 0' para otras variables
                x[(i, p, t)] = solver.IntVar(0, solver.infinity(), f'x_{i}_{p}_{t}')
                s[(i, p, t)] = solver.IntVar(0, solver.infinity(), f's_{i}_{p}_{t}')
```

```
                for j in J:
                    y[(j, i, p, t)] = solver.IntVar(0, solver.infinity(), f'y_{j}_{i}_{p}_{t}')
                for k in K:
                    z[(i, k, p, t)] = solver.IntVar(0, solver.infinity(), f'z_{i}_{k}_{p}_{t}')

    # Integración de las restricciones:
    # Restricciones de balance de inventario
    for i in I:
        for p in P:
            for t_idx, t in enumerate(T[1:], start=1):  # Comienza en el primer periodo real
                # Balance de inventario en periodos subsiguientes
                constraint = solver.Constraint(0, 0)
                constraint.SetCoefficient(s[(i, p, t)], 1)
                constraint.SetCoefficient(s[(i, p, T[t_idx-1])], -1)
                constraint.SetCoefficient(x[(i, p, t)], -1)
                for j in J:
                    constraint.SetCoefficient(y[(j, i, p, t)], -1)
                for k in K:
                    constraint.SetCoefficient(z[(i, k, p, t)], 1)

    # Restricciones de capacidad de producción
    for i in I:
        for t in T[1:]:  # Comenzamos en 1 para saltar 'Periodo 0'
            constraint = solver.Constraint(0, Q[(i, t)])
            for p in P:
                constraint.SetCoefficient(x[(i, p, t)], q[(i, p)])

    # Restricciones de demanda de clientes
    for k in K:
        for p in P:
            for t in T[1:]:  # Saltamos 'Periodo 0'
                constraint = solver.Constraint(0, D[(k, p, t)])
                for i in I:
                    constraint.SetCoefficient(z[(i, k, p, t)], 1)

    # Restricciones de capacidad de oferta de plantas
    for i in I:
        for k in K:
            for p in P:
                for t_idx, t in enumerate(T[1:], start=1):  # Empezamos desde 'Periodo 1'
                    constraint = solver.Constraint(0, solver.infinity())
                    # Usamos T[t_idx-1] que correctamente se refiere al período anterior sin salirnos del rango
                    constraint.SetCoefficient(s[(i, p, T[t_idx-1])], 1)
                    constraint.SetCoefficient(x[(i, p, t)], 1)
                    for j in J:
                        constraint.SetCoefficient(y[(j, i, p, t)], 1)
                    constraint.SetCoefficient(z[(i, k, p, t)], -1)

    # Restricciones de compra máxima
    constraint = solver.Constraint(0, Y)
    for i in I:
        for j in J:
            for p in P:
                for t in T[1:]:  # Excluimos el 'Periodo 0' para compras
                    constraint.SetCoefficient(y[(j, i, p, t)], d[(j, i, p, t)])

    # Función objetivo: Maximizar rentabilidad
    objective = solver.Objective()
    for k in K:
        for i in I:
            for p in P:
                for t in T[1:]:  # Excluimos el 'Periodo 0'
                    objective.SetCoefficient(z[(i, k, p, t)], e[(i, k, p, t)])  # Ingreso por ventas
                    objective.SetCoefficient(x[(i, p, t)], -c[(i, p)])          # Costo de producción
                    objective.SetCoefficient(s[(i, p, t)], -h[(i, p)])          # Costo de almacenamiento
                    for j in J:
                        objective.SetCoefficient(y[(j, i, p, t)], -d[(j, i, p, t)])  # Costo de compra y transporte
```

