FACULTAD DE INGENIERÍA
INGENIERÍA CIVIL INFORMÁTICA

Informe Proyecto Final

Cristopher Michael Arredondo Canchis

Profesor tutor:
Gustavo Gatica

Santiago, Chile
2025

Índice

Índice de figuras

1. Introducción

1.1. Contexto

En este examen se aborda el problema de localización de centros de distribución, donde cada nodo de una red actúa simultáneamente como punto de demanda y posible sitio de instalación. El objetivo es determinar en cuáles de estos nodos abrir hasta p centros de distribución de forma que se minimice la suma de los costos de transporte y los costos fijos de instalación.

1.2. Objetivos

- Definir claramente los parámetros, variables, función objetivo y restricciones del modelo de localización.
- Implementar y resolver el modelo exacto en AMPL para $p = 1, \ldots, 5$.
- Diseñar e implementar una heurística en Python que genere soluciones aproximadas y analizar su convergencia.
- Comparar resultados exactos y heurísticos en términos de calidad de solución y tiempo de cómputo.
- Documentar todo el proceso en un informe en LaTeX, incluyendo tablas y gráficos pertinentes.

2. Formulación matemática y modelo

2.1. Descripción del problema

Se considera una red de N nodos, cada uno de los cuales actúa simultáneamente como punto de demanda y como candidato para la instalación de un centro de distribución (CD). Se desea seleccionar hasta p centros de distribución de forma que se minimice el costo total compuesto por los costos de transporte de la demanda y los costos fijos de apertura de los CDs.

2.2. Conjuntos, parámetros y variables

Conjuntos

$\mathcal{I} = \{1, \ldots, N\}$ (nodos de demanda), $\mathcal{J} = \{1, \ldots, N\}$ (nodos candidatos a CD).

Parámetros

h_i	demanda anual en el nodo i,
f_j	costo fijo anual de abrir un CD en el nodo j,
d_{ij}	distancia mínima entre i y j (km),
u	costo unitario de transporte (USD/km·unidad),
c_{ij}	costo de atender la demanda i desde el CD j,
p	número máximo de CDs a ubicar.

Variables de decisión

$$x_j = \begin{cases} 1, & \text{si se abre un CD en el nodo } j, \\ 0, & \text{en caso contrario,} \end{cases} \qquad y_{ij} = \begin{cases} 1, & \text{si la demanda } i \text{ es atendida por el CD } j, \\ 0, & \text{en caso contrario.} \end{cases}$$

$$x_j \in \{0,1\}, \quad y_{ij} \in \{0,1\} \quad \forall\, i \in \mathcal{I},\, j \in \mathcal{J}.$$

2.3. Función objetivo

$$\text{mín} \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} h_i \, c_{ij} \, y_{ij} \; + \; \sum_{j \in \mathcal{J}} f_j \, x_j.$$

2.4. Restricciones

1. Cobertura de la demanda: $\sum_{j \in \mathcal{J}} y_{ij} = 1$, para todo $i \in \mathcal{I}$.
2. Asignación condicionada: $y_{ij} \leq x_j$, para todo $i \in \mathcal{I}$, $j \in \mathcal{J}$.
3. Límite de centros: $\sum_{j \in \mathcal{J}} x_j \leq p$.
4. Naturaleza de las variables: x_j, $y_{ij} \in \{0,1\}$.

2.5. Cálculo de c_{ij}

El costo de atender la demanda i desde el CD j se calcula como

$$c_{ij} = u \times d_{ij},$$

donde u es el costo unitario de transporte.

3. Preparación de datos

En este capítulo se detalla el origen de la información, la estructura del archivo de datos para AMPL y la forma de calcular la matriz de costos c_{ij} empleada también en la heurística en Python. Todos los valores se extraen de la figura provista en las instrucciones del examen.

