.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(TB024) Curso Buchwald - Genender

Trabajo Práctico 3
Problemas NP-Completos para la defensa de la Tribu del Agua

30 de octubre de 2025

Alen Davies
107.084

Hugo Huzan
67.910

Joaquin Vedoya
110.282

Franco Altieri Lamas
105.400

Trabajo Práctico 3:
Problemas NP-Completos para la defensa de la Tribu del Agua

1. Introducción

Es el año 95 DG. La Nación del Fuego sigue su ataque, esta vez hacia la Tribu del Agua, luego de una humillante derrota a manos del Reino de la Tierra, gracias a nuestra ayuda. La tribu debe defenderse del ataque.

El maestro Pakku ha recomendado hacer lo siguiente: Separar a todos los Maestros Agua en k grupos $(S_1, S_2, \cdots, S_k)$. Primero atacará el primer grupo. A medida que el primer grupo se vaya cansando entrará el segundo grupo. Luego entrará el tercero, y de esta manera se busca generar un ataque constante, que sumado a la ventaja del agua por sobre el fuego, buscará lograr la victoria.

En función de esto, lo más conveniente es que los grupos estén parejos para que, justamente, ese ataque se mantenga constante.

Conocemos la fuerza/maestría/habilidad de cada uno de los maestros agua, la cual podemos cuantificar diciendo que para el maestro i ese valor es x_i, y tenemos todos los valores $x_1, x_2, \cdots, x_n$ (todos valores positivos).

Para que los grupos estén parejos, lo que buscaremos es minimizar la adición de los cuadrados de las sumas de las fuerzas de los grupos. Es decir:

$$\text{mín} \sum_{i=1}^{k} \left(\sum_{x_j \in S_i} x_j \right)^2$$

El Maestro Pakku nos dice que esta es una tarea difícil, pero que con tiempo y paciencia podemos obtener el resultado ideal.

Resolución

2. El Problema es NP

Se mostrará que el problema pertenece a la clase de complejidad **NP** presentando un *Certificador Eficiente* que puede validar un *certificado* en tiempo polinomial.

Certificado.

Un vector $C = (S_1, \cdots, S_k)$ siendo cada S_j un vector de índices que indica que x_i pertenecen a cada grupo.

Algoritmo

El certificador debe verificar:

- Que la cantidad de vectores S_j sea K
- Que cada x_i pertenezca a algún vector S_j
- Que cada x_i aparezca una sola vez.
- Que la suma de los cuadrados de las sumas de los elementos de cada S_j sea menor o igual a B.

TODO: Mostrar que es polinómico

3. Backtracking

```python
import math


def suma_cuadrados(sumas):
    return sum(s * s for s in sumas)


def backtracking(i, valores, k, sumas, grupos, mejor_valor, mejor_particion):

    if i == len(valores):
        actual = suma_cuadrados(sumas)
        if actual < mejor_valor:
            return actual, [list(g) for g in grupos]
        return mejor_valor, mejor_particion

    if suma_cuadrados(sumas) >= mejor_valor:
        return mejor_valor, mejor_particion

    for g in range(k):
        grupos[g].append(i)
        sumas[g] += valores[i]

        mejor_valor, mejor_particion = backtracking(
            i + 1, valores, k, sumas, grupos, mejor_valor, mejor_particion
        )

        sumas[g] -= valores[i]
        grupos[g].pop()

    return mejor_valor, mejor_particion


def resolver(k, valores):
    sumas = [0] * k
    grupos = [[] for _ in range(k)]
    mejor_valor = math.inf
    mejor_particion = None

    mejor_valor, mejor_particion = backtracking(
        0, valores, k, sumas, grupos, mejor_valor, mejor_particion
    )

    return mejor_valor, mejor_particion
```

4. Programación Lineal

5. Ejemplo de Ejecuciones

6. Mediciones

7. Conclusiones