```
    objective.SetMaximization()

    solver.Solve()
    first_objective_value = solver.Objective().Value()

    for i in I:
        for p in P:
            for t in T[1:]:
                x[(i, p, t)]
                for j in J:
                    y[(j, i, p, t)]
                for k in K:
                    z[(i, k, p, t)]
```

Listing 5: Implementación del modelo matemático

Hasta este punto se presentó la implementación para la resolución del modelo con una sola función objetivo, es así que para proporcionar la solución multi-objetivo se utilizó el método lexicográfico. Se indica como prioritaria la función que indica la rentabilidad, optimizando para esta y se incluye el resultado para esta función como una restricción, en este caso al construir la restricción se indica que la primera función objetivo puede tomar como mínimo el 80 % del resultado obtenido tras maximizar esta.

$$\text{máx } Z_2 = \sum_{k \in K} \sum_{i \in I} \sum_{p \in P} \sum_{t \in T} z_{ikpt}$$

$$s.t. \quad Z_1 \geq 0{,}8\, Z_1^*$$

Donde Z_1 es la función objetivo 1 y Z_1^* es el valor de la función obtenido tras maximizar la función anteriormente. Esto se puede observar en la siguiente porción de código.

```

    # Primer función objetivo como restricción
    constraint = solver.Constraint(0.8* first_objective_value, solver.infinity())
    for k in K:
        for i in I:
            for p in P:
                for t in T[1:]:  # Excluimos el 'Periodo 0'
                    constraint.SetCoefficient(z[(i, k, p, t)], e[(i, k, p, t)])  # Ingreso
    por ventas
                    constraint.SetCoefficient(x[(i, p, t)], -c[(i, p)])          # Costo de
    producción
                    constraint.SetCoefficient(s[(i, p, t)], -h[(i, p)])          # Costo de
    almacenamiento
                    for j in J:
                        constraint.SetCoefficient(y[(j, i, p, t)], -d[(j, i, p, t)])  # Costo
     de compra y transporte

    # Función objetivo: Maximizar demanda satisfecha
    objective.Clear()
    for k in K:
        for i in I:
            for p in P:
                for t in T[1:]:
                    objective.SetCoefficient(z[(i, k, p, t)], 1)  # Producto vendido
    objective.SetMaximization()

    # Devolver el solver y las variables y conjuntos necesarios
    return solver, I, J, K, P, T, x, y, z, s, c, d, h, e
```

Listing 6: Solución para la segunda función objetivo.

```
def solve_model(solver, I, J, K, P, T, x, y, z, s, c, d, h, e):
    """ Resuelve el modelo y imprime resultados óptimos incluyendo el valor de la función
    objetivo. """
    status = solver.Solve()
    if status == pywraplp.Solver.OPTIMAL:
        # Evaluar el valor de la primera función objetivo con la solución actual
```

```
        first_objective_value = sum(e[(i, k, p, t)] * z[(i, k, p, t)].solution_value() for k in K for i in I for p in P for t in T[1:]) - sum(c[(i, p)] * x[(i, p, t)].solution_value() + h[(i, p)] * s[(i, p, t)].solution_value() for i in I for p in P for t in T[1:]) - sum(d[(j, i, p, t)] * y[(j, i, p, t)].solution_value() for j in J for i in I for p in P for t in T[1:])
        print(f'Valor de la primera función objetivo con la solución de la segunda optimización: {first_objective_value}')

        # Valor actual de la función objetivo (segunda optimización)
        print(f'Valor de la función objetivo (segunda optimización): {solver.Objective().Value()}')
        # Imprimir los valores de las variables de decisión que no son cero
        for i in I:
            for p in P:
                for t in T[1:]:
                    if x[(i, p, t)].solution_value() != 0:
                        print(f'x[{i},{p},{t}] = {x[(i, p, t)].solution_value()}')
        for i in I:
            for p in P:
                for t in T[1:]:
                    for j in J:
                        if y[(j, i, p, t)].solution_value() != 0:
                            print(f'y[{j},{i},{p},{t}] = {y[(j, i, p, t)].solution_value()}')
        for i in I:
            for k in K:
                for p in P:
                    for t in T[1:]:
                        if z[(i, k, p, t)].solution_value() != 0:
                            print(f'z[{i},{k},{p},{t}] = {z[(i, k, p, t)].solution_value()}')
        for i in I:
            for p in P:
                for t in T[1:]:
                    if s[(i, p, t)].solution_value() != 0:
                        print(f's[{i},{p},{t}] = {s[(i, p, t)].solution_value()}')