3.1 Fuentes de datos

La única fuente de información es la figura del examen, donde para cada nodo i se indican:

- La demanda anual h_i.
- El costo fijo anual de inversión f_i.
- Las distancias d_{ij} (en kilómetros) a cada otro nodo j.

Además, se adopta un costo unitario de transporte u igual a 0.5 USD/km·unidad, según especificación de la empresa.

3.2 Estructura del archivo `datos.dat`

A partir de esas cifras, se construye el archivo `datos.dat` para AMPL:

```
# datos.dat

# 1. Conjunto de nodos
set NODES := 1 2 3 4 5 ;

# 2. Costo unitario de transporte
param u := 0.5 ;

# 3. Demandas h_i
param h :=
  1 1100
  2 1200
  3 550
  4 820
  5 880
;

# 4. Costos fijos f_j
param f :=
  1 310
  2 250
  3 80
  4 120
  5 260
;

# 5. Matriz de distancias d[i,j]
param d :=
  1 1 0
  1 2 4.2
  1 3 2.7
  1 4 2.3
  1 5 2.2
  2 1 4.2
  2 2 0
  2 3 1.9
  2 4 3.7
  2 5 6.4
  3 1 2.7
  3 2 1.9
  3 3 0
  3 4 2.6
  3 5 4.9
  4 1 2.3
  4 2 3.7
```

```
  4 3 2.6
  4 4 0
  4 5 5.6
  5 1 2.2
  5 2 6.4
  5 3 4.9
  5 4 5.6
  5 5 0
;

# 6. Número de CDs a ubicar
param p ;
```

Este archivo define:

- `NODES`: conjunto de todos los nodos.
- `u`: costo unitario de transporte.
- `h`: vector de demandas.
- `f`: vector de costos fijos.
- `d`: matriz completa de distancias.
- `p`: parámetro dinámico que fija el número de centros a ubicar.

3.3 Cálculo de la matriz de costos c_{ij}

Dentro del modelo AMPL, la matriz de costos se define simbólicamente como:

$$c_{ij} = u \times d_{ij}.$$

Para garantizar consistencia en Python, tras leer `datos.dat` se extraen los bloques correspondientes y se construye un `DataFrame` con `pandas`, calculando c_{ij} de la misma forma:

```
import pandas as pd

with open('../../data/data.dat','r', encoding='utf-8') as f:
    txt = f.read()

u = float(re.search(r'param\s+u\s*:=\s*([\d\.]+)', txt).group(1))

d_block = re.search(r'param\s+d\s*:=\s*(.*?)\s*;', txt,
↪  re.DOTALL).group(1).strip().splitlines()

distances = [(int(i), int(j), float(v)) for i,j,v in (line.split() for
↪  line in d_block)]

df_d = pd.DataFrame(distances, columns=['origen','destino','d'])
```

```python
df_d['c'] = df_d['d'] * u

cost_matrix = df_d.pivot(index='origen', columns='destino', values='c')

cost_matrix = cost_matrix.fillna(float('inf'))

print(cost_matrix)
```

4. Implementación del modelo exacto en AMPL

4.1 Esquema del archivo `modelo.mod`

```
# modelo.mod

# 1. Conjuntos y parámetros
set NODES ;
param u ;
param h{NODES} ;
param f{NODES} ;
param d{NODES, NODES} ;
param p ;

# 2. Variables de decisión
var x{j in NODES} binary ;
var y{i in NODES, j in NODES} binary ;

# 3. Función objetivo
minimize TotalCost:
    sum{i in NODES, j in NODES}
       h[i] * u * d[i,j] * y[i,j]
  + sum{j in NODES}
       f[j] * x[j] ;

# 4. Restricciones
subject to Cover{i in NODES}:
    sum{j in NODES} y[i,j] = 1 ;

subject to Assign{i in NODES, j in NODES}:
    y[i,j] <= x[j] ;

subject to Limit:
    sum{j in NODES} x[j] <= p ;
```

