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 polinomialmente 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 en 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 Problema 2-Partition

Para demostrar que el problema de la tribu del agua es NP-Completo haremos uso del problema 2-Partition. Este se define de la siguiente manera: se tiene un conjunto de n elementos, cada uno con un valor, y se desea separarlos en 2 subconjuntos tal que la suma de los valores de cada uno sea igual para ambos. Procederemos a justificar que se trata de un problema NP-Completo.

2.2.1 Problema en NP

Enunciamos el problema de decisión asociado: dado un conjunto $\{a_1, a_2, \ldots, a_n\}$, ¿existen dos subconjuntos S_1 y S_2 tales que $\sum_{x \in S_1} x = \sum_{x \in S_2} x$?

A continuación mostramos la implementación de un validador:

```
1  def validador_2_partition(valores, s1, s2):
2      if sum(s1) != sum(s2) or len(s1) >= len(valores) or len(s2) >= len(valores):
3          return False
```

```
4
5       apariciones = {}
6       for elemento in s1:
7           apariciones[elemento] = apariciones.get(elemento, 0) + 1
8
9       for elemento in s2:
10          apariciones[elemento] = apariciones.get(elemento, 0) + 1
11
12      valores_no_repetidos = {}
13      for valor in valores:
14          nuevoValor = valores_no_repetidos.get(valor, 0) + 1
15          valores_no_repetidos[valor] = nuevoValor
16
17      if len(apariciones) != len(valores_no_repetidos):
18          return False
19
20      for valor, cantidad in apariciones:
21          if valor not in valores_no_repetidos:
22              return False
23          elif valores_no_repetidos[valor] != cantidad:
24              return False
25
26      return True
```

Hay tres ciclos `for`. Uno de ellos recorre todos los valores del conjunto. Considerando que hay `n`, es una operación $O(n)$. Los conjuntos S_1 y S_2 deberían contener en total n elementos, pudiendo uno de ellos tener hasta $n-1$. En total, ambos conllevan $O(n)$. Por lo tanto, la complejidad temporal del validador es $O(n)$, constituyendo un certificador eficiente. Como el problema 2-Partition puede ser verificado en tiempo polinomial, podemos afirmar que se encuentra en NP.

2.2.2 Reducción

Como hemos visto en clase, el problema de subset sum es NP-Completo. Utilizaremos esta conclusión para demostrar que 2-Partition también lo es.

Dado un conjunto de números $S = \{a_1, a_2, \ldots, a_n\}$ y un número B, queremos saber si hay un subconjunto de S cuya suma sea igual a B.

Para realizar la reducción $SS \leq_p 2P$, debemos definir cuál es la entrada (conjunto) de la caja negra que resuelve 2-Partition. Sea $s = sum(S)$ la suma de los valores del conjunto S, definimos $S' = S \cup \{s - 2 \cdot B\}$. Aplicamos el algoritmo en S'. Agregar un elemento a un conjunto es $O(1)$, mientras que si se desea copiar S para no modificarlo, la operación es lineal en el tamaño del set. Por lo tanto, esta transformación es polinomial.

Si $T \subseteq S$ es un subconjunto con suma B:

- Sea $O = S \setminus T$, con suma $o = s - B$, los elementos que "sobran".

- Sea $T' = T \cup \{s - 2 \cdot B\}$, cuya suma es $B + (s - 2 \cdot B) = s - B$. Es el subconjunto buscado, unido con el valor agregado a S.
- $O \cup T' = S'$

Entonces, la suma de T' y O es la misma, $s-B$, lo que significa que podemos dividir S' en dos subconjuntos con la misma suma.

Luego, existe un subset sum de valor B en S si y solo si existe un 2-Partition en S':

$\rightarrow$ Si existe un conjunto de números en S que suman B, entonces los números restantes en S suman $s - B$. Por lo tanto, existe una partición de `S'` en dos tal que cada partición suma $s - B$: $O = S \setminus T$ y $T' = T \cup \{s - 2 \cdot B\}$.

$\leftarrow$ Digamos que existe una partición de `S'` en dos conjuntos tal que la suma de cada conjunto es $s - B$. Uno de estos conjuntos contiene el número $s - 2 \cdot B$. Eliminando este número, obtenemos un conjunto de números cuya suma es B, y todos estos números están en S.

Como esta reducción conlleva una transformación polinomial, podemos afirmar que 2-Partition es un problema NP-Completo.