    elif status == pywraplp.Solver.FEASIBLE:
        print('Solución factible encontrada, pero no necesariamente óptima.')
        print(f'Valor de la función objetivo (factible): {solver.Objective().Value()}')
    else:
        print('No se encontró una solución factible o hubo algún otro error.')
```

Listing 7: Ejecución del modelo y obtención de resultados

```
def main(file_path):
    data = load_data(file_path)
    solver = initialize_solver()

    # Asegurarse de que build_model devuelve tanto el solver como las variables y conjuntos necesarios
    solver, I, J, K, P, T, x, y, z, s, c, d, h, e = build_model(data, solver)

    # Pasar todos los parámetros necesarios a solve_model
    solve_model(solver, I, J, K, P, T, x, y, z, s, c, d, h, e)

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Error: No se especificó el archivo de datos.")
    else:
        main(sys.argv[1])
```

Listing 8: Main del modelo matemático de planeación

4. Análisis de Resultados

Al contar con el modelo matemático y su implementación, en código se generaron parámetros para poder observar el comportamiento del código y asegurarse que los resultados que arroja estén dentro de lo esperado. Para la prueba se cuenta con:

- **Plantas (I):** 2
- **Proveedores (J):** 2

- **Clientes (K):** 2
- **Periodos (T):** 2
- **Productos (P):** 2

4.1. Valores de prueba

```
# Definición de los conjuntos como DataFrames
plants = pd.DataFrame({'Plant': ['PlantaA', 'PlantaB']})
suppliers = pd.DataFrame({'Supplier': ['ProveedorA', 'ProveedorB']})
clients = pd.DataFrame({'Client': ['ClienteA', 'ClienteB']})
periods = pd.DataFrame({'Period': ['Semestre1', 'Semestre2']})
products = pd.DataFrame({'Product': ['Artículo1', 'Artículo2']})

# Costos de producción por planta y producto
production_costs = pd.DataFrame({
    'Plant': ['PlantaA', 'PlantaA', 'PlantaB', 'PlantaB'],
    'Product': ['Artículo1', 'Artículo2', 'Artículo1', 'Artículo2'],
    'Cost': [5, 8, 10, 4]
})

# Costos de compra incluyendo envío de proveedor a planta por producto y período
purchase_costs = pd.DataFrame({
    'Supplier': ['ProveedorA','ProveedorA','ProveedorA','ProveedorA','ProveedorA','ProveedorA
    ','ProveedorA','ProveedorA', 'ProveedorB','ProveedorB','ProveedorB','ProveedorB','
    ProveedorB','ProveedorB','ProveedorB','ProveedorB'],
    'Plant': ['PlantaA', 'PlantaA', 'PlantaA', 'PlantaA', 'PlantaB', 'PlantaB', 'PlantaB', '
    PlantaB','PlantaA', 'PlantaA', 'PlantaA', 'PlantaA', 'PlantaB', 'PlantaB', 'PlantaB', '
    PlantaB'],
    'Product': ['Artículo1', 'Artículo2', 'Artículo1', 'Artículo2', 'Artículo1', 'Artículo2',
     'Artículo1', 'Artículo2', 'Artículo1', 'Artículo2', 'Artículo1', 'Artículo2', 'Artículo1
    ', 'Artículo2', 'Artículo1', 'Artículo2'],
    'Period': ['Semestre1', 'Semestre1', 'Semestre2', 'Semestre2', 'Semestre1', 'Semestre1',
    'Semestre2', 'Semestre2', 'Semestre1', 'Semestre1', 'Semestre2', 'Semestre2', 'Semestre1'
    , 'Semestre1', 'Semestre2', 'Semestre2'],
    'Cost': [15,20,8,12,13,17,11,13,12,9,9,10,13,12,12,11]
})

