Práctico 2 Parte 1 Complejidad Computacional

Julián Viera

jviera@fing.edu.uy
julviera44@gmail.com

2 de septiembre de 2021

Agenda

Fórmula booleana 2CNF

Fórmula $2CNF$: es un AND de cláusulas, en el que cada cláusula es un OR de a lo sumo 2 literales.

Ejemplo: $\Phi = (x_3 \vee \overline{x_1}) \wedge \overline{x_2} \wedge (x_1 \vee x_2)$

Se verifica que Φ es satisfactible, con la siguiente asignación de valores de verdad:

- $x_1 = 1$
- $x_2 = 0$
- $x_3 = 1$

Ejercicio 1 - Planteo del problema (7.42 de Sipser)

Se considera el lenguaje:
$2SAT = \{< \Phi > / \Phi$ es una fórmula $2CNF$ satisfactible $\}$.

Probar que $2SAT \in P$

Propiedad de las fórmulas 2CNF

Si se le ha asignado un valor de verdad al primer literal de una cláusula de una fórmula $2CNF$, puede evaluarse inmediatamente dicha cláusula:

- Si el literal vale $1 \Longrightarrow$ la cláusula evalúa en 1 independientemente del valor del segundo literal
- Si el literal vale $0 \Longrightarrow$ el segundo literal debe necesariamente valer 1 para que la cláusula evalúe en 1 y la fórmula tenga chance de ser satisfactible

Esto implica que puede eliminarse dicha cláusula de la fórmula original, obteniéndose una fórmula $2CNF$ reducida, para esa elección del primer literal de la cláusula.
Ejemplo: $\Phi = (x_3 \vee \overline{x_1}) \wedge \overline{x_2} \wedge (x_1 \vee x_2)$
Si se elige $x_3 = 1 \Longrightarrow \Phi' = \overline{x_2} \wedge (x_1 \vee x_2)$ con $x_3 = 1$
Si se elige $x_3 = 0 \Longrightarrow \Phi' = \overline{x_2} \wedge (x_1 \vee x_2)$ con $x_3 = 0, x_1 = 0$

Ejercicio 1 - una posible solución

La idea asignar valor 1 al primer literal de la primera cláusula, y evaluar lo más que se pueda dicha fórmula (lo que puede implicar recorrerla varias veces, ya que al evaluar pueden ir quedando fijados los valores de otros literales de la fórmula).
Se pueden presentar tres casos:

1. Se simplifica la primera cláusula y eventualmente otras, y queda una fórmula reducida que no se puede evaluar más. En este caso da valor 1 al primer literal de la primera cláusula de la fórmula reducida y se repite el procedimiento.
2. Todas la cláusulas de la fórmula evalúan en TRUE, lo que prueba que la fórmula es satisfactible.
3. Alguna cláusula evalúa en 0. En este caso se asigna valor 0 al literal al que inicialmente se le habia asignado 1, y se repite la evaluación. Si se llega a que alguna cláusula evalúa en 0 entonces la fórmula original no es satisfactible, pues ya se probaron los dos valores de verdad que puede tener un literal.

Ejemplos del algoritmo propuesto

1. $\Phi_0 = (x_3 \vee \overline{x_1}) \wedge \overline{x_2} \wedge (x_1 \vee x_2)$
   Se elige $x_3 = 1 \Longrightarrow (x_3 \vee \overline{x_1}) = 1 \Longrightarrow \Phi_1 = \overline{x_2} \wedge (x_1 \vee x_2)$
   Elegimos ahora $\overline{x_2} = 1 \Longrightarrow \Phi_2 = (x_1 \vee x_2)$
   Como $x_2 = 0 \Longrightarrow$ debe ser $x_1 = 1$ para satisfacer Φ_2.
   $\Phi_2 = 1 \Longrightarrow \Phi_1 = 1 \Longrightarrow \Phi_0 = 1$ con $(x_1, x_2, x_3) = (1, 0, 1)$.

