

Proyecto de Optimización Lineal para la Planificación de Plantas de Generación Eléctrica

Noel Pérez Calvo (C311)
Jabel Resendiz Aguirre (C312)

Definición del Problema

En este proyecto se busca optimizar la planificación de la construcción y operación de plantas de generación eléctrica, minimizando los costos totales de construcción y operación, mientras se cumplen ciertas restricciones, como la demanda energética y las emisiones de CO₂.

Datos del Problema

El problema incluye tres tipos de plantas:

- Térmica (T)
- Hidroeléctrica (H)
- Renovable (R)

Cada tipo de planta tiene costos asociados de construcción y operación, así como emisiones de CO₂. Además, hay límites en la cantidad mínima y máxima de generación de cada planta y restricciones en el número de plantas a construir.

- Costos de construcción y generación:

Térmica: 1,000,000 USD, 50 USD/MW

Hidroeléctrica: 2,000,000 USD, 30 USD/MW

Renovable: 800,000 USD, 20 USD/MW

- Restricciones:

- Demanda total: 1000 MW.
- Límite de emisiones de CO₂: 500 toneladas.

Variables

Se definen las siguientes variables:

- $x_T, x_H, x_R \in \mathbb{Z}^+$: Número de plantas a construir de cada tipo.
- $g_{Ti}, g_{Hj}, g_{Rk} \geq 0$: Generación de cada planta térmica, hidroeléctrica y renovable.

Función Objetivo

La función objetivo busca minimizar el costo total de construcción y operación de las plantas de generación:

$$\text{mín } Z = \sum_{i=1}^{x_T} (1,000,000) + \sum_{j=1}^{x_H} (2,000,000) + \sum_{k=1}^{x_R} (800,000) + \sum_{i=1}^{x_T} (50g_{Ti}) + \sum_{j=1}^{x_H} (30g_{Hj}) + \sum_{k=1}^{x_R} (20g_{Rk})$$

Restricciones

- Demanda energética:

$$\sum_{i=1}^{x_T} g_{Ti} + \sum_{j=1}^{x_H} g_{Hj} + \sum_{k=1}^{x_R} g_{Rk} \geq 1000$$

- Restricciones de generación por planta:

$$50 \leq g_{Ti} \leq 200, \quad 100 \leq g_{Hj} \leq 300, \quad 20 \leq g_{Rk} \leq 150$$

- Límite de emisiones de CO₂:

$$\sum_{i=1}^{x_T} 0,8g_{Ti} \leq 500$$

Estrategia de Resolución

El problema se resolverá en dos fases:

1. **Fase 1:** Resolver el problema relajado utilizando el método Simplex.
2. **Fase 2:** Resolver el problema utilizando Ramificación y Acotación (Branch & Bound).