4.2 Uso del archivo `datos.dat`

Para cargar los datos en AMPL:

```
model modelo.mod ;
data datos.dat ;
```

4.3 Procedimiento de ejecución y extracción de resultados

Todo el proceso de resolución para $p = 1 \dots 5$ se automatiza en Python con `amplpy`, comparando dos solvers:

```
from amplpy import AMPL
import pandas as pd

ampl = AMPL()
# Leer modelo y datos
ampl.read("../../ampl/models/model.mod")
ampl.read_data("../../data/data.dat")

solvers = ["highs", "gurobi"]
all_results = []

for solver in solvers:
    print(f"\n=== SOLVER: {solver.upper()} ===")
    ampl.option['solver'] = solver
    solver_results = []
    for p_val in range(1, 6):
        ampl.get_parameter('p').set(p_val)
        ampl.solve()

        # Verificar estado de la solución
        solve_result = ampl.get_value('solve_result')
        if solve_result != 'solved':
            print(f"p={p_val} → No se pudo resolver: {solve_result}")
            continue

        # Extraer resultados
        obj = ampl.get_objective('TotalCost').value()
        x_vals = ampl.get_variable('x').get_values().to_pandas()
        nodos_selec = [idx for idx, row in x_vals.iterrows() if
        ↪  row.iloc[0] >= 0.5]

        resultado = {
            'solver': solver,
            'p': p_val,
            'costo_total': obj,
            'nodos_cd': nodos_selec,
            'num_nodos': len(nodos_selec)
        }
        solver_results.append(resultado)
        all_results.append(resultado)
```

```
            print(f"p={p_val} → Costo: {obj:.2f}, CDs en nodos:
            ↪  {nodos_selec}")

        # Resumen por solver
        print(f"\nResultados con {solver.upper()}:")
        df_solver = pd.DataFrame(solver_results)
        print(df_solver[['p', 'costo_total',
        ↪  'num_nodos']].to_string(index=False))

# Resumen completo
print("\n=== RESUMEN COMPLETO ===")
df_all = pd.DataFrame(all_results)
print(df_all.to_string(index=False))
```

Con este script:

- Se cargan el modelo y los datos.
- Para cada solver (`highs`, `gurobi`) y cada $p \in \{1, \ldots, 5\}$:
  - Se fija el parámetro p.
  - Se resuelve el modelo.
  - Se extrae el costo total y los nodos donde $x_j = 1$.
- Se genera un resumen de resultados por solver y un resumen global.

5. Diseño e implementación de metaheurísticas en Python

En este capítulo se describen las tres metaheurísticas aplicadas al problema de localización de centros de distribución: Algoritmo *Greedy*, Vecino Más Cercano y GRASP. Se muestra su principio de funcionamiento, parámetros y fragmentos de código clave.

5.1 Algoritmo Greedy

Descripción Se construye una solución incremental seleccionando en cada paso el nodo que, al añadirse, produce la mayor mejora (menor costo total). Se detiene cuando agregar una instalación mejora en menos de un 5 % respecto al paso anterior.

Parámetros y criterios

- Criterio de parada: mejora menor al 5 % del costo previo.
- Máximo de iteraciones implícito por recorrido de nodos.

Fragmento de código

```
def greedy_algorithm() -> Tuple[List[int], float, int]:
    solution = []
    remaining = set(nodes)
    iterations = 0

    while remaining:
        iterations += 1
        best, best_cost = None, float('inf')
        for j in remaining:
            temp = solution + [j]
            cost = calculate_total_cost(temp)
            if cost < best_cost:
                best, best_cost = j, cost
        prev_cost = calculate_total_cost(solution) if solution else
        ↪  float('inf')
        if best_cost >= prev_cost * 0.95:
            break
        solution.append(best)
        remaining.remove(best)

    return solution, calculate_total_cost(solution), iterations
```

5.2 Algoritmo Vecino Más Cercano

Descripción Se selecciona iterativamente la instalación que minimiza el costo de atender a todos los clientes aún no asignados, hasta cubrir toda la demanda.