2. $\Phi_0 = (x_3 \vee \overline{x_1}) \wedge \overline{x_2} \wedge (x_2 \vee \overline{x_3})$
   Se elige $x_3 = 1 \Longrightarrow (x_3 \vee \overline{x_1}) = 1 \Longrightarrow \Phi_1 = \overline{x_2} \wedge (x_2 \vee \overline{x_3})$
   Como $x_3 = 1 \Longrightarrow$ debe ser $x_2 = 1$ para satisfacer Φ_1.
   $\Longrightarrow \Phi_2 = \overline{x_2}$. Como $x_2 = 1 \Longrightarrow \Phi_2 = 0$
   Volvemos a la fórmula original Φ_0 y elegimos $x_3 = 0 \Longrightarrow$ debe ser $x_1 = 0$ para satisfacer la primera cláusula.
   $\Longrightarrow \Phi_1 = \overline{x_2} \wedge (x_2 \vee \overline{x_3})$. Como $x_3 = 0 \Longrightarrow$ la última cláusula evalúa en 1 y se puede eliminar $\Longrightarrow \Phi_2 = \overline{x_2}$
   Elegimos ahora $\overline{x_2} = 1 \Longrightarrow \Phi_2 = 1 \Longrightarrow \Phi_1 = 1 \Longrightarrow \Phi_0 = 1$ con $(x_1, x_2, x_3) = (0, 0, 0)$.

Pseudocodigo del algoritmo

$\phi^* = \phi$

$\boxed{1}$ $\bar{\phi} = \phi^*$ guardo la subfórmula actual

Elegir el primer literal de la primera cláusula de ϕ^* y asignarle el valor 1

Evaluar en profundidad ϕ^*, descartando de ϕ^* todas las cláusulas que evalúan en 1

Si alguna cláusula evalúa en 0, ir a $\boxed{2}$.

Si $\phi^* = \varnothing$ PARAR, $\phi \in$ 2SAT

Si $\phi^* \neq \varnothing$ volver a $\boxed{1}$

$\boxed{2}$ $\phi^* = \bar{\phi}$ recupero la subfórmula guardada

Elegir el primer literal de la primera cláusula de ϕ^* y asignarle el valor 0

Evaluar en profundidad ϕ^*, descartando de ϕ^* todas las cláusulas que evalúan en 1

Si alguna cláusula evalúa en 0 PARAR, $\phi \notin$ 2SAT

Si $\phi^* = \varnothing$ PARAR, $\phi \in$ 2SAT

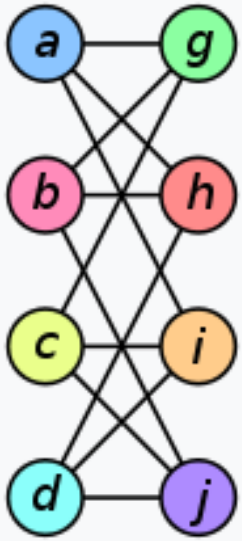
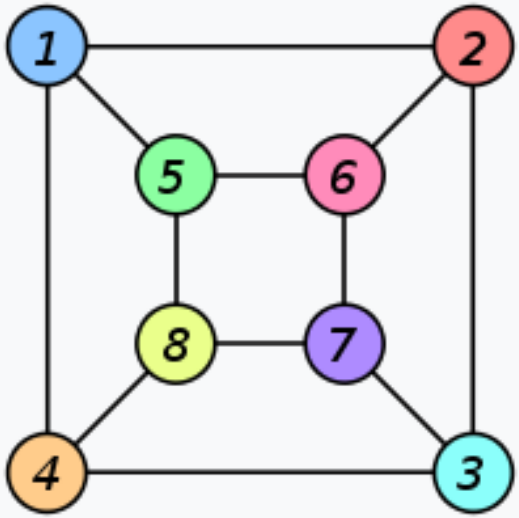
Si $\phi^* \neq \varnothing$ volver a $\boxed{1}$

