Set Covering Problem

- Juan Pablo Echeagaray González
- A00830646

- Diseño de algoritmos matemáticos bio-inspirados
- 30 de septiembre del 2022

In [ ]:
```
from ortools.linear_solver import pywraplp
import numpy as np
```

Modelación matemática

Sean los conjuntos:

$$M = \{1, 2, \dots, m\}$$
$$N = \{1, 2, \dots, n\}$$

Sean los parámetros:

$$c_j := \text{costo de la instalación } j, \forall j \in N$$
$$a_{ij} := \text{Si la región } i \in M \text{ puede ser atendida por el centro } j \in N, \forall i \in M, \forall j \in N$$

Sean las variables:

$$x_j := \text{Si se instala el centro } j, \forall j \in N$$

Entonces:

$$\begin{aligned} \min \quad & \sum_{j \in N} c_j x_j \\ \text{st:} \quad & \sum_{j \in N} a_{ij} x_j \geq 1, \forall i \in M \\ & x_j \in \mathbb{B}, \forall j \in N \end{aligned}$$

Datos del problema

In [ ]:
```
def create_data(heuristic: bool = False):
    c = np.array([20, 10, 30, 25])
    A = np.array([[0, 1, 1, 0],
                  [1, 1, 0, 1],
                  [1, 0, 1, 0],
                  [0, 1, 0, 1],
                  [1, 1, 0, 1]])
    cities, centers = A.shape

    if heuristic:
        return c, A

    return c, A, cities, centers
```

Aproximación por un método heurístico

- **Idea**: Calcular relevancia de un centro como la proporción de su cobertura y el costo. Ir removiendo los centros abiertos, y detener el código cuando todas las ciudades hayan sido cubiertas.

```
Algoritmo:
init costos
init A
abiertos = []
costs = []
while True:
    relevancias = relevancia(A, costos)
    j = argmax(relevancias)
    Remover ciudades cubiertas por j
    Remover j de A, de costos
    Agregar j a abiertos
    Agregar costos[j] a costs

    Si A ya no tiene filas:
        break

    return abiertos, costos * abiertos
```

In [ ]:
```python
def heuristic_approach(cost, A, res: bool = False):

    open_centers = []
    # Helper list to keep track of the opened centers (fuzzy logic)
    costs = []

    while True:
        reach = np.sum(A, axis=0)
        relevance = reach / cost

        best_center = np.argmax(relevance)
        open_centers.append(best_center)
        costs.append(cost[best_center])

        covered = np.where(A[:, best_center] == 1)[0]
        A = np.delete(A, best_center, axis=1)
        A = np.delete(A, covered, axis=0)
        cost = np.delete(cost, best_center)

        # If all cities are covered, stop
        if A.shape[0] == 0:
            break

    results = {'Z': np.sum(costs), 'centers': open_centers}

    if res:
        return results
```

In [ ]:
```python
heuristic_approach(*create_data(True), True)
```

Out[ ]:
```
{'Z': 30, 'centers': [1, 0]}
```

Solución con `solver`

In [ ]:
```python
def solver_solution(res: bool):
    solver = pywraplp.Solver.CreateSolver('BOP')
    costs, A, cities, centers = create_data()

    # Variables
    # x[i] abrir centro i
    x = [solver.BoolVar(f'x_{i}') for i in range(centers)]

    # Constraints
    for j in range(cities):
        solver.Add(sum(A[j, i] * x[i] for i in range(centers)) >= 1)

    # Objective Function
    solver.Minimize(sum(x * costs))

    status = solver.Solve()

    if status == pywraplp.Solver.OPTIMAL and res:
        return {'Z': solver.Objective().Value(), 'Centers': [i for i in range(centers) if x[i].solution_value()
```

In [ ]:
```python
solver_solution(True)
```

Out[ ]:
```
{'Z': 30.0, 'Centers': [0, 1]}
```

Comparación de tiempos de cómputo

In [ ]:
```python
heuristic = %timeit -n 1000 -r 10 -o heuristic_approach(*create_data(True))
solver = %timeit -n 1000 -r 10 -o solver_solution(False)
```

```
131 µs ± 27.7 µs per loop (mean ± std. dev. of 10 runs, 1000 loops each)
904 µs ± 68.2 µs per loop (mean ± std. dev. of 10 runs, 1000 loops each)
```

De esta pequeña prueba, puedo ver que el desempeño del algoritmo heurístico que propongo es bastante buena. Al menos para un caso sencillo, logra llegar a la optimalidad en un tiempo de cómputo aproximadamente 1 orden de magnitud menor que el de un solver de talla comercial.

El que en esta prueba el resultado obtenido haya sido el óptimo no es evidencia suficiente de que mi propuesta encuentre soluciones de alta calidad para casos más complicados. Tendría que hacer más pruebas con más problemas para ver si su desempeño es el suficiente como para ser aplicado en casos más complejos.