Fase 1: Resolución con el Método Simplex Relajado

En esta fase, se resuelve el problema sin considerar la restricción de enteros, utilizando el método Simplex. El problema se modela como un problema de programación lineal con variables continuas.

```
1 import pulp
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # Definir el problema
6 problem = pulp.LpProblem("LP_PowerPlant_Optimization", pulp.LpMinimize)
7
8 # Variables continuas para la cantidad de plantas
9 x_T = pulp.LpVariable("x_T", lowBound=0, upBound=10, cat='Continuous')
10 x_H = pulp.LpVariable("x_H", lowBound=0, upBound=10, cat='Continuous')
11 x_R = pulp.LpVariable("x_R", lowBound=0, upBound=10, cat='Continuous')
12
13 # Variables continuas para la generacion por planta
14 G_T = pulp.LpVariable("G_T", lowBound=0)
15 G_H = pulp.LpVariable("G_H", lowBound=0)
16 G_R = pulp.LpVariable("G_R", lowBound=0)
17
18 # Restringir G_T, G_H, G_R segun las plantas
19 problem += G_T >= 50 * x_T
20 problem += G_T <= 200 * x_T
21 problem += G_H >= 100 * x_H
22 problem += G_H <= 300 * x_H
23 problem += G_R >= 20 * x_R
24 problem += G_R <= 150 * x_R
25
26 # Función objetivo: Minimizar costos totales
27 problem += (
28     x_T * 1_000_000 + x_H * 2_000_000 + x_R * 800_000 +
29     50 * G_T + 30 * G_H + 20 * G_R
30 ), "Total_Cost"
31
32 # Restricción de demanda
33 problem += (G_T + G_H + G_R >= 1000), "Demand"
34
35 # Restricción de emisiones de CO2
```

```

36 problem += (0.8 * G_T <= 500), "CO2_Limit"
37
38 # Resolver el problema con Simplex
39 problem.solve()
40
41 # Imprimir resultados
42 print("Estado de la solución:", pulp.LpStatus[problem.status])
43 print("Costo total:", pulp.value(problem.objective))
44 print("Plantas térmicas:", pulp.value(x_T))
45 print("Plantas hidroeléctricas:", pulp.value(x_H))
46 print("Plantas renovables:", pulp.value(x_R))
47 print("Generación térmica:", pulp.value(G_T))
48 print("Generación hidroeléctrica:", pulp.value(G_H))
49 print("Generación renovable:", pulp.value(G_R))
50
51 # Visualización de la generación
52 labels = ["Térmica", "Hidroeléctrica", "Renovable"]
53 generacion = [pulp.value(x_T), pulp.value(x_H), pulp.value(x_R)]
54
55 plt.figure(figsize=(8, 6))
56 plt.bar(labels, generacion, color=['red', 'blue', 'green'])
57 plt.xlabel("Tipo de Planta")
58 plt.ylabel("Generación (MW)")
59 plt.title("Distribución de la Generación de Energía")
60 plt.show()

```

Resultados

- Estado de la solución: Óptima.
- Costo total: 5,163,750 USD.
- Plantas térmicas: 3.125.
- Plantas hidroeléctricas: 0.0.
- Plantas renovables: 2.5.