Complejidad en el tiempo del algoritmo propuesto

Tomamos como medida del tamaño de la entrada el número de cláusulas n de la fórmula $2CNF$.
La observación clave es que en cada paso se elimina al menos una cláusula, la correspondiente al literal cuyo valor de verdad se asigna.
Cada evaluación en profundidad de la fórmula requiere en el peor caso tiempo $\mathcal{O}(n^2)$ $\left(\sum_{k=1}^{n}(n-k)\right)$.
Como hay a lo sumo n cláusulas en la fórmula, y cada primer literal de una cláusula puede tomar el valor 0 o 1, el tiempo total del algoritmo en el peor caso es $\mathcal{O}(n^3)$.
$\Longrightarrow 2SAT \in P$ ■

Grafos isomorfos

Dos grafos simples G y H con el mismo número de nodos n y el mismo número de aristas m son *ISOMORFOS* si los nodos de G pueden ser reordenados de tal modo que el grafo reordenado quede idéntico a H. Ejemplo:

Grafo G	Grafo H	Un isomorfismo entre G y H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

Ejercicio 2 - Planteo del problema (7.11de Sipser)

Se considera el lenguaje:

$ISO = \{< G, H > / G \text{ y } H \text{ son grafos isomorfos } \}$

Probar que $ISO \in NP$

Definición de la clase *NP*

Un lenguaje $L \subseteq \{0,1\}^*$ está en **NP** si existe un polinomio $p : \mathbb{N} \longrightarrow \mathbb{N}$ y una MT de tiempo polinomial M (llamada *VERIFICADOR* de L) de modo que $\forall x \in \{0,1\}^*$:

$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)}$ que verifica $M(x,u) = 1$

Si $u \in \{0,1\}^{p(|x|)}$ y $M(x,u) = 1 \Longrightarrow$ se dice que u es un *CERTIFICADO* para x.

Un posible certificado para el problema *ISO*

Suponemos que G tiene un número de vértices n.
Un certificado u para el problema *ISO* queda definido por la función de mapeo f de los vértices de G en los vértices de H. Es decir que dada la lista o vector V de los n vértices de G, el certificado u es otra lista o vector de largo n dada por $u = f(V)$.
La función de mapeo f debe ser **biyectiva**.
Es claro que el tamaño del certificado u es polinomial en el tamaño de la entrada n (es *lineal* en n), tal como lo requiere la definición de certificado.

Un algoritmo *VERIFICADOR* para el problema *ISO*

La idea para implementar el verificador del lenguaje *ISO* consiste en considerar todos los $\frac{n(n-1)}{2}$ pares de nodos de G y verificar que:

1. Si hay arista (v, w) entre dos nodos v y w de G, entonces debe existir arista $(f(v), f(w))$ entre los nodos $f(v)$ y $f(w)$ de H.
2. Si no hay arista (v, w) entre dos nodos v y w de G, entonces no debe existir arista $(f(v), f(w))$ entre los nodos $f(v)$ y $f(w)$ de H.

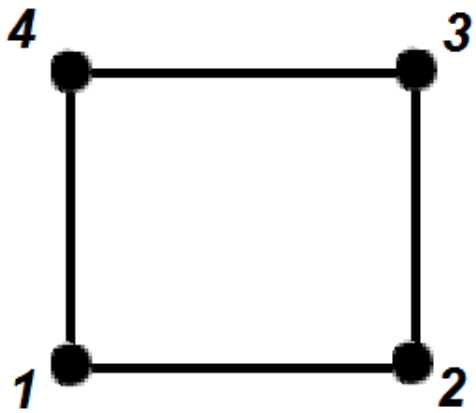
Si se verifican ambas condiciones, los grafos son isomorfos.

