

# Ejercicio 3

## El Problema de los Subuniversos

### 1. Ejercicio

Se tiene un universo principal  $U$  con varias leyes.

$$U = \{l_1, l_2, \dots, l_n\}$$

Sean  $C_1, C_2, \dots, C_k$  subuniversos donde cada uno presenta un subconjunto de leyes del universo principal. Se desea saber si existe un conjunto de estos subuniversos tal que su unión formen el universo principal y que sean disjuntos entre sí.

### 2. Solución

#### 2.1. Cobertura exacta de conjuntos / Exact Set Cover

Analizando el ejercicio podemos ver que es una instancia de un problema NP-Completo conocido, Exact Set Cover o Exact Cover.

Definición (Exact Set Cover):

- Entrada:
  - Un conjunto finito  $X$ (universo)
  - Una colección  $S$  de subconjuntos de  $X$
- Pregunta:
  - ¿Existe una subcolección  $T \subseteq S$  tal que cada elemento de  $X$  pertenece exactamente a un subconjunto en  $T$ ?

Mapeo de Instancias

- Universo:
  - Asignamos  $U = X$
- Subconjuntos:
  - Asignamos  $C = S$
- Objetivo:
  - Encontrar una subcolección de  $C$  que cubra cada elemento de  $U$  exactamente una vez.

Como podemos ver la estructura y pregunta de ambos problemas son idénticas. Por lo que podemos afirmar que el problema de los subuniversos es una instancia de Exact Set Cover.

## 2.2. Demostración NP-Completo

### Demostración NP:

Para demostrar que el problema está en NP, tenemos que verificar que en tiempo polinomial una solución candidata sea válida.

### Verificación:

**Cobertura Completa:** Verificamos que la unión de los subconjuntos seleccionados es igual a  $U$ . Podemos construir la unión iterando sobre cada conjunto seleccionado y agregando sus elementos a un conjunto auxiliar.

**Cobertura Exacta:** Verificamos que no hay solapamientos entre los conjuntos seleccionados. Mantenemos un conjunto auxiliar y comprobamos que cada elemento se agrega exactamente una vez. Si encontramos un elemento que ya está en el conjunto auxiliar al procesar un nuevo subconjunto, entonces hay solapamiento.

**Tiempo Requerido:**  $O(\sum_{s \in S'} |s|)$  donde  $S'$  es la subcolección seleccionada.

La verificación se puede realizar en tiempo polinomial respecto al tamaño del universo  $U$  y el total de elementos en los subconjuntos seleccionados.

**Demostrar NP-Hard:** Demostremos que el enigma de los subuniversos es NP-Hard mediante una reducción desde el problema SAT que es conocido por ser NP-Completo.

### Descripción de SAT:

#### ■ Entrada:

- Un conjunto de variables booleanas  $B = \{b_1, b_2, \dots, b_n\}$
- Una fórmula  $f$  en forma normal conjuntiva (FNC), donde cada cláusula  $C_i$  es la disyunción de literales (variables o su negación).

#### ■ Pregunta:

- ¿Existe una asignación de valores de verdad a las variables en  $B$  que satisfaga todas las cláusulas de  $f$ ?

### Reducción

Demostremos que Exact Set Cover es NP-Hard. Para ello vamos a utilizar una reducción de un problema conocido que sabemos que es NP-Hard a este. Utilizaremos SAT, en este problema se tiene una expresión booleana en forma normal conjuntiva.

Dada una instancia de SAT:

$f = C_1 \wedge C_2 \wedge \dots \wedge C_n$  donde  $C_i = \{x_{i1} \vee x_{i2} \vee \dots \vee x_{ik}\}$  y  $x_{i,j} = \{\text{True or False, de la forma } \neg x_{i,j} \text{ o } x_{i,j}\}$

Definamos un conjunto  $U$  y una familia de subconjuntos  $F$ .

El conjunto universo  $U$  consistirá de 3 tipos de elementos:

- $x_i$ : por cada variable  $i$
- $C_j$ : por cada cláusula  $j$
- $p_{i,j}$  por cada aparición de la variable  $i$  en la cláusula  $j$

Para el conjunto  $F$  tendremos:

- Por cada variable  $i$  creamos dos conjuntos, el conjunto verdadero  $x_i^T$  y el conjunto falso  $x_i^F$ . El conjunto verdadero contiene  $x_i$  en conjunto a las apariciones de  $i$  en cada cláusula  $j$  donde aparezca negada (los  $p_{i,j}$  donde  $i$  sea falso) y el conjunto falso contiene  $x_i$  junto a las apariciones de  $i$  en cada cláusula  $j$  donde no aparezca negada (los  $p_{i,j}$  donde  $i$  sea verdadero).

**Ejemplo:** Si  $x_1$  aparece en la cláusula 2 y  $\neg x_1$  aparece en las cláusulas 3 y 4 creamos los conjuntos:

$$x_1^T = \{x_1, p_{1,2}\} \text{ y } x_1^F = \{x_1, p_{1,3}, p_{1,4}\}$$

- Por cada elemento  $p_{i,j}$  creamos un conjunto  $\{C_j, p_{i,j}\}$
- Luego por cada  $p_{i,j}$  crearemos subconjuntos por cada uno donde sea su único elemento.  $\{p_{i,j}\}$ .