Ejemplo 1: La entrada de Subset Sum es $S = \{3, 1, 4, 2, 2\}$ y $B = 6$. Como la suma de los valores de S es 12, agregamos el elemento $12 - 2 \cdot 6 = 0$ $S' = \{3, 1, 4, 2, 2, 0\}$. Este conjunto puede ser particionado en $\{3, 1, 2\}$ y $\{4, 2, 0\}$, ambas con suma 6, donde $T = \{3, 1, 2\}$ es un subconjunto de S que suma 6. Por lo tanto, la caja negra que resuelve el problema de partición, nos devolverá que es posible encoentrar un subset sum que sume B.

Ejemplo 2: La entrada de Subset Sum es $S = \{5, 3, 2, 7\}$, que suma 17, y $B = 10$. Agregamos el valor $17 - 2 \cdot 10 = -3$. Entonces, $S' = \{5, 3, 2, 7, -3\}$ y se puede dividir en $\{7\}$ y $\{2, 5, 3, -3\}$, donde $T = \{2, 5, 3\}$ es el subconjunto que suma 10. Nuevamente existe solución. En cambio, si $B = 6$, tendríamos $S' = \{5, 3, 2, 7, 5\}$ no puede dividirse en dos subconjuntos que sumen $\frac{22}{2=11}$, por lo que no existirá un subset sum con $B = 6$ en S.

2.3 Reducción del Problema del Balanceo de Carga

2.3.1 Introducción

Habiendo establecido que el problema pertenece a NP, ahora vamos a realizar una reducción de un problema NP-Completo a este problema. Para esta reducción, decidimos reducir el problema del Balanceo de Carga (que llamaremos B.C.) al problema de la tribu del agua (que llamaremos T.A.). Esto es decir, nuestra reducción polinomial va a ser de la forma:

$$BC \leq_p TA$$

BC plantea lo siguiente: *Dadas m Máquinas y un conjunto n de trabajos, donde cada trabajo j toma un tiempo Tj, se desea asignar el trabajo en las Máquinas de forma balanceada. Dada una asignación A(i) para la Máquina i, su tiempo de trabajo es:* $T_i = \sum_{j \in A(i)} t_j$. *Y queremos encontrar la asignación que*

minimice el máx valor de Ti, que también representa el tiempo que se tardará en finalizar todos los trabajos.

Si escribimos en forma de ecuación lo que plantea BC, obtenemos la siguiente expresión:

$$\min(\max(\sum_{j \in A(i)} t_j))$$

La versión de decisión de este problema dice: Dado un numero m de maquinas, una succione n de trabajos, y un numero C, determinar si existe un asignación de trabajos en las m maquinas, tal que el máximo T_i sea menor o igual a C.

Cabe destacar que en clase nosotros estudiamos al problema de BC como un problema NP-Hard; resta aun demostrar que es un problema NP-Completo.

2.3.2 Demostración de problema NP-Completo

FALTA REDUCCIÓN DESDE 2-PARTITION!

2.3.3 Desarrollo

El problema de decisión de BC, busca minimizar el máximo valor de las maquinas; mientras que TA busca lograr que todos los subgrupos tengan una duración pareja. La pregunta entonces es: ¿Hay relación entre estas dos búsquedas?. ¡La respuesta es que si! Buscar minimizar el máximo valor de las maquinas, está implícitamente buscando lograr que todas las máquinas tengan una duración pareja. Para reducir el tiempo de uso de una maquina, yo tengo que darle alguno de sus trabajos a otra máquina. Cuanto mas trabajos yo pueda darle a otra maquina (sin que esta se convierta en la nueva maquina de mayor duración), mas voy a lograr minimizar el valor máximo de las maquinas. Esto implica que en el momento que tenga todas las máquinas con igual T_i, es el momento en donde va a estar el mínimo valor máximo de T_i. Esto se ve en que si una maquina le da un trabajo a otra, una maquina va a estar por debajo de todas las anteriores, pero la otra va a tener un valor por encima de todas estas. Esto implica un mayor nivel máximo de T_i.

Basándonos, podemos realizar la siguiente reducción: Por cada trabajo n_i, definimos un maestro agua x_i. Por cada máquina m_i definimos un subgrupo S_i (m sería nuestro k). Luego tenemos C, el cual representa una cota máxima para el máximo valor de T_i. Para transformalo a B en tiempo polinomial, tenemos que elevar C al cuadrado y sumarle a este nuevo valor todo los otros valores de T_i al cuadrado. Esto nos va a dar una cota equivalente al problema de desicion de TA. Entonces, despues de realizar estas transformaciones, le tenemos que pasar esta instancia de problema al validador de problemas de TA, el cual nos va a decir si la solución es válida o no.