# Costos de almacenamiento por planta y producto
storage_costs = pd.DataFrame({
    'Plant': ['PlantaA', 'PlantaA', 'PlantaB', 'PlantaB'],
    'Product': ['Artículo1', 'Artículo2', 'Artículo1', 'Artículo2'],
    'Cost': [3, 1, 2, 2]
})

# Ingresos netos por ventas de planta a cliente por producto y período
sales_revenue = pd.DataFrame({
    'Plant': ['PlantaA', 'PlantaA', 'PlantaA', 'PlantaA', 'PlantaA', 'PlantaA', 'PlantaA', '
    PlantaA', 'PlantaB', 'PlantaB', 'PlantaB', 'PlantaB', 'PlantaB', 'PlantaB', 'PlantaB', '
    PlantaB'],
    'Client': ['ClienteA', 'ClienteA', 'ClienteA', 'ClienteA', 'ClienteB', 'ClienteB', '
    ClienteB', 'ClienteB', 'ClienteA', 'ClienteA', 'ClienteA', 'ClienteA', 'ClienteB', '
    ClienteB', 'ClienteB', 'ClienteB'],
    'Product': ['Artículo1', 'Artículo2', 'Artículo1', 'Artículo2', 'Artículo1', 'Artículo2',
     'Artículo1', 'Artículo2', 'Artículo1', 'Artículo2', 'Artículo1', 'Artículo2', 'Artículo1
    ', 'Artículo2', 'Artículo1', 'Artículo2'],
    'Period': ['Semestre1', 'Semestre1', 'Semestre2', 'Semestre2', 'Semestre1', 'Semestre1',
    'Semestre2', 'Semestre2', 'Semestre1', 'Semestre1', 'Semestre2', 'Semestre2', 'Semestre1'
    , 'Semestre1', 'Semestre2', 'Semestre2'],
    'Revenue': [12, 11, 11, 12, 10, 10, 12, 11, 11, 12, 10, 10, 12, 11, 11, 12]
})

# Tiempo requerido para producir cada producto en cada planta
production_time = pd.DataFrame({
    'Plant': ['PlantaA', 'PlantaA', 'PlantaB', 'PlantaB'],
    'Product': ['Artículo1', 'Artículo2', 'Artículo1', 'Artículo2'],
    'Time': [2, 1, 1, 2]
})

# Inventario inicial por planta y producto
Initial_inventory = pd.DataFrame({
    'Plant': ['PlantaA', 'PlantaA', 'PlantaB', 'PlantaB'],
    'Product': ['Artículo1', 'Artículo2', 'Artículo1', 'Artículo2'],
```

```
    'Inventory': [5, 10, 10, 5]
})

# Presupuesto máximo de compra
Max_purchase_budget = pd.DataFrame({'Y': [9999999999999]})
```

4.2. Caso optimista: Mucho tiempo de producción disponible y poca demanda

```
# Capacidad máxima de tiempo de producción en cada planta por periodo
production_capacity = pd.DataFrame({
    'Plant': ['PlantaA', 'PlantaA', 'PlantaB', 'PlantaB'],
    'Period': ['Semestre1', 'Semestre2', 'Semestre1', 'Semestre2'],
    'Capacity': [300, 300, 300, 300]
})