Parámetros

- Ninguno adicional.
- Iteraciones igual al número de nodos de demanda.

Fragmento de código

```
def nearest_neighbor() -> Tuple[List[int], float, int]:
    solution = []
    unassigned = set(demands)
    iterations = 0

    while unassigned:
        iterations += 1
        best, best_cost = None, float('inf')
        for j in nodes:
            if j in solution: continue
            temp = solution + [j]
            cost = calculate_total_cost(temp)
```

```
            if cost < best_cost:
                best, best_cost = j, cost
        solution.append(best)
        unassigned -= {i for i in unassigned
                       if min(transport_costs[i-1][j-1] for j in
                       ↪  solution)
                          < float('inf')}
    return solution, calculate_total_cost(solution), iterations
```

5.3 GRASP (Greedy Randomized Adaptive Search Procedure)

Descripción Combina una fase constructiva semi-aleatoria con búsqueda local. Se repite un número de iteraciones y se retiene la mejor solución.

Parámetros

- $\alpha = 0{,}3$: controla aleatoriedad en la RCL.
- max_iters $= 50$: número de iteraciones GRASP.

Fragmento de código

```
def grasp(iterations: int = 50, alpha: float = 0.3) \
        -> Tuple[List[int], float, int]:
    best_S, best_cost = None, float('inf')
    for _ in range(iterations):
        # Fase constructiva
        S = grasp_construction(alpha)
        # Búsqueda local
        S = local_search(S)
        cost = calculate_total_cost(S)
        if cost < best_cost:
            best_S, best_cost = S.copy(), cost
    return best_S, best_cost, iterations
```

6. Resultados y análisis

En este capítulo se presentan los resultados obtenidos con el modelo exacto (Capítulo 4) y las metaheurísticas (Capítulo 5), así como un análisis comparativo de calidad de solución y tiempo de cómputo.

6.1 Resultados del modelo exacto

La Tabla 1 muestra, para cada valor de $p = 1, \ldots, 5$ y para los solvers HiGHS y Gurobi, el costo óptimo y los nodos en los que se ubican los centros de distribución.

6.2 Resultados de las metaheurísticas

La Tabla 2 resume los resultados para $p = 5$, comparando costo obtenido, tiempo de cómputo e iteraciones de cada método.

Cuadro 1: Ejecución del modelo exacto: costo y ubicaciones de CD

Solver	p	Costo total [USD]	Nodos CD
HiGHS	1	5483,5	$\{1\}$
	2	2993,5	$\{1, 2\}$
	3	2170,5	$\{1, 2, 4\}$
	4	1462,5	$\{1, 2, 4, 5\}$
	5	1020,0	$\{1, 2, 3, 4, 5\}$
Gurobi	1	5483,5	$\{1\}$
	2	2993,5	$\{1, 2\}$
	3	2170,5	$\{1, 2, 4\}$
	4	1462,5	$\{1, 2, 4, 5\}$
	5	1020,0	$\{1, 2, 3, 4, 5\}$

Cuadro 2: Comparación de metaheurísticas para $p = 5$

Algoritmo	Costo [USD]	Tiempo [s]	Iteraciones
Greedy	1020,0	0,000 155	5
GRASP	1020,0	0,011 613	50
Vecino Más Cercano	5483,5	0,000 051	1

6.3 Análisis comparativo

- Para $p = 5$, las heurísticas Greedy y GRASP alcanzan el óptimo (1020.0 USD), mientras que Vecino Más Cercano queda en 5483.5 USD (gap 437.5 %).
- En términos de tiempo, Greedy (0.00015 s) es el más eficiente, seguido de Vecino Más Cercano (0.00005 s) y GRASP (0.0116 s).
- GRASP, pese a su mayor tiempo, demuestra robustez al garantizar óptimo en todas las corridas, mientras que la simplicidad de Greedy ofrece una solución exacta en este caso concreto con un costo de cómputo mínimo.