**Ejemplo de la construcción:**

$$f = \{x_1 \vee x_2 \vee \neg x_3\} \wedge \{\neg x_1 \vee x_2 \vee x_3\}$$

Creamos los conjuntos verdaderos y falsos

- $x_1^T = \{x_1, p_{1,2}\}$
- $x_1^F = \{x_1, p_{1,1}\}$
- $x_2^T = \{x_2\}$
- $x_2^F = \{x_2, p_{2,1}, p_{2,2}\}$
- $x_3^T = \{x_3, p_{3,1}\}$
- $x_3^F = \{x_3, p_{3,2}\}$

Creamos los conjuntos de cláusulas con sus  $p_{i,j}$

- $\{C_1, p_{1,1}\}, \{C_1, p_{2,1}\}, \{C_1, p_{3,1}\}$
- $\{C_2, p_{1,2}\}, \{C_2, p_{2,2}\}, \{C_2, p_{3,2}\}$
- $\{C_3, p_{1,3}\}, \{C_3, p_{2,3}\}, \{C_3, p_{3,3}\}$

y los conjuntos independientes de  $p_{i,j}$ :

$$\{p_{1,1}\}, \{p_{1,2}\}, \{p_{1,3}\}, \{p_{2,1}\}, \{p_{2,2}\}, \{p_{2,3}\}, \{p_{3,1}\}, \{p_{3,2}\}, \{p_{3,3}\}.$$

Sea el conjunto

$$F = \{\{p_{1,1}\}, \{p_{2,1}\}, \{p_{3,1}\}, \{p_{1,2}\}, \{p_{2,2}\}, \{p_{3,2}\}, \{p_{1,3}\}, \{p_{2,3}\}, \{p_{3,3}\}, x_1^T, x_1^F, x_2^T, x_2^F, x_3^T, x_3^F, \{C_1, p_{1,1}\}, \{C_1, p_{2,1}\}, \{C_1, p_{3,1}\}, \{C_2, p_{1,2}\}, \{C_2, p_{2,2}\}, \{C_2, p_{3,2}\}, \{C_3, p_{1,3}\}, \{C_3, p_{2,3}\}, \{C_3, p_{3,3}\}\} \quad (1)$$

Sea el universo

$$U = \{x_1, x_2, x_3, C_1, C_2, p_{1,1}, p_{1,2}, p_{1,3}, p_{2,1}, p_{2,2}, p_{2,3}, p_{3,1}, p_{3,2}, p_{3,3}\}$$

Definamos una instancia de Exact Set Cover donde el conjunto universo sera representado por el conjunto  $U$  y el conjunto de subconjuntos por  $F$ . Tendremos que:

$$U = \{x_i, C_j, p_{i,j}\}$$

$$F = \{x_i^T, x_i^F, \{C_j, p_{i,j}\}, \{p_{i,j}\}\}$$

para  $0 \leq i \leq n$  y  $0 \leq j \leq k$ .

Vamos a demostrar que si se tiene una asignacion de las variables que satisface la expresi3n(o sea, que resuelve SAT), entonces existe un subconjunto  $T \subseteq F$  que satisface la instancia de Exact Set Cover que se creo.

Teniendo la asignaci3n que resuelve SAT vamos a armar un subconjunto:

- Tomamos el conjunto  $x_i^T$  o  $x_i^F$  dependiendo si  $x_i$  es verdadero o falso.
- Tomamos los conjuntos  $C_j, p_{i,j}$  dependiendo si  $x_i$  se encuentra en la cl1usula  $j$ .
- Tomamos los conjuntos  $p_{i,j}$  de los restantes  $x_i$  ya que solo podemos escoger  $k$  de los subconjuntos  $C_j, p_{i,j}$ , por ser  $k$  la cantidad de cl1usulas. Para evitar la interseccion entre conjuntos.

**Podemos ver que:**

- Construimos un subconjunto que forma el universo ya que al escoger los conjuntos  $x_i^T$  o  $x_i^F$ , obtenemos todos los  $x_i$  y los  $p_{i,j}$ , donde era falso o verdadero respectivamente. Luego obtenemos los  $C_j$  de los  $C_j, p_{i,j}$  y los  $p_{i,j}$  donde se encontraban los  $x_i$  y luego completamos con los  $p_{i,j}$  restantes.
- Los subconjuntos son disjuntos entre s3 ya que al escoger los  $x_i^T$  o  $x_i^F$  se escogen los  $p_{i,j}$  donde es falso o verdadero respectivamente pues no habr1 intersecci3n con los  $p_{i,j}$  escogidos en  $C_j, p_{i,j}$  pues se seleccionar1n donde aparece ese  $x_i$  en su respectiva cl1usula que es lo contrario de los  $x_i^T$  o  $x_i^F$ .