# Demanda mínima de cada cliente por producto y periodo
client_demand = pd.DataFrame({
    'Client': ['ClienteA', 'ClienteA', 'ClienteA', 'ClienteA', 'ClienteB', 'ClienteB', '
    ClienteB', 'ClienteB'],
    'Product': ['Artículo1', 'Artículo1', 'Artículo2', 'Artículo2', 'Artículo1', 'Artículo1',
     'Artículo2', 'Artículo2'],
    'Period': ['Semestre1', 'Semestre2', 'Semestre1', 'Semestre2', 'Semestre1', 'Semestre2',
    'Semestre1', 'Semestre2'],
    'Demand': [30, 40, 25, 35, 45, 20, 15, 50]})
```

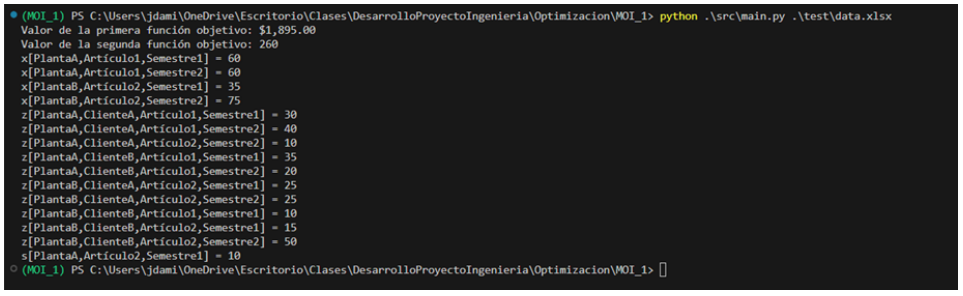

Figura 1: Resultados de caso optimista.

Al observar el caso optimista, notamos el cómo resulta mejor el producir los productos en planta directamente, con pocos o sin apoyo de proveedores externos. Para el caso de prueba se puede apreciar lo obtenido en la Fig[1].

4.3. Caso pesimista: Poco tiempo de producción disponible y mucha demanda

```
# Capacidad máxima de tiempo de producción en cada planta por periodo
production_capacity = pd.DataFrame({
    'Plant': ['PlantaA', 'PlantaA', 'PlantaB', 'PlantaB'],
    'Period': ['Semestre1', 'Semestre2', 'Semestre1', 'Semestre2'],
    'Capacity': [30, 40, 40, 30]
})

# Demanda mínima de cada cliente por producto y periodo
client_demand = pd.DataFrame({
    'Client': ['ClienteA', 'ClienteA', 'ClienteA', 'ClienteA', 'ClienteB', 'ClienteB', '
    ClienteB', 'ClienteB'],
    'Product': ['Artículo1', 'Artículo1', 'Artículo2', 'Artículo2', 'Artículo1', 'Artículo1',
     'Artículo2', 'Artículo2'],
    'Period': ['Semestre1', 'Semestre2', 'Semestre1', 'Semestre2', 'Semestre1', 'Semestre2',
    'Semestre1', 'Semestre2'],
    'Demand': [250, 350, 450, 600, 400, 300, 200, 150]})
```

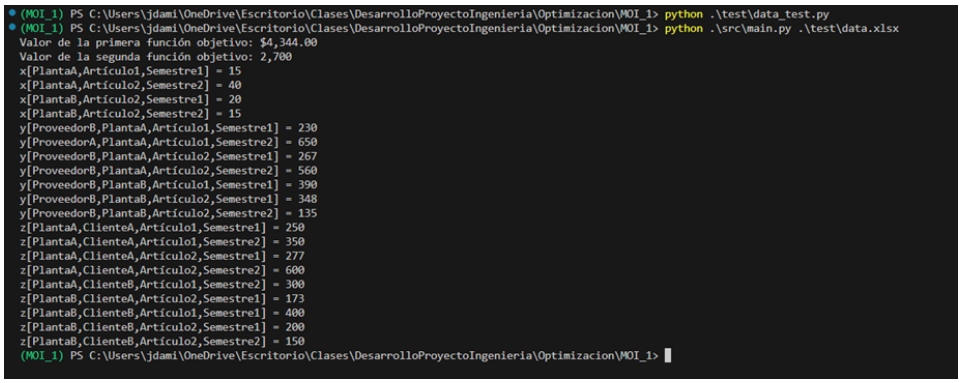

Figura 2: Resultados de caso pesimista.

Al observar el caso pesimista, notamos el cómo resulta mejor el comprar los productos a proveedores, con poca producción en plantas propiamente. Para el caso de prueba se puede apreciar lo obtenido en la Fig[2].

4.4. Caso termino normal: Tiempo de producción disponible y demanda moderada