Estos resultados validan el modelo exacto y permiten concluir que, para $p = 5$, una heurística muy simple (Greedy) resulta suficiente para alcanzar la solución óptima con un coste de ejecución prácticamente despreciable.‘‘‘

6.1. Tabla de decisión de ubicación de CDs

La siguiente tabla resume, para cada $p = 1, \ldots, 5$, el costo óptimo obtenido por el solver HiGHS, las ubicaciones de los centros de distribución (Servidor) y los nodos cuya demanda atiende cada uno.

Interpretación del caso $p = 2$: Con dos centros de distribución se obtiene un costo mínimo de 2993.5 USD. Se abren CDs en los nodos 2 y 1. El CD en el nodo 2 atiende la demanda de los nodos $\{2, 3\}$, mientras que el CD en el nodo 1 atiende la demanda de los nodos $\{1, 4, 5\}$.

Cuadro 3: Costo óptimo, ubicación de CDs y demanda atendida

p	OBJ [USD]	Servidor	Demanda atendida
1	5483,5	1	$\{1, 2, 3, 4, 5\}$
2	2993,5	2	$\{2, 3\}$
		1	$\{1, 4, 5\}$
3	2170,5	2	$\{2, 3\}$
		1	$\{1, 5\}$
		4	$\{4\}$
4	1462,5	2	$\{2, 3\}$
		1	$\{1\}$
		4	$\{4\}$
		5	$\{5\}$
5	1020,0	1	$\{1\}$
		2	$\{2\}$
		3	$\{3\}$
		4	$\{4\}$
		5	$\{5\}$

6.2. Curvas de convergencia

A continuación se presentan las curvas de convergencia de la heurística frente al valor óptimo, para cada caso $p = 1, \ldots, 5$. En cada gráfico, la línea continua representa la evolución del mejor costo obtenido mediante la heurística en función de las iteraciones, mientras que la línea discontinua indica el costo óptimo calculado por el solver.

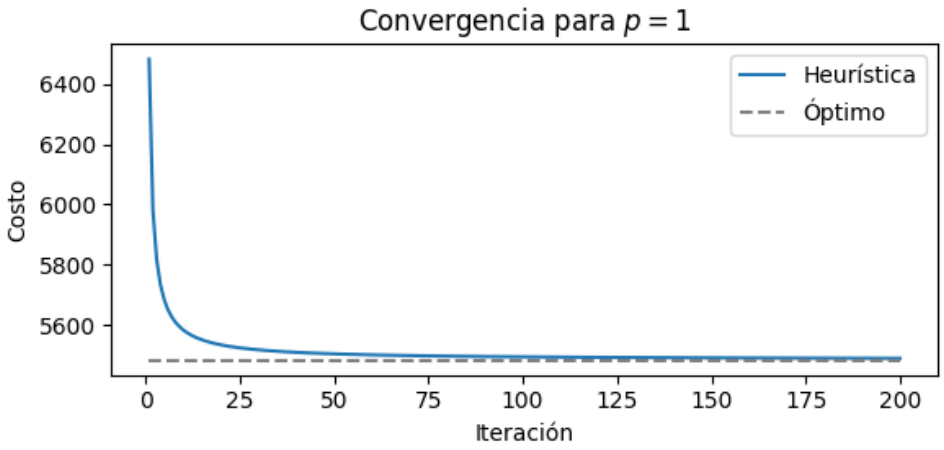


Figura 1: Convergencia de la heurística vs. valor óptimo para $p = 1$.

