TECNOLÓGICO NACIONAL DE MÉXICO

MATERIA:

DESARROLLO DE SOFTWARE PARA LA TOMA DE DECISIONES

CLAVE DE MATERIA

DSED2302

CARRERA:

INGENIERIA EN SISTEMAS COMPUTACIONALES

“DOCUMENTO DE LOGICA”

PRESENTAN:

CRUZ SANCHEZ JHOAN MARVIN

ESPINOZA DE LA ROSA URIEL

JOSE SEBASTIAN JAFET

JUAREZ MONJARAZ GRISELDA ITZEL

ORTEGA PATIÑO NIMSI JOANA

NOMBRE DEL CATEDRÁTICO

MARTINEZ NIETO ADELINA

EQUIPO: **2**

GRUPO: **8SB**

OAXACA DE JUAREZ, OAXACA A 25 DE FEBRERO DEL 2026

Implementación del Modelo 4.9 *Mathematical Programming Optimization* (Turban) – LP Product-Mix (MBI Corp.)

1) Modelo del libro en el que se basa el programa

Nuestro programa se basa en el modelo explicado en el PDF de Turban, **Sección 4.9 "Mathematical Programming Optimization"**, específicamente el ejemplo de **mezcla de productos (LP Product-Mix Model Formulation)** donde se decide cuánto producir de dos productos para **maximizar la utilidad** bajo restricciones de recursos y requerimientos mínimos de mercado.

En este modelo se identifican los elementos clásicos de la Programación Lineal:

- **Variables de decisión (controlables):**
  (X1) = unidades de CC-7 a producir
  (X2) = unidades de CC-8 a producir
- **Variable de resultado:**
  (Z) = utilidad total (lo que se quiere maximizar).
- **Restricciones (variables no controlables / del entorno):**
  Limitaciones de **mano de obra**, **presupuesto de materiales** y **mínimos obligatorios** de producción por marketing.

2) Problema que resuelve el modelo

El problema real es: **decidir cuántas unidades producir de CC-7 y CC-8** para obtener la **mayor utilidad posible**, sin exceder:

1. Capacidad de mano de obra mensual
2. Presupuesto de materiales mensual
3. Requisitos mínimos de mercado (marketing exige producir al menos cierta cantidad)

En otras palabras, el DSS entrega una **recomendación de producción óptima** (una estrategia) en lugar de solo hacer cálculos sueltos, porque considera todas las restricciones al mismo tiempo y elige la mejor combinación factible.

3) Ejemplo implementado y programado MBI Corp.

En el caso MBI del libro, se tienen dos productos:

- CC-7: requiere 300 días de trabajo y $10,000 de materiales por unidad; utilidad $8,000.

- CC-8: requiere 500 días de trabajo y $15,000 de materiales por unidad; utilidad $12,000.
- Límites: 200,000 días de mano de obra y $8,000,000 de presupuesto.
- Marketing: mínimo 100 unidades CC-7 y mínimo 200 unidades CC-8.

El programa permite capturar esos valores y resolver el modelo automáticamente con Programación Lineal, devolviendo la combinación óptima (por ejemplo, con los datos del caso, una solución típica es producir **200 de CC-8** y alrededor de **333.33 de CC-7** con utilidad máxima aproximada de **$5,066,666.64**, según el escenario mostrado en la documentación/capturas).

4) Cómo se construyó el código y por qué se programó así

Para que el programa sea claro y fácil de mantener, se aplicó una separación por "capas", que coincide con la idea de un DSS:

1. **Interfaz** (captura de datos / interacción)
2. **Motor lógico (modelo matemático)**
3. **Salida / reporte (interpretación de resultados)**

Esta división evita mezclar "pantallas" con "matemáticas" y permite cambiar la interfaz (consola o GUI) sin tocar la lógica del modelo.

Estructura del proyecto (módulos)

- **gui_main.py:** interfaz gráfica (Tkinter) para capturar datos y mostrar el reporte.
- **logic.py:** "cerebro" matemático: arma y resuelve el modelo LP con PuLP.
- **main.py:** orquestador en versión consola (flujo inputs → lógica → outputs).
- **inputs.py:** captura de datos por consola.
- **outputs.py:** presentación e interpretación de resultados por consola.
- **README.md:** descripción general, instalación y prueba.