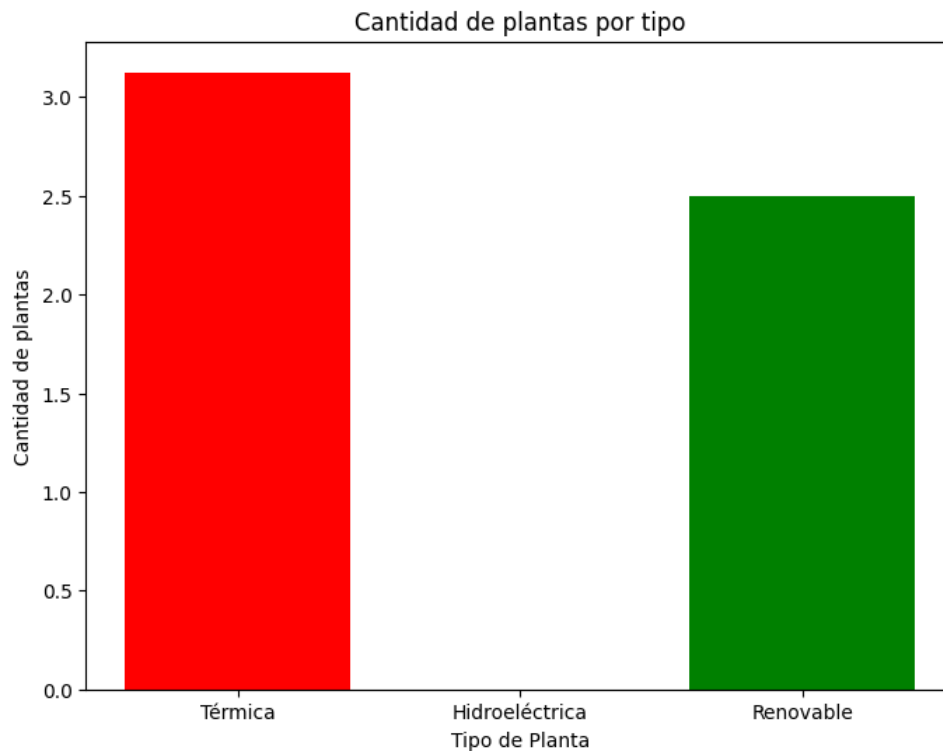


Figura 1: Distribución de la generación de energía por tipo de planta

Fase 2: Resolución con Ramificación y Acotación

En esta fase, se utiliza el método de Ramificación y Acotación para manejar las restricciones de enteros en las variables. A continuación, se muestra la implementación en Python del método:

```

1  import pulp
2  import time
3  import pandas as pd
4  import matplotlib.pyplot as plt
5
6  def branch_and_bound(problem, vars_ent, max_iters=100):
7      """ Implementación del método Branch and Bound """
8      iteraciones = 0
9      cola = [(problem, [])] # Inicializar la cola con el problema original
10
11     mejor_sol = None
12     mejor_costo = float("inf")
13
14     while cola and iteraciones < max_iters:
15         iteraciones += 1
16         nodo, ramas = cola.pop(0)
17
18         # Resolver la relajación LP del nodo actual
19         nodo.solve()
20
21         if nodo.status != pulp.LpStatusOptimal:
22             continue # No hay solución óptima para este nodo
23
24         # Obtener solución óptima relajada
25         costo_actual = pulp.value(nodo.objective)
26         valores_vars = {var.name: pulp.value(var) for var in vars_ent}
27
28         # Si la solución es peor que la mejor encontrada, descartar
29         if costo_actual >= mejor_costo:

```

```

30         continue
31
32     # Verificar si todas las variables enteras están en valores enteros
33     if all(abs(val - int(val)) < 1e-6 for val in valores_vars.values()):
34         mejor_sol = valores_vars
35         mejor_costo = costo_actual
36         continue # No hay necesidad de seguir explorando
37
38     # Seleccionar una variable no entera para bifurcar
39     var_frac = next(var for var in vars_ent if abs(valores_vars[var.name]
40         ] - int(valores_vars[var.name])) > 1e-6)
41
42     # Crear dos nuevos subproblemas con restricciones adicionales
43     nuevo_nodo1 = nodo.deepcopy()
44     nuevo_nodo1 += var_frac <= int(valores_vars[var_frac.name])
45
46     nuevo_nodo2 = nodo.deepcopy()
47     nuevo_nodo2 += var_frac >= int(valores_vars[var_frac.name]) + 1
48
49     cola.append((nuevo_nodo1, ramas + [f"{var_frac.name} <= {int(
50         valores_vars[var_frac.name])}"]))
51     cola.append((nuevo_nodo2, ramas + [f"{var_frac.name} >= {int(
52         valores_vars[var_frac.name]) + 1}"]))
53
54     return mejor_sol, mejor_costo, iteraciones
55
56 # ** Definir el problema inicial **
57 def crear_problema():
58     problem = pulp.LpProblem("MILP_PowerPlant_Optimization", pulp.LpMinimize
59         )
60
61     x_T = pulp.LpVariable("x_T", lowBound=0, upBound=10, cat='Continuous')
62     x_H = pulp.LpVariable("x_H", lowBound=0, upBound=10, cat='Continuous')
63     x_R = pulp.LpVariable("x_R", lowBound=0, upBound=10, cat='Continuous')
64
65     G_T = pulp.LpVariable("G_T", lowBound=0)
66     G_H = pulp.LpVariable("G_H", lowBound=0)
67     G_R = pulp.LpVariable("G_R", lowBound=0)
68
69     problem += G_T >= 50 * x_T
70     problem += G_T <= 200 * x_T
71     problem += G_H >= 100 * x_H
72     problem += G_H <= 300 * x_H
73     problem += G_R >= 20 * x_R
74     problem += G_R <= 150 * x_R
75
76     problem += (x_T * 1_000_000 + x_H * 2_000_000 + x_R * 800_000 +
77         50 * G_T + 30 * G_H + 20 * G_R), "Total_Cost"
78
79     problem += (G_T + G_H + G_R >= 1000), "Demand"
80     problem += (0.8 * G_T <= 500), "CO2_Limit"
81
82     return problem, [x_T, x_H, x_R]
83
84 if __name__ == "__main__":
85     problema, vars_ent = crear_problema()
86
87     start_time = time.time()
88     solucion, costo, iteraciones = branch_and_bound(problema, vars_ent)
89     end_time = time.time()