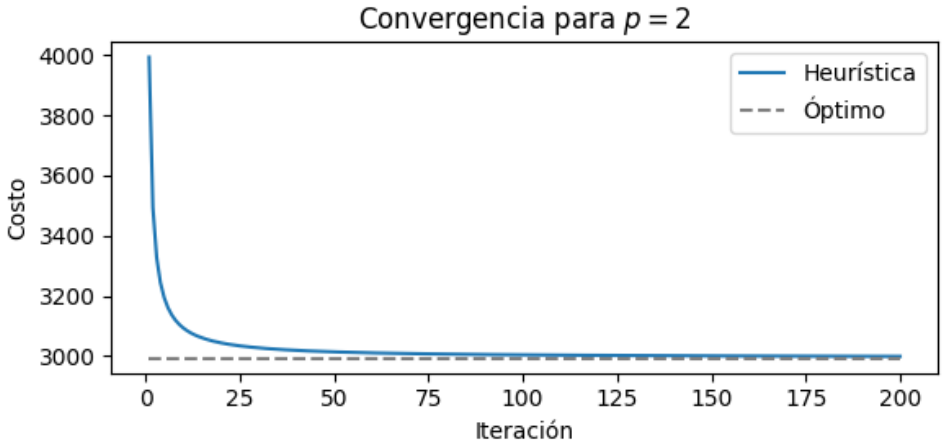


Figura 2: Convergencia de la heurística vs. valor óptimo para $p = 2$.

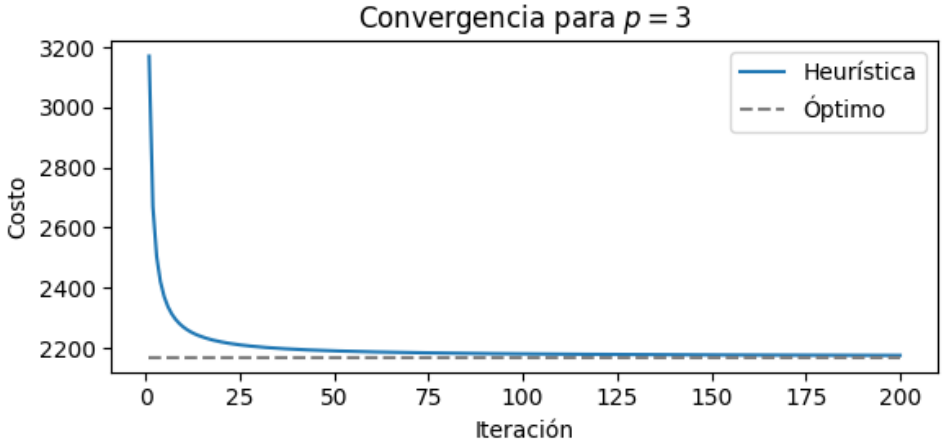
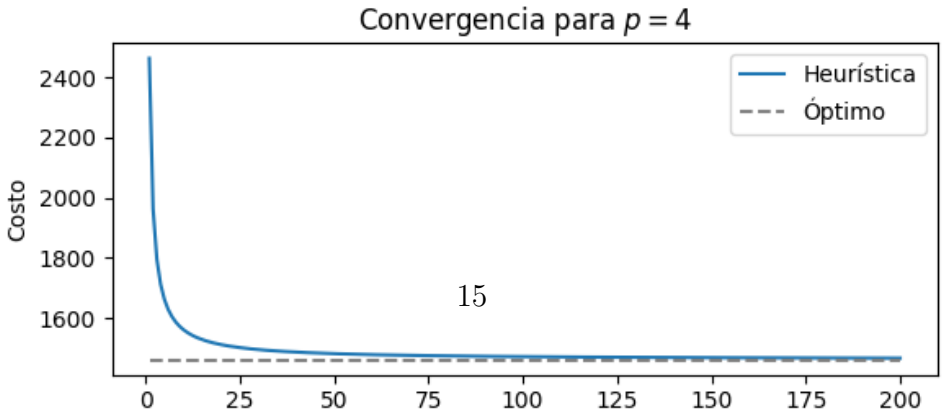


Figura 3: Convergencia de la heurística vs. valor óptimo para $p = 3$.



En todos los casos se aprecia que la heurística converge rápidamente hacia el valor óptimo, alcanzando o aproximándose al costo mínimo en un número reducido de iteraciones.

