



FACULTAD DE INGENIERÍA

TEORÍA DE ALGORITMOS  
(TB024) CURSO BUCHWALD - GENENDER

# Trabajo Práctico 3

## Problemas NP-Completo para la defensa de la Tribu del Agua



12 de noviembre de 2025

Alen Davies  
107.084

Hugo Huzan  
67.910

Joaquin Vedoya  
110.282

Franco Altieri Lamas  
105.400

## Trabajo Práctico 3: Problemas NP-Completo 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  $(S_1, S_2, \dots, S_k)$ . 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  $x_1, x_2, \dots, x_n$  (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:

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

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  $C = (S_1, \dots, S_k)$  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

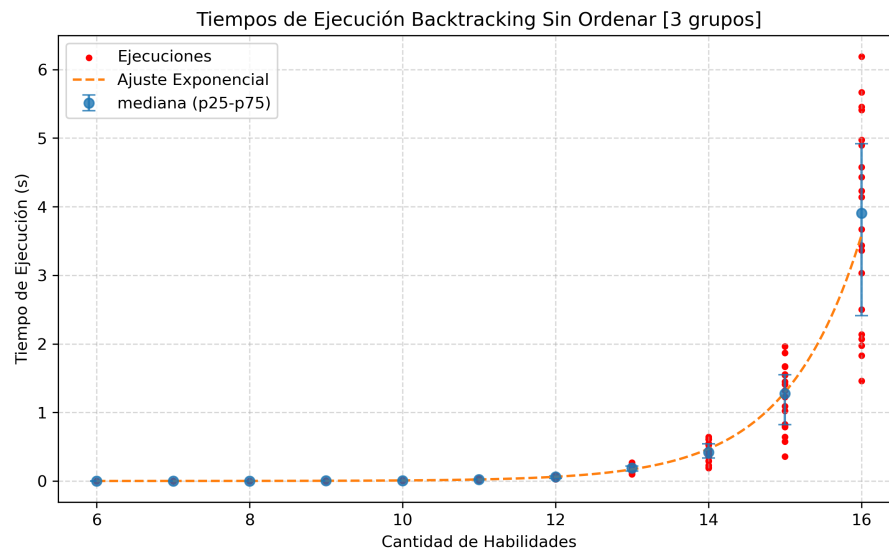
### 3. Backtracking

```
1 import math
2
3
4 def suma_cuadrados(sumas):
5     return sum(s * s for s in sumas)
6
7
8 def backtracking(i, habilidades, k, sumas, particion, mejor_valor, mejor_particion)
9     :
10
11     sum_cuad_actual = suma_cuadrados(sumas)
12
13     if sum_cuad_actual >= mejor_valor:
14         return mejor_valor, mejor_particion
15
16     if i == len(habilidades):
17         if sum_cuad_actual < mejor_valor:
18             mejor_valor = sum_cuad_actual
19             mejor_particion = [list(g) for g in particion]
20         return mejor_valor, mejor_particion
21
22     valor, indice = habilidades[i]
23
24     # orden = sorted(range(k), key=lambda x: sumas[x])
25     ya_uso_vacio = False # para evitar permutaciones de los conjuntos
26     # for g in orden:
27     for g in range(k):
28         if sumas[g] == 0:
29             if ya_uso_vacio:
30                 continue
31             ya_uso_vacio = True
32
33     particion[g].append(indice)
34     sumas[g] += valor
35
36     mejor_valor, mejor_particion = backtracking(
37         i + 1, habilidades, k, sumas, particion, mejor_valor, mejor_particion)
38
39     sumas[g] -= valor
40     particion[g].pop()
41
42     return mejor_valor, mejor_particion
43
44 def resolver(k, habilidades, ordenar=True):
45
46     tuplas = sorted([(v, i)
47                     for i, v in enumerate(habilidades)], reverse=True) if ordenar
48     else [(v, i) for i, v in enumerate(habilidades)]
49
50     mejor_valor, mejor_particion = math.inf, None
51
52     sumas = [0] * k
53     particion = [[] for _ in range(k)]
54
55     return backtracking(0, tuplas, k, sumas, particion, mejor_valor,
56                         mejor_particion)
```

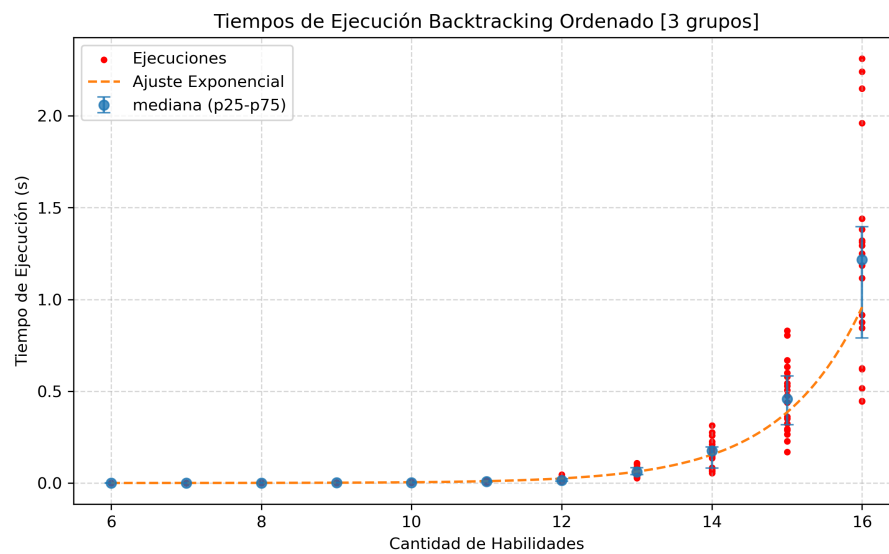
## 4. Programación Lineal

## 5. Ejemplo de Ejecuciones

## 6. Mediciones



(a) Sin Ordenar



(b) Datos Ordenados

Figura 1: Comparación de Tiempos de Ejecución Según Orden

## 7. Conclusiones