```
# Capacidad máxima de tiempo de producción en cada planta por periodo
production_capacity = pd.DataFrame({
    'Plant': ['PlantaA', 'PlantaA', 'PlantaB', 'PlantaB'],
    'Period': ['Semestre1', 'Semestre2', 'Semestre1', 'Semestre2'],
    'Capacity': [90, 120, 100, 130]
})

# Demanda mínima de cada cliente por producto y periodo
client_demand = pd.DataFrame({
    'Client': ['ClienteA', 'ClienteA', 'ClienteA', 'ClienteA', 'ClienteB', 'ClienteB', '
    ClienteB', 'ClienteB'],
    'Product': ['Artículo1', 'Artículo1', 'Artículo2', 'Artículo2', 'Artículo1', 'Artículo1',
     'Artículo2', 'Artículo2'],
    'Period': ['Semestre1', 'Semestre2', 'Semestre1', 'Semestre2', 'Semestre1', 'Semestre2',
    'Semestre1', 'Semestre2'],
    'Demand': [130, 90, 100, 120, 120, 100, 90, 130]})
```

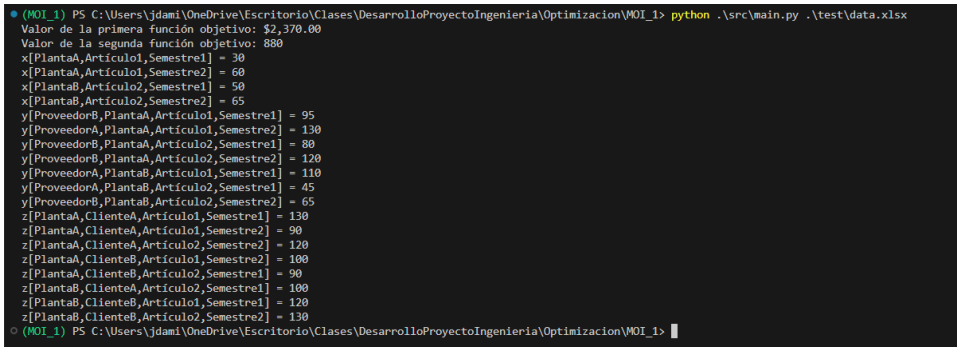

Figura 3: Resultados de caso normal.

Al observar el caso de término normal, notamos que lo mejor resulta en una combinación de producción en planta y de compra a proveedores externos. Para el caso de prueba se puede apreciar lo obtenido en la Fig[3].

5. Generación de Valor

El modelo matemático desarrollado genera valor de manera significativa al proporcionar una herramienta estratégica que optimiza los costos operativos y mejora la eficiencia global. Este modelo permite a las empresas balancear de manera eficiente la producción, la compra a proveedores, y la gestión del inventario, reduciendo los costos asociados mientras maximizan las ventas.

Al satisfacer las demandas de los clientes de manera precisa y eficiente, el modelo contribuye a mejorar la lealtad y satisfacción del cliente, aspectos críticos en mercados altamente competitivos. Además, su capacidad para adaptarse a fluctuaciones en la demanda y condiciones cambiantes del mercado permite a las empresas responder de manera ágil, lo que facilita la toma de decisiones estratégicas informadas y mejora la rentabilidad a largo plazo. Este modelo no solo fomenta una operación más rentable y sostenible, sino que también posiciona a la empresa para aprovechar oportunidades futuras y enfrentar desafíos en un entorno dinámico.