5) Explicación del código

5.1 GUI: gui_main.py

Responsabilidad: convertir el modelo matemático en una herramienta visual (captura de datos + botón + reporte).

Aquí no se “resuelve” el LP; solo se capturan parámetros y se mandan al motor lógico.

a) Importaciones y propósito

- Se usa **Tkinter** para la GUI.
- Se importa logic para ejecutar el modelo matemático.

b) __init__(self, root)

Construye la ventana y organiza el formulario en 3 pasos (igual que el planteamiento del modelo):

- **Paso 1 (Rentabilidad):** utilidad por unidad CC-7 y CC-8 (coeficientes de la función objetivo).
- **Paso 2 (Recursos):** límites de mano de obra y presupuesto (restricciones).
- **Paso 3 (Mercado):** mínimos de producción (cotas inferiores).
- Botón: **CALCULAR RECOMENDACIÓN ÓPTIMA**.
- Área de texto: reporte final.

Se programó así para que el usuario vea el mismo “mapa mental” del modelo de Turban: objetivo → restricciones → mínimos.

```
# --- SECCIÓN 1: RENTABILIDAD (Variables de Resultado) ---
tk.Label(root, text="PASO 1: Rentabilidad por Producto", font=style_header, fg="blue").pack(pady=5)

self.u_cc7 = self.crear_entrada("Utilidad CC-7 ($):", "8000") # Datos de MBI
self.u_cc8 = self.crear_entrada("Utilidad CC-8 ($):", "12000")

# --- SECCIÓN 2: RECURSOS (Variables Incontrolables / Restricciones) ---
tk.Label(root, text="PASO 2: Disponibilidad de Recursos", font=style_header, fg="blue").pack(pady=5)

self.limit_labor = self.crear_entrada("Días de Labor disponibles:", "200000")
self.limit_budget = self.crear_entrada("Presupuesto de Materiales ($):", "8000000")

# --- SECCIÓN 3: MERCADO (Límites Inferiores) ---
tk.Label(root, text="PASO 3: Requerimientos de Mercado", font=style_header, fg="blue").pack(pady=5)

self.min_cc7 = self.crear_entrada("Mínimo CC-7 a producir:", "100")
self.min_cc8 = self.crear_entrada("Mínimo CC-8 a producir:", "200")

# --- BOTÓN DE ACCIÓN: Ejecutar Modelo ---
self.btn_resolver = tk.Button(root, text="CALCULAR RECOMENDACIÓN ÓPTIMA",
                              command=self.ejecutar_modelo, bg="green", fg="white", font=("Arial", 11
self.btn_resolver.pack(pady=20)

# --- SECCIÓN DE RESULTADOS ---
tk.Label(root, text="REPORTE ESTRATÉGICO FINAL", font=style_header).pack()
self.txt_resultados = tk.Text(root, height=12, width=55, state="disabled", bg="#f0f0f0")
self.txt_resultados.pack(pady=5)
```

c) crear_entrada(self, texto, default_val)

Qué hace: función auxiliar para no repetir código.

- Crea un Frame con una etiqueta y un Entry con valor por defecto.
- Se programó así para mantener el GUI limpio y consistente.

```python
def crear_entrada(self, texto, default_val):
    """Crea una etiqueta y un cuadro de texto con un valor por defecto."""
    frame = tk.Frame(self.root)
    frame.pack(fill="x", padx=40, pady=2)
    tk.Label(frame, text=texto, width=25, anchor="w").pack(side="left")
    entry = tk.Entry(frame)
    entry.insert(0, default_val)
    entry.pack(side="right", expand=True, fill="x")
    return entry
```

d) ejecutar_modelo(self)

Qué hace (flujo DSS):

1. Lee los datos del formulario y los convierte a números (float).
2. Los empaqueta en un diccionario datos.
3. Llama a logic.calcular_mezcla_optima(datos) (aquí ocurre la optimización real).
4. Manda el resultado a mostrar_reporte(res, datos).
5. Si hay error de captura (texto en lugar de números), muestra un messagebox.