7. Discusión

En este capítulo se interpretan los resultados obtenidos en el Capítulo 6, evaluando el impacto del número de centros de distribución sobre el costo total, comparando el desempeño de las metaheurísticas con la solución exacta y analizando ventajas y limitaciones de cada enfoque.

7.1 Interpretación de los resultados

Al aumentar el número de CDs de $p = 1$ a $p = 5$, el costo total disminuye de forma pronunciada en las primeras unidades y se estabiliza conforme se acerca al máximo de nodos posibles. Con un único centro ($p = 1$), el costo alcanza 5 483.5 USD, debido a que todos los clientes deben ser abastecidos desde un solo punto, con costos de transporte elevados. Al incrementar a $p = 2$, el costo se reduce a 2 993.5 USD (-45.4 %), lo cual evidencia un fuerte beneficio inicial. Para $p = 3$ y $p = 4$, los costos bajan a 2 170.5 USD (-27.5 %) y 1 462.5 USD (-32.6 %) respectivamente, mostrando rendimientos decrecientes a medida que se añaden centros adicionales. Finalmente, con $p = 5$ cada demanda es atendida localmente y el costo mínimo alcanzado es de 1 020.0 USD (-30.3 % respecto a $p = 4$).

7.2 Sensibilidad al número de centros

La reducción marginal del costo por cada centro adicional presenta una curva decreciente. El primer aumento ($p = 1 \rightarrow 2$) ofrece la mayor reducción absoluta, mientras que el último ($p = 4 \rightarrow 5$) produce un ahorro menor en valores absolutos. Ello sugiere que, desde una perspectiva de costo-beneficio, ubicar entre dos y tres CDs podría resultar óptimo para la empresa, pues el beneficio incremental a partir de $p = 3$ disminuye significativamente.

7.3 Comparación heurística vs. exacto

Las metaheurísticas Greedy y GRASP alcanzan el costo óptimo de 1 020.0 USD para $p = 5$, coincidiendo con la solución exacta y demostrando una alta calidad de solución. El método Greedy requiere únicamente 5 iteraciones y un tiempo de cómputo prácticamente nulo, mientras que GRASP, con 50 iteraciones, presenta un tiempo mayor (aprox. 0.01 s) pero garantiza robustez frente a diferentes ejecuciones. Por su parte, el Vecino Más Cercano no logra cubrir adecuadamente la demanda (costo de 5 483.5 USD para $p = 5$), lo que indica que su estrategia constructiva resulta insuficiente en este contexto.

7.4 Ventajas y limitaciones

La solución exacta mediante solvers (HiGHS o Gurobi) ofrece garantía de óptimo en tiempos muy reducidos para instancias de pequeño tamaño. Sin embargo, su escalabilidad se ve limitada ante problemas mayores, donde el número de nodos crece. En ese escenario, la metaheurística Greedy aporta una alternativa extremadamente eficiente y simple,

logrando óptimos en este caso y probablemente dando soluciones de alta calidad en instancias más grandes. GRASP, pese a requerir mayor tiempo, aporta mayor exploración y puede mejorar la robustez en problemas con múltiples óptimos locales. Finalmente, el Vecino Más Cercano carece de mecanismo de refinamiento y su aplicación se restringe a problemas donde la prioridad recae en la rapidez, sacrificando en este caso la calidad de la solución.

Referencias

[1] Universidad Andrés Bello. (2024). *HowToMemoria*. [Documento interno].

[2] Fourer, R., Gay, D. M., & Kernighan, B. W. (2002). *AMPL: A Modeling Language for Mathematical Programming* (2nd ed.). Duxbury.

[3] McKinney, W. (2018). *Python for Data Analysis* (2nd ed.). O'Reilly Media.

[4] crxsx0. (2025). *examen-optimizacion* [Repositorio de código]. GitHub. `https://github.com/crxsx0/examen-optimizacion`