Pseudocodigo del algoritmo verificador

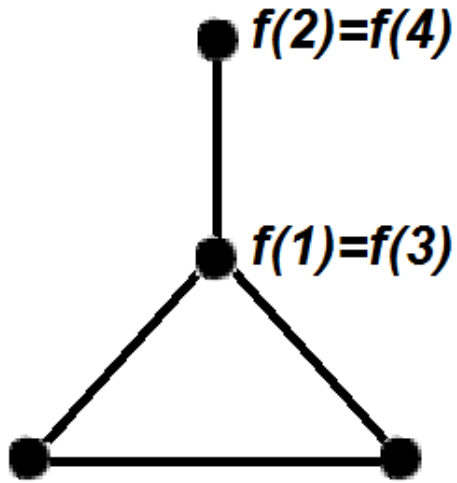
```
Chequear que el certificado u tenga todos los elementos distintos
Si no lo son, PARAR y devolver 0
Para cada vertice w de G
    Para cada vertice v de G distinto de w
        Si hay arista (w,v) en G
            Si no hay arista (f(w),f(v)) en H, PARAR y devolver 0
            fin Si
        sino
            Si hay arista (f(w),f(v)) en H, PARAR y devolver 0
            Fin Si
        Fin Si
    Fin Para
Fin Para
Devolver 1
```

Porqué es necesario chequear que los elementos de u sean distintos

Si los elementos del certificado u no son todos distintos (hay vértices de G con el mismo vértice correspondiente en H), la función de mapeo no es biyectiva. Si no se controla la biyectividad de u, el algoritmo puede fallar, como lo muestra el ejemplo de la figura. En este caso G y H no son isomorfos y el algoritmo sin dicho control devolvería que sí lo son.



G



H

Complejidad en el tiempo del algoritmo verificador

Tomamos como medida del tamaño de la entrada el número de nodos n de los grafos G y H.
Asumimos que los grafos se representan mediante una matriz de aydacencia nxn.
El chequeo de que el mapeo es biyectivo se puede hacer en $\mathcal{O}(n)$, recorriendo el certificado u de largo n.
El bucle externo se ejecuta n veces, una vez por cada nodo del grafo G.
El bucle interno se ejecuta también n veces, una vez por cada posible nodo vecino al nodo del bucle externo.
La comprobación de existencia o inexistencia de arista lleva $\mathcal{O}(1)$.
$\Longrightarrow$ el algoritmo verificador es $\mathcal{O}(n^2)$
$\Longrightarrow ISO \in NP$ ■

Comentarios sobre el problema *ISO*

1. Se desconoce si el problema está en P o si es NP-completo.
2. Dos grafos G y H con matrices de adyacencia A y B respectivamente son isomorfos si y solo si existe una matriz permutación P tal que $B = PAP^t$.
   Matriz permutación: matriz de ceros y unos con un solo uno por fila y columna.
3. El isomorfismo de grafos es una relación de equivalencia, por lo que existen clases de grafos isomorfos para cada par (n, m).

Problema de la parada (halting problem)

Se define la función *HALT* como aquella que tiene como entrada una pareja (α, x) y devuelve 1 si y solo si la MT M_α, representada por el string α, PARA con entrada x en un número finito de pasos.

Ejercicio 3 - Planteo del problema (2.8 de Arora)

1. Probar que el lenguaje $HALT$ es $NP - hard$
2. ¿Es $HALT$ $NP - completo$?

Para demostrar ① hay que probar que dado un lenguaje $L \in NP \Longrightarrow L \leq_P HALT$.

Reducción de tiempo polinomial de Karp ($\leq_P$)

Un lenguaje $L \subseteq \{0,1\}^*$ se dice que es reducible en tiempo polinomial a un lenguaje $L' \subseteq \{0,1\}^*$, denotado como $L \leq_P L'$, si existe una función computable en tiempo polinomial $f : \{0,1\}^* \to \{0,1\}^*$ tal que:
$\forall x \in \{0,1\}^*, x \in L \Leftrightarrow f(x) \in L'$

Idea para reducir $L \in NP$ al problema *HALT*

Dados $L \in NP$ y una instancia x de L, la función de mapeo $f(x)$ se va a basar en el verificador M que sabemos que existe para L, por pertenecer L a NP, y en el polinomio p que determina el largo del certificado u para x.
La idea es construir otra MT M^*, la cual dado x tiene un bucle que genera secuencialmente todos los certificados u posibles para x y los verifica invocando al verificador M.
Si para algún certificado u el verificador M devuelve 1, entoces M^* para (devolviendo 1 por ejemplo).
Si M devuelve 0 para todos los certificados posibles de largo $p(|x|)$ (que son $2^{p(|x|)}$), M^* no para y se queda en un bucle infinito.
Si llamamos α a la codificación en bits de esta máquina M^*, la función de mapeo de la reducción va a estar dada por $f(x) = (\alpha, x)$.