Se programó con diccionario porque el motor lógico necesita recibir **parámetros del escenario** (coeficientes y límites) de forma clara y flexible.

```python
def ejecutar_modelo(self):
    """Extrae los datos, llama al cerebro lógico y muestra el reporte."""
    try:
        # Captura dinámica de datos
        datos = {
            'utilidad_cc7': float(self.u_cc7.get()),
            'utilidad_cc8': float(self.u_cc8.get()),
            'limite_labor': float(self.limit_labor.get()),
            'limite_presupuesto': float(self.limit_budget.get()),
            'min_cc7': float(self.min_cc7.get()),
            'min_cc8': float(self.min_cc8.get())
        }

        # Ejecución del motor lógico (Llamada al Integrante 1)
        res = logic.calcular_mezcla_optima(datos)

        # Presentación de resultados
        self.mostrar_reporte(res, datos)

    except ValueError:
        messagebox.showerror("Error de Datos", "Por favor, ingrese solo números válidos.")
```

e) mostrar_reporte(self, res, datos)

Qué hace: transforma números en una recomendación gerencial:

- Si status == 1:
    - imprime unidades óptimas (X1, X2)
    - imprime utilidad total (Z)
    - calcula un análisis simple tipo “holgura” (slack) para mostrar consumo de recursos (por ejemplo, mano de obra usada vs disponible y presupuesto sobrante).
- Si no: indica escenario infactible.

Esta parte se programó para que el DSS no solo “dé números”, sino que explique **qué significan** (en especial el consumo de recursos).

```python
def mostrar_reporte(self, res, datos):
    """Muestra el análisis de la solución óptima y la holgura (Slack)."""
    self.txt_resultados.config(state="normal")
    self.txt_resultados.delete("1.0", tk.END)

    if res['status'] == 1:
        reporte = (
            f"ESTADO: Solución Óptima Encontrada conforme al modelo de Turban.\n\n"
            f"ESTRATEGIA SUGERIDA:\n"
            f" > Fabricar CC-7: {res['x1']:,.2f} unidades\n"
            f" > Fabricar CC-8: {res['x2']:,.2f} unidades\n\n"
            f"UTILIDAD TOTAL MAXIMIZADA: ${res['ganancia_total']:,.2f}\n"
            f"-----------------------------------------------------------\n"
            f"ANÁLISIS DE RECURSOS (HOLGURA):\n"
        )

        # Cálculo de recursos usados para análisis de Slack
        labor_usada = 300 * res['x1'] + 500 * res['x2']
        sobrante_pres = datos['limite_presupuesto'] - (10000 * res['x1'] + 15000 * res['x2'])

        reporte += f" > Mano de Obra: {labor_usada:,.0f} / {datos['limite_labor']:,.0f} días (AGOT
        reporte += f" > Presupuesto Sobrante: ${sobrante_pres:,.2f} (DISPONIBLE)\n"

        self.txt_resultados.insert(tk.END, reporte)
    else:
        self.txt_resultados.insert(tk.END, "ERROR: El escenario es INFACTIBLE.\nNo hay recursos su

    self.txt_resultados.config(state="disabled")
```

5.2 Motor matemático: logic.py (función calcular_mezcla_optima)

Responsabilidad: construir el modelo LP y resolverlo.

a) Librería usada

Se usa **PuLP** (solver CBC) para ejecutar Programación Lineal (método tipo Simplex/solver LP).

```python
# logic.py
# Responsabilidad: Motor matematico de optimizacion (Linear Programming Solver)

#Se importa las funciones necesarias de la libreria PuLP
#El LpMaximize Indica que queremos maximizar ganancias y no minimizar costos
#LpProblem es el contenedor del modelo matematico
#LpVariable este representa las Variables de Decisión queson X1, X2
from pulp import LpMaximize, LpProblem, LpVariable, value, PULP_CBC_CMD
```

b) Paso 1: Definición del problema

prob = LpProblem("Optimizacion_Ganancias_MBI", LpMaximize)

Se define que es un problema de **maximización** (por la utilidad).

```
#1.- DEFINICION DEL PROBLEMA
#Creamos el objeto del modelo indicando que el objetivo es maximizar (LpMaximize)
prob = LpProblem("Optimizacion_Ganancias_MBI", LpMaximize)
```

c) Paso 2: Variables de decisión (X1 y X2)

x1 = LpVariable("Unidades_CC7", lowBound=d['min_cc7'], cat='Continuous')

x2 = LpVariable("Unidades_CC8", lowBound=d['min_cc8'], cat='Continuous')

- lowBound implementa directamente los mínimos de marketing: (X1>=min_cc7), (X2 >= min_cc8).
- Continuous permite decimales (en LP se permite).

```
x1 = LpVariable("Unidades_CC7", lowBound=d['min_cc7'], cat='Continuous')
x2 = LpVariable("Unidades_CC8", lowBound=d['min_cc8'], cat='Continuous')
```

d) Paso 3: Función objetivo

prob += d['utilidad_cc7'] * x1 + d['utilidad_cc8'] * x2

Esto corresponde al modelo:
max Z = 8000X_1 + 12000X_2
pero parametrizado, para que no sea “fijo”, sino configurable por el usuario.

```
#3.- Funcion objectivo (Z)
#Z = (Ganancia_CC7 * X1) + (Ganancia_CC8 * X2)
#Esta es la meta matematica que el algoritmo intentara hacer lo más grande posible.
prob += d['utilidad_cc7'] * x1 + d['utilidad_cc8'] * x2, "Ganancia_Total_Z"
```

e) Paso 4: Restricciones (mundo real)

prob += 300*x1 + 500*x2 <= d['limite_labor']

prob += 10000*x1 + 15000*x2 <= d['limite_presupuesto']

Estas ecuaciones implementan exactamente las restricciones del caso:

- Mano de obra
- Presupuesto de materiales

```
#4.- Restricciones
#Representan las limitaciones del mundo real.
#El algoritmo buscara valores para X1 y X2 que no violen estas ecuaciones.

#Restricciom de mano de obra: (300 días * X1) + (500 días * X2) <= Disponibilidad Total
prob += 300 * x1 + 500 * x2 <= d['limite_labor'], "Restriccion_Mano_de_Obra"

#Restriccion de presupuesto: ($10,000 * X1) + ($15,000 * X2) <= Presupuesto Total
prob += 10000 * x1 + 15000 * x2 <= d['limite_presupuesto'], "Restriccion_Presupuesto"
```

f) Paso 5: Resolver

prob.solve(PULP_CBC_CMD(msg=0))

Se resuelve con CBC y se ocultan logs para no saturar la salida.

```
#5.- solucion (ALGORITMO)
#Ejecuta el solver interno para encontrar los valores optimos de X1 y X2
#msg=0 oculta los logs tecnicos de la libreria para limpiar la consola
prob.solve(PULP_CBC_CMD(msg=0))
```

g) Paso 6: Retorno estandarizado

Se devuelve un diccionario:

- status (óptimo o infactible)
- valores óptimos x1, x2
- ganancia_total (Z)

Esto se programó así para que **cualquier interfaz** (GUI o consola) pueda consumir el resultado de la misma forma.

```
#6.- Retorno de resultados
#Empaquetamos el estado de la solucion y los valores encontrados
#para enviarlos al modulo de presentacion (outputs.py de Nimsi).
return {
    "status": prob.status,          #1 = optimo, -1 = Infactible
    "x1": value(x1),                #Cantidad optima de CC-7
    "x2": value(x2),                #Cantidad optima de CC-8
    "ganancia_total": value(prob.objective) #Valor maximo de Z
}
```

6) Cómo nos guiamos para construirlo

El método fue literalmente traducir el modelo de Turban a código:

1. Identificar **variables de decisión** (X1, X2).
2. Definir **función objetivo** (maximizar Z).
3. Definir **restricciones** (mano de obra, presupuesto, mínimos).
4. Elegir un solver de LP (PuLP CBC) para automatizar el cálculo.
5. Separar el DSS en módulos: interfaz, lógica, salida.

Así, el programa no “inventa” reglas: **ejecuta el modelo** como lo plantea el capítulo, pero empaquetado en un DSS funcional.

7) Por qué el programa está hecho así

Este DSS se programó con una arquitectura modular porque:

- hace más fácil entender la lógica (cada archivo tiene una responsabilidad),
- permite cambiar la interfaz sin modificar el modelo matemático,
- refleja la estructura de un DSS real (entrada → modelo → recomendación),
- y mantiene el enfoque del libro: **optimizar decisiones bajo restricciones**.