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

Facultad de Ingeniería de la Universidad de Buenos Aires

Teoría de Algoritmos

Cátedra Buchwald-Genender

Gómez Belis, Sofía
Padrón: 109358
email: sgomezb@fi.uba.ar

Llanos Pontaut, Valentina
Padrón: 104413
email: vllanos@fi.uba.ar

Orsi, Tomas Fabrizio
Padrón: 109735
email: torsi@fi.uba.ar

Indice

1 Introducción

1.1 Descripción y objetivo

Continuando con el ataque de la Nación del Fuego sobre el resto de las naciones, esta vez es la Tribu del Agua la que requiere de nuestra ayuda para defenderse.

Cada maestro agua tiene una fuerza o habilidad positiva x_i, y contamos con el conjunto de todos los valores $(x_i, x_2, \ldots, x_n)$. Basándonos en estos, el maestro Pakku desea separar los maestros en `k` grupos $(S_1, S_2, \ldots, S_k)$ parejos tal que cuando un grupo se canse, entrará el siguiente en el combate, obteniendo un ataque constante que les permita salir victoriosos, aprovechando también la ventaja del agua por sobre el fuego.

Para que los grupos estén lo más parejos posibles, nos han encomendado minimizar la adición de los cuadrados de las sumas de las fuerzas de los grupos:

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

En este trabajo desarrollaremos algoritmos de backtracking, programación lineal y posibles aproximaciones buscando resolver el problema planteado con el objetivo de ayudar a los maestros de la Tribu del Agua a derrotar a la Nación del Fuego.

2 Demostración de problema NP-Completo

Los problemas NP-Completos son los problemas más difíciles de NP. Cualquier problema en NP puede ser reducido a uno NP-Completo.

Para demostrar que el problema de la Tribu del Agua es NP-Completo, primero debemos demostrar que se encuentra en NP. Posteriormente, si logramos hacer una reducción polinomial de un problema NP-Completo a éste, entonces esto implica que también se trata de un problema NP-Completo. Ésto se debe a que la reducción $Y \leq_p X$ implica que la dificultad de resolver `Y` se reduce a la dificultad de resolver `X`, o que `X` es al menos tan difícil de resolver como `Y`. Si `Y` es un problema NP-Completo, entonces `X` es al menos tan difícul de resolver como un problema NP-Completo.

Para realizar la demostración y la reducción, necesitamos basarnos en el problema de decisión:

Dados una secuencia de `n` fuerzas de maestros agua $(x_i, x_2, \ldots, x_n)$, y dos números `k` y `B`, ¿existe una partición en `k` subgrupos $(S_1, S_2, \ldots, S_k)$ tal que:

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

y cada elemento x_i esté asignado a solamente un grupo?

2.1 Problema NP

Un problema se encuentra en NP si existe un certificador eficiente para el mismo. Es decir, si puede ser verificado o validado en tiempo polinomial.

Tenemos los siguientes datos:

- Habilidades de los maestros → se presentan en forma de tupla (nombre maestro, fuerza)
- Cantidad de subconjuntos → `k`
- Cota del coeficiente → `B`
- Resultado a validar → subconjuntos S_i que contienen los nombres de los maestros del grupo correspondiente

También tenemos las siguientes restricciones:

- Cada maestro debe estar asignado a un solo grupo
- El resultado debe contener `k` grupos
- La sumatoria resultante debe ser a lo sumo `B`

Dados estos datos y restricciones, proponemos el siguiente certificador.

```python
def validador_problema_tribu_del_agua(habilidades_maestros_agua, k, B, S):
    if len(S) != k: #O(1)
        return False
    suma = 0 #O(1)
    maestros = set() #O(1) Cada maestro debe estar en solo un grupo
    dic_habilidades = dict() #O(1)

    for info_maestro in habilidades_maestros_agua: #O(n)
        maestro, habilidad = info_maestro #O(1)
        dic_habilidades[maestro] = habilidad #O(1)

    # O(n * k)
    for grupo in S: # k grupos: O(k)
        suma_grupo = 0 #O(1)
        if len(grupo) == 0:
            return False
        for maestro in grupo: # En el peor caso O(n)
            if maestro in maestros or maestro not in dic_habilidades:
                return False # El maestro dado está en más de un grupo
            maestros.add(maestro) #O(1)
            suma_grupo += dic_habilidades[maestro] #O(1)
        suma += suma_grupo ** 2 #O(1)

    if suma > B:  # O(1)
```

```
25          return False
26
27      for maestro, _habilidad in habilidades_maestros_agua: #O(n)
28          if maestro not in maestros:  # O(1)
29              return False # No tiene grupo asignado
30
31      return True
```

Para que el verificador presentado pueda ser considerado un certificador eficiente, debe ejecutarse en tiempo polinomial. Por lo tanto, procedemos a explicar y justificar la complejidad del código.

Realizamos operaciones aritméticas, asignaciones y verificaciones. Todas estas son operaciones $O(1)$ puesto que consumen tiempo constante según la documentación oficial . Recorremos dos veces la lista de maestros y habilidades con diferentes propósitos. Como contiene `n` tuplas, cada ciclo `for` tiene un costo $O(n)$. Con el objetivo de obtener la adición de los cuadrados de las sumas de las fuerzas de los grupos y compararla con `B`, tenemos dos ciclos anidados. El segundo recorre los maestros de un grupo que, en el peor caso, son `n`. Esto se realiza `k` veces, por lo que en total es $O(k \times n)$.

Entonces,

$$T(n) = O(1) + 2 \cdot O(n) + O(k \times n) = O(k \times n)$$

En el peor caso, $k = n$ y

$$T(n) = O(n \times n) = O(n^2)$$

Ambas cotas, $O(k \times n)$ y $O(n^2)$ conllevan tiempo polinomial. Por lo tanto, podemos afirmar que existe un certificador eficiente para el problema de la Tribu del Agua y entonces este problema está en NP.

2.2 Reducción

3 Complejidad algorítmica

3.1 Complejidad lectura de archivos

3.2 Complejidad algoritmo de Backtracking

3.3 Complejidad algoritmo de programación lineal

3.4 Complejidad algoritmo de aproximación

3.5 Efecto de las variables sobre el algoritmo

4 Ejemplos de ejecución