2.4 Reducción del Problema 2-Partition

Como hemos demostrado que el Problema 2-Partition es NP-Completo, lo utilizaremos para realizar una segunda reducción y comprobar nuevamente que

el problema de la tribu del agua es también NP-Completo. Además, debido a la propiedad transitiva de las reducciones tenemos que, como $2P \leq_p BC$ y $BC \leq_p TA$, entonces $2P \leq_p TA$.

Dados los datos de entrada del problema de 2-Partition, $\{a_1, a_2, \ldots, a_n\}$, podemos realizar la siguiente transformación polinomial que nos permitirá resolver el problema de decisión asociado (y enunciado previamente) utilizando la caja negra que resuelve el de la tribu del agua:

- $k = 2$ porque según la definición de 2-Partition necesitamos dividir el conjunto en dos grupos.
- La fuerzas de los maestros $\{x_1, x_2, \ldots, x_n\}$ serán $\{a_1, a_2, \ldots, a_n\}$.
- B es el valor obtenido cuando las sumas de los elementos en cada subconjunto son iguales. Es decir, suponiendo que el problema de decisión de 2-Partition tiene solución y sabiendo que en ese caso la suma de cada subconjunto es s = $\frac{sum(A)}{2}$, entonces $B = s^2 + s^2$.

Esta transformación requiere recorrer el conjunto $A = \{a_1, a_2, \ldots, a_n\}$ para crear las tuplas (maestro, habilidad), lo cual implica una operación $O(n)$, que es polinomial.

El problema original de la partición tiene una solución si y solo si es posible obtener dos grupos de maestros tal que la adición de los cuadrados de las sumas de las habilidades en cada grupo sea mínima y menor o igual a B.

$\rightarrow$ Si el conjunto A puede ser particionado en 2 subconjuntos tales que la suma de los elementos en cada uno sea igual, significa que los 2 grupos de maestros estarán perfectamente parejos. Obtener conjuntos de habilidades parejas es justamente el objetivo de minimizar la adición de los cuadrados de las sumas de las fuerzas. En este caso, los cuadrados de las sumas serán iguales y, por lo tanto, la suma de los mismos será mínima. Si resolviéramos el problema de decisión para $V > B$, también llegaríamos a la conclusión de que existe solución. En cambio, si $V = B - 1$, no se puede resolver. Por lo tanto, B es el coeficiente mínimo.

$\leftarrow$ Si existe una solución del problema de la tribu del agua, significa que se puede dividir a los maestros en 2 grupos tal que la adición de los cuadrados de las sumas de cada uno es a lo sumo B. Como B es el resultado de sumar dos veces s^2, esto implica que los 2 subconjuntos tienen suma s, por lo que el problema de 2-partition tiene solución.

Ejemplo: $A = \{1, 2, 3\}$. Realizando la transformación, $k = 2$, $B = (\frac{6}{2})^2 + (\frac{6}{2})^2 = 18$ y $maestros = \{(\text{""}, 1), (\text{""}, 2), (\text{""}, 3)\}$. Los nombres de los maestros no nos interesan porque no queremos obtener una solución para 2-Partition, sino saber si existe una. Existen múltiples combinaciones de valores para los dos subconjuntos, pero nos quedaremos con la que minimice la suma de los cuadrados, respetando que debe ser a lo sumo B,

1. $S_1 = 1$ y $S_2 = 2, 3 \rightarrow$ El coeficiente es $1^2 + (2 + 3)^2 = 26 > B$. No es solución.

2. $S_1 = 1, 2$ y $S_2 = 3 \rightarrow$ El coeficiente es $(1+2)^2+3^2 = 18 = B$. Es solución.

3. $S_1 = 1, 3$ y $S_2 = 2 \rightarrow$ El coeficiente es $(1+3)^2 + 2^2 = 20 > B$. No es solución.

Como existe solución para el problema de la tribu del agua, entonces también la existe para 2-Partition.

De esta forma, hemos demostrado que el problema de la tribu del agua es al menos tan difícil como el problema de partición, que es NP-completo. Como la reducción realizada es polinomial, podemos afirmar que el problema de la tribu del agua es también NP-Completo.

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

5 Mediciones de tiempo

5.1 Algoritmo de backtracking

5.2 Algoritmo de programación lineal

5.3 Algoritmo de aproximación

6 Conclusión