**Ejemplo de soluci3n:**

Dada una soluci3n cualquiera de SAT, sea esta:

$$s = \neg x_1 \wedge x_2 \wedge x_3$$

cubriremos el universo, dando como resultado el subconjunto

$$T = \{x_2^T, x_1^F, x_3^T, \{C_1, p_{2,1}\}, \{C_2, p_{1,2}\}, \{p_{2,2}\}, \{p_{3,2}\}\}$$

Por tanto queda demostrado que si existe una soluci3n al problema SAT podemos construir a partir de esta una soluci3n al Exact Set Cover en tiempo polinomial.

Ahora vamos a hacer la demostraci3n en el otro sentido. Demostremos que si existe una soluci3n de Exact Set Cover entonces podemos contruir una soluci3n para SAT.

Teniendo un subconjunto  $T \subseteq F$  que de foma que la union de sus elementos formen el universo y no exista intersecci3n entre ellos. Sea esta una soluci3n de Exact Set Cover.

- Asignamos verdadero a  $x_i$  si aparece  $x_i^T$  en el subconjunto, mientras que asignamos falso a  $x_i$  si aparece  $x_i^F$ .

- Luego verificamos las clausulas, donde por cada clausula  $C_j$  al menos uno de los conjuntos  $C_j, p_{i,j}$  debe aparecer en  $T$ . Si  $C_j$  no esta cubierta por ninguno de estos conjuntos entonces la fórmula SAT no es satisfacible.

De modo general tenemos que en cada cláusula debe haber una variable en verdadero, lo que hace que se satisfaga la ecuación y por tanto la asignacion realizada resuelve el SAT.

La solución de Exact Set Cover proporciona una cobertura exacta del universo, lo que permite determinar directamente los valores de verdad de las variables en la fórmula SAT original.

Finalmente hemos demostrado que se puede construir una solución para SAT a partir de una para Exact Set Cover en tiempo polinomial, como SAT es NP-Hard entonces Exact Set Cover es NP-Hard. Finalmente como Exact Set Cover es NP y NP-Hard es también NP-Completo.

## 2.3. Algoritmo

### Backtrack

Para la solución de este problema, planteamos un algoritmo de backtrack que ite-ramos por cada conjunto y vamos agregando si no existe solapamiento. Termina si encuentra una solución y la devuelve, si no, devuelve vacío.

```
def exact_set_cover(universe, subsets):
    used_subsets = []

    def backtrack(covered, index):
        if covered == universe:
            return True
        if index >= len(subsets):
            return False

        for i in range(index, len(subsets)):
            subset = subsets[i]

            if not covered & subset:
                used_subsets.append(subset)
                if backtrack(covered | subset, i + 1):
                    return True
                used_subsets.pop()
        return False

    if backtrack(set(), 0):
        return used_subsets
    else:
        return None
```

Podemos ver que es un algoritmo bastante sencillo pero ineficiente ya que recorre todos los posibles subconjuntos.

### Análisis de Complejidad

- Búsqueda exhaustiva: La función backtrack explora todas las posibles combinaciones de subconjuntos. En el peor caso, esto implica considerar  $2^n$  combinaciones, donde  $n$  es el número de subconjuntos.
- Verificación de cobertura: En cada llamada recursiva de backtrack, se verifica si la unión de los subconjuntos seleccionados cubre el universo. Esto puede requerir hasta  $O(n * m)$  operaciones, donde  $m$  es el número de elementos en el universo.

La complejidad temporal total de la solución de fuerza bruta es  $O(2^n * n * m)$ .

### **Greedy Conjunto de Mayor Tamaño**

Como el anterior es muy ineficiente para mejorar la eficiencia del algoritmo usamos un enfoque greedy donde siempre a la hora de agregar tomamos el conjunto con mayor cantidad de elementos.

A pesar de que puede ser útil para encontrar soluciones aproximadas al problema de Exact Cover, sin embargo, no es un algoritmo polinomial en el sentido estricto y su rendimiento puede variar significativamente dependiendo de la instancia del problema. No garantiza optimalidad pero en este problema no se busca optimalidad.

Heurística intuitiva: La idea de seleccionar en cada paso el conjunto que cubre la mayor cantidad de elementos restantes parece una estrategia lógica. Reducción del espacio de búsqueda: Al eliminar los elementos cubiertos en cada iteración, se reduce significativamente el espacio de búsqueda, lo que puede acelerar la búsqueda de una solución.

Entonces, ¿cuándo funciona bien el algoritmo greedy para Cobertura Exacta?

Conjuntos con poco solapamiento: Si los conjuntos tienen pocos elementos en común, el algoritmo greedy tiene más probabilidades de encontrar una solución rápidamente. Instancias "fáciles": En instancias donde la estructura de los conjuntos favorece la estrategia greedy, el algoritmo puede encontrar una solución óptima o cercana a la óptima.

### **Resultados:**

Para un universo  $U$  de 30 elementos y un conjunto  $S$  de 75 subuniversos.

**Backtrack:** 0.03481864929199219 ms.

**Greedy Conjunto Mayor Tamaño:** 0.006000041961669922ms.

Podemos observar que el algoritmo greedy es más eficiente que el de backtrack.