Pseudocodigo de la MT M^* que implementa la reducción

Generar un primer certificado u a partir de x y p

Repetir

Calcular $M(x,u)$

Si $M(x,u)=1 \Rightarrow$ PARAR y devolver 1

Calcular el siguiente certificado u en la secuencia cíclica de los $2^{p(|x|)}$ posibles certificados del conjunto $\{1,0\}^{p(|x|)}$

Fin Repetir

Análisis de la reducción propuesta

1. La función $f(x) = (\alpha, x)$ se computa en tiempo polinomial (lineal en $n = |x|$), ya que realizar la codificación α de M^* es independiente de x, dependiendo sólo de p y M.
2. $x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)}$ que verifica $M(x,u) = 1$
   $\Leftrightarrow M^*$ PARA en un número finito de pasos
   $\Leftrightarrow HALT(\alpha, x) = 1$

¿Es *HALT* $NP - completo$?

HALT no es $NP - completo$ porque $HALT \notin NP$.
Para ver esto supongamos por absurdo que $HALT \in NP$. Entonces por definición de la clase *NP* existe un verificador M' para *HALT*.
$\Rightarrow$ dada una entrada (α, x) para *HALT*, puedo generar todos los $2^{p(|\alpha|+|x|)}$ certificados posibles de largo $p(|\alpha| + |x|)$ para esa entrada, y usando el verificador M' puedo probar dichos certificados y decidir *HALT* para esa entrada.
Pero sabemos que no existe una MT que compute *HALT* pues *HALT* es un problema indecidible $\Rightarrow HALT \notin NP$.

Ejercicio 7 - Planteo del problema Conjunto Diverso

Una tienda que trata de analizar el comportamiento de sus clientes a menudo mantiene una tabla bidimiensional A, donde las filas corresponden a sus clientes y las columnas a los productos que vende. El elemento $A(i, j)$ especifica la cantidad del producto j que ha sido comprada por el cliente i. Un ejemplo de tabla se muestra a continuación.

Cliente	Detergente	Cerveza	Pañales	Arena para gatos
Raj	0	6	0	3
Alanis	2	3	0	0
Chelsea	0	0	0	7

Se dice que un subconjunto S de clientes es ***diverso*** si ningún par de clientes en S ha comprado nunca el mismo producto, es decir que para cada producto a lo sumo un cliente de S lo ha comprado alguna vez. Un conjunto diverso de clientes resulta útil por ejemplo como grupo objetivo para estudios de mercado. En la tabla ejemplo el conjunto de clientes {Alanis, Chelsea} es diverso.

Se define el problema de Conjunto Diverso como sigue: dada una tabla $A_{m \times n}$ con m clientes y n productos y un número $k \leq m$, ¿existe un subconjunto de al menos k clientes que sea diverso?

Probar que Conjunto Diverso es NP-completo.

La clase de los lenguajes NP-completos

Un lenguaje L es NP-completo si:

1. $L \in NP$
2. L es NP-hard

Hay que probar entonces que *ConjuntoDiverso* (en adelante *CD*) satisface ① y ②

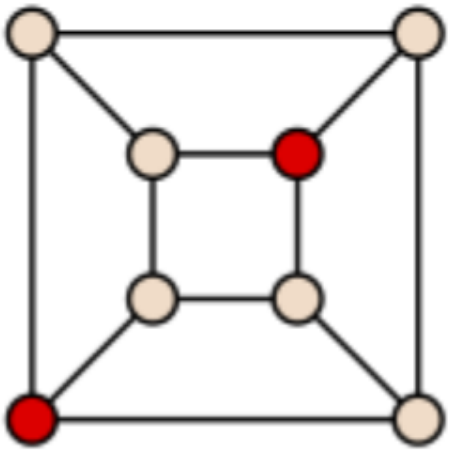
Prueba de que $CD \in NP$

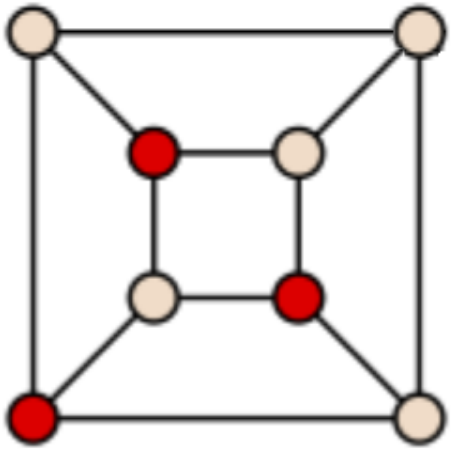
Sea A la tabla mxn que define la instancia del problema.
Un cerificado para CD consiste en un conjunto C de clientes.
El algoritmo verificador lo primero que comprueba es que $k \leq |C|$.
Esto lo hace en $\mathcal{O}(m)$, contando los elementos de C.
Para verificar que C es un conjunto diverso de clientes, se recorre C una vez por cada producto (por cada columna de A) verificando que haya a lo sumo un único comprador de ese producto en C.
Esto se puede hacer en tiempo $\mathcal{O}(mn)$, polinomial en el tamaño de la entrada A.
$\Longrightarrow CD \in NP$

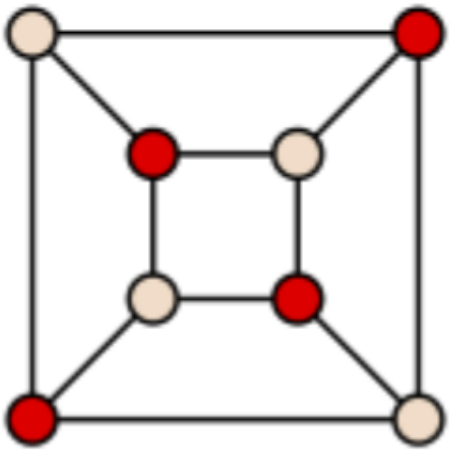
Prueba de que CD es $NP - hard$

Para probar que CD es NP-hard, se mostrará que existe una reducción polinomial del problema NP-completo *independent set* (en adelante IS) a CD,es decir que $IS \leq_P CD$.

$$IS = \{< G, k > / \exists S \subseteq V(G), |S| \geq k \ y \ \forall u, v \in S, (u, v) \notin E(G)\}$$







Reducción del problema IS al problema CD

Una instancia $< G, k >$ de IS se transforma en una instancia $< A, k' >$ de CD de la siguiente forma:

- Cada nodo v_i de G se transforma en un cliente c_i de A
- Por cada arista (v_i, v_j) de G se incluye en A un producto p_{ij} con cantidad vendida 1 para los clientes c_i y c_j y cantidad vendida 0 para todo otro cliente c_k, $k \neq i, j$
- Se toma $k' = k$

Se observa que la tabla A_{nxm} de la instancia de CD se puede construir en tiempo polinomial $\mathcal{O}(nm)$ a partir de la instancia $< G, k >$ de IS, siendo n el número de nodos y m la cantidad de aristas de G.

Correctitud de la reducción propuesta

Probaremos que $< G, k >$ es una instancia SI de $IS \Leftrightarrow < A, k >$ es una instancia SI de CD.

$< G, k >$ es una instancia SI de $IS \Leftrightarrow$ existe un independent set S de G con $|S| \geq k \Leftrightarrow$ no hay arista (v_i, v_j) entre dos nodos $v_i, v_j \in S \Leftrightarrow$ no hay producto p_{ij} en las columnas de $A \Leftrightarrow$ los clientes $c_i, c_j \in S'$, con $S' = f(S)$, no han comprado el mismo producto $\Leftrightarrow S'$ es un conjunto diverso para $< A, k >$, con $|S'| \geq k \Leftrightarrow < A, k >$ es una instancia SI de CD