```

```

89     tiempo_total = end_time - start_time
90
91     print("\n **Resultados Branch and Bound**")
92     print(f" Costo Optimo: {costo}")
93     print(f" Iteraciones: {iteraciones}")
94     print(f" Tiempo de ejecución: {tiempo_total:.4f} segundos")
95     print(f" Solución óptima encontrada: {solucion}")
96
97     # ** Visualización de generación de energía **
98     if solucion:
99         etiquetas = ["Térmica", "Hidroeléctrica", "Renovable"]
100         valores = [solucion["x_T"], solucion["x_H"], solucion["x_R"]]
101
102         plt.figure(figsize=(8, 6))
103         plt.bar(etiquetas, valores, color=['red', 'blue', 'green'])
104         plt.xlabel("Tipo de Planta")
105         plt.ylabel("Cantidad de Plantas")
106         plt.title("Distribución de la Generación de Energía - Branch and
107                    Bound")
108         plt.show()

```

Resultados

- Estado de la solución: Óptima.
- Costo total: 5232000.0 USD.
- Plantas térmicas: 2.0.
- Plantas hidroeléctricas: 0.0.
- Plantas renovables: 4.0.

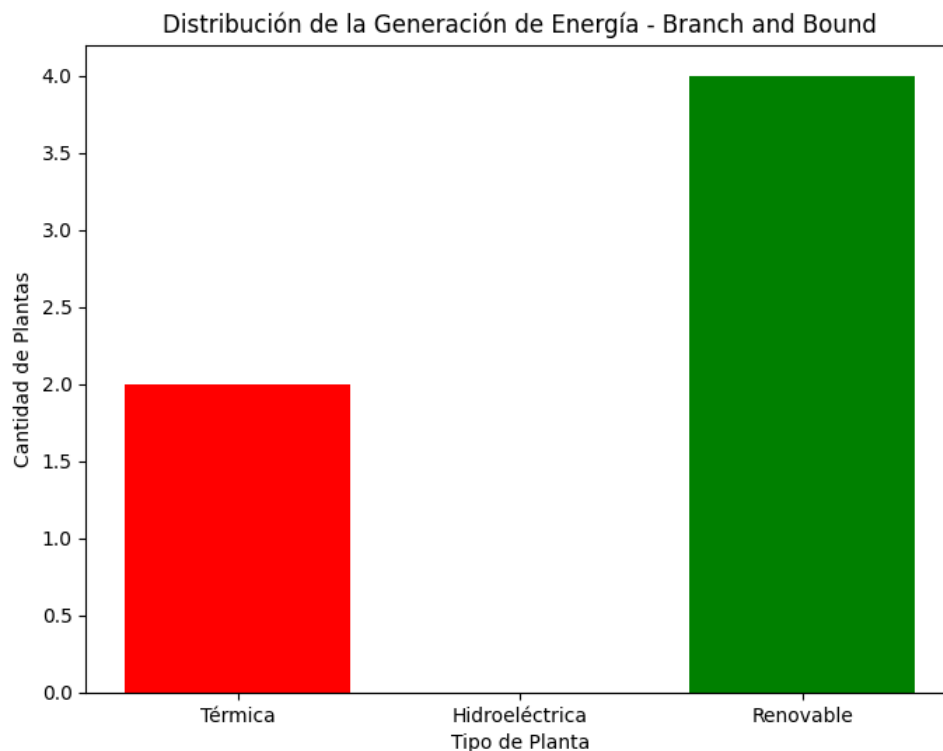


Figura 2: Distribución de la generación de